Volkan AKTAŞ

EĞİTİM VİDEOLARI İÇİN
KODLAB.com



30.
BASKI
ÇOK SATAN

HER YÖNÜYLE C#9.0

KODLAB®
www.kodlab.com

HER YÖNÜYLE C# 9.0

Volkan Aktaş

KODLAB®
Yayın Dağıtım Yazılım ve Eğitim
Hizmetleri San. ve Tic. Ltd. Şti.

HER YÖNÜYLE C# 9.0

VOLKAN AKTAŞ

ISBN 978-605-9118-87-3
Yayıncılık Sertifika No: 13206

30. Baskı: 2021

Genel Yayın Yönetmeni
Gizem Atlı

Editör
İrem Soylu

Grafik Tasarım
Tamer Takmaz

Satış ve Pazarlama
Can Üstünel
Yunus Emre Üstünel

Baskı: İnkılâp Kitabevi Yayın San. Tic. A.Ş. Sertifika No: 44066

Bu kitabın bütün yayın hakları Kodlab Yayın Dağıtım Yazılım ve Eğitim Hizmetleri San. ve Tic. Ltd. Şti.'ne aittir. Yayınevimizin yazılı izni olmaksızın kısmen veya tamamen alıntı yapılamaz, kopya çekilemez, çoğaltılamaz ve yayınlanamaz.

KODLAB Yayın Dağıtım Yazılım ve Eğitim Hizmetleri San. ve Tic. Ltd. Şti.
15 Temmuz Mh. 1481. Sk. No : 44/A Bağcılar / İSTANBUL

tel: 0(212) 514 55 66 **web:** www.kodlab.com
0(212) 514 66 61 **e-posta:** bilgi@kodlab.com

VOLKAN AKTAŞ

1976 Mersin doğumlu olup aslen Diyarbakır Ergani'lidir. Lise öğrenimini Mersin Atatürk Teknik Lisesi Bilgisayar Bölümünden, üniversite öğrenimini Gazi Üniversitesi Bilgisayar Öğretmenliği bölümünden mezun olarak tamamlamıştır. Yüksek Lisansını ise Mersin Üniversitesi Bilgisayar ve Öğretim Teknolojileri fakültesinde tamamlamıştır. 1999-2003 yılları arasında Şanlıurfa Merkez Endüstri Meslek Lisesinde bilgisayar öğretmeni olarak görev yapmış olup, 2003 yılından bu yana da Mersin'de görev yapmaktadır. Hâlen Mersin Kadri Şaman MTSO Mesleki ve Teknik Anadolu Lisesinde görev yapmaktadır.

Visual Basic .NET, C#, Visual C++, Python, Go dilleri ve Arduino, Raspberry Pi ve ESP kartları ile aktif olarak uygulamalar geliştirmektedir.

Ayşe AKTAŞ ile evli olup Talha ile Sümeyye adlarında iki çocuğu vardır.

Yazarımız, uygulamalarını *https://github.com/volkanaktas* adresinde yayınlamaktadır.

Bu kitabı rahmetli babam

Osman AKTAŞ'a ithaf ediyorum.

ÖN SÖZ

Bu kitap yıllardır C# .NET ile geliştirdiğim projelerin, kullandığım uygulamaların ve bu süreçte edindiğim tecrübelerin birleşiminden oluşmuştur. Elbette bu süreçte gerek öğrencilerime ders anlatmak üzere hazırladığım ders notları gerekse çeşitli web sitelerinde yazdığım makaleler de etkili olmuştur.

.Net yapısının sağladığı güçlü altyapı birçok projemi C#. NET ile geliştirmemde etkili olmuştur. .NET altındaki diğer dillerden farklı olarak tamamen .NET için geliştirilen bir dil olması nedeniyle, .NET altyapısını en iyi olarak kullanabilen dildir diyebiliriz. C, C++ ve Java dillerinin güçlü yanlarını bünyesinde toplamıştır.

Elbette bir programcının tek bir dile bağımlı olması düşünülemez ve düşünülmemelidir. Benim görüşüme göre iyi bir programcı birden fazla dil bilmeli ve kullanacağı proje tipine göre dil seçimi yapmalıdır. Çünkü programlama dillerinin birbirlerine göre daha güçlü olduğu kullanım alanları ve sahip olduğu bazı nitelikler vardır.

Bu kitapta genel olarak C# .Net temel ve orta seviye konularıyla birlikte C# .NET 9.0 sürümüne ait bir kısım yeni özelliklere yer vermeye çalıştım. Dolayısı ile gerek C# .NET'i yeni öğrenmeye başlayan gerekse daha önceden C# .NET ile çalışmış bir okuyucu, kitapta kendisine göre konular bulacaktır.

Kitapta elden geldiği kadar çok örneğe yer vermeye çalıştım ve hemen her konuyu örneklerle destekleyerek anlattım. Anlatılan her örnek denenmiş ve çalıştığı görüldükten sonra kitaba eklenmiştir. Kitapta konu sıralaması programcılık mantığının doğru bir şekilde gelişimine katkıda bulunacak şekilde yapılmıştır. Temel seviye konularda konsol uygulamalarına ağırlık verilmiş, bölümler ilerledikçe uygulamalarda Windows form uygulamalarına yer verilmiştir. Windows form uygulamalarında yalnızca anlatılan konuya dair kontroller kullanılmış gereksiz kontrollerin kullanımından kaçınılmıştır. Ayrıca form üzerindeki kontrollerin tasarımı ve özellikleri gibi bazı ayrıntılar mümkün oldukça atlanarak kullanıcı gereksiz ayrıntılarla boğulmamıştır.

Kitabımın, tüm okuyucularına faydalı olmasını temenni ederim.

TEŞEKKÜR

Bu kitabın yazım sürecinde öncelikle anneme, kardeşime, bana destek olan ve sabır gösteren eşime ve çocuklarıma teşekkür ederim.

Ayrıca içerik ve örneklerin kontrolünde destek olan çalışma arkadaşlarım Bünyamin KARAHAN, Ömer YİĞİT, Mehmet ÖZTOPRAK, Hale KATMER, Şerif Mehmet ÖZOĞUL, Serhat KARAKULLUKÇU, Ahmet UYSAL, Uğur DELİKURT, İrfan MERGAN'a, yazım sürecinde bana destek olan Aziz BEKTAŞ, Tahsin SEVİNÇ ve Vedat TAYLAN'a, manevi desteklerini eksik etmeyen Mustafa AKSU, Enver ARSLAN, Ayhan ÇELİK, Ahmet IŞIK, Said KOTAY, Murat ÇIRAK, Serkan ŞENDAĞ'a ve adlarını burada sayamadığım dostlarıma ve öğrencilerime teşekkür ederim.

İÇİNDEKİLER

1 C# .NET KOD GELİŞTİRME ORTAMI 1
Kod Geliştirme Ortamı Bileşenleri 5
Menü Window 5
Menü Çubuğu 5
Araç Simgesi Çubuğu 7
Solution Explorer 7
Properties Window 10
Server Explorer 11
Error List 12
ToolBox 13
Notifications 14
Live Share 16

2 KOD YAZMAYA GİRİŞ 19
.NET Framework Nedir? 20
Bir Programın Derleme Süreci 23
Yeni Proje Açmak 26
Kayıtlı Projeleri Açmak 37
Projeleri Kaydetmek 38
Projeleri Derlemek 40
Proje Özelliklerini Ayarlamak 45
Uygulama Geliştirme 72

3 KONSOL EKRANI 89
Konsol Ekranında Değer Yazma ve Okuma 91
Write ve WriteLine Metotları 91
Read ve ReadLine Metotları 94
BackgroundColor ve ForegroundColor 100
Metotları 100

ConsoleKeyInfo Yapısı 101
SetCursorPosition Metodu 106
Konsol Uygulamalarından Çıkış 107

4 SABİTLER VE DEĞİŞKENLER 109
Sabitler 109
Değişkenler 115
Değişken Tipleri 118
Tanımlandıkları Yerlere Göre Değişken Türleri 125
Değer ve Referans Tipi Değişken Tanımlama 133
Boxing ve Unboxing 135
Değişken Tiplerini Öğrenmek 139
1) GetType Metodu 139
2) TypeOf Metodu 140
Değişken Tipleri Arasında 141
Dönüşüm Yapmak 141
1) Parse Metodu 145
2) TryParse Metodu 148
3) Convert Metodu 151
Kullanıcı Tanımlı Veri Tipleri 154
1) Struct (Yapı) Oluşturmak 154
Struct İçerisinde Struct Tanımlamak 163
2) Enum Oluşturmak 167
IsDefined Metodu 167
Enum İçerisindeki Elemanlara Erişim 168
System.Enum Sınıfı 177
GetName Metodu 177
GetNames Metodu 179
GetValues Metodu 179
Parse Metodu 180

5 OPERATÖRLER **181**
Aritmetik Operatörler 182
Aktarma Operatörleri 183
Mantıksal Operatörler 184
& (Ve) Operatörü 184
| (Veya) Operatörü 185
&& (VeDeğil) Operatörü 185
|| (VeyaDeğil) Operatörü 185
^ (Özel veya) Operatörü 185
! (Değil) Operatörü 185
?? (Null Coalescıng) Operatörü 186
?: (Koşul) Operatörü 187
Karşılaştırma Operatörleri 187
Birleştirme Operatörleri 188

6 KONTROL YAPILARI **189**
if Yapısı 189
Switch Case Yapısı 200
Pattern Matching 204
Pattern Matching'in switch-case Yapısı 206
İçerisinde Kullanımı 206
Generic Pattern Matching 212
Switch Expressions 214

7 DÖNGÜLER **219**
for Döngüsü 219
foreach Döngüsü 231
while Döngüsü 233

8 DİZİLER VE KOLEKSİYONLAR **239**
Diziler (Arrays) 240
Boyutlarına Göre Diziler 242

Tek Boyutlu Diziler 242
Çok Boyutlu Diziler 243
Dikdörtgensel Diziler (Rectangular Array) 243
Düzensiz Diziler (Jagged Array) 245
Dizileri Yeniden Boyutlandırma 247
Dizi Değişkenlerinde Kullanılan Metodlar 250
CreateInstance Metodu 250
IsArray Metodu 251
SetValue Metodu 252
GetValue Metodu 253
Join Metodu 255
Copy Metodu 256
Diziler İçerisinde Arama İşlemleri 259
Contains Metodu ile Arama Yapma 259
IndexOf Metodu ile Arama Yapma 260
LastIndexOf Metodu ile Arama Yapma 262
BınarySearch Metodu ile Arama Yapma 263
Koleksiyonlar 264
Non Generic (Generic Olmayan) Koleksiyonlar 265
ArrayList Sınıfı 265
Add Metodu 265
Insert Metodu 267
Sort Metodu 269
Count Özelliği 271
Remove Metodu 272
RemoveAt Metodu 273
Clear Metodu 274
Reverse Metodu 274
Capacity Özelliği 275
GetRange Metodu 276

AddRange Metodu 277
InsertRange Metodu 278
RemoveRange Metodu 279
CopyTo Metodu 279
Clone Metodu 282
ToArray Metodu 284
Hashtable Sınıfı 284
SortedList Sınıfı 290
Generic Koleksiyonlar 296
List Sınıfı 296
Stack (Yığın) Sınıfı 301
Queue (Kuyruk) Sınıfı 303
LinkedList (Bağlı Liste) Sınıfı 305
Dıctionary (Sözlük) Sınıfı 312
Dictionary Inıtıalızer 319
SortedDictionary Sınıfı 319
SortedSet Sınıfı 321
HashSet Sınıfı 324
Specialized (Özelleştirilmiş) 327
Koleksiyonlar 327
ListDictionary Sınıfı 327
HybridDictionary Sınıfı 330
OrderedDictionary Sınıfı 333
StringCollection Sınıfı 336
.Net Core Projelerinde Kullanılan Dizi ve Koleksiyon Metotları 338
Index Yapısı 338
Range Yapısı 339
Span<T> Yapısı 341

9 HATALARI BULMAK VE KONTROL ETMEK 343
Exception Nesnesi 344
Finally Deyimi 349
Throw Deyimi 350
Throw Expressions 352
Exception.Data Özelliği 353
Exception Filters 355
Kullanıcı Tanımlı Exception Nesnesi Oluşturmak 356

10 FORMLAR VE KONTROLLER 359
Kontrollerin Yerleşimi ve Düzeni 362
Formlar 364
Formlar Arası Geçiş 364
Form Nesnesinin Özellikleri 371
Form Boyutu ile İlgili Özellikler 372
Form Olayları 373
Owned Formlar 380
Diğer Kontroller 381
Label Kontrolü 381
LinkLabel Kontrolü 382
TextBox Kontrolü 383
TextBox Kontrolünün Özellikleri 384
TextBox Kontrolünün Metodları 386
TextBox Kontrolünün Olayları 389
Button Kontrolü 390
Button Kontrolünün Özellikleri 390
Button Kontrolünün Olayları 390
ComboBox Kontrolü 391
ComboBox Kontrolünün Özellikleri 391
ComboBox Kontrolünün Metodları 392
ListBox Kontrolü 394

Drag And Drop (Fare ile Sürükleme ve Bırakma) Olayları 397
TreeView Kontrolü 399
TreeView Kontrolünün Özellikleri 400
TreeView Kontrolünün Metodları 401
TreeView Kontrolünün Olayları 402
CheckBox Kontrolü 407
CheckBox Kontrolünün Özellikleri 407
CheckBox Kontrolünün Olayları 407
CheckedListBox Kontrolü 409
CheckedListBox Kontrolünün Özellikleri 409
CheckedListBox Kontrolünün Metodları 409
RadıoButton Nesnesi 411
RadıoButton Kontrolünün Özellikleri 411
GroupBox Kontrolü 412
ImageList Kontrolü 414
ImageList Kontrolünün Özellikleri 414
ListView Kontrolü 416
ListView Kontrolünün Özellikleri 416
Panel Kontrolü 424
FlowLayoutPanel Kontrolü 426
FlowLayoutPanel Kontrolünün Özellikleri 426
TableLayoutPanel Kontrolü 429
TabControl Nesnesi 433
TabControl Kontrolünün Özellikleri 433
TabControl Kontrolünün Metodları 434
TabControl Nesnesi İçerisine 435
Sekme Eklemek ve Silmek 435
Timer Kontrolü 441
Timer Kontrolünün Özellikleri 441
Timer Kontrolünün Metodları 441
Timer Kontrolünün Olayları 441

ProgressBar Kontrolü 452
ProgressBar Kontrolünün Özellikleri 452
PictureBox Kontrolü 452
PictureBox Kontrolünün Özellikleri 452
PictureBox Kontrolünün Metodları 453
MenuStrip Kontrolü 455
MenuStrip Kontrolünün Özellikleri 455
MenuStripTasks 456
ContextMenuStrip Kontrolü 460
NotifyIcon Kontrolü 462
NotifyIcon Kontrolünün Özellikleri 462
NotifyIcon Kontrolünün Metodları 462
ToolStrip Kontrolü 464
ToolStrip Kontrolünün Özellikleri 464
StatusStrip Kontrolü 470
ToolTip Kontrolü 474
Tooltip Kontrolünün Özellikleri 474
Tooltip Kontrolünün Metodları 476
ErrorProvider Kontrolü 479
ErrorProvider Kontrolünün Özellikleri 479
ErrorProvider Kontrolünün Metodları 479
HelpProvider Kontrolü 481
HelpProvider Kontrolünün Özellikleri 481
HelpProvider Kontrolünün Metodları 482
DateTımePicker Kontrolü 485
DateTımePicker Kontrolünün Özellikleri 485
DateTımePicker Kontrolünün Olayları 488
MonthCalendar Kontrolü 489
MonthCalendar Kontrolünün Özellikleri 489
MonthCalendar Kontrolünün Metodları 492

MonthCalendar Kontrolünün Olayları 492
DrıveListBox Kontrolü 493
DriveListBox Kontrolünün Özellikleri 493
DirListBox Kontrolü 493
DirListBox Kontrolünün Özellikleri 493
FileListBox Kontrolü 493
FileListBox Kontrolünün Özellikleri 494

11 DİYALOG PENCERELERİ 497
Inputbox (Giriş Penceresi) 497
Messagebox (Mesaj Penceresi) 500
File Dialog Pencereleri 506
Color Dialog 506
Folder Browser Dialog 507
Font Dialog 509
Open File Dialog 510
Save File Dialog 514

12 STRING İŞLEMLERİ 517
String Sınıfı 517
ToString Metodu 517
Copy Metodu 518
Concat Metodu 519
Empty Metodu 520
Compare Metodu 520
Split Metodu 525
Join Metodu 528
IndexOf, IndexOfAny, LastIndex ve 529
LastIndexOfAny Metodları 529
Substring Metodu 531
Replace Metodu 532
ToUpper ve ToLower Metodları 533

PadLeft ve PadRight Metodları 534
Strıngbuilder Sınıfı 536
Strıngbuilder Sınıfı Özellikleri 538
Length Özelliği 538
Capacity Özelliği 538
Ensurecapacity Özelliği 538
Maxcapacity Özelligi 540
Stringbuilder Sınıfı Metodları 540
Append ve Appendline Metodları 540
Appendformat Metodu 541
Remove Metodu 544
Replace Metodu 545
Clear Metodu 546

13 TARİH-ZAMAN METODLARI 547
DateTime Nesnesinin Özellikleri 547
Tarih ve Zaman Üzerinde Ekleme ve 550
Çıkarma Yapmak 550
TimeSpan Nesnesi 551
Tarih ve Zamanları Karşılaştırmak 555
Strıng Sınıfını Kullanarak 557
Biçimlendirme Yapmak 557
Standart Date and Time Format Strings 558
Custom Date and Time Format Strings 559

14 MATEMATİKSEL METODLAR 561
Math Sınıfı 562
String Sınıfı Kullanarak Biçimlendirme Yapmak 565
Standart Numeric Format Strings 566
Custom Numeric Format Strings 567
system.numerics Sınıfı 568
BigInteger Yapısı 568

BigInteger Yapısı Özellikleri 569
BigInteger Yapısı Metodları 570
Complex Yapısı 572
Complex Yapısı Özellikleri 572
Complex Yapısı Metodları 573

15 REGULAR EXPRESSIONS (DÜZENLİ İFADELER) 575
Özel Karakterler (Special Characters) 575
Yazdırılmayan Karakterler 576
(Non-Printable Characters) 576
Regex Sınıfı ve Metodları 576
IsMatch Metodu 576
Match Metodu 578
Matches Metodu 581
Replace Metodu 585
Split Metodu 585

16 RESOURCE DOSYALARI 587
Resource Dosyası İşlemleri 592
ResourceManager 593
ResXResourceReader 593
GetEnumerator Metodu 593
ResXResourceWriter 593
AddResource Metodu 593

17 SYSTEM.DRAWING SINIFI 599
Graphics Sınıfı 599
Form Üzerine Yazı Çizimi 599
Form Üzerine Şekil Çizimi 604
Şekil ve Yazının Birlikte Kullanımı 613
Image Sınıfı 615

18 YAZDIRMA İŞLEMLERİ **627**
PrintDocument Sınıfı 627
PrintDocument Sınıfının 627
Özellik ve Metotları 627
PrintDialog Sınıfı 639
PrintDialog Nesnesi 639
PrintPreviewDialog Nesnesi 641
PageSetupDialog Nesnesi 647

19 METOTLAR **651**
Değer Döndürmeyen Metotlar 651
Metot İçerisinde Parametre Kullanımı 653
Değer Döndüren Metotlar 658
Params IEnumerable 662
Tuple Deconstruction 664
Tuple İle Geriye Çoklu Değer Döndürme 668
ref return 670
ref Conditional 671
in Parametreler 673
ref readonly return 674
Kontrollere Metot Bağlamak 675
(Event Handler) 675
Generic Metotlar 677
Metotların Aşırı Yüklenmesi 679
(Overloading) 679
Temsilciler (Delegates) 681
Anonim (Anonymous) Metotlar 686
Lambda Operatörü 687
Action Delegate 688
Func Delegate 691
Predicate Delegate 692

Local Function 692
Caller Info Attributes 695

20 GENİŞLETME METODLARI (EXTENSION METHODS) 697
Genişletme Metotlarının Aşırı Yüklenmesi (Overloading) 701

21 CLASS (SINIF) YAPISI 707
Primary Constructors 715
Parametre Dizisi 716
XML Yorum Satırı Eklemek 723
Property Kullanımı 727
Auto Property Initializers 733
nameof İfadesi 739
Inheritence (Miras, Kalıtım) 739
Access Modifiers 743
(Erişim Belirleyiciler) 743
Encapsulation (Kapsülleme) 746
Inıt-Only Propertıes 749
Overriding (Metotları Ezme) 750
Polymorphism (Çok Biçimlilik) 756
Interface (Arayüz) 759
Record 761

22 USER CONTROL HAZIRLAMA 769

23 REGISTRY İŞLEMLERİ 773
Registry Anahtar ve Değerlerine Erişim 773
Registry'e Anahtar ve Değer Ekleme 775
Anahtar ve Değer Sayısını Bulma 778
Anahtar ve Değer Listesini Alma 779
Anahtar ve Değerleri Silme 781

24 WORD VE EXCEL PROGRAMLARINI KULLANMA 785
Word Programını Kullanma 785
Belge Açma, Kapama ve Kaydetme Olayları 785
Yazı Ekleme ve Düzenleme İşlemleri 792
Range Nesnesi Kullanılarak 792
Sentences Nesnesi Kullanılarak 794
Paragraph Nesnesi Kullanılarak 795
Selection Nesnesi Kullanılarak 796
Önizleme Yapma ve Yazdırma İşlemleri 797
Tablo İşlemleri 801
Grafik İşlemleri 805
Excel Programını Kullanma 809
Belge Açma, Kapama ve Kaydetme Olayları 809
Hücreye Erişim, Yazı Ekleme ve
Biçimlendirme İşlemleri 815
Seçim İşlemleri 818
Önizleme Yapma ve Yazdırma İşlemleri 820
Arama İşlemleri 823
Grafik İşlemleri 826

25 İŞLETİM SİSTEMİ KAYNAKLARINA ERİŞİM 831
Wındows API'lerini Kullanma 831
Disk Toplam Alanını ve Boş Alanını Bulma 832
Mikroİşlemci Hakkında Bilgi Alma 833
Ram Bellek Hakkında Bilgi Alma 836
Masaüstü Resmini Değiştirme 838
Ekran Çözünürlüğünü Bulma ve Değiştirme 839
Geri Dönüşüm Kutusunu Kullanma 845
İnternet Bağlantısını Test Etme 848
Ses Dosyalarını Çalma 849
Ses Ayarlarını Değiştirme 853

Video Dosyalarını Yürütme 855
WMI (Windows Management Instrumentation) Sınıfı 858

26 DOSYA VE KLASÖR İŞLEMLERİ 867
System.IO Sınıfı Kullanılarak Dosya ve Klasör İşlemlerini Gerçekleştirme 867
Dosyalar Üzerinde Yapılan İşlemler 867
File Nesnesi ve Metotları 868
File.Create Metodu 870
File.WriteAll Metodu 872
File.ReadAllLines Metodu 872
Dosya Özelliklerini Elde Etme ve Değiştirme 874
Dosyaları Kopyalama, Taşıma ve Silme 875
FileStream Nesnesi 876
FileStream Nesnesinin Özellikleri 878
FileStream Nesnesinin Metodları 879
Klasörler Üzerinde Yapılan İşlemler 881
Klasör Oluşturma, Silme 883
Dosya İsimlerini Listeleme 884
Klasör Özelliklerini Elde Etme ve Değiştirme 885
Klasörleri Kopyalama ve Taşıma 889
Isolated Storage File Sınıfı ile Dosya ve Klasör İşlemlerini Gerçekleştirme 891
Isolated Storage File Sınıfı Özellikleri 891
Isolated Storage File Sınıfı Metotları 891
Klasör Oluşturma 893
Dosya Oluşturma 894
Dosyaya Yazma 896
Dosyadan Okuma Yapma 898
Dosya ve Klasör Silme 898
System.Text.Json Sınıfı Kullanılarak Dosya ve Klasör İşlemlerini Gerçekleştirme 902

27 VERİ TABANI İŞLEMLERİ 913
Sorgular 913
Access Veri Tabanı İşlemleri 919
OLEDBCOMMAND 927
Kayıt Ekleme 930
Kayıt Güncelleme 932
Kayıt Silme 933
OLEDBDATAADAPTER 943
Kayıt Listeleme 944
Kayıt Ekleme 946
Kayıt Güncelleme 947
Kayıt Silme 948
Kayıt Arama 949
Veri Tabanını Düzenleme 950
Sql Veri Tabanı İşlemleri 960
SQLCOMMAND 966
Kayıt Listeleme 967
Kayıt Ekleme 968
Kayıt Güncelleme 969
Kayıt Silme 970
SQLDATAADAPTER 976
Kayıt Listeleme 977
Kayıt Ekleme 979
Kayıt Güncelleme 980
Kayıt Silme 981

28 LINQ (LANGUAGE INTEGRATED QUERY) 987
LINQ Operatörleri 988
WHERE Operatörü 990
SELECT Operatörü 990
DISTINCT Operatörü 993

CONTAINS operatörü 994
MAX, MIN Operatörleri 994
COUNT Operatörü 995
SQL ve LINQ 995
XML ve LINQ 999

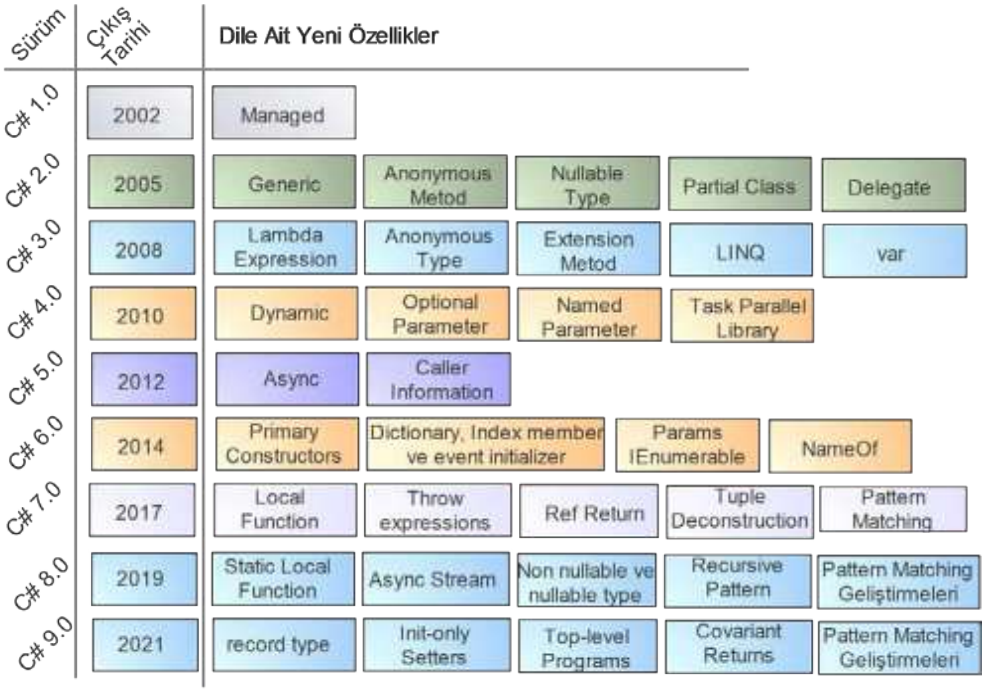
29 THREADING 1005
Threadler ile Çalışmak 1006
Sleep Metodu 1009
Suspend ve Resume Metodları 1011
Abort Metodu 1012
Jo ın Metodu 1012

30 PARALEL PROGRAMLAMA 1015
Parallel Sınıfı 1015
Parallel.Invoke Metodu 1016
Parallel.For ve 1019
Parallel.ForEach Metotları 1019
ParallelLoopState Nesnesi 1021
Task Yapısı 1022
Yeni Bir Task Olusturmak 1022
Task İçerisinde Yürütülen İşlemleri 1025
Durdurmak 1025
ContinueWith( ) 1027
IsCompleted( ) 1028
Child Task 1029
Asenkron Metotlar 1030
async ve await 1030
async main 1035

C# .NET KOD GELİŞTİRME ORTAMI

1

Microsoft firması tarafından 1999 yılının sonu itibari ile geliştirilmeye başlanan **Visual C#** programlama dili, 2002 yılı itibari ile **1.0** sürümü ile **.NET** dilleri arasında yerini almıştır.



C# 1.0 versiyonu ile birlikte yönetilebilir kod yapısı kullanılmaya başlanmıştır.

2.0 versiyonu ile **generics**, **anonymous methods** (*isimsiz metotlar*) gibi programlama yapıları kullanılmaya başlandı.

3.0 versiyonu ile birlikte **lambda ifadeleri**, **extension methods** (*genişletme metotları*), **LINQ** gibi programlama yapıları kullanılmaya başlandı.

4.0 versiyonu ile birlikte **dynamic veri tipi**, **named** ve **optional parameters** gibi programlama yapıları kullanılmaya başlandı.

5.0 versiyonunda, **asenkron programlama yapısı** kullanılmaya başlandı.

6.0 versiyonunda, **primary constructors**, **dictionary**, **index member** ve **event initializer**, **params ienumerable** programlama yapıları kullanılmaya başlanmıştır.

7.0 versiyonunda, **local function**, **throw expressions**, **ref return**, **tuple deconstruction**, **pattern matching** programlama yapıları kullanılmaya başlanmıştır.

8.0 versiyonunda **static local function**, **async stream** yapısı, **non-nullable** ve **nullable reference** tipleri ve **recursive pattern** gibi daha gelişmiş pattern yapıları kullanılmaya başlanılmıştır.

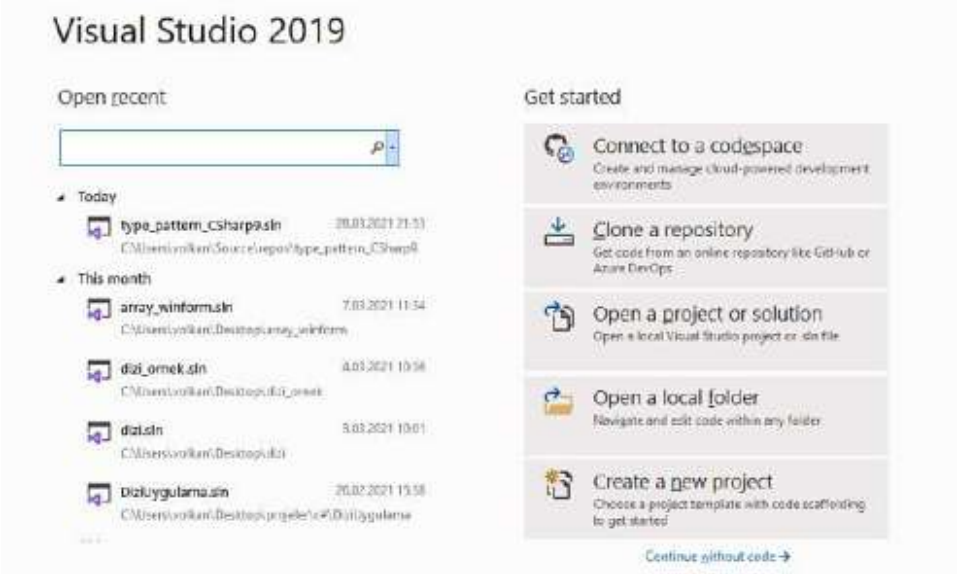
9.0 versiyonunda ise **record type, Init-Only Setters, Top-level Programs, Covariant Returns** gibi yenilikler ve **Relational Patterns, Logical Patterns, Simple Type Pattern** gibi daha gelişmiş pattern yapıları kullanılmaya başlanılmıştır. Ayrıca bu sürümden itibaren kullanılmaya başlanan **.NET Framework 5.0** ile birlikte **desktop, mobil, web, iot** gibi farklı platformların tek bir **.NET Framework** çatısı altında birleştirilmesi hedeflenmiştir.

C# programlama dili, **TurboPascal**, **Delphi**, **J++** gibi dillerin gelişiminde öncülük etmiş olan **Anders Hejlsberg**'in başında bulunduğu bir ekip tarafından geliştirilmiştir. .NET dil ailesi içerisindeki en güçlü dillerden birisidir. *Java, C, C++ gibi dillerin güçlü yönlerini bünyesinde birleştiren bir yapıya sahiptir*.

C# programlama diline, Visual Studio kod geliştirme ortamı içerisinde erişebiliyoruz. Visual Studio içerisinde **C#** dilinin yanı sıra Visual Basic .NET, Visual C++, F#, TypeScript dilleri ve web sitesi tasarımı, kodlaması yapabileceğimiz ASP.Net adlı web programlama dili mevcuttur.

Visual Studio'nun **Enterprise**, **Professional**, ve **Community** olmak üzere **3** sürümü mevcuttur. **Community sürümü**, *ücretsiz* olarak indirip kullanabileceğimiz bazı özellikleri **kısıtlanmış** olan sürümüdür. **En kapsamlı** özelliklere sahip olan sürümü, **Enterprise** sürümüdür. Diğer sürümlerdeki özelliklere ek olarak gelişmiş debugging, test, analiz ve yönetim araçlarına sahiptir. Sürümler hakkındaki detaylı bilgiye *https://www.visualstudio.com/tr/vs/compare/* adresinden ulaşabilirsiniz.

Visual Studio .NET ilk açıldığında karşımıza, **Start Page** adlı başlangıç ekranı gelir. Buradan yeni proje oluşturabilir ve önceden kayıtlı olan projeleri açabilirsiniz.



Open recent bölümünde son açılmış olan **proje isimleri** listelenir.

Connect to a codespace seçeneği ile web tarayıcısı üzerinden Visual Studio Codespace isimli online kod düzenleme ortamına bağlanabilirsiniz.

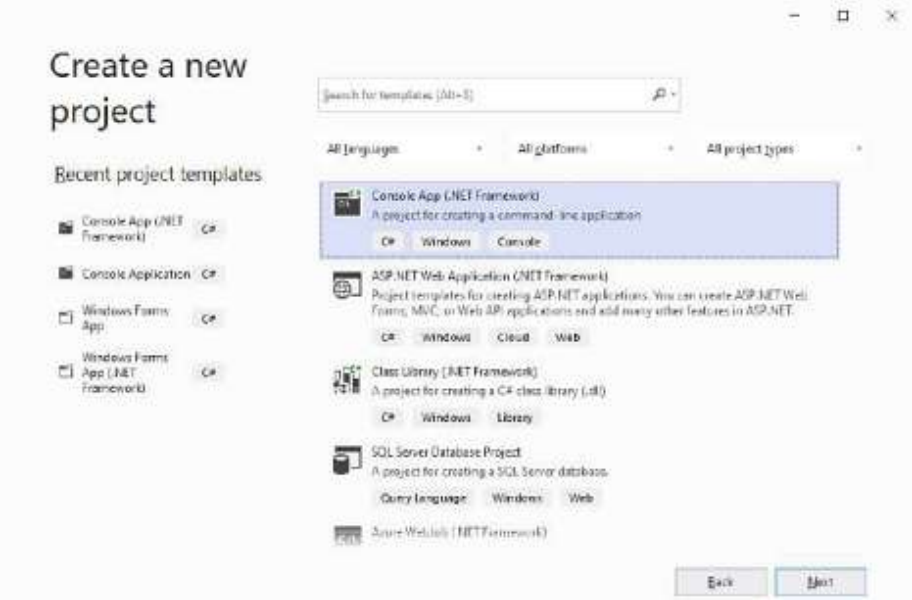
Clone a repository bölümünde var olan bir github hesabındaki mevcut projeler belirtilen yerel bir klasöre kopyalanır.

Open a project or solution bölümünde ise daha önceden mevcut olan proje ya da web sitelerinden biri seçilerek açılır.

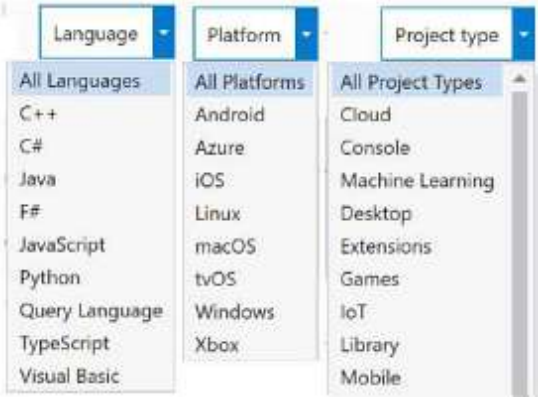
Open a local folder seçeneği ile önceden mevcut olan yerel bir bir proje klasörü açılır.

Create a new project seçeneği ile yeni bir proje açılır. Bu pencerede bulunan diller içerisinden ve mevcut olan proje şablonlarından birisi seçilerek proje geliştirilebilir.

Continue without code seçeneği ile herhangi bir proje dosyası seçilmeden boş bir proje açılır.



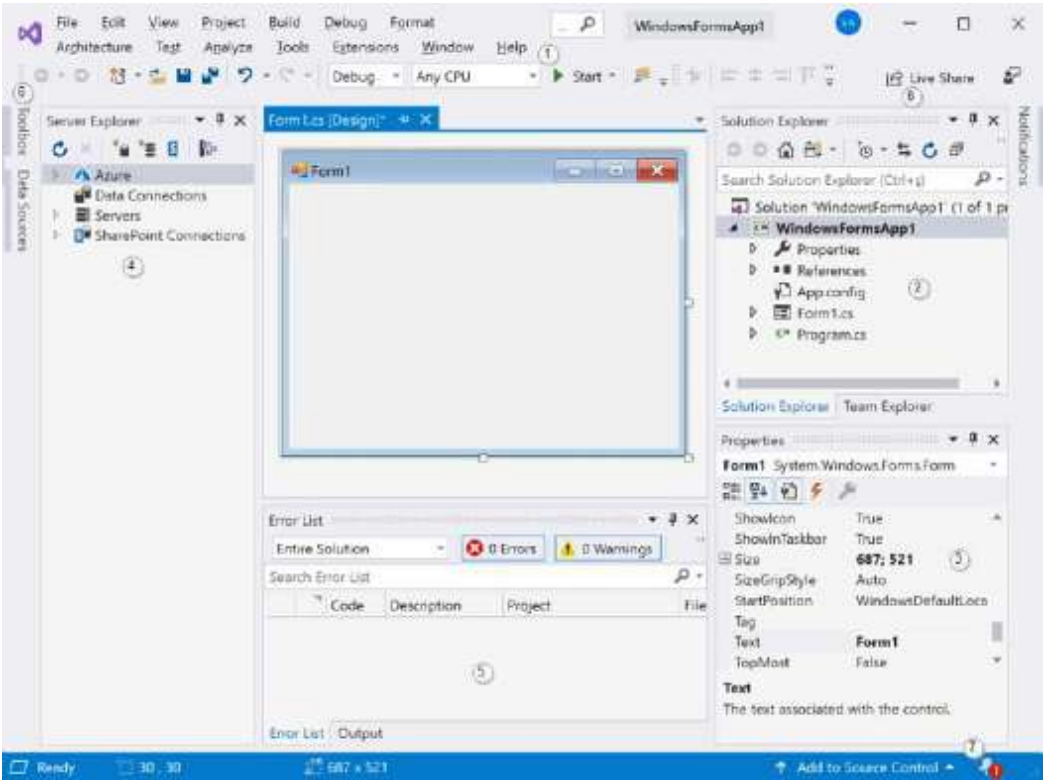
Recent project templates ile son kullanılan proje şablonları seçilebilir.



Language, **Platform** ve **Project type** bölümlerinden, seçilen seçeneklere uygun proje şablonları listelenir.

Kod Geliştirme Ortamı Bileşenleri

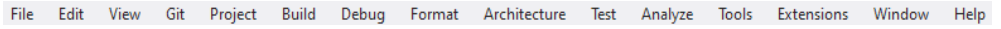
C# .NET kod geliştirme ortamı başlıca şu bileşenlerden oluşmuştur:



Menü Window

İki bölümden oluşur:

Menü Çubuğu



Şeklinde sıralanan menülerden oluşur. Bu menülerin başlıcaları şunlardır:

File menüsü altında yeni proje açma, daha önceden kayıtlı projeleri açma, projeleri kaydetme ve farklı isimle kaydetme ve Visual Studio penceresini kapatma gibi işlemler gerçekleştirilir.

Edit menüsü altında, kesme, kopyalama, yapıştırma gibi düzenleme işlemleri, arama ve değiştirme gibi işlemler gerçekleştirilir.

View başlığı altında, Visual Studio IDE'sinde çeşitli amaçlar için kullanılan pencereleri ekleme, kaldırma, düzenleme gibi işlemler gerçekleştirilir.

Git menüsü altında **GitHub** ortamında projelerinizi saklamak için gerekli menü seçenekleri mevcuttur.

Project menüsü altında, aktif projeye bileşenler ve referans ekleme işlemleri gerçekleştirebilir ve projenizle ilgili çeşitli ayarları yapabilirsiniz.

Build başlığı altında, projelerimizi derleyerek başka bilgisayarlarda da kullanıma hazırlamak amacıyla kullanılan menüler mevcuttur.

Debug başlığı altında, üzerinde çalışılan uygulamayı çalıştırmak ve hataları ayıklamak için kullanılan menüler mevcuttur.

Format başlığı altında, form ara yüzlerindeki nesnelerin hizalama ve yerleşim gibi düzenlemeleri gerçekleştirilir.

Architecture başlığı altında, projenizin uygulama yapısını görsel olarak görüntüleyecek çeşitli araçlar mevcuttur.

Test başlığı altında yazdığımız kodları ve uygulamaları test etmek amaçlı menüler mevcuttur.

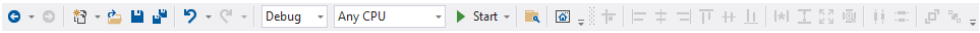
Analyze başlığı altında yazdığımız kod ve uygulamaları analiz etmek amaçlı menüler mevcuttur.

Tools başlığı altında, Visual Studio içerisinde uygulamalarınızı geliştirirken gerekli bazı araçları çalıştırabilirsiniz.

Extensions başlığı altında, Visual Studio içerisine eklentiler (*extensions*) ekleyebilir ya da var olan eklentileri kaldırabilirsiniz.

Window başlığı altında, Visual Studio IDE'sindeki pencerelerin yerleşim şekilleri ile ilgili ayarlamaları yapabileceğiniz menüler mevcuttur.

Araç Simgesi Çubuğu

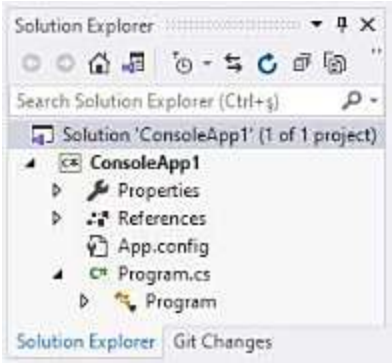


Visual Studio menülerinde yer alan işlemleri araç çubuğundaki simgeler yardımıyla daha kısa sürede gerçekleştirebilirsiniz.

NOT

Gerek menüler, gerekse araç çubuğunun ayrıntılı açıklamasına burada değinmeyeceğiz. İlgili konularda gerekli menü ve araç simgeleri detaylıca açıklanacaktır.

Solution Explorer



Visual Studio .NET içerisinde projeleri gruplandırarak bir çatı altında toplamak amacı ile **solution** adı verilen yapılar kullanılır. Bir solution içerisinde birden fazla proje bulunabileceği gibi tek bir proje de bulunabilir. Projede mevcut olan kontrol, sınıf, metot gibi proje bileşenlerini görüntülemek amacı ile **Solution Explorer** adlı Visual Studio bileşeni kullanılır.



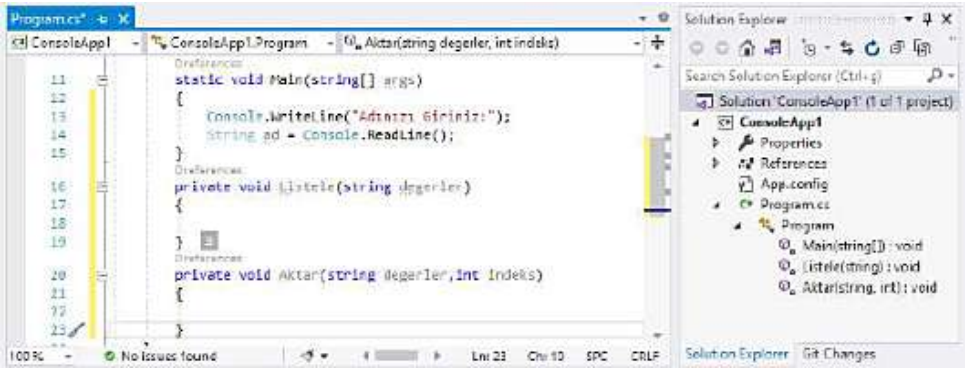
Properties başlığı altında projemiz ile ilgili ayarların saklandığı **AssemblyInfo.cs** dosyası yer almaktadır.

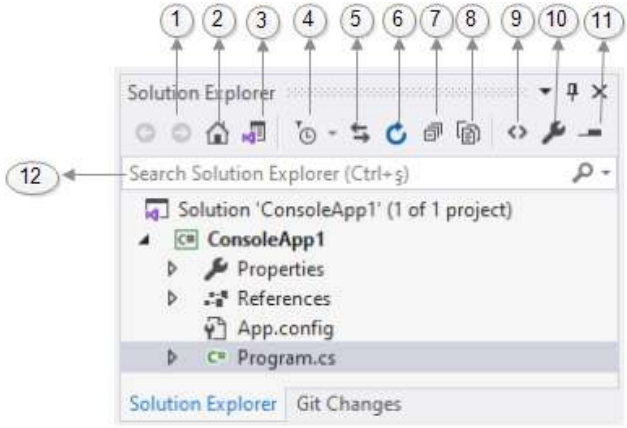
References başlığı altında uygulamamıza eklenmiş olan referans dosyalarının isimleri listelenmektedir. **Analyzers** kısmında projemize eklemiş olduğumuz analiz araçlarının referansları listelenecektir.

App.config dosyası, uygulamanız ile ilgili genel tanımlama ve bilgilerin yazıldığı dosyadır. Örneğin, **Framework** versiyonu varsayılan olarak burada yazılıdır.

Program.cs, C# kodlarını yazdığımız dosyadır. Uygulamamızın gerçekleştirmesini istediğimiz işlemler burada yazılır. Bu dosya konsol uygulamaları için bu şekilde iken bir Windows form uygulamasında kod sayfası **Form1.cs** şeklinde olacaktır.

Proje içerisindeki kod sayfalarında yer alan metotlar da **Solution Explorer** içerisinde listelenmektedir.





Solution Explorer penceresi üzerinde bulunan butonlar ve görevleri şu şekildedir:

(1) **Back – Forward:** Sayfalar arasında ileri geri hareket eder.

(2) **Home:** Ana sayfayı görüntüler.

(3) **Switch between solutions and available view:** Proje görünümleri arasında geçiş yapar.

(4) **Filter:** Filtreleme işlemi yapar.

(5) **Sync:** Geçerli dosyaya Solution Explorer içerisinde konumlanır.

(6) **Refresh:** Yapılan değişikliklerin ardından Solution Explorer penceresini yeniler.

(7) **Collapse:** Solution Explorer içerisinde açık olan alt dalları kapatır.

(8) **Show All Files:** Solution Explorer içerisinde listelenen ögelerin alt dalları varsa hepsini açarak listeler.

(9) **View Code:** Kod görünümüne geçiş yapar.

(10) **Properties**: Seçili olan nesnenin özelliklerini Properties Window içerisinde görüntüler.

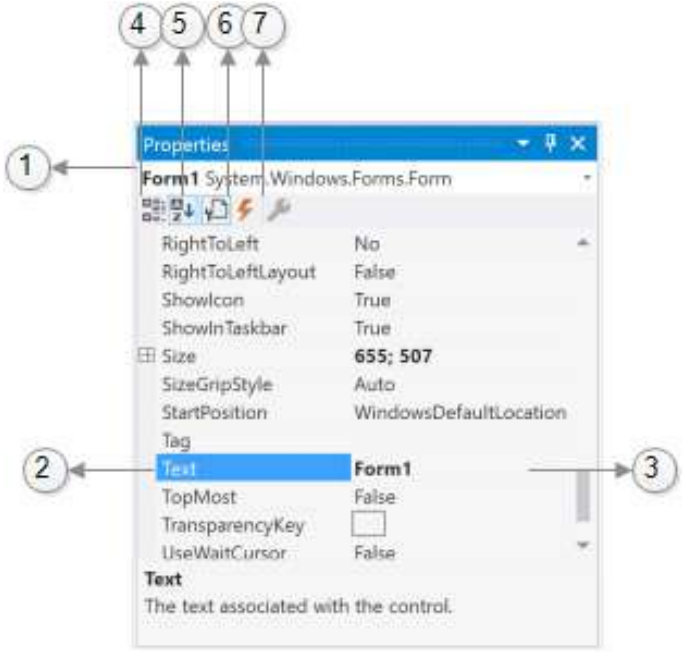
(11) **Preview:** Seçili olan nesneye önizleme yapar. Örneğin bir resim dosyasının ön izleme şeklinde görüntülenmesi gibi.

(12) **Search:** Solution Explorer içerisinde aranacak ifade ve arama çubuğuna yazılarak bulunan ifadenin geçtiği kısımlar, Solution Explorer içerisinde işaretlenir.

Properties Window

Seçili olan nesnenin özelliklerini görüntüler. Ayrıca seçili olan özelliği değiştirmek amacı ile de kullanılır.

Properties penceresi üzerinde yer alan butonlar ve görevleri şu şekildedir:



(1) **Object List:** Proje içerisinde yüklü olan nesneleri listeler.

(2) **Property:** Seçili olan nesnenin özelliğidir.

(3) **Value:** Seçili olan nesnenin özelliğine ait değeri belirtir.

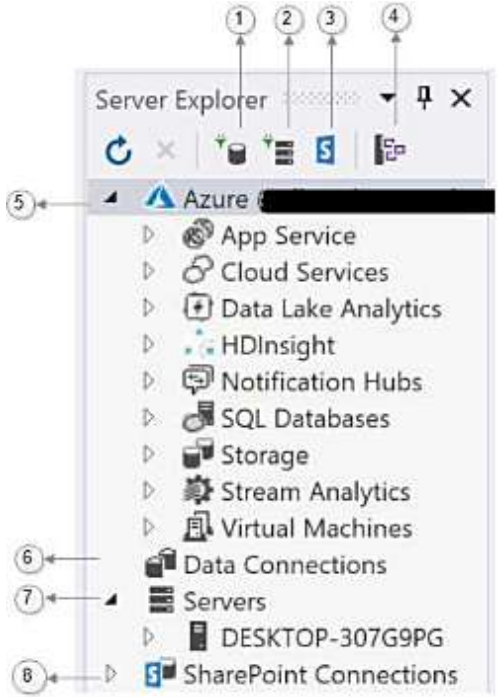
(4) **Categorized:** Properties penceresinde listelenen özelliklerin, benzer niteliklerine göre gruplandırılarak listelenmesini sağlar.

(5) **Alphabetical:** Properties penceresinde listelenen özelliklerin, alfabetik sıraya göre gruplandırılarak listelenmesini sağlar.

(6) **Properties:** Properties penceresinde seçili olan nesneye ait özelliklerin listelenmesini sağlar.

(7) **Events:** Properties penceresinde seçili olan nesneye ait olayların listelenmesini sağlar.

Server Explorer



Yeni bir veritabanı oluşturmak, oluşturulmuş olan veritabanları içerisinde düzenlemeler yapmak ve sunucu bağlantısı sağlamak amacı ile kullanılır.

Server Explorer penceresi üzerinde yer alan butonlar ve görevleri şu şekildedir:

(1) **Connect to Database:** Bir veritabanı sürücüsüne bağlanarak veritabanı oluşturmak ve oluşturulmuş veritabanlarına bağlanmak amacıyla kullanılır.

(2) **Connect to Server:** Bir sunucuya bağlanmak amacıyla kullanılır.

(3) **Add Sharepoint Connection:** Sharepoint bağlantısı eklemek amacıyla kullanılır.

(4) **SQL Server Object Explorer:** SQL veritabanları oluşturmak ve üzerlerinde düzenleme yapmak amacıyla kullanılır.

(5) **Azure:** Windows Azure servislerinin listesi. Azure, kullanıcıların uygulamaları kendi sistemlerine kurmadan, sunucu sistemleri vasıtası ile web tarayıcıları üzerinde çalıştırabildiği sistemdir.

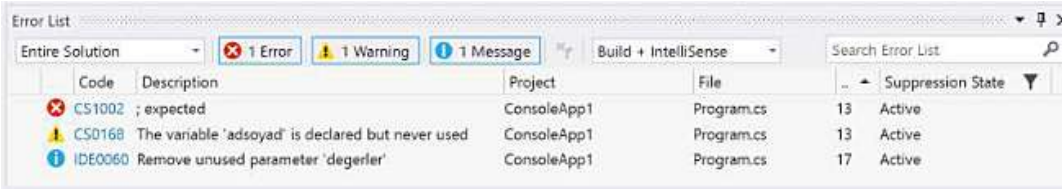
(6) **Data Connections:** Yeni veritabanı bağlantıları yapmak ve mevcut veritabanı bağlantılarını listelemek amacıyla kullanılır.

(7) **Servers:** Yüklü olan server listesi.

(8) **SharePoint Connections:** Sharepoint bağlantılarının listesi. Sharepoint, özellikle iş uygulamaları geliştirmek üzere kullanılan bir sistemdir. Sharepoint ile doküman ve içerik yönetimi, portal uygulamaları geliştirme, raporlama, projeler üzerinde ekip hâlinde çalışabilme ve kurumsal nitelikli birçok işlem gerçekleştirebilirsiniz.

Error List

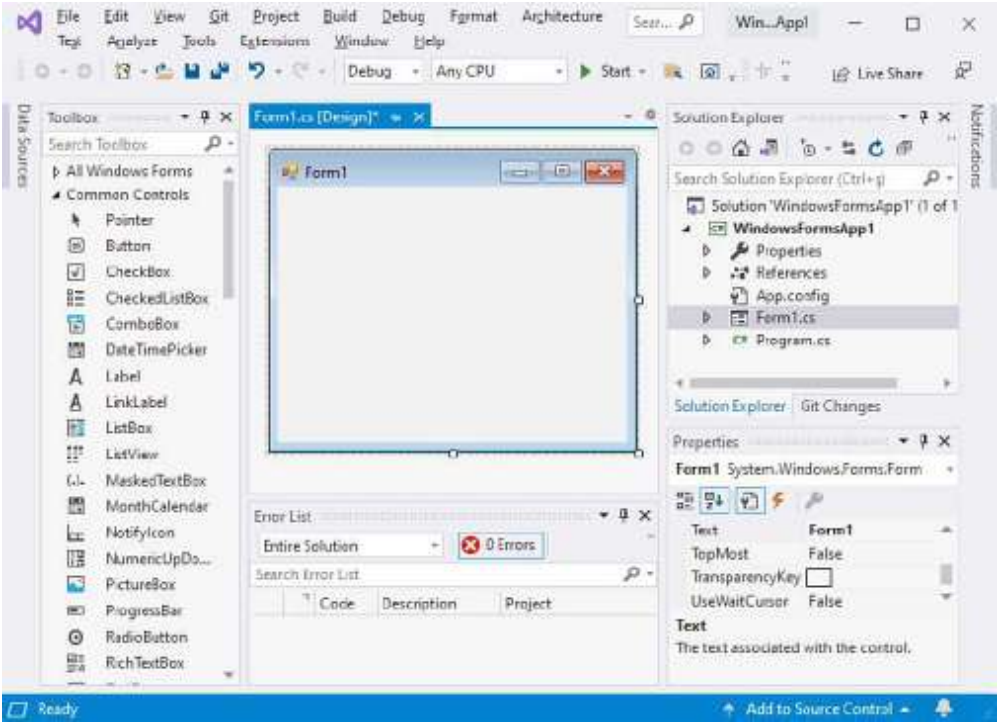
Proje süreci içerisinde oluşan hata ve uyarı mesajlarının görüntülendiği penceredir. Programın çalışmasını kesintiye uğratacak hatalar işareti ile; programın çalışmasını kesintiye uğratmayıp programcıya yol gösterme amaçlı uyarılar ise işareti ile gösterilir. Kod içerisinde bilgilendirme ve kullanıcıya yol gösterme amaçlı mesajlar ise işareti ile gösterilir.



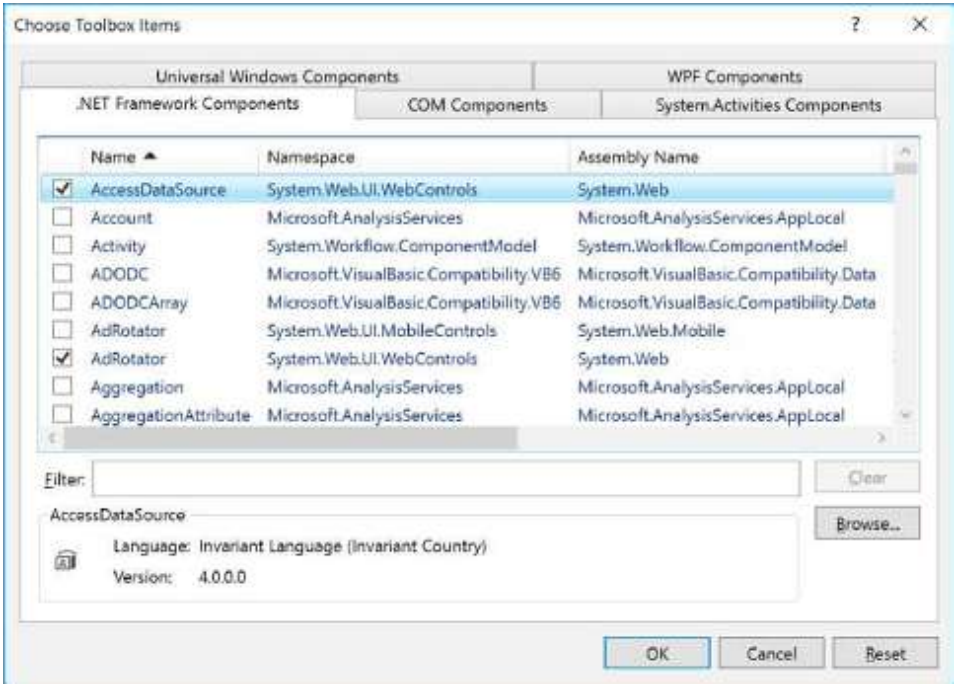
Örneğin yukarıdaki **Error List** penceresini incelediğimizde program içerisinde tanımlanıp kullanılmayan `adsoyad` değişkeni programın çalışmasını engellemediği için uyarı biçiminde görüntülenmektedir. Ancak bir komut satırının sonuna noktalı virgül (;) konulmaması programın çalışmasını engellediği için hata biçiminde görüntülenmektedir. Ayrıca kod içerisinde "`değerler`" adında tanımlanıp da kullanılmayan parametrenin silinmesi programcıya tavsiye edilmektedir.

ToolBox

Visual Studio arabirim ekranının sol tarafında bulunan ve projemize ekleyebileceğimiz nesneleri barındıran bölümdür.

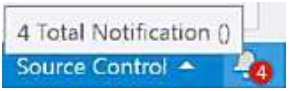


Toolbox içerisine yeni bir nesne eklemek için Toolbox'a farenin **sağ tuşu** ile tıklanarak gelen menüden **Choose Items** seçeneği tıklanır. Karşımıza gelen pencerede istenilen nesne adı seçilerek ya da **Browse** seçeneği ile nesnenin ait olduğu dosya adı seçilerek Toolbox içerisine istenilen nesneler eklenebilir.

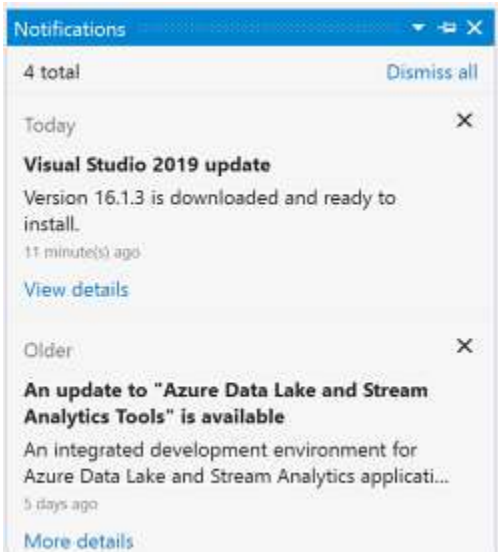


NOTIFICATIONS

Visual Studio ile ilgili güncellemeler ve ek paketler konusunda geliştiricileri bilgilendirmek amacıyla kullanılmaktadır. Bildirim simgesinin hemen sağında bildirim adedi rakamla bildirilmektedir. 

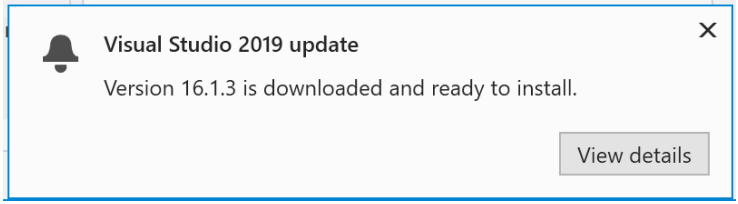


Bildirim simgesinin üzerine fare ile gelinip beklenildiğinde okunmamış bildirim sayısı (*New notifications*) ve toplam bildirim sayısı (*Total notifications*) görüntülenmektedir.



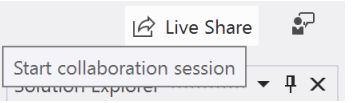
Bildirim simgesinin üzerine fare ile tıklatıldığında ise yüklenebilecek güncellemeler ve ek paketler listelenecek ve ilgili linklere tıklayarak güncelleme ve yükleme işlemini gerçekleştirebileceksiniz.

Visual Studio **update** bildirimleri bildirim simgesine ek olarak Visual Studio açıldığında ayrı bir pencere ile de görüntülenmektedir. İster yukarıdaki gibi **Notifications** penceresinden isterseniz bu pencereden **Visual Studio update** işlemini gerçekleştirebilirsiniz.



Live Share

Visual Studio Live Share ile projelerinizi ve kodlarınızı arkadaşlarınızla paylaşarak uygulamalarınızı birlikte geliştirebilirsiniz. Projelerinizi ister yalnızca okumaya açar, isterseniz de katılımcıların kodlarda değişiklik yapmalarına izin verebilirsiniz.

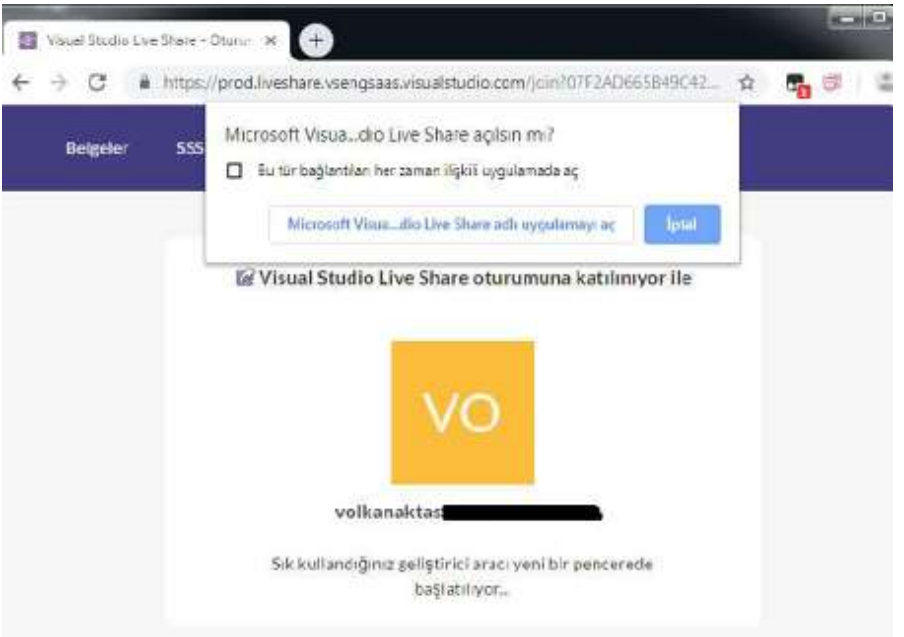


Visual Studio penceresinin sağ üst tarafındaki **Live Share** butonuna tıklayarak yeni bir oturum başlatıyoruz.

Kullanıcıların oturuma katılabilmeleri için bir web linki oluşturulacaktır. "**Make read only**" seçeneğine tıklayarak kullanıcıların uygulamada değişiklik yapamayacağı, yalnız **okunabilir** bir oturum da başlatabilirsiniz.

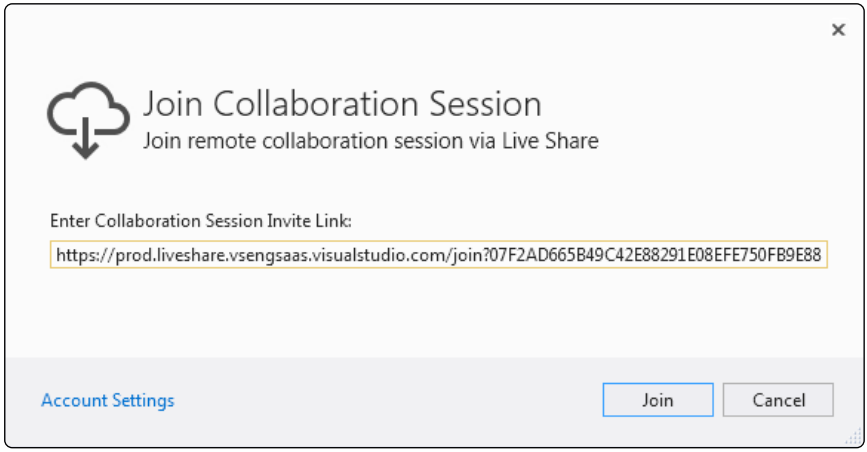


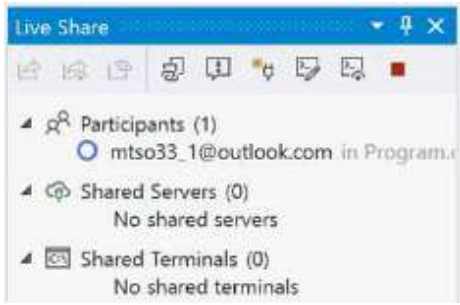
Oturum linki "*https://prod.liveshare.vsengsaas.visualstudio.com/join?*" şeklinde başlayan bir link olacaktır.



Ayrıca **Live Share** penceresindeki "**Join Session**" butonuna tıklayarak da bir **Live Share** oturumuna katılabilirsiniz.

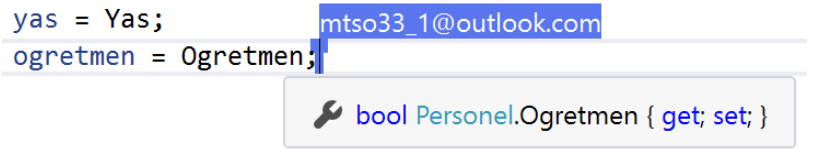




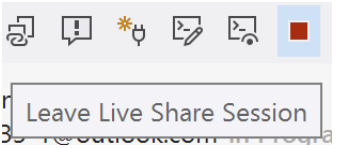


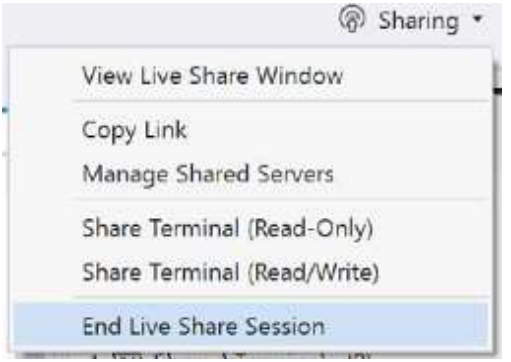
Hangi katılımcının hangi kod dosyası içerisinde bulunduğu, **Live Share** penceresi içerisinde "**Participants**" bölümünde listelenecektir.

Katılımcıların kod üzerinde değişiklik yaptığı yerler işaretlenecektir.



Live Share oturumundan çıkmak için Live Share penceresindeki "**Leave Live Share Session**" butonunu ya da **Sharing** menüsünden "**End Live Share Session**" seçeneğini kullanabiliriz.





KOD YAZMAYA GİRİŞ

2

Visual Studio arabirimi (IDE) bizlere; form tasarımı, kod yazımı, hata yakalama, derleme ve çalıştırma gibi işlemler için gerekli araçları sağlamaktadır. **Visual Studio arabirimi** içerisinde; masaüstü, web ve mobil programlama yapabilmek için gerekli diller ve araçlar mevcuttur. Visual Studio içerisindeki diller **.NET Programlama Dilleri** olarak adlandırılır. Bu diller;

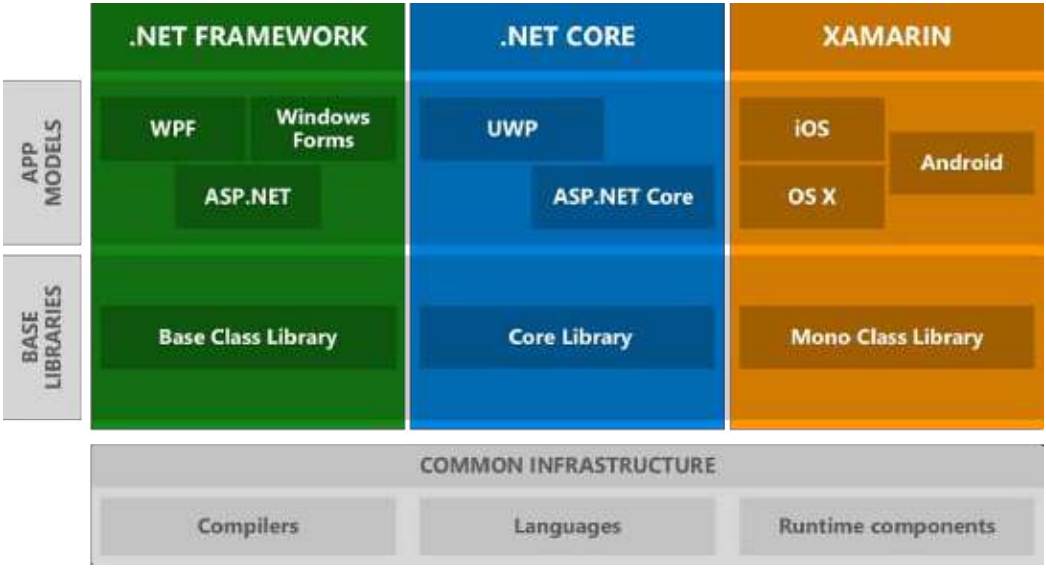
- **C#.NET**
- **Visual Basic .NET**
- **Visual C++ .NET**
- **F#**
- **ASP.NET**
- **Typescript**

şeklindedir. Ayrıca bu dillerle birlikte kullanılan **WPF**, **WCF**, **Silverlight**, **Xamarin** gibi teknolojilerle hemen her alanda ihtiyacı karşılayabilecek uygulamalar geliştirebilirsiniz.

Visual Studio arabiriminin kullanıcıya sağladığı destek ile bazı işlemler kod yazmadan bazıları kod yazılarak bazıları da her iki şekilde Visual Studio arabirimi ile gerçekleştirilebilmektedir.

.NET Framework Nedir?

.NET Framework, yeni nesil Windows uygulamaları oluşturulması, çalıştırılması ve yönetilmesi gibi işlemler için gerekli altyapıyı sağlayan bir platformdur. Windows uygulamaları ile Windows işletim sistemleri arasında bir katman görevi görür. Dolayısı ile birbirinden farklı programlama dilleri, tek bir iletişim katmanını kullanarak işletim sistemine erişim sağlarlar. İşletim sistemleri ile bağlantıyı ve entegrasyonu sağlayan kütüphaneler ve araçlara sahiptir. Bu sayede işletim sisteminden ve donanımdan bağımsız olarak çalışmayı ve uygulama taşınabilirliğini sağlar.



İşletim sistemi ile uygulamalar arasındaki ilk katman **CLR** (*Common Language Runtime*) katmanıdır. Bu katman, programların çalıştırılmasını yönetir. İşletim sistemi ile uygulamalarımız arasında aracılık yapar. İkinci katman ise **Base Libraries** katmanıdır. Base Libraries, içerisinde .NET ile uygulama geliştirirken kullanacağınız komutlar ve metotlar bulunan geniş içerikli bir kütüphanedir. Kullanılacak framework yapısına göre kullanılacak library de (*Base Class, Core, Mono Class Libraries*) şeklinde değişiklik göstermektedir.

.NET Framework 5.0 sürümüne kadar .NET Framework yapısı birden çok Framework yapısından oluşmaktadır. Standart masaüstü ve web uygulamaları için .NET Framework yapısı kullanılırken, mobil uygulamalar geliştirmek için **Mono Framework** tabanlı Xamarin yapısı kullanılmakta, platform bağımsız uygulamalar geliştirmek için ise **.NET Core mimarisi** kullanılmaktadır. .Net Core, Microsoft'un açık kaynak kodlu olarak geliştirdiği, platform bağımsız olarak Windows, Linux, MacOS gibi farklı platformlarda çalışabilen güçlü, hızlı ve esnek bir uygulama geliştirme platformudur.

.NET Framework 5.0 sürümünden itibaren ise artık platform bağımsız uygulamalar geliştirmeye imkân verecek biçimde, farklı **base class libraries** yerine tek bir **class libraries** yapısının kullanılmaya başladığı görülmektedir. NET Standard yapısı ile farklı Frameworkler ve proje tipleri için farklı .net api'leri ve kütüphaneleri kullanmak yerine bütün Frameworkler tek bir çatı altında birleştirilerek hepsinde ortak olarak kullanılabilen .net api'leri ve kütüphaneler kullanılmaktadır. .NET Framework 5 sürümünden itibaren artık tek bir **Base Class Library** kullanılmaktadır.



Framework yalnızca .NET dillerine değil bunun yanı sıra .NET dışında; **Python**, **Pascal**, **C++, Cobol**, **Java**, **Scheme** gibi elli civarında programlama diline de destek vermektedir.

2002 yılından itibaren **1.0** versiyonu ile birlikte kullanıma sunulan framework, artık Microsoft Windows işletim sistemleri ile birlikte de işletim sistemine kurulu olarak kullanıma sunulmaktadır. Örneğin, Windows 7 Framework 3.5

kurulu olarak kullanıma sunulurken Windows 10, Framework 4.6 ve üstü versiyonlar kurulu olarak kullanıma sunulmaktadır.

Visual Studio kurulduğu zaman kurulan sürüme göre framewok sürümleri yüklenmektedir. Örneğin Visual Studio 2015, **Framework 4.6** versiyonu ile birlikte gelirken Visual Studio 2019 ise **Framework 4.8** versiyonu ile birlikte gelmektedir.

Uygulamalarda bellek yönetimi **Managed** ve **Unmanaged** olarak iki şekilde gerçekleştirilmektedir.

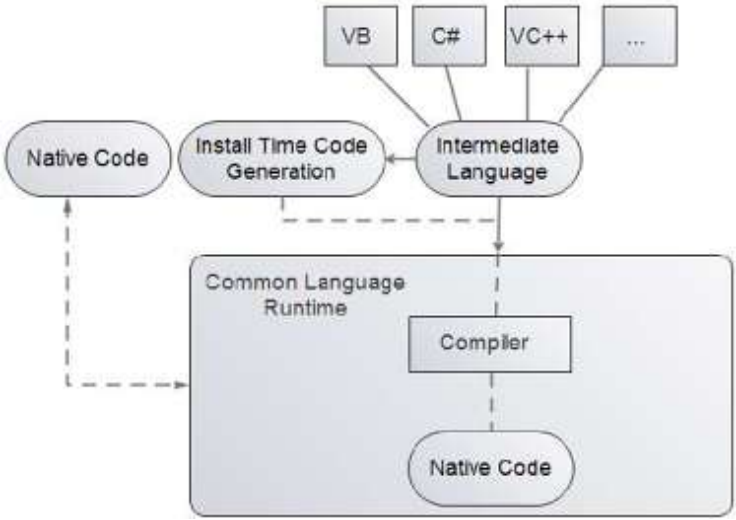
Managed uygulamalarda bellek yönetimi .NET tarafından, Garbage Collector gibi bir mekanizma ile yürütülerek işlemler için hafızada yer açma, işlemler bittiğinde ya da artık kullanılmadığında hafızayı temizleme gibi işlemler gerçekleştirilir. **.NET** ile yapılan uygulamalar varsayılan olarak **managed bellek** yönetimi gerçekleştirmektedir.

Unmanaged uygulamalarda ise bellek yönetimi işlemi, kullanıcının kendisi tarafından **manuel** olarak gerçekleştirilmek zorundadır.

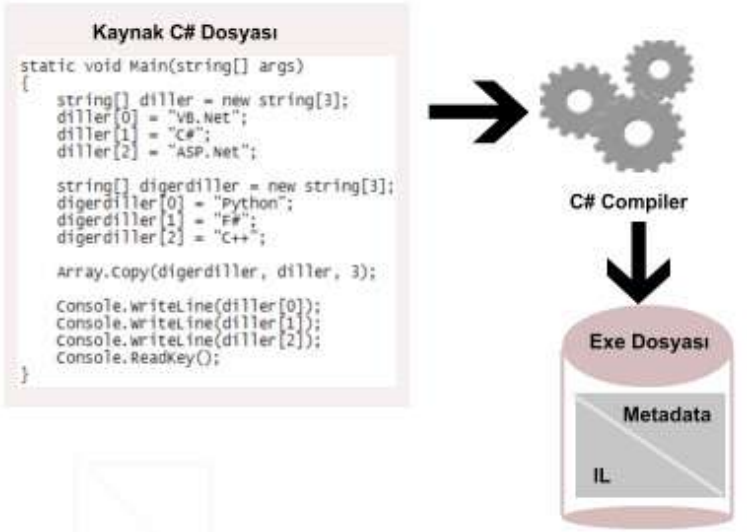
Managed uygulamalar genellikle, .NET ile yapılan uygulamalar olurken **Unmanaged** uygulamalar ise .NET dışındaki uygulamalardır.

Bir Programın Derleme Süreci

Derleme (*compile*) işlemi, yazılan programı çalıştırılabilir **exe** dosyası, **dll** dosyası gibi formatlara dönüştürmek amacıyla kullanılır. C# içerisinde bir programı derlemek için **F5** tuşuna basarız ve programımız çalışarak ekranda sonuçları görüntülenir. *Acaba bu süreçte ne olmaktadır?* Bu sürece kısaca değinelim:



Program kodu derleme anında, **IL** (*Microsoft Intermediate Language*) denilen bir ara dile çevrilir. Oluşturulan bu kod kendi başına çalışabilir bir kod değildir. Çalışabilmesi için bazı ek işlemlere ihtiyaç duyar. Programı yazdığınız dil hangi dil olursa olsun hepsi de ortak dil olan **IL** diline çevrilir. Bu sayede üretilen kodlar, işlemciden bağımsız olarak her platformda çalışabilirler. Oluşturulan kod, **CLR** (*Common Language Runtime*) içerisinde **compiler** tarafından derlenerek makine diline çevrilir ve hafızaya yüklenerek çalıştırılır. Bu dönüştürme işlemi **CTS** (*Common Type System*) kurallarına göre yapılır. Bu sayede farklı dillerden, ortak bir standart altında dönüşüm yapılmış olur. Oluşturulan kod içerisinde yalnızca IL değil, aynı zamanda metadata bilgileri de bulunur. **Metadata** içerisinde; veri tipleri, kullanılan metotlar ve sınıflarla ilgili bilgiler bulunur.



CLR, .NET ile işletim sistemi arasında **köprü vazifesi** kurarak iletişimi sağlar. CLR genel olarak aşağıdaki işlemleri gerçekleştirir:

- Otomatik hafıza yönetimi
- Thread yönetimi
- Kod derleme ve çalıştırma
- Kod doğrulama
- Yüksek güvenlik seviyesi
- Yapılandırılmış hata yakalama
- Managed ve Unmanaged kodların birlikte çalıştırılabilmesi

Varsayılan olarak derleme işlemi **Standart JIT Compiler** tarafından gerçekleştirilirken bunun dışında compiler olarak EconoJIT, PreJIT ve Multicore Background JIT, RyuJIT ve Roslyn Compiler da kullanılabilir.

Standart JIT compiler, yalnızca çalışma zamanında çağrılan metotları derler. Bu metotlar ilk çağrıldıkları zaman derlenerek **cache** belleğe aktarılır. Cache belleğe eklenen metotlar ihtiyaç duyulduğunda çağrılıp derlendiği için, derlenen kod **EconoJIT compiler** ile derlenen koda göre daha **hızlı** çalışmaktadır.

EconoJIT compiler ise sınırlı sistem kaynakları için dizayn edilmiştir. Ram belleğin yeterli olmadığı durumlarda kullanılabilir. Derlenen metotlar ihtiyaç duyulmadığında silinir. Derlenen kısımlar bellekte depolanmayıp her defasında yeniden çalıştırıldığı için hafıza kaynaklarını az kullanır, ancak Standart JIT Compiler'e göre daha **yavaş** çalışır.

PreJIT ile derleme işlemi program çalıştırılmadan önce gerçekleştirilir. Derlenen tüm kısım bir defada makine koduna dönüştürülür.

Multicore Background JIT Compiler, .NET Framework 4.5 ile birlikte kullanılmaya başlanmıştır. Birden fazla işlemci çekirdeğini kullanarak uygulamanın derleme sürecini hızlandırır. Uygulama ilk kez çalıştırıldığında sık kullanılan metotlara dair veriler toplanır. Bu veriler profil dosyası şeklinde saklanır ve uygulamanın sonraki çalışmalarında profil dosyası içerisindeki veriler, arka planda çalışan **JIT Compiler** tarafından kullanılır. Bu özelliği kullanabilmek için uygulamanıza `ProfileOptimization.SetProfileRoot`**(yol adı)** ve `ProfileOptimization.StartProfile`**(dosya adı)** şeklinde metotları eklemeniz gerekmektedir. Silverlight 5 ve ASP.NET uygulamalarında bu özellik varsayılan olarak **aktif**tir. Özellikle ARM tabanlı işlemcilerde uygulamaların açılış zamanını hızlandırarak daha performanslı çalışmalarını sağlamak amacıyla geliştirilmiştir.

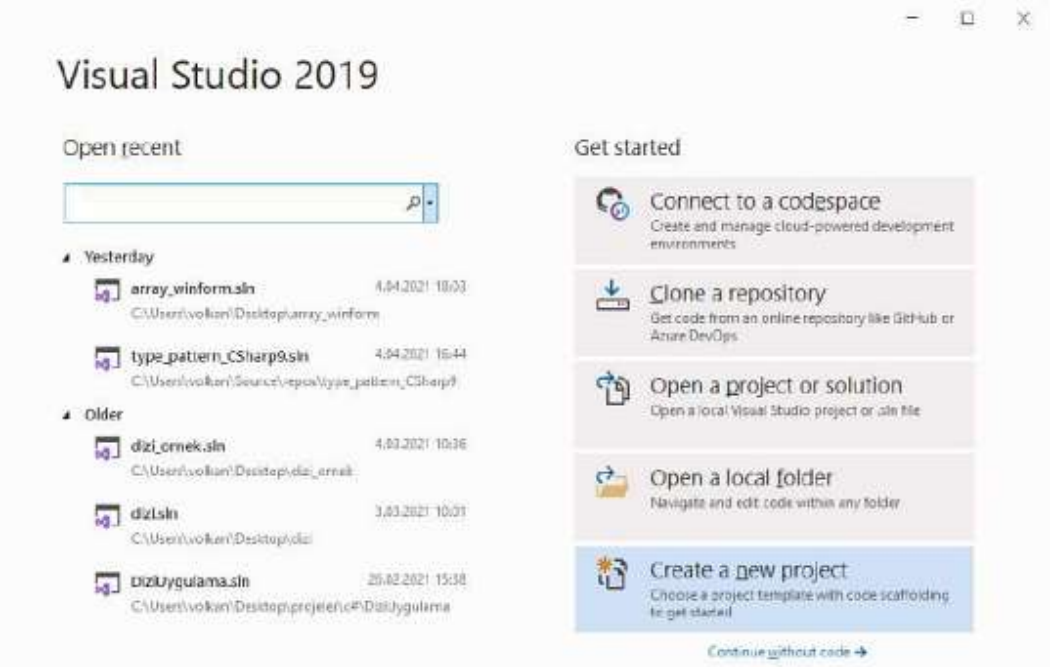
RyuJIT Compiler, başlangıçta açık kaynak kodlu bir proje iken Visual Studio 2015 Update 1 sürümüyle birlikte Visual Studio içerisine dâhil edilen 64 bitlik bir derleyicidir. WPF ve Windows Form uygulamalarının grafik gücünü artırmak için geliştirilen bu compiler, diğer derleyicilere göre **%30**'a kadar daha **performanslı** çalışmaktadır.

Roslyn Compiler, açık kaynak kodlu bir derleyici olup ayrıca kod analizi için araçları da içermektedir. Roslyn ile geliştiriciler uygulamaların arka planında gerçekleşen işlemler hakkında bilgi sahibi olarak kendi kod analiz ve denetleyici uygulamalarını geliştirebilirler. **Visual Studio 2015** sürümünden itibaren **Roslyn derleyicisi** kullanılmaya başlanmıştır. Roslyn ile uygulama geliştirebilmek için **Extensions/Manage** Extensions menüsü altından **.NET Compiler Platform SDK**'sini yüklemelisiniz.

Yeni Proje Açmak

Yeni bir proje açmak için şu seçenekler kullanılabilir:

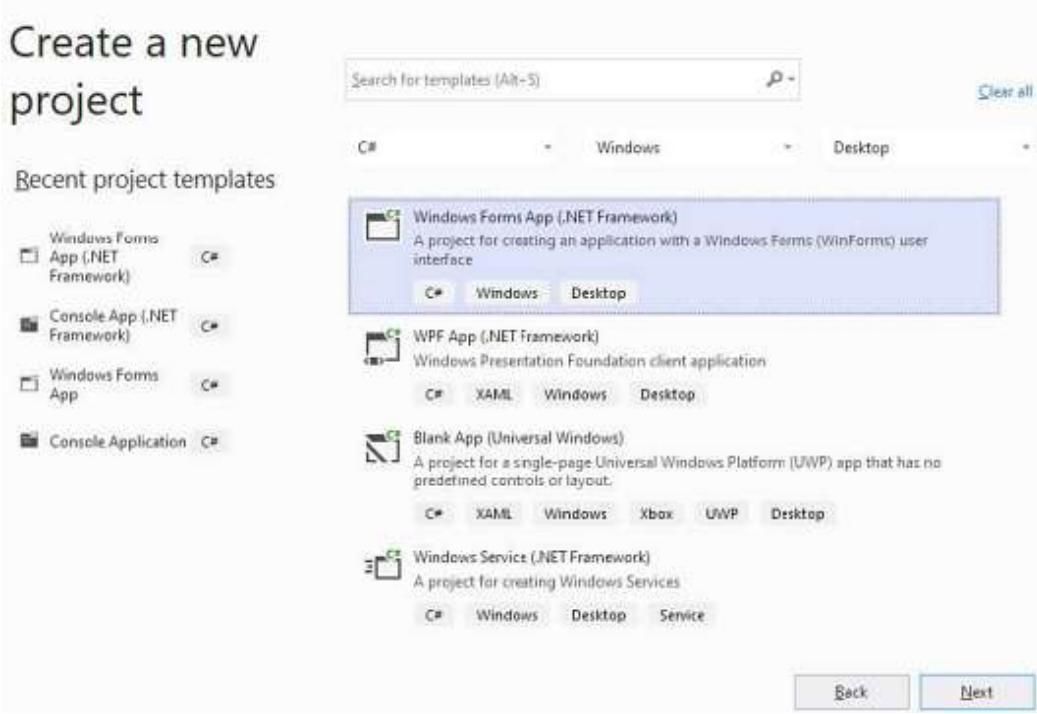
- **Start Page** ekranından **Get Started** başlığı altındaki **Create a new project** seçeneği tıklanabilir.



- **File** menüsünden **New / Project** seçeneği kullanılabilir.
- **Ctrl+Shift+N** tuşları kullanılabilir.
- Araç çubuğundan simgesi tıklanabilir.

Bu seçeneklerden herhangi biri kullanılarak **New Project** penceresi görüntülenir. Bu pencerede mevcut olan proje şablonlarından birisi seçilerek yeni bir proje oluşturulabilir. .NET 5.0 ile desktop ve console projelerini **.Net Framework** ve **.Net Core** uygulaması olmak üzere 2 farklı biçimde oluşturabilirsiniz. Yapacağınız seçime göre proje açma aşamaları **farklılık** gösterecektir.

Yeni bir .NET Framework projesi açmak için **Console App** (.NET Framework) ya da Windows **Forms App** (.NET Framework) seçeneklerinden birisini seçebilirsiniz.



C# .NET ile kullanılabilecek başlıca proje tipleri şunlardır:

Windows Forms App (.NET Framework): İçerisinde Windows formları bulunan .NET Framework masaüstü uygulamaları geliştirmek için kullanılır.

Console App (.NET Framework): İçerisinde Windows formları bulunmayan .NET Framework komut satırı uygulamaları geliştirmek için kullanılır.

Windows Forms App: İçerisinde Windows formları bulunan .NET Core masaüstü uygulamaları geliştirmek için kullanılır.

Console Application: İçerisinde Windows formları bulunmayan .NET Core komut satırı uygulamaları geliştirmek için kullanılır.

WPF App (.NET Framework): Görsel açıdan zengin niteliklere sahip XAML tabanlı .NET Framework WPF (**Windows Presentation Foundation**) uygulamaları geliştirmek için kullanılır.

WPF Application: Görsel açıdan zengin niteliklere sahip XAML tabanlı .NET Core WPF (**Windows Presentation Foundation**) uygulamaları geliştirmek için kullanılır.

ASP.NET Core Web App: İnternet ortamında çalışacak ASP.NET Core uygulamaları ve web siteleri geliştirmek için kullanılır.

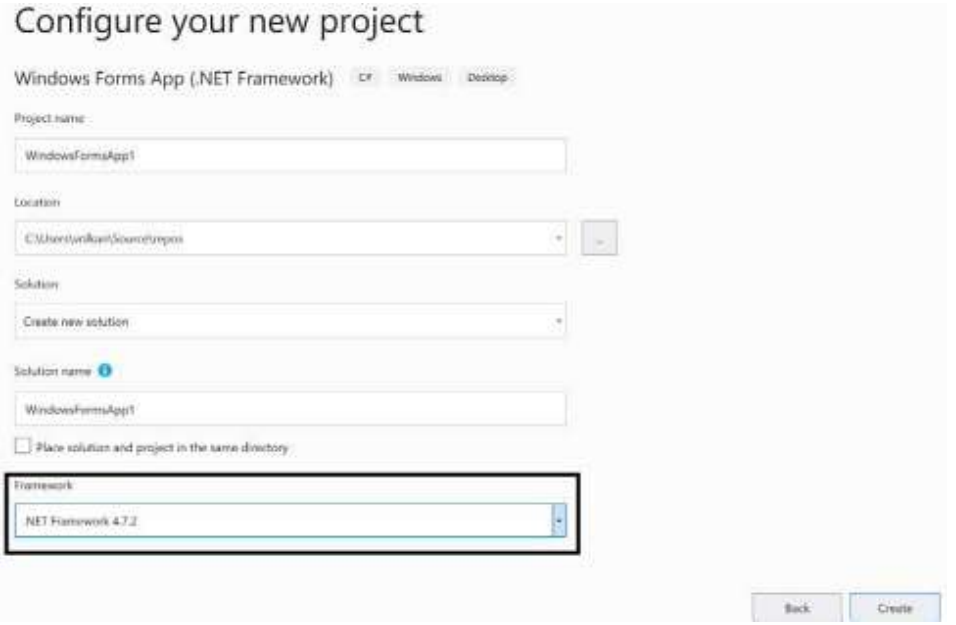
ASP.NET Core Web App (Model-View-Controller): Web Forms, MVC (**Model View Controller**) Framework ve Web Api'lerini bir arada kullanabilmek amacıyla ASP.NET Core uygulamaları kullanılmaktadır.

Class Library: Dll dosyalarından oluşan sınıflar geliştirmek için kullanılır.

Blank App (Universal Windows): Windows masaüstü, mobil ve iot projelerini ayrı ayrı geliştirmek yerine, tek bir projeyi aynı anda farklı platformlar için geliştirebileceğiniz proje tipidir.

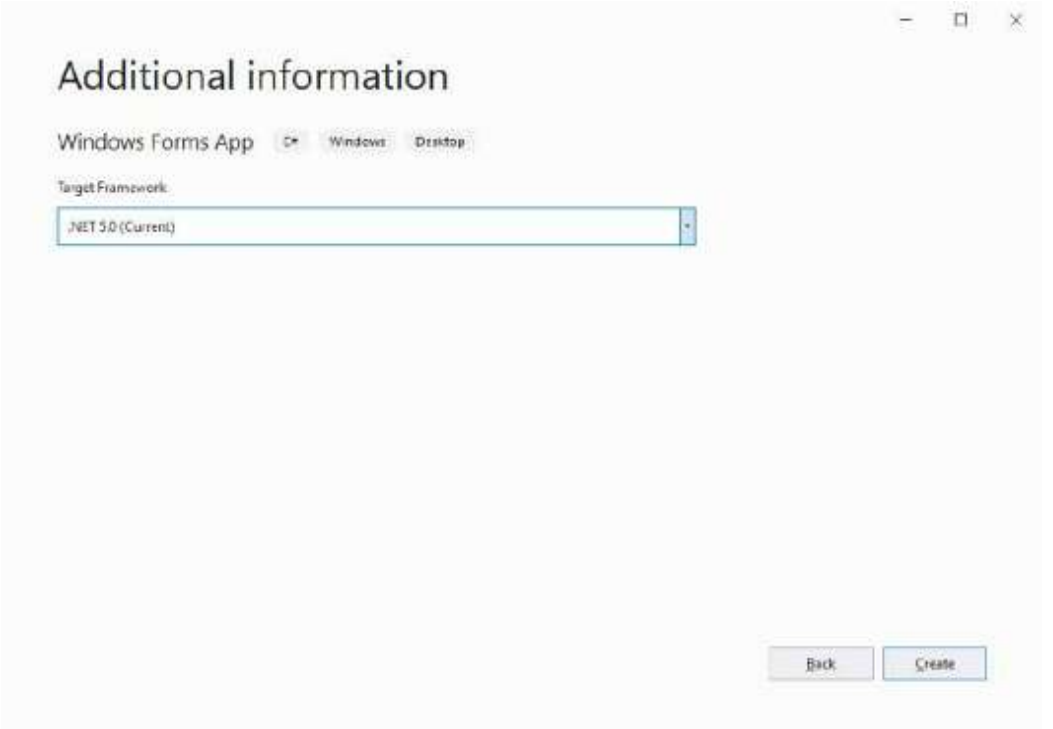
Burada listelenen şablon sayısı, seçili olan Framework versiyonuna göre değişmektedir.

Framework versiyonunu "**New Project**" penceresinde **Next** butonuna tıklayarak gelen "**Configure your new project**" penceresinde değiştirebilirsiniz. Bu pencerede istenilen Framework sürümünü seçtikten sonra **Create** butonuna tıklanarak proje oluşturulur.



Framework 2.0 versiyonunda, yalnızca 7 adet şablon mevcutken **Framework 4.7.2** versiyonunda WPF, WCF, VSTO, Silverlight, Azure, Sharepoint, Universal Windows, .NET Core gibi teknolojilerin de devreye girmesiyle şablon sayısı **80**'e yükselmiştir. Elbette bunlar varsayılan olarak Visual Studio kurulumu ile birlikte yüklü gelen şablonlardır. Geliştireceğiniz uygulama türüne göre yeni şablonlar ekleyebilir ve kullanabilirsiniz.

Windows Forms App ve **Console Application** proje şablonlarını kullanarak Yeni bir .NET Core projesi açmak istediğimizde ise karşımıza yukarıdaki gibi bir Framework seçme penceresi yerine aşağıdaki gibi bir pencere gelecektir. Karşımıza gelen bu pencerede "**Target Framework**" bölümünde Framework versiyonunu değiştirebilirsiniz.

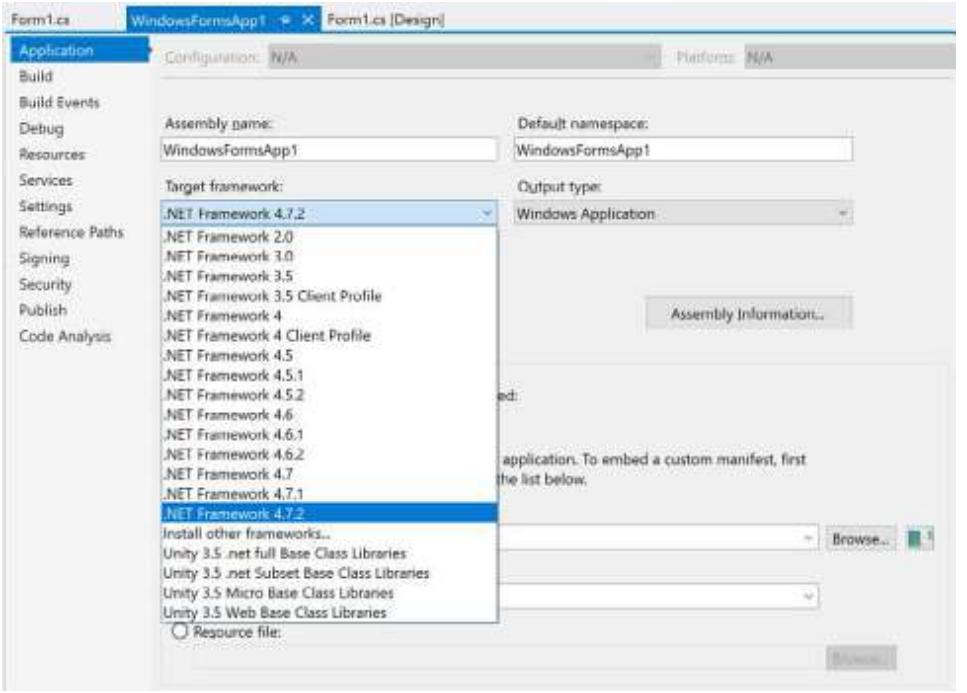


Bu pencerede istenilen Framework sürümünü seçtikten sonra **Create** butonuna tıklanarak proje oluşturulur.

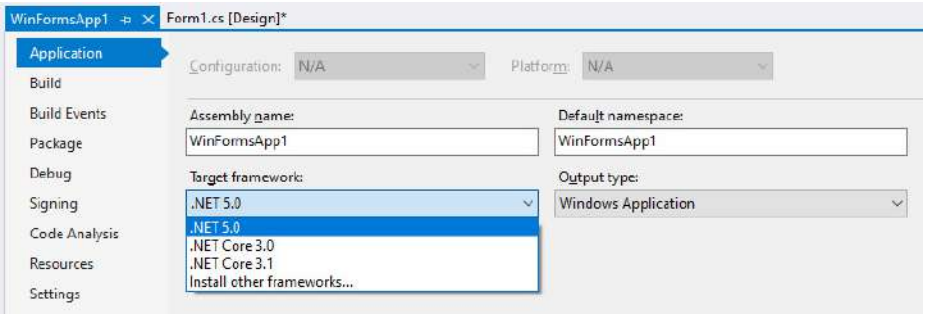
Framework versiyonu yükseldikçe daha farklı projeler geliştirebileceğiniz proje **şablonu sayısı** da artmaktadır. Elbette bunlar varsayılan olarak Visual Studio kurulumu ile birlikte yüklü gelen şablonlardır. Geliştireceğiniz uygulama türüne göre yeni şablonlar ekleyebilir ve kullanabilirsiniz.

.NET Framework projelerinde Framework versiyonu yeni bir proje oluşturulurken değiştirilmekteyken, daha önceden yazdığımız programların Framework versiyonunu değiştirmek içinse aşağıdaki işlemleri gerçekleştirmek gerekmektedir.

- Solution Explorer içerisinde, proje ismine sağ tıklayarak gelen menüden **Properties** seçeneği ile ya da **Project** menüsünden proje adı Properties seçeneğine tıklayarak proje ile ilgili **ayarlar** penceresini görüntüleriz.
- Gelen pencerede **Application** sekmesi altında **Target Framework** başlığı altında istediğimiz Framework versiyonunu seçeriz.



.NET Framework projeleri için yukarıdaki **Target Framework** seçenekleri listelenmektedir.

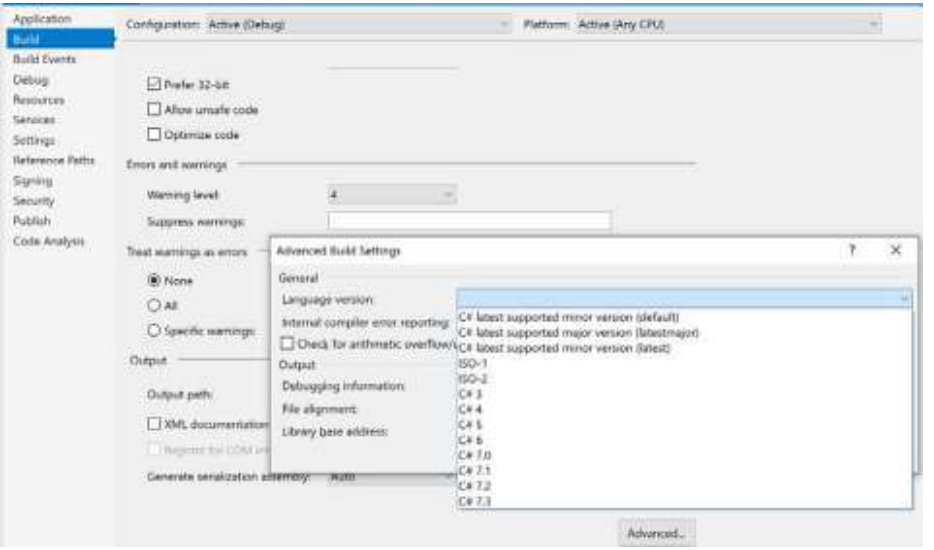


.NET Core projeleri içinse yukarıdaki Target Framework seçenekleri listelenmektedir.

- Bu işlemin ardından projemiz otomatik olarak kapatılarak seçtiğimiz framework versiyonu ile yeniden açılacaktır.

NOT

C# ile geliştirdiğiniz .NET Framework uygulamalarında bazı özellik ve metotları çalıştırabilmek için C#'ın üst versiyonlarını kullanmanız gerekebilir. Kullandığınız C# versiyonunu **ayarlar** penceresinden değiştirebilirsiniz. Proje ayarları penceresini görüntüleyerek, bu pencerede Build başlığı altındaki **Advanced** butonuna tıklayarak **Advanced Build Settings** penceresini açıyoruz.



Language version bölümünden istediğiniz C# versiyonunu seçebilirsiniz.

"**C# latest supported major version (latestmajor)**" seçeneği, yüklü olan en son C# ana sürümünü seçmenizi sağlar.

"**C# latest supported minor version (latest)**" seçeneği ise yüklü olan en son C# sürümünü (*küçük sürümler dâhil*) seçmenizi sağlar. Bu seçenekle her yeni gelen C# güncellemesini otomatik olarak kullanabilirsiniz.

.NET Core uygulamalarında ise kullandığınız Framework sürümüne göre programlama dili sürümü de **otomatik** olarak ayarlanacaktır.

NOT

Visual Studio içerisinde C# 9 dilini .NET 5 Core sürümü ile birlikte kullanabilmek için öncelikle

https://dotnet.microsoft.com/download/dotnet/5.0

linkinden .Net 5.0 SDK'sını yüklemeliyiz. Bu SDK

- NET Runtime 5.0
- ASP.NET Core Runtime 5.0
- .NET Desktop Runtime 5.0

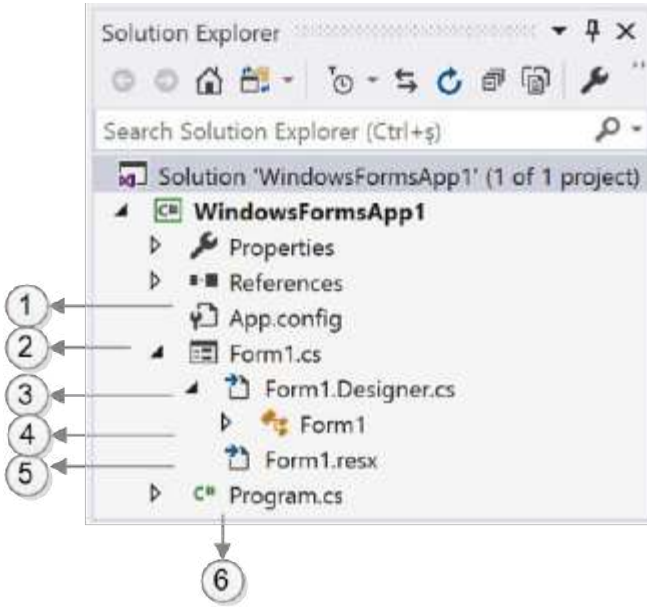
Kütüphanelerini içermektedir.

Bu sayfadan

dotnet-sdk-5.0.X-win-x64.exe

şeklinde versiyonlanan kurulum dosyasını indirerek kuruyoruz.

Yeni bir proje oluşturup **solution explorer** içerisini incelediğimizde, standart olarak bazı dosyaların oluşturulduğunu görürüz. NET Framework projelerinde bu dosyaların isimleri ve açıklamaları şu şekildedir:



(1) **App.config:** Uygulamanız ile ilgili genel tanımlama ve bilgilerin yazıldığı dosyadır. Örneğin Framework versiyonu varsayılan olarak burada yazılıdır.

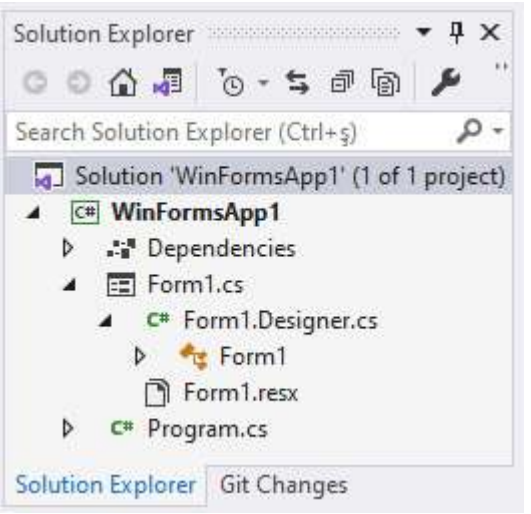
(2) **Form1.cs:** Formumuzun tasarım görünümünü ve kodlarını içeren kısımdır.

(3) **Form1.Designer.cs:** Form içerisinde kullanılan nesnelerin, görsel özellikleri ile ilgili kodların C# tarafından otomatik olarak oluşturulduğu dosyadır.

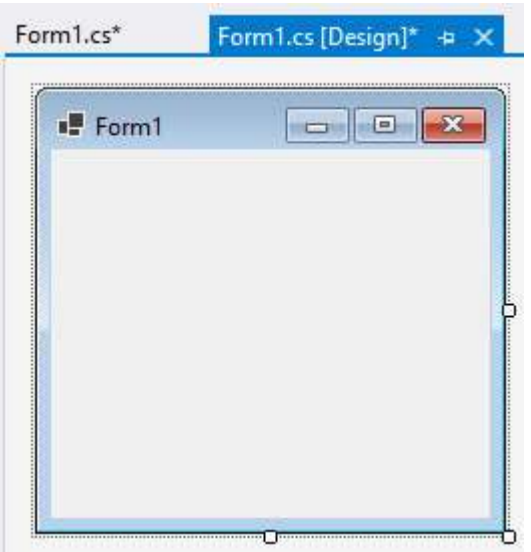
(4) Formumuzun kodlarını içeren kısımdır. Formun kod dosyası disk üzerinde **Form1.cs** ismi ile saklanır. Bu simgeyi tıklayarak kod sayfasına geçiş yapabilirsiniz.

(5) **Form1.resx:** Formunuzda kullanmış olduğunuz *resim, müzik, yazı vb.* dosyaları saklamak için bu **resource** dosyasını kullanabilirsiniz.

(6) **Program.cs:** Projemizin ortam ayarları ile ilgili işlemlerin yapıldığı dosyadır. Örneğin bu dosya içerisinden proje içerisinde birden fazla form var ise bunlardan hangisinin başlangıç formu olarak görüntüleneceğini belirleyebiliriz.

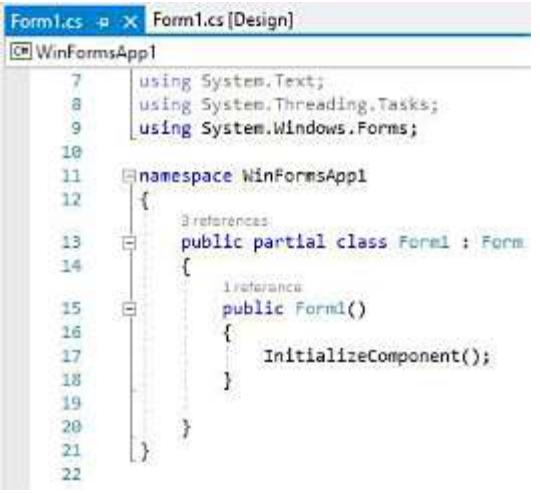


.NET Core projelerinde ise App.config dosyası proje klasöründe yer almadığı için Solution Explorer içerisinde görüntülenmez.



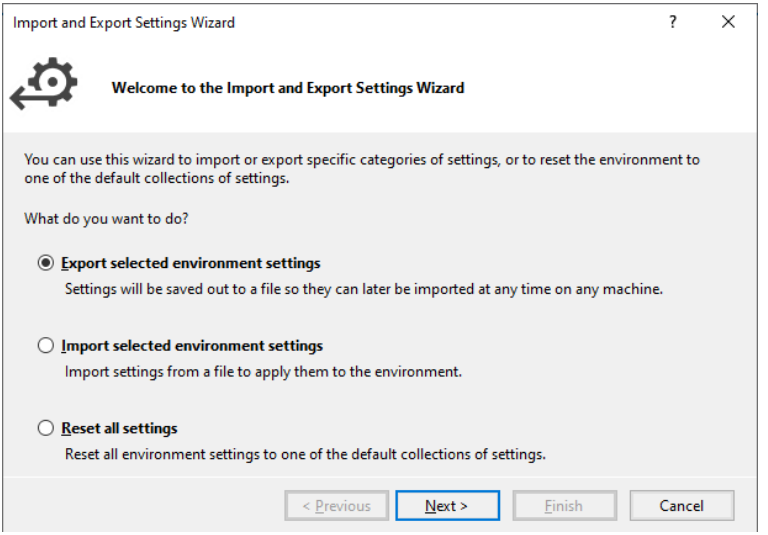
Açılan form ya da kod pencereleri **IDE** penceresi içerisinde sekmeler hâlinde sıralanır.

Tasarım görünümünde açılan form, "**FormAdı.cs[Design]**" sekmesi şeklinde isimlendirilerek görüntülenecektir.



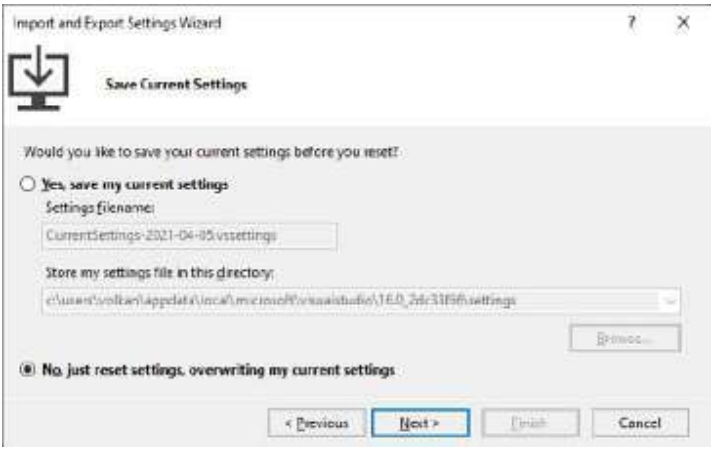
Kod yazma penceresi içerisinde açılan kod ise `[FormAdı.cs]` biçiminde isimlendirilecektir.

Visual Studio ilk kurulum esnasında varsayılan olarak hangi geliştirme ortamını kullanmak istediğinizi size sorarak daha sonraki açışlarda, bu açış ayarını kullanarak ilgili programlama dili ve şablonlarını listeler. Eğer daha sonra bu ayarı değiştirmek istersek **Tools** menüsünden, **Import and Export Setting Wizard** seçeneği tıklanarak gelen pencereden **Reset all settings** seçeneği işaretlenir ve **Next** butonu tıklanır.



Bu işlemi yapmadan önce **Export selected environment settings** seçeneğini kullanarak ayarlarınızı yedekleyebilir ya da aynı penceredeki **Import selected environment settings** seçeneği ile daha önceden kaydettiğiniz ayarları yeniden yükleyebilirsiniz.

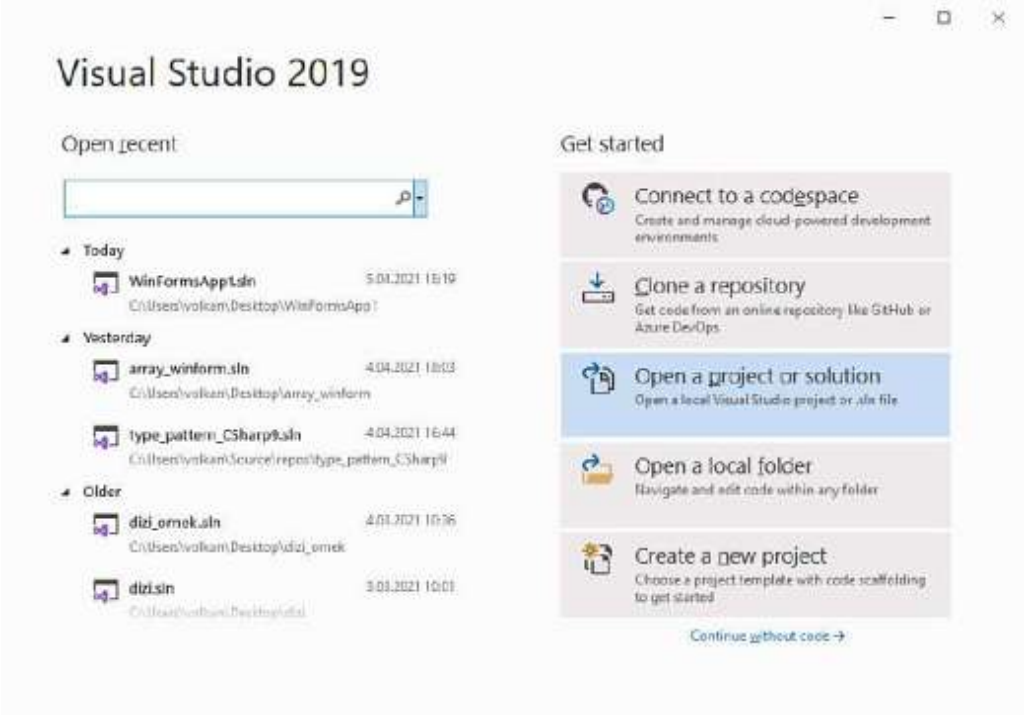
Reset All Settings seçeneği seçilip **Next** butonu tıklanarak gelen son penceremizde ayarlarımızı sıfırlarız.



KAYITLI PROJELERİ AÇMAK

Daha önceden mevcut olan projeleri açmak için şu seçenekler kullanılabilir:

- **Start Page** ekranından **Get Started** başlığı altındaki **Open a Project or solution** seçeneği tıklanabilir.



- **File** menüsünden **Project/Solution** seçeneği kullanılabilir.
- **Ctrl+Shift+O** tuşları kullanılabilir.
- Araç çubuğundan simgesi tıklanabilir.

Bu seçeneklerden herhangi biri kullanılarak **Open Project** penceresi görüntülenir.

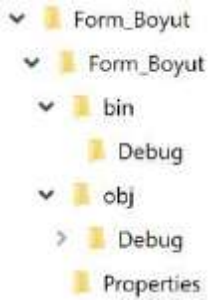
Projeleri Kaydetmek

Bir projeyi kaydetmek için şu seçenekler kullanılabilir:

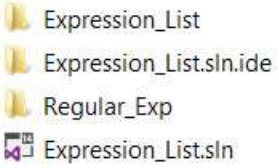
- **File** menüsünden **Save All** seçeneği kullanılabilir.
- **Ctrl+Shift+S** tuşları kullanılabilir.
- Araç çubuğundan  simgesi tıklanabilir.

Bu seçeneklerden herhangi biri kullanılarak **Save Project** penceresi görüntülenir.

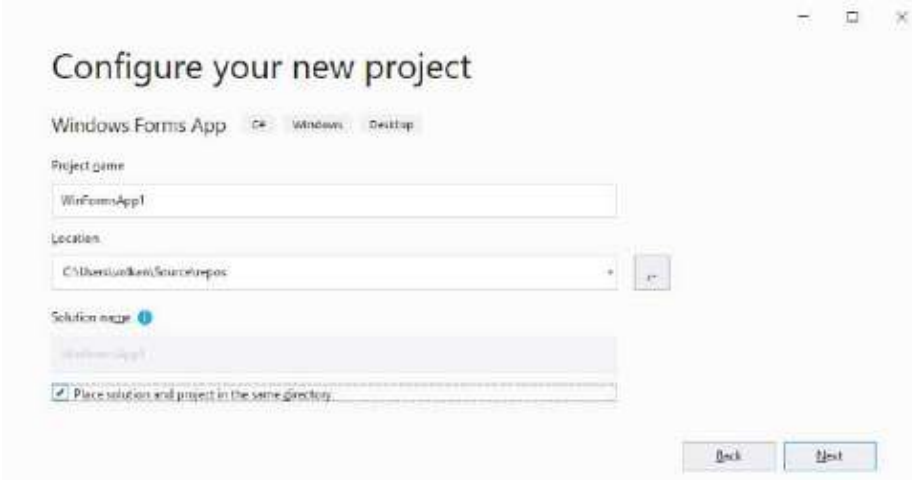
Varsayılan olarak **solution** ile aynı ada sahip bir proje klasörü oluşturulur.

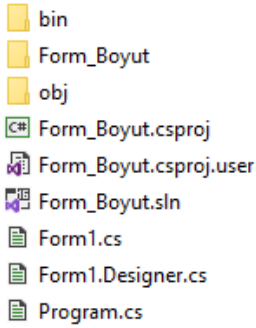


Eğer **solution** içerisinde birden fazla proje mevcutsa her proje için ayrı bir klasör oluşturulur.



Eğer solution ile proje dosyalarının aynı klasörde olması istenirse yeni bir proje oluşturulurken "**Configure your new project**" penceresinde yer alan "**Place solution and project in the same directory**" seçeneği işareti kaldırılmalıdır.





Proje klasörü içerisine baktığımızda farklı uzantılara sahip dosyalar görürüz.

Bu uzantıların başlıcaları şunlardır:

. csproj: Proje dosyası uzantısıdır.

.user: Kullanıcılara ait IDE ayarlarının saklandığı dosya uzantısıdır.

.sln: Solution dosyası uzantısıdır. Solution içerisinde yüklü olan projelerin bilgisini içerir.

.cs: Proje içerisinde kullanılan formların kaynak kodlarını içerir.



Yukarıdaki dosya uzantıları .NET Core proje klasöründe yer almaktadır. .NET Framework proje klasörü içeriği de .user uzantılı dosya hariç aynıdır. .NET Framework proje klasörü içerisinde .user uzantılı dosya yer almaz.

Projeleri Derlemek

Derleme işlemi bir uygulamanın çalıştırılarak eğer varsa hatalarından arındırılması demektir.

Bir projeyi derlemek için şu seçenekler kullanılabilir:

- **Debug** menüsünden **Start Debugging** seçeneği kullanılabilir.
- **F5** tuşu kullanılabilir.
- .NET Framework projeleri için araç çubuğundan 

simgesi tıklanabilir. .NET Core projeleri içinse aynı simge 

şeklinde proje ismi ile görüntülenecektir.



Derleme seçeneğinde varsayılan olarak seçili olan Debug mod'u kullanmak yerine Release modunu da kullanabilirsiniz.

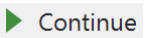
Debug mod, özellikle proje geliştirme aşamasındayken çalıştırma amaçlı olarak kullanılır. Sembolik hata ayıklama bilgisi yer alır. Kodlar **optimize edilmez**.

Release mod ise proje bitip, son kullanıcıya sunulurken derleme yapmak amacıyla kullanılır. Sembolik hata ayıklama bilgisi yer almaz. Ayrıca Release modda kodlar daha **optimize edilmiş hâlde** derlenir.

Derleme esnasında projeyi duraklatmak için ise:

- **Debug** menüsünden **Break All** seçeneği kullanılabilir.
- **Ctrl+Alt+Break** tuşları kullanılabilir.
- Araç çubuğundan  simgesi tıklanabilir.

Durdurulmuş olan projeyi devam ettirmek için:

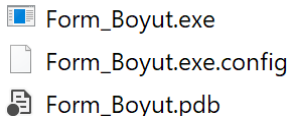
- **Debug** menüsünden **Continue** seçeneği kullanılabilir.
- **F5** tuşu kullanılabilir.
- Araç çubuğundan 

simgesi tıklanabilir.

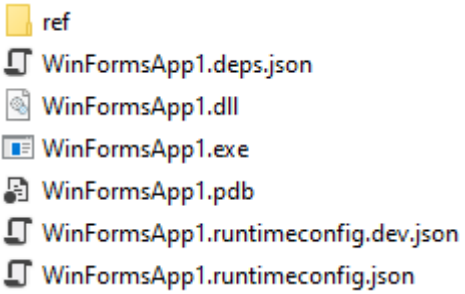
Derleme işlemini sonlandırmak için:

- **Debug** menüsünden **Stop Debugging** seçeneği kullanılabilir.
- **Shift+F5** tuşları kullanılabilir.
- Araç çubuğundan  simgesi tıklanabilir.

.NET Framework projeleri için derleme işlemi esnasında oluşan exe dosyası proje klasörü içerisindeki **Bin** klasörü altında bulunan **Debug** klasörü içerisinde oluşturulur.



.NET Core projeleri içinse derleme işlemi esnasında oluşan exe dosyası proje klasörü içerisindeki **Bin/Debug** klasörü altında bulunan **net5.0-windows** klasörü içerisinde oluşturulur.



Dll dosyalarını derlemek için ise **Build** menüsü kullanılır.

Build menüsü altındaki;

Build Solution seçeneği ya da **Ctrl+Shift+B** kısayol tuşları ile dll dosyası derlenir.

Rebuild Solution seçeneği ile proje üzerinde yapılan değişiklikler uygulanarak proje yeniden derlenir.

Clean Solution seçeneği ile oluşturulmuş olan dll dosyası ve ilgili proje dosyaları silinir.

Derleme işlemi esnasında oluşan **dll** dosyasına, proje klasörü içerisindeki **Bin** klasörü altında bulunan **Debug** klasöründen erişebilirsiniz.



C# .NET komut satırı derleyicisini kullanarak **not defteri** programıyla bile kod yazıp bunları komut satırında derleyebilirsiniz. C# .NET komut satırı derleyicisi **C:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise\MSBuild\Current\Bin\Roslyn** klasörü altında **csc.exe** adı ile bulunmaktadır.

Örneğin:

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
       Console.Write("Konsol Uygulaması");
                Console.ReadLine();
        }
    }
}
```

Şeklindeki kodu not defterinde yazıp **konsol.cs** adı ile kaydedelim.

cmd komut satırını açıp **C:\Program Files (x86)\ Microsoft Visual Studio\2019\Enterprise\ MSBuild\Current\Bin\Roslyn** şeklindeki klasör içerisinde geçiş yapalım.

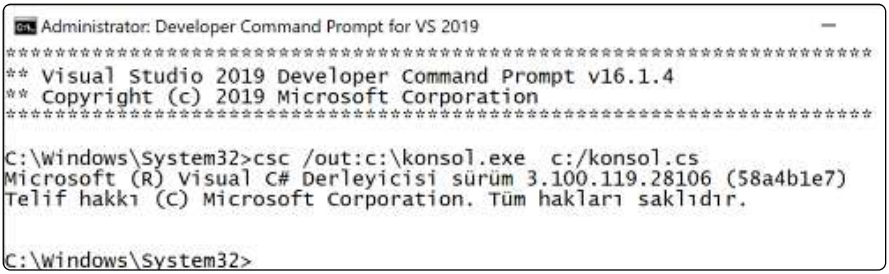
Komut satırını Visual Studio için düzenlenmiş hâli ile çalıştırmak için **Start** menüsü altında bulunan **Visual Studio 2019** menüsünde, altındaki **Developer Command Prompt for VS2019** menüsünü kullanabilirsiniz.

Komut satırında iken:

```
csc /out:c:\konsol.exe  c:/konsol.cs
```

ifadesini yazarak `out` parametresini kullanarak **C** sürücüsünde mevcut bulunan

konsol.cs. uygulamasını yine **C** sürücüsünde **konsol.exe** adıyla derliyoruz. Eğer herhangi bir hata bulunmazsa aşağıda görüldüğü gibi kodumuz derlenir ve **exe** uzantılı hâle getirilir.



CSC komut satırı derleyicisi ile birlikte **`out`** parametresi dışında, aşağıdaki parametreleri de kullanabiliriz:

`/platform`: Uygulamamızı farklı işletim sistemi platformları için kaydetmek amacı ile kullanılır.

Örneğin;

```
csc /platform:x64 c:\konsol.cs
```

konsol.cs dosyası, **konsol.exe** adı ile **64 bit** olarak derlenir.

`/target`: Derlenecek olan dosyanın kayıt biçimini belirler. Aşağıdaki değerlerle kullanılır;

`/target:exe`: Konsol uygulamasında kullanılır.

`/target: library` : Dll uygulamasında kullanılır.

`/target:winexe` : Windows form uygulamasında kullanılır.



`/target` parametresi **`/t`** şeklinde de kullanılabilir.

Örneğin;

```
csc /target:exe c:\konsol.cs
```

konsol.cs dosyası **konsol.exe** adı ile derlenir.

/reference : Derlenecek dosya ile birlikte belirtilen **dll** dosyasının da **exe** dosyası içerisine eklenmesini sağlar. Eğer birden fazla **dll** dosyası eklenecekse aralarında "**;**" işareti kullanılarak eklenmelidir.

NOT **/reference** parametresi **/r** şeklinde de kullanılabilir.

```
csc /r:System.Windows.Forms.dll c:\konsol+winform.cs
```

System.Windows.Forms.dll sınıfını uygulamamıza dâhil ederek bu sınıfa ait olan mesaj penceresi nesnesini kullandık.

NOT Birden fazla dll dosyasını eklemek için aralarında "," işareti ile yazılabilir.

```
csc /r:System.Windows.Forms.dll,System.Management.dll c:\konsol+winform.cs
```

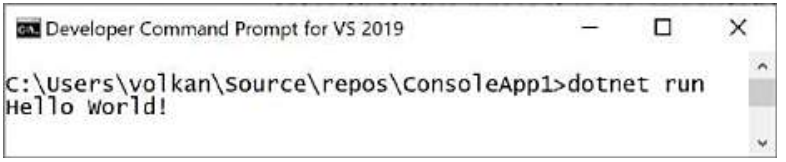
Eğer bir **dll** dosyasını birden fazla uygulamaya eklemek istiyorsak uygulamalar aralarında boşluk bırakarak yazabiliriz.

```
csc /r:System.Windows.Forms.dll c:\konsol+winform.cs c:\mesaj.cs
```

NOT NET Core projelerini komut satırında iken

`dotnet run`

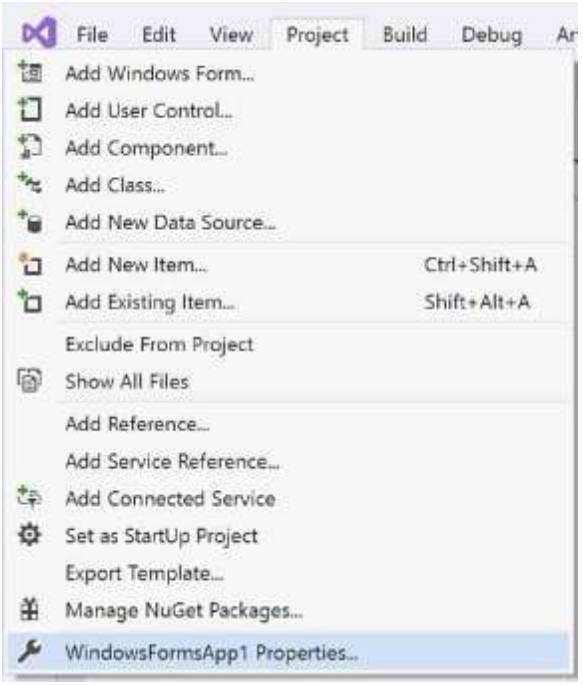
komutunu yazarak derleyebiliriz.



Uygulama çıktısı aynı komut ekranında görüntülenecektir.

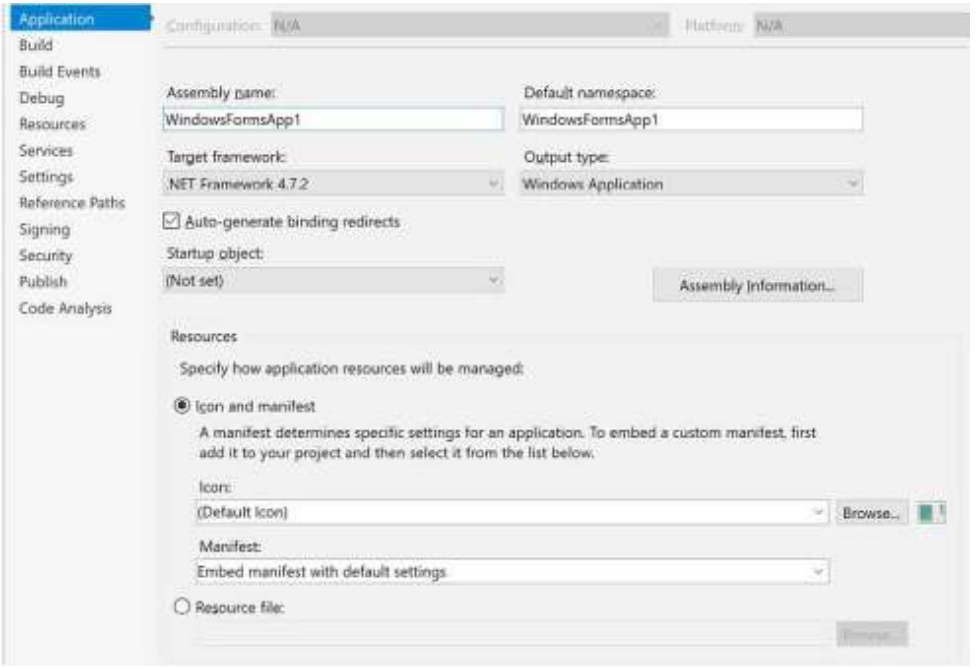
Proje Özelliklerini Ayarlamak

Üzerinde çalışılan aktif proje ile ilgili özellikleri ayarlamak için **Project** menüsünden, **Proje Adı Properties** seçeneği kullanılabilir.



Aynı ayar penceresine, **Solution Explorer** içerisinde farenin sağ tuşuna tıklanarak gelen menüden, **Properties** seçeneği ya da **Alt + Enter** tuşları ile ulaşılabilir.

Bu işlemler sonucunda proje ile ilgili ayarları yapabileceğimiz **ayarlar** penceresi görüntülenecektir.



Bu pencerede yer alan sekmeleri sırasıyla inceleyerek önemli olan bazı özellikleri açıklayalım.

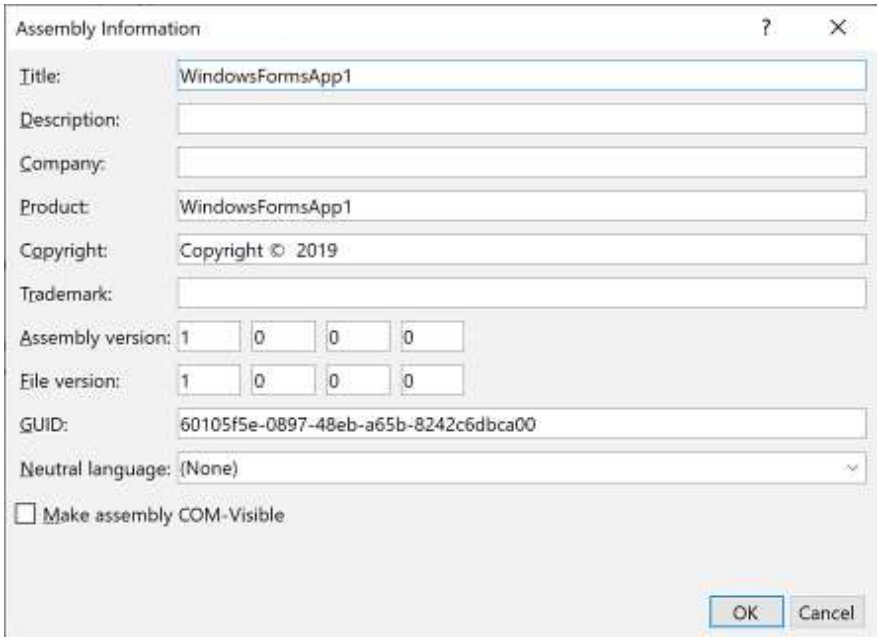
- **Application** sekmesinde uygulama ile ilgili genel ayarlar gerçekleştirilir.

  Assembly name seçeneği ile oluşturulacak olan **exe** dosyasının ismi, **Default namespace** seçeneği ile varsayılan **namespace** ismi belirlenir.

  Target framework seçeneği ile projemizin framework versiyonu görüntülenir. İstenildiği takdirde framework versiyonu değiştirilebilir.

  Output type seçeneği ile proje tipi görüntülenir. İstenildiği takdirde proje tipi değiştirilebilir. 3 çeşit değer alabilir; **Windows Application**, **Console Application**, **Class Library**.

Startup object seçeneği ile projenin başlangıcında hangi formun görüntüleneceği belirlenir.



Assembly Information butonuna tıklanarak proje ile ilgili açıklama, dil ve versiyon bilgilerini girebileceğimiz bir pencere görüntülenir.

NOT

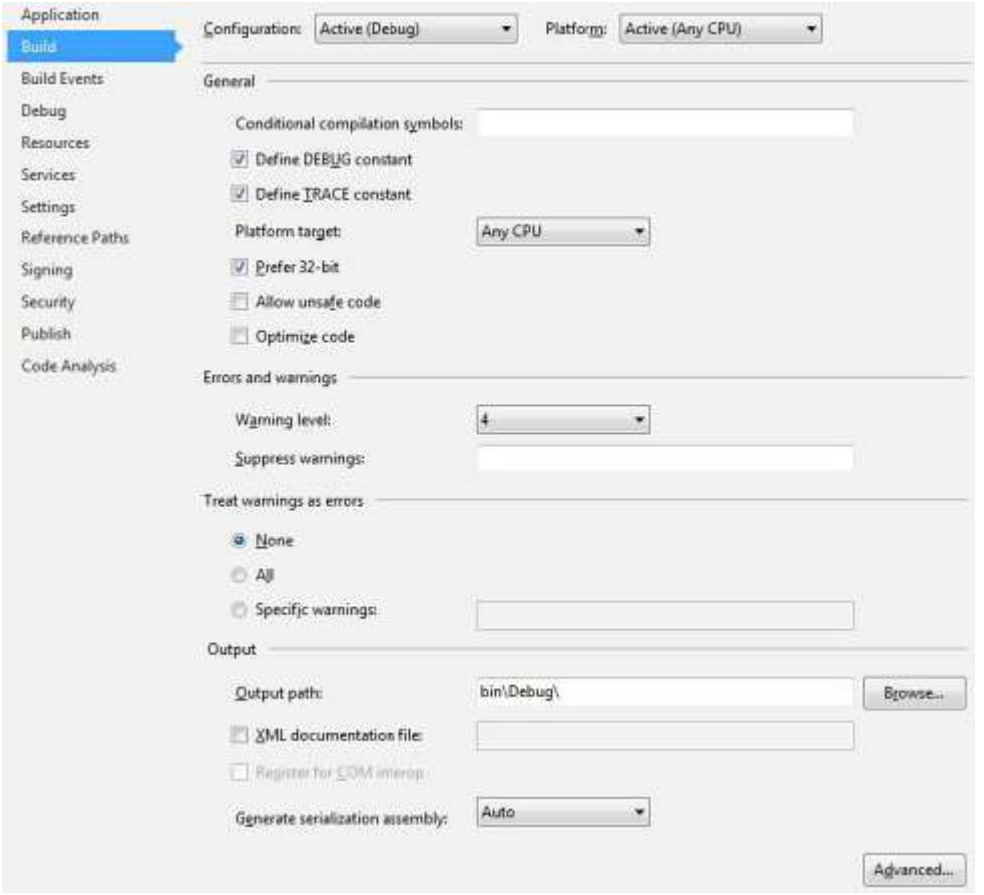
Assembly Information butonu .NET Framework projeleri için geçerlidir. .NET Core projelerinde ise bu seçenekler **Package** sekmesi altında yer almaktadır.

Resources bölümünde projede kullanılacak olan kaynaklarla (*resim, simge, yazı, ayar bilgileri vb.*) ilgili belirlemeler yapılır. **Manifest dosyaları**, projelerimizde kullandığımız **assembly**'ler ile ilgili bilgilerin saklandığı dosyalardır.

Bu bölümde **Icon and Manifest** seçeneği seçili iken **Icon** seçeneği ile programımızın simgesini belirleyebiliriz. **Manifest** bölümünde "**Embed manifest with default settings**" seçeneği ile manifest dosyalarının proje içerisinde gömülü hâlde bulunmasını sağlarken; "**Create application without a manifest**" seçeneği ile de manifest dosyalarının proje içerisinde bulunmamasını sağlar.

Resource file seçeneğinde ise projede kullanacağımız kaynaklarımızı bir resource dosyası içerisinde oluşturmuşsak oluşturmuş olduğumuz resource dosyasını seçeriz.

- **Build** sekmesinde sınıfların derlenmesi ile ilgili genel ayarlar gerçekleştirilir.



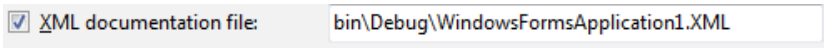
General başlığı altındaki;

Platform target seçeneği ile projemizin hangi tip işlemciler ile çalıştırılacağı belirlenir. 3 adet değer alır. Bunlar, **Any CPU**, **x86** ve **x64** değerleridir.

Output başlığı altındaki;

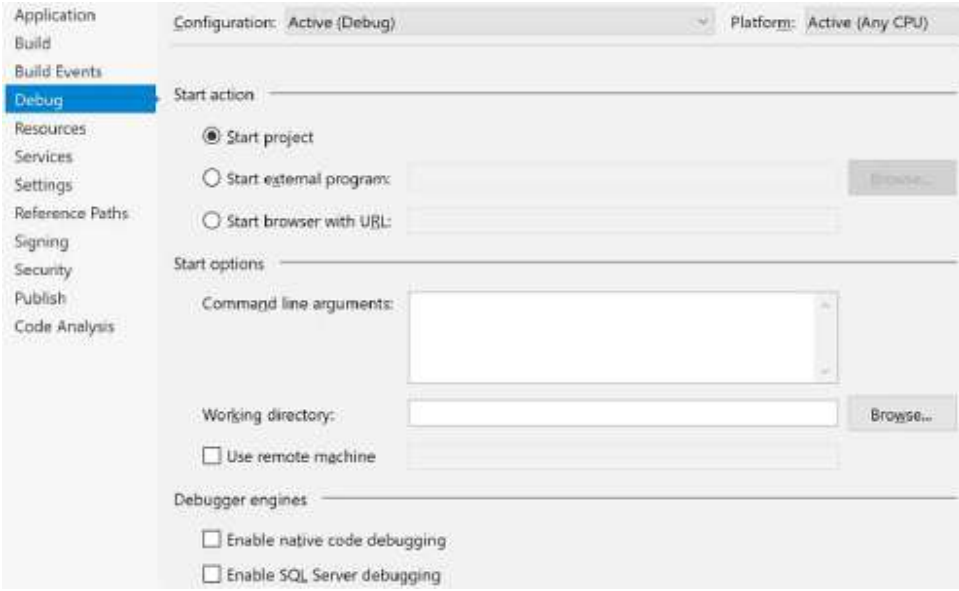
Output path bölümünde, sınıfımızın derleneceği klasör yolu belirtilir.

XML documentation file seçeneği işaretli iken sınıfımız içerisinde kullanacağımız XML yorum dosyasının ismi ve klasör yolu belirlenir.



- **Build events** sekmesinde derleme işlemi başlamadan önce ya da tamamlandıktan sonra çalıştırılacak **özel komutlar** yazılabilir. Bir proje derlenmeden önce gerçekleştirilecek olaylar **prebuildevent.bat** adlı bir dosyaya eklenirken, derleme işleminin bitiminden sonra gerçekleştirilecek olaylar postbuildevent.bat adlı bir dosyaya eklenir.

  Debug sekmesinde, projelerin derlenmesi ile ilgili genel ayarlar gerçekleştirilir.



Start Action başlığı altındaki seçenekler ile projenin başlatılması esnasında gerçekleştirilecek işlemler belirlenir.

Start Project seçeneği işaretli iken herhangi bir işlem yapılmadan proje başlatılır.

Start external program seçeneği işaretli iken belirtilen program, projenin başlatılması esnasında çalıştırılır.

Start browser with URL seçeneği işaretli iken belirtilen internet adresi projenin başlatılması esnasında görüntülenir.

Working directory seçeneği ile derleme sonucu oluşan **exe** uzantılı dosyanın hangi klasörde oluşturulacağı belirlenir.

Debugger Engines başlığı altındaki;

Enable native code debugging seçeneği ile **native code debugging** desteği sağlanır.

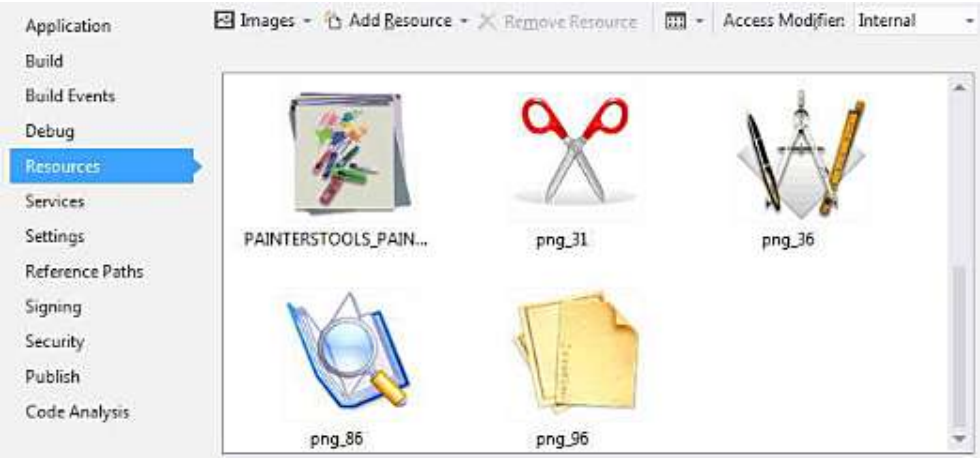
Enable SQL Server debugging seçeneği ile **SQL server** uygulamaları için **debugging** desteği sağlanır. Projelerimizin kaynak kodlarının derlenerek bilgisayarın anlayacağı biçime dönüştürülmüş hâline **native code** adı verilir.

NOT .NET Core projeleri için Debug sekmesinde yukarıdaki seçeneklere ek olarak **Launch** bölümü de yer almaktadır.



Launch başlığı altında uygulamanın çalışma şekli ile ilgili seçenekler yer alır. Burada yer alan;

- **Snapshot debugger** seçeneği projeniz üzerinde yaptığınız değişiklikler istediğiniz zaman geri almanızı sağlar.
- **Project** seçeneği ile herhangi bir işlem gerçekleştirilmeden proje başlatılır.
- **Executable** seçeneği uygulamanızın çalıştırılabilir bir .exe dosyası hâline getirilmesi sağlanır.
- **Resources** sekmesinde, proje içerisinde kullanacağımız yazı, resim gibi ögeleri saklayabileceğimiz **resource** dosyaları ile ilgili işlemleri gerçekleştirebiliriz.



Resim, yazı gibi dosyaları **ekleyebilir**, ya da **silebiliriz**. Bu dosyalar **resource** dosyası içerisinde gömülü olarak proje içerisinde yer alacaktır.

- **Settings** sekmesinde, projelerimiz ile ilgili basit ayarlar yapabiliriz. Örneğin proje formlarının ekran koordinatları, çözünürlük değerleri, renk tercihleri gibi…

  Settings sekmesinde 4 adet özellik vardır:

  Name: Ayar bilgisinin kod içerisinde temsil edilen ismini belirtir.

  Type: Ayar bilgisinin türünü belirtir. Örneğin renk ayarları ile ilgili bir işlem yapılacaksa bu ayar bilgisinin türünü **System.Color** türünden belirtmemiz gerekmektedir.

  Scope: Ayar bilgisine erişim şeklini belirtir. **User** ve **Application** biçiminde iki değer alabilir.

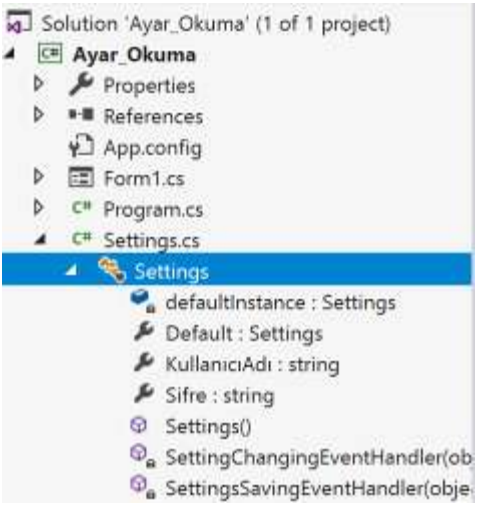
  Value: Ayar bilgisinin içerdiği değeri belirtir.

  Örnek olarak kullanıcı adı ve şifre bilgilerini settings dosyasından okuyan bir **Windows form** uygulaması yapalım.

  Settings sekmesinde değerleri aşağıdaki gibi giriyoruz:

	Name	Type	Scope	Value
▶	KullanıcıAdı	string	User	Volkan
	Sifre	string	User	123

Solution Explorer içerisinde listelenen **Settings.Designer.cs** dosyasını inceleyelim.



Kaynak kodu incelediğimizde **`KullanıcıAdı`** ve **`Sifre`** isimli değerler için metotlar oluşturulduğunu görebiliriz.

```
[global::System.Configuration.UserScopedSettingAttribute()]
[global::System.Diagnostics.DebuggerNonUserCodeAttribute()]
[global::System.Configuration.DefaultSettingValueAttribute("Volk
an")]
public string KullanıcıAdı {
    get {
        return ((string)(this["KullanıcıAdı"]));
    }
    set {
        this["KullanıcıAdı"] = value;
    }
}

[global::System.Configuration.UserScopedSettingAttribute()]
```

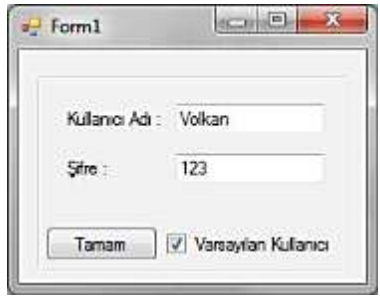
```
[global::System.Diagnostics.DebuggerNonUserCodeAttribute()]
[global::System.Configuration.DefaultSettingValueAttribute("123")]
public string Sifre {
    get {
        return ((string)(this["Sifre"]));
    }
    set {
        this["Sifre"] = value;
    }
}
```

Form üzerine kullanıcı adı için bir adet, şifre için de bir adet olmak üzere iki adet **`textBox`**, bir de bir adet **`checkBox`** nesnesi ekliyoruz.

Kod olarak da aşağıdaki kod satırlarını yazalım:

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    if (checkBox1.Checked)
    {
        textBox1.Text = Properties.Settings.Default.KullanıcıAdı;
        textBox2.Text = Properties.Settings.Default.Sifre;
    }
    else
    {
        textBox1.Clear();
        textBox2.Clear();
    }
}
```

Görüldüğü gibi **Settings** dosyasındaki **`KullanıcıAdı`** ve **`Sifre`** isimli değerlere **`Properties.Settings.Default`** özelliği altından ulaşabiliyoruz.



Uygulamamızı çalıştırıp **Varsayılan Kullanıcı** adlı **checkBox**'ı işaretlediğimizde kullanıcı adı ve şifre değerleri settings dosyasından okunarak **`textBox`**'lar içerisine aktarılır.

> **NOT** Aynı işlemi App.config dosyasını kullanarak da gerçekleştirebiliriz. App.config dosyasının uygulamanız ile ilgili genel tanımlama ve bilgilerinin saklandığı bir dosya olduğunu daha önce belirtmiştik.

Örneğimiz için **App.config** dosyasının içeriğini aşağıdaki gibi düzenliyoruz:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".
NETFramework,Version=v4.5" />
  </startup>
<appSettings>
  <add key="KullanıcıAdı" value="Volkan"/>
    <add key="Sifre" value="123"/>
  </appSettings>
</configuration>
```

Kodumuzda **`<appSettings> </appSettings>`** tag'leri arasında yazılı olan kod, bizim sonradan eklediğimiz koddur. **`key`**, **`value`** şeklinde anahtar ve değerleri ekleyip **App.config** dosyasını kaydediyoruz.

Ardından form arayüzünde aşağıdaki kod satırlarını yazıyoruz.

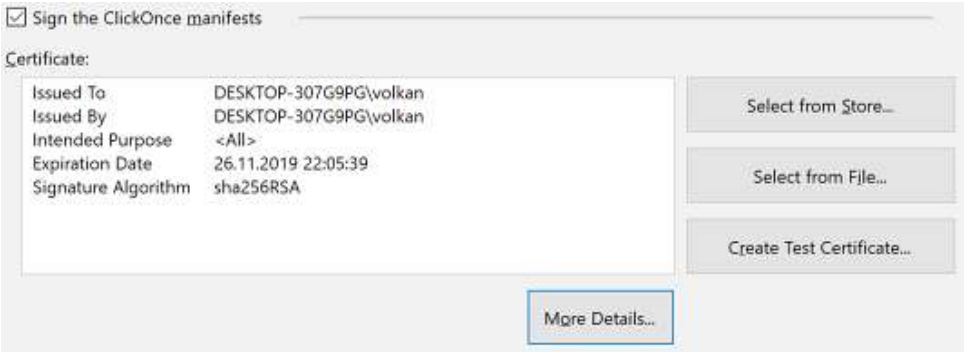
```
using System;
using System.Windows.Forms;
using System.Configuration;
namespace App_Config
{
  public partial class Form1 : Form
  {
    public Form1()
    {
        InitializeComponent();
    }
    private void checkBox1_CheckedChanged(object sender, EventArgs
```

```
e)
    {
        if (checkBox1.Checked)
        {
            textBox1.Text = ConfigurationManager.
AppSettings["KullanıcıAdı"];
            textBox2.Text = ConfigurationManager.
AppSettings["Sifre"];
        }
        else
        {
            textBox1.Clear();
            textBox2.Clear();
        }
    }
  }
}
```

Kodumuzda **`ConfigurationManager`** sınıfını kullanarak **App.config** dosyası içerisindeki değerlere ulaşıyoruz. Ancak bu sınıfı kullanabilmek için **System. Configuration** adlı referans dosyasını Add Reference seçeneği ile projemize ekleyip **`using`** ifadesi ile de projemize dâhil etmemiz gerekmektedir.

- **Reference Paths** sekmesinde, projemiz içerisinde kullanacağımız **dll** dosyalarının yolu ayarlanabilir.
- **Signing** sekmesi, projemiz ile ilgili sertifika ya da güvenlik anahtarı oluşturmak amacıyla kullanılır. Bir uygulamayı **ClickOnce** dağıtımını kullanarak yayınlamak istiyorsanız uygulamalarınızın özel bir anahtar dosyası ile imzalanması gerekir. Windows sertifika dosyalarını veya bir güvenlik anahtarı dosyasını kullanarak uygulamalarınızı imzalayabilirsiniz. .Net Core için **ClikOnce** dosyaları projenizin /Build/Publish alt klasörü altında yer alır.

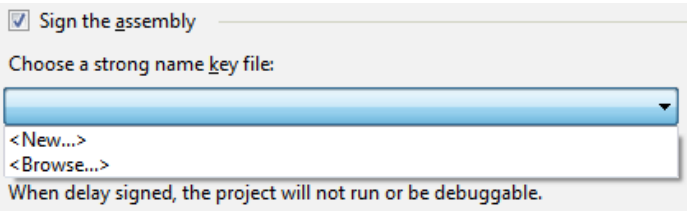
  Bu bölümdeki **Sign the ClickOnce manifests** kısmında oluşturmuş olduğunuz uygulamalarınızı 3 farklı şekilde imzalayabilirsiniz.



- **Select from Store...** seçeneği, mevcut sertifikalardan birinin belli bir süre sınırı ile kullanılabilmesini sağlar.
- **Select from File...** seçeneği, önceden mevcut olan **.pfx** uzantılı sertifika dosyalarının kullanılabilmesini sağlar.
- **Create Test Certificate...** test amaçlı bir sertifika dosyasının oluşturulmasını sağlar.

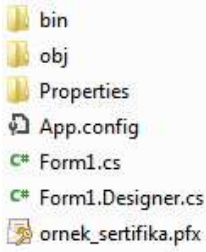
Sign the assembly bölümünde ise **exe** dosyalarınız için **.pfx** uzantılı sertifika dosyaları oluşturabilirsiniz. Bu şekilde uygulamalarınız için benzersiz bir kimlik numarası oluşturarak uygulamalarınızın güvenli olarak kabul edilmesini sağlarsınız.

Bir **assembly** dosyası için sertifika dosyası oluşturmak amacıyla **Sign the assembly** seçeneğini işaretleyerek **New** seçeneğine tıklayalım.



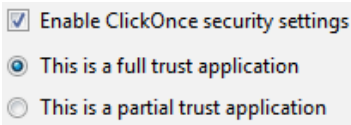
Protect my key file with a password seçeneğini işaretleyerek sertifika dosyalarımıza şifre verilmesini sağlarız. Oluşturacağımız sertifika dosyası için en az **6** karakterden oluşan şifre girmeliyiz.





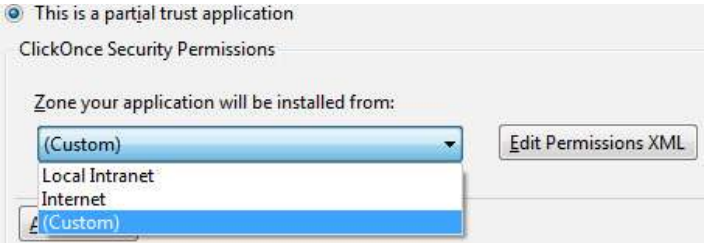
Oluşturmuş olduğumuz sertifika dosyası **solution explorer** içerisine eklenecektir.

- **Security** sekmesinde, projemiz ile ilgili güvenlik ayarlarını yapabiliriz.

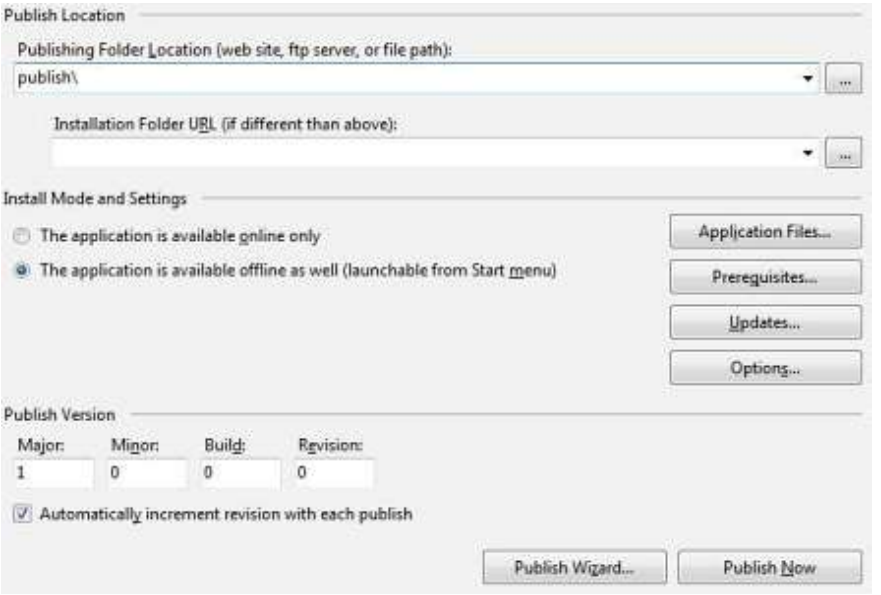


Enable ClickOnce security setting seçeneğini işaretleyerek **ClickOnce** güvenlik ayarlarını yapabiliriz. ClickOnce, Microsoft'un yeni nesil güncelleme sistemidir. **This is a full trust application** seçeneği işaretli iken uygulamanın tamamı **güvenli** olarak kabul edilir.

This is a partial trust application seçeneği işaretli iken uygulamanın belirtilen kısmı güvenli olarak kabul edilir. Bu seçenek işaretli olduğunda **Zone your application installed from** başlığı **aktif** olur. Bu başlık altında listelenen seçenekler ile uygulamanın sadece belirtilen kısımları **güvenli** olarak kabul edilir. **Local Intranet**, **Internet** ve **Custom** değerlerini alır. **Custom** seçeneği seçildiğinde aktifleşen **Edit Permissions XML** butonuna tıklanarak **app.manifest** dosyasının içeriği düzenlenebilir.



- **Publish** sekmesinde, projemizin yayınlanması ile ilgili ayarları yapabiliriz.



Publish Location bölümünde, uygulamamızın yayınlanacağı klasör yolu belirtilir. Bu bir web sitesi, `ftp server` ya da herhangi bir dizin olabilir.

Install Mode and Settings başlığı altında, uygulamanın kurulum modu ve ayarları belirlenir. Bu başlık altındaki;

The application is avalaible online only seçeneği ile uygulamamızın yalnızca **online** olarak çalıştırılmasını,

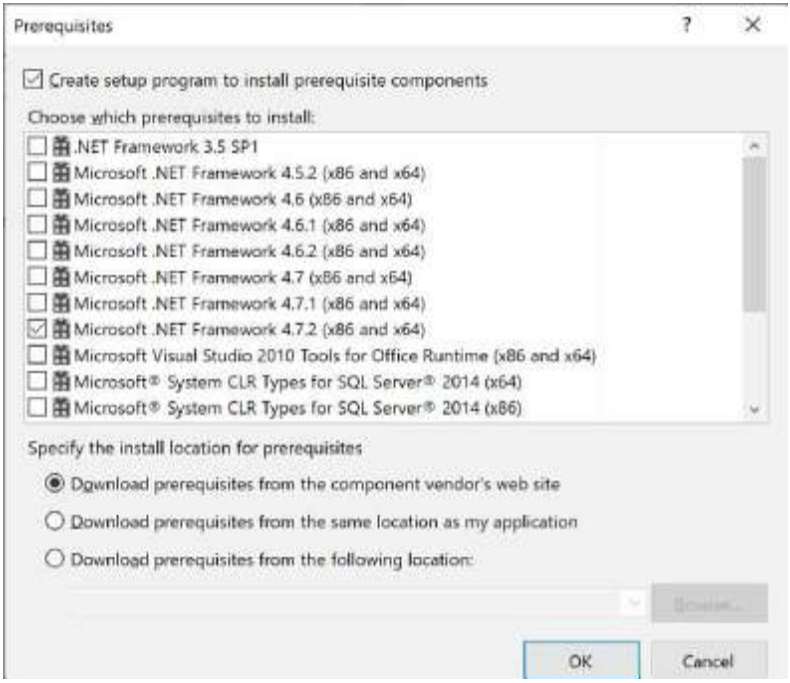
The application is avalaible offline as well (*launchable from Start menu*) seçeneği ile uygulamamızın **offline** olarak çalıştırılabilmesini ayarlarız. Varsayılan olarak bu seçenek seçilidir.

Publish Version seçeneği ile uygulamamızın sürüm numaralarını belirleriz. **Automatically increment revison with each publish** seçeneği işaretli iken uygulamanın her yayınlanmasında sürüm numaraları **otomatik** olarak arttırılır.

Application Files butonuna tıklanarak uygulamamıza ait dosyaları ve bu dosyaların kurulum içerisine dâhil edilip edilmeyeceğini belirleriz.

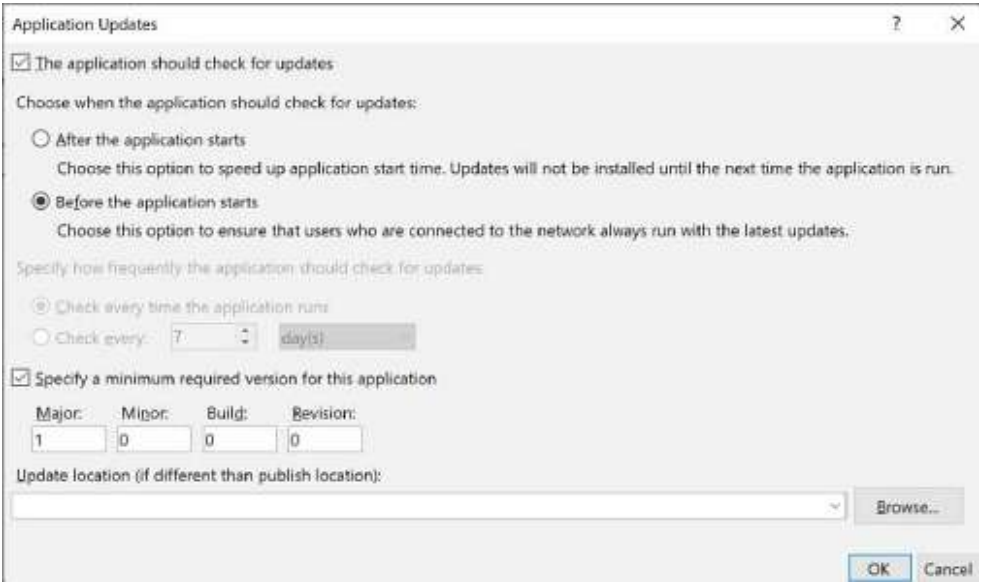
File Name	Publish Status	Download Group	Hash
Expression_List.exe	Include (Auto)	(Required)	Include
Expression_List.exe.config	Include (Auto)	(Required)	Include
Expression_List.exe.manifest	Include (Auto)	(Required)	Include

Prerequisites butonuna tıklanarak uygulamamıza ait dosyaları ve bu dosyaların kurulum içerisine dâhil edilip edilmeyeceğini belirleriz.



Bu penceredeki ilk seçenek olan, "**Download Prerequisites from the component vendor's web site**" seçeneği işaretli iken seçili olan dosyalar bilgisayarda mevcut olmadığı zaman internet sitesinden yüklenir. İkinci seçenek olan "**Download Prerequisites from the same location as my application**" seçeneği ile **setup** uygulamamız içerisine, eklemiş olduğumuz dosyalardan yükleme yapılmasını sağlarız. Son seçenek olan "**Download Prerequisites from the following location**" seçeneği seçilip **Browse** butonu ile belirtilen klasörden yükleme yapılması sağlanır.

Updates butonuna tıklayarak uygulamamızın güncellemesi ile ilgili ayarları yapabiliriz.



"**The application should check for updates**" seçeneğini işaretleyerek uygulamamızın güncellemeleri kontrol etmesini sağlayabiliriz. Bunun için iki ayar seçeneği mevcuttur.

Birinci ayar olan "**After the application starts**" seçeneği ile güncelleme kontrolünün uygulama çalıştırıldıktan sonra yapılması sağlanır.

İkinci ayar olan "**Before the application starts**" seçeneği ile güncelleme kontrolünün, uygulamanın çalıştırılması esnasında yapılması sağlanır.

"**Specify a minimum required version for this application**" seçeneği işaretli iken uygulamanın çalışması için gereken en düşük sürüm numarasını belirleyebiliriz.

Update location bölümünde ise eğer uygulamamızın setup dosyalarının bulunduğu klasör ya da web adresi ile güncelleme dosyalarının bulunduğu klasör ya da web adresi birbirinden farklı ise bu klasör ya da web adresi **Browse** butonuna tıklanarak belirlenir.

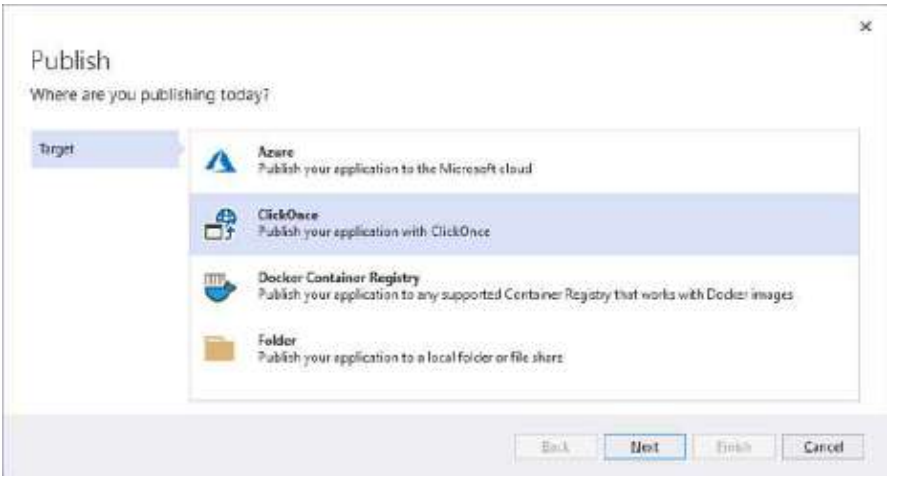
Publish Wizard butonuna tıklayarak ve **Yayınlama Sihirbazı** yardımıyla gereken ayarları yaparak uygulamamızı yayınlarız.

Publish Now butonuna tıklayarak uygulamamızın geçerli ayarlar ile yayınlanması sağlanır.

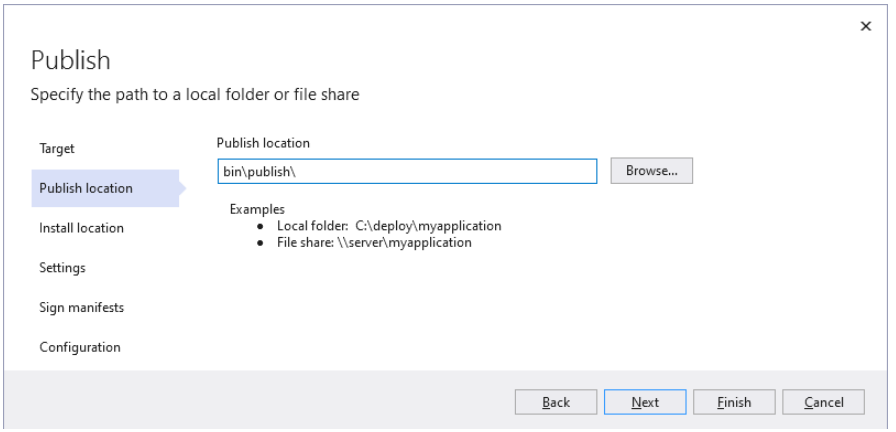
NOT

.NET Core projeleri için properties penceresinde **publish sekmesi yer almamaktadır.** Bu işlemi Build/Publish menüsünden gerçekleştirebilirsiniz. Ayrıca Security sekmesi de properties penceresinde yer almaz. .Net Framework projeleri için properties penceresinde Signing, Security, Publish sekmelerinde yer alan seçenekler ve ayarlar .NET Core projeleri için Build/Publish menüsü içerisinden gerçekleştirilmektedir. Uygulamanız ile ilgili açıklama, dil, versiyon, telif hakkı ve lisans bilgilerini ise properties penceresinden **Package** sekmesi altında girebilirsiniz.

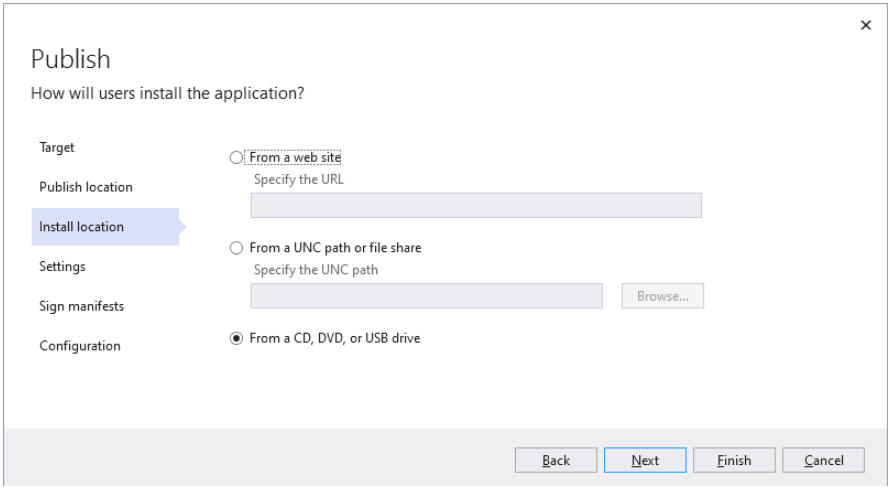
.NET Core projeleri için **Publish** işlemini Build/Publish menüsü içerisinden adım adım gerçekleştirelim. İlk olarak karşımıza aşağıdaki gibi bir pencere gelmektedir:



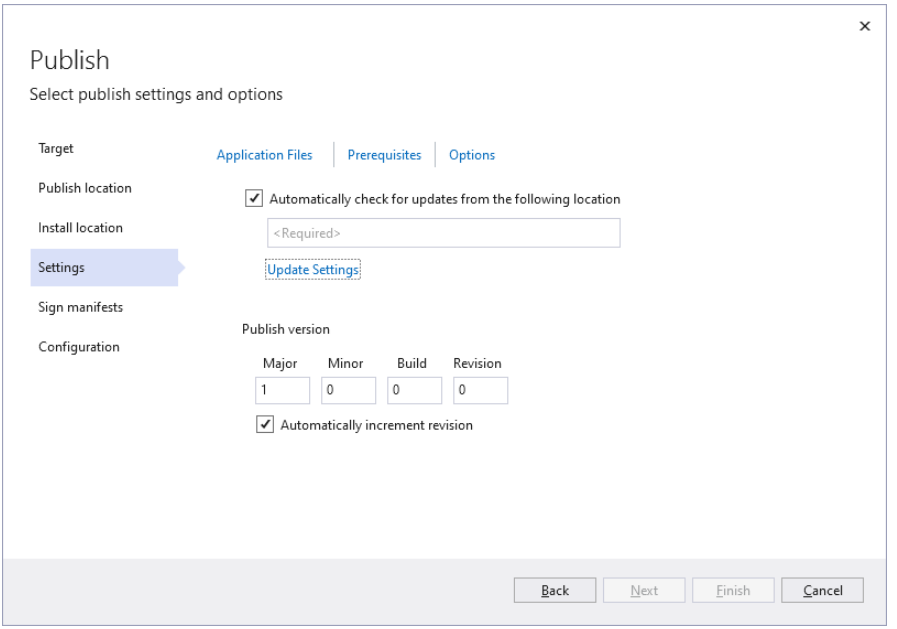
Bu pencerede yer alan seçeneklerle belirtilen ortamlarda uygulamanızı yayınlayabilirsiniz. Azure seçeneği ile Microsoft Azure ortamında, Docker Container Registry seçeneği ile ise docker imajı olarak uygulamalarınızı yayınlayabilirsiniz. Folder seçeneği ile uygulamanızı yerel ya da web ortamında bir klasörde yayınlayabilirsiniz. Aynı seçenek ClickOnce bölümünde de gerçekleştirilebilir. Biz örnek olarak **ClickOnce** seçeneğini kullanarak bir kurulum hazırlayacağız. ClickOnce seçeneğini seçerek **Next** butonuna tıkladığımızda karşımıza aşağıdaki seçenekler gelir:



"Publish location" bölümünde kurulumun hazırlanacağı yerel ya da web klasörü yolu belirtilerek **Next** butonuna tıklanır.



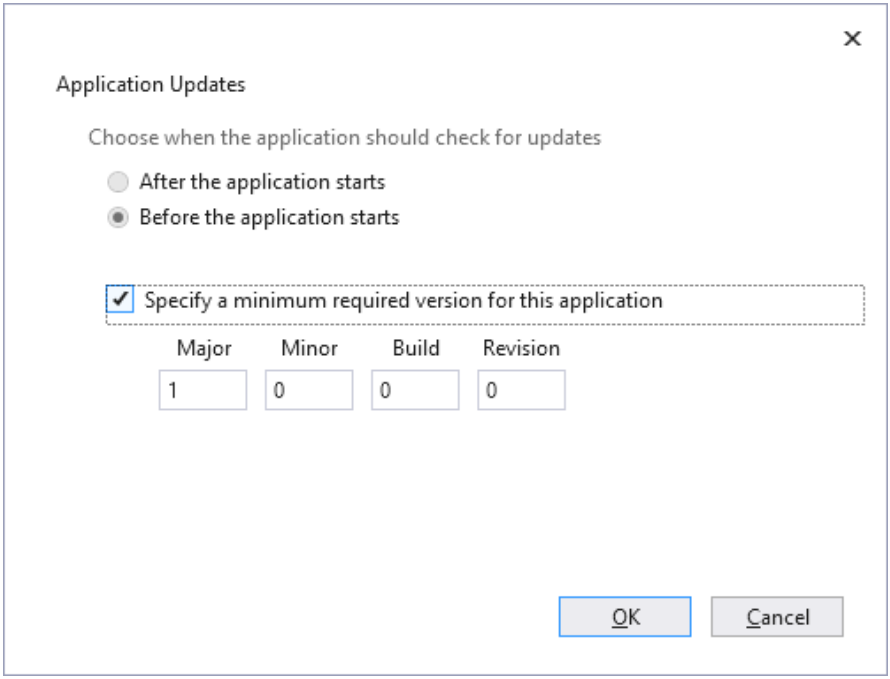
“Install location” bölümünde kullanıcıların uygulamayı hangi ortamdan indirip kuracakları belirtilir ve Next butonuna tıklanır.



Settings bölümünde yayınlanan uygulama ile ilgili çeşitli seçenekler mevcuttur.

- **“Automatically check for updates from the following location”** seçeneğiyle uygulama güncellemelerinin hangi web adresinden yapılacağı bilgisi girilir.
- **“Publish version”** seçeneği ile uygulamanın başlangıç sürüm bilgisi girilir. Eğer **“Automatically increment revision”** seçeneği işaretlenirse uygulamanın yeni sürümünü publish ettikçe sürüm numarası otomatik olarak arttırılacaktır.

Güncelleme ile ilgili daha fazla ayar için **“Settings”** bölümündeki **“Update Settings”** seçeneği tıklanır.



Birinci ayar olan **"After the application starts"** seçeneği ile güncelleme kontrolünün uygulama çalıştırıldıktan sonra yapılması sağlanır.

İkinci ayar olan **"Before the application starts"** seçeneği ile güncelleme kontrolünün, uygulamanın çalıştırılması esnasında yapılması sağlanır.

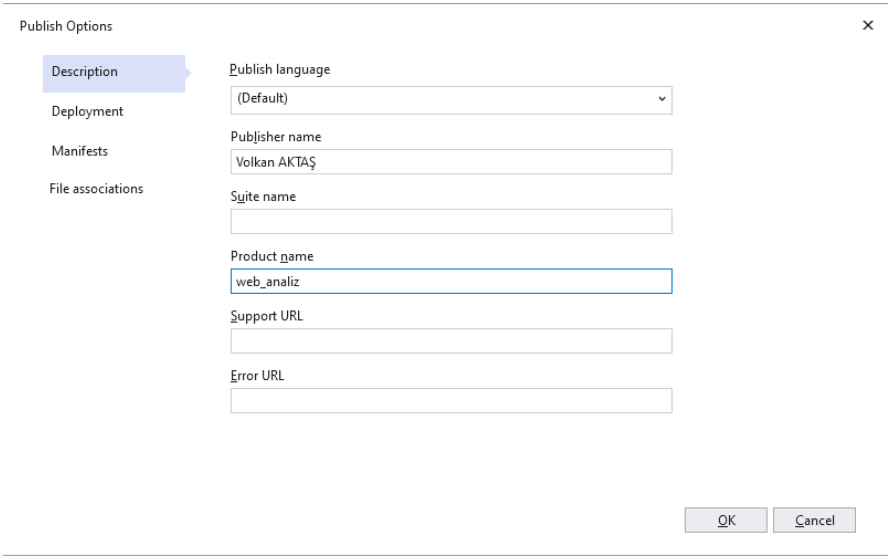
"Specify a minimum required version for this application" seçeneği işaretli iken uygulamanın çalışması için gereken en düşük sürüm numarasını belirleyebiliriz.

"Settings" bölümündeki **"Application Files"** seçeneğine tıklanarak uygulamamıza ait dosyaları ve bu dosyaların kurulum içerisine dâhil edilip edilmeyeceğini belirleriz.

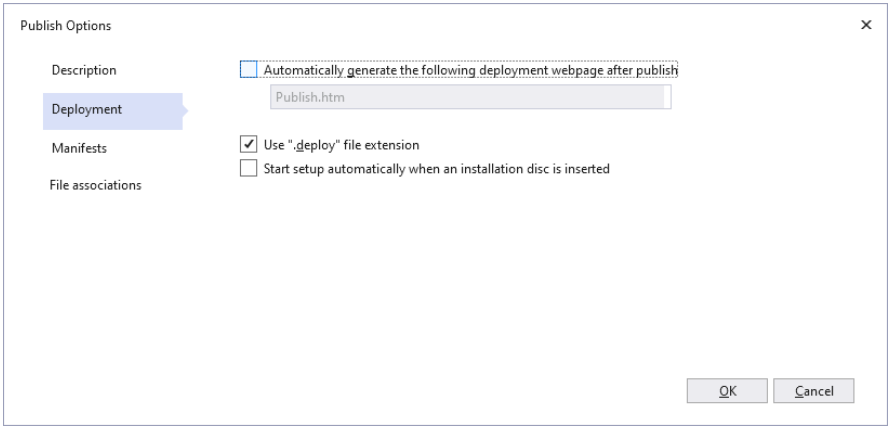
"Settings" bölümündeki **"Prerequisites"** seçeneğine tıklanarak uygulamamıza ait dosyaları ve bu dosyaların kurulum içerisine dâhil edilip edilmeyeceğini belirleriz.

“Settings” bölümündeki **“Options”** seçeneğine tıklanarak yayınlanacak uygulama ile ilgili çeşitli ayarların bulunduğu pencereye ulaşılır. Bu pencerede aşağıdaki gibi seçenekler yer almaktadır:

“Description” sekmesinde yayınlanacak uygulama ile ilgili açıklama, dil ve ürün bilgileri girilir.



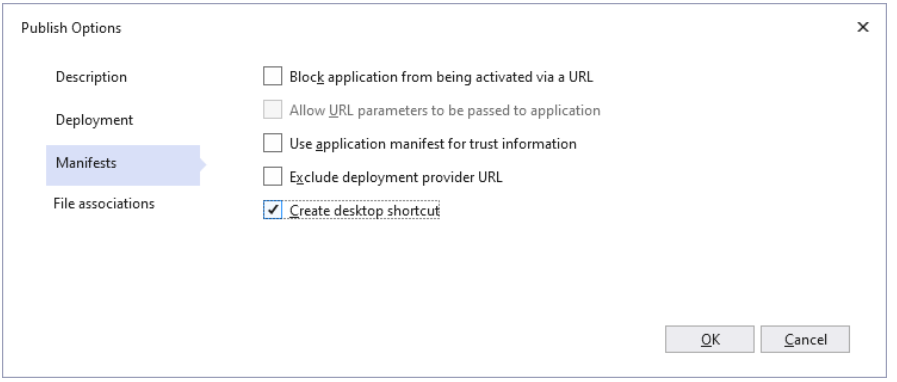
“Deployment” sekmesinde uygulama yayınlandıktan sonra gerçekleştirilecek olan işlemlere dair ayarlar yer alır.



- **"Automatically generate the following deployment webpage after publish"** seçeneği uygulama yayınlandıktan sonra yayınlanan uygulamaya dair bilgilerin yer aldığı bir web sayfasının görüntülenmesini sağlar.
- **"Use ".deploy" file extension"** seçeneği uygulama yayınlandığında dağıtımın ortamındaki tüm dosyalara ".deploy" dosyalarının eklenmesini sağlar. Bu seçenek .exe ve .dll gibi dosya türleri yüklemelerinin engellendiği web ortamlarında ClickOnce kurulumunun başarıyla tamamlanmasını sağlar.
- **"Start setup automatically when an installation disc is inserted"** seçeneği kurulumun yer aldığı medya ortamı bilgisayara takıldığında kurulumun otomatik olarak başlamasını sağlar.

"Manifests" sekmesinde ise aşağıdaki seçenekler yer alır:

- **"Block application from being activated via a URL":** Uygulamanın bir URL araması ile değil de Başlat menüsünden çalıştırılmasını sağlar. Ayrıca bu seçeneğin pasif yapılması uygulamanın kurulum bittikten sonra otomatik olarak çalıştırılmasını sağlar.
- **"Allow URL parameters to be passed to application":** Uygulamanın URL parametrelerini kabul etmesine izin verir.
- **"User application manifest for trust information":** Dağıtımın varsayılan Microsoft sertifika bilgilerini mi yoksa kendi sertifikanızı mı kullanacağını belirler.
- **"Exclude deployment provider URL":** Uygulamanın ilk sürümü yayınlandığında güncelleme konumu belirsizse kullanılabilir. Bu durumda uygulama, başlangıçta yüklendiği yerden güncellemeleri kontrol edecektir.
- **"Create desktop shortcut":** Uygulama için masaüstünde kısayol oluşturulup oluşturulmayacağını belirler.

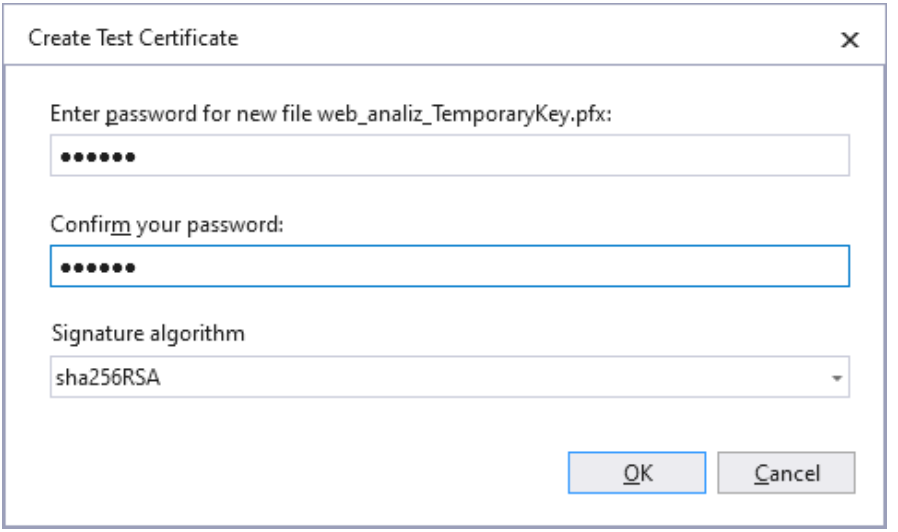


- **“File associations”** bölümünde uygulamanıza özel dosya türleri tanımlayabilirsiniz.

  Tüm bu ayarların yapılmasının ardından Next butonuna tıklanarak **“Sign manifests”** sekmesi görüntülenir. Bu sekme, projemiz ile ilgili güvenlik sertifikası ya da güvenlik anahtarı oluşturmak amacıyla kullanılır. Bir uygulamayı ClickOnce dağıtımını kullanarak yayınlamak istiyorsanız, uygulamalarınızın özel bir anahtar dosyası ile imzalanması gerekir. **Windows sertifika dosyalarını** veya **bir güvenlik anahtarı dosyasını** kullanarak uygulamalarınızı imzalayabilirsiniz.

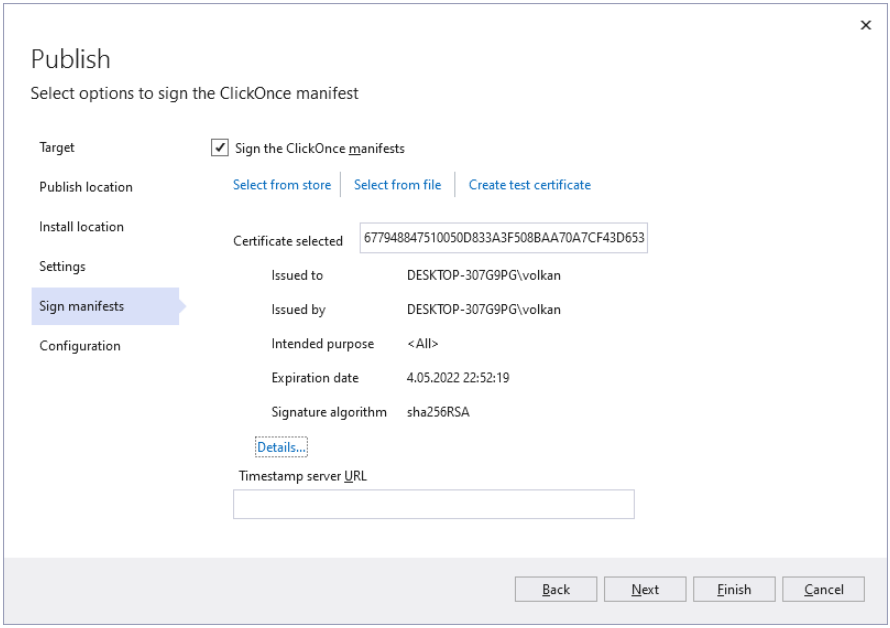
  Bu bölümdeki **“Sign the ClickOnce manifests”** kısmında oluşturmuş olduğunuz uygulamalarınızı 3 farklı şekilde imzalayabilirsiniz.

- **Select from Store…** seçeneği mevcut sertifikalardan birinin belli bir süre sınırı ile kullanılabilmesini sağlar.
- **Select from File…** seçeneği önceden mevcut olan .pfx uzantılı sertifika dosyalarının kullanılabilmesini sağlar.
- **Create Test Certificate…** test amaçlı bir sertifika dosyasının oluşturulmasını sağlar. Bu seçeneği tıklayıp gelen pencereden oluşturulacak olan sertifika dosyası için gerekli şifreleri girelim.

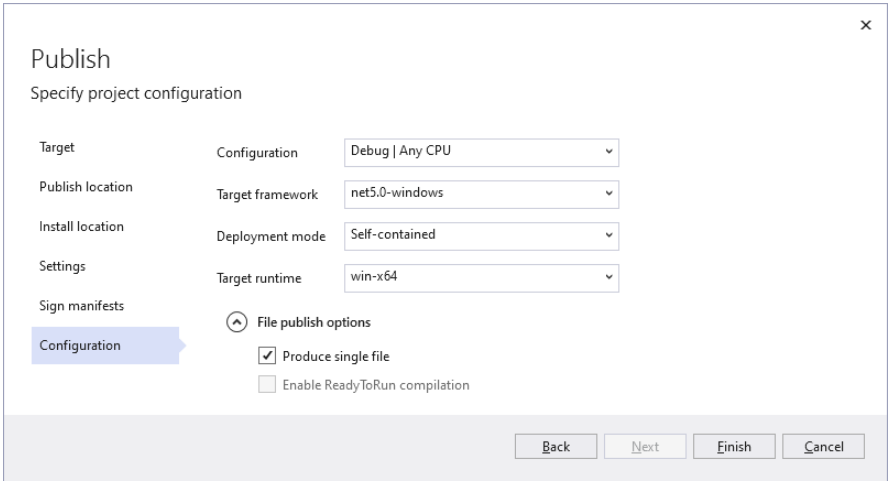


OK butonuna tıkladığımızda ilgili sertifika dosyası **“projeadı_TemporaryKey.**

pfx" ismiyle proje klasöründe oluşturulacaktır. Bu şekilde uygulamalarınız için benzersiz bir kimlik numarası oluşturarak uygulamalarınızın güvenli olarak kabul edilmesini sağlarsınız.



Next butonuna tıklayarak son sekmemiz olan "**Configuration**" sekmesini görüntülüyoruz.

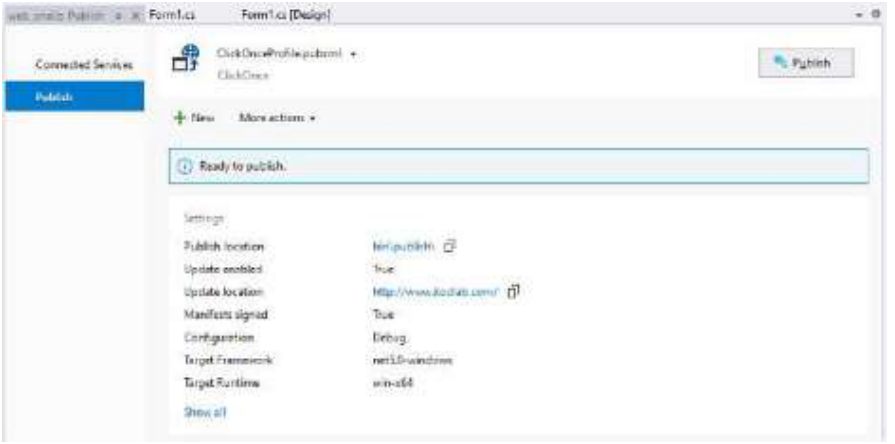

Publish
Specify project configuration
Target
Publish location
Install location
Settings
Sign manifests
Configuration
Configuration
Debug | Any CPU
Target framework
net5.0-windows
Deployment mode
Self-contained
Target runtime
win-x64
File publish options
Produce single file
Enable ReadyToRun compilation
Back
Next
Finish
Cancel

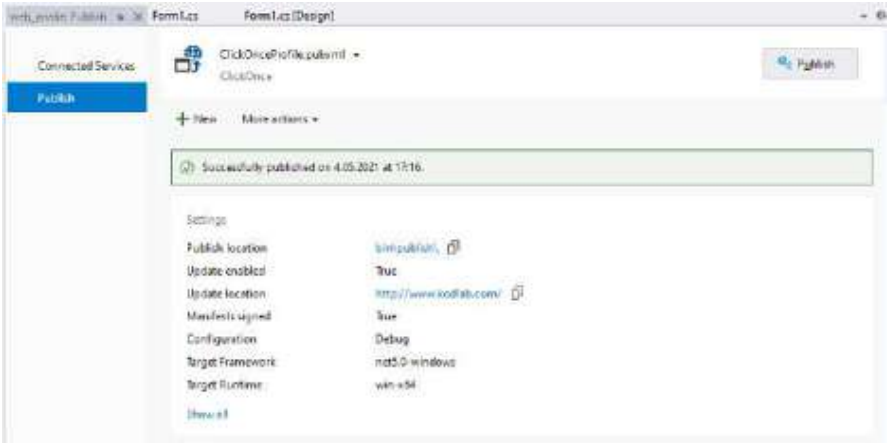
Deployment Mode bölümünden uygulamamızı .exe uzantılı hâle getireceğimiz için "**Self-contained**" seçeneğini seçiyoruz.

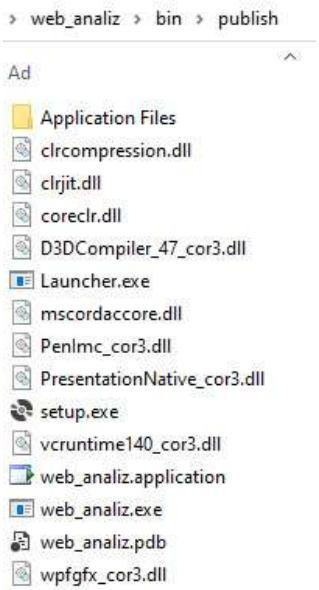
Target Runtime bölümünden ise uygulamamızı .exe uzantılı hâle getireceğimiz için "**win-x64**" seçeneğini seçmeliyiz.

Uygulamamızı tek bir .exe dosyası hâlinde yayınlamak için "**Produce single file**" seçeneğini işaretlemeliyiz.

Bu işlemlerin ardından Finish butonuna tıklayarak gelen pencerede **Publish** butonuna tıklayıp uygulamamızı yayınlıyoruz.





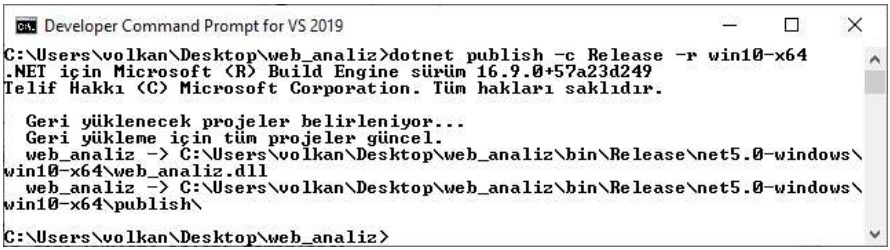


Uygulamamız proje klasörü altındaki **bin**/**publish** klasöründe yayınlanacaktır. Klasör içerisinde kurulum esnasında gerekli olan tüm dosyalar bu klasöre kopyalanır. "**setup.exe**" dosyasına tıklanarak kurulum gerçekleştirilir.

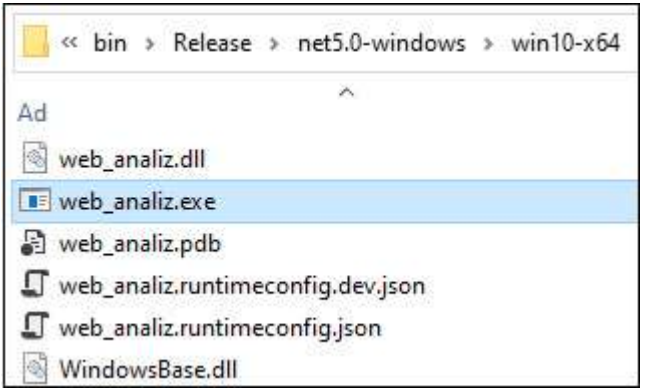
Komut satırını kullanarak uygulamamızı Windows işletim sistemi için .exe uzantılı hâle getirmek istersek komut satırından **publish** komutunu kullanmalıyız. publish komutunu

```
dotnet publish -c Release -r win10-x64
```

şeklinde yazıyoruz.



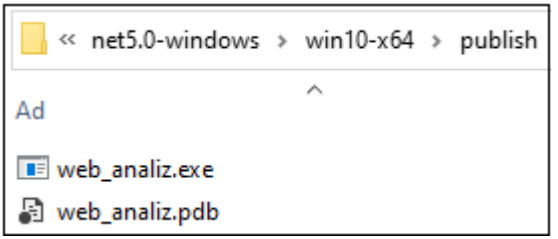
Uygulamamızın .exe uzantısı **win10-x64\publish** klasöründe oluşturulacaktır.



Uygulamamızı tek bir exe dosyası paketinden oluşacak şekilde yayınlayabiliriz. Bunun için komut satırından aşağıdaki ifadeyi yazıyoruz:

```
dotnet publish -r win10-x64 --self-contained=false
/p:PublishSingleFile=true
```

win10-x64\publish klasörüne baktığımızda klasörde yalnızca **projeadı.exe** ve **projeadı.pdb** dosyalarının olduğunu görürüz.



Bu şekilde projeadı.dll, projeadı.dev.json, projeadı.runtimeconfig.json ve projeadı.runtimeconfig.dev.json dosyaları projeadı.exe dosyasının içerisine gömülecektir.

Uygulama Geliştirme

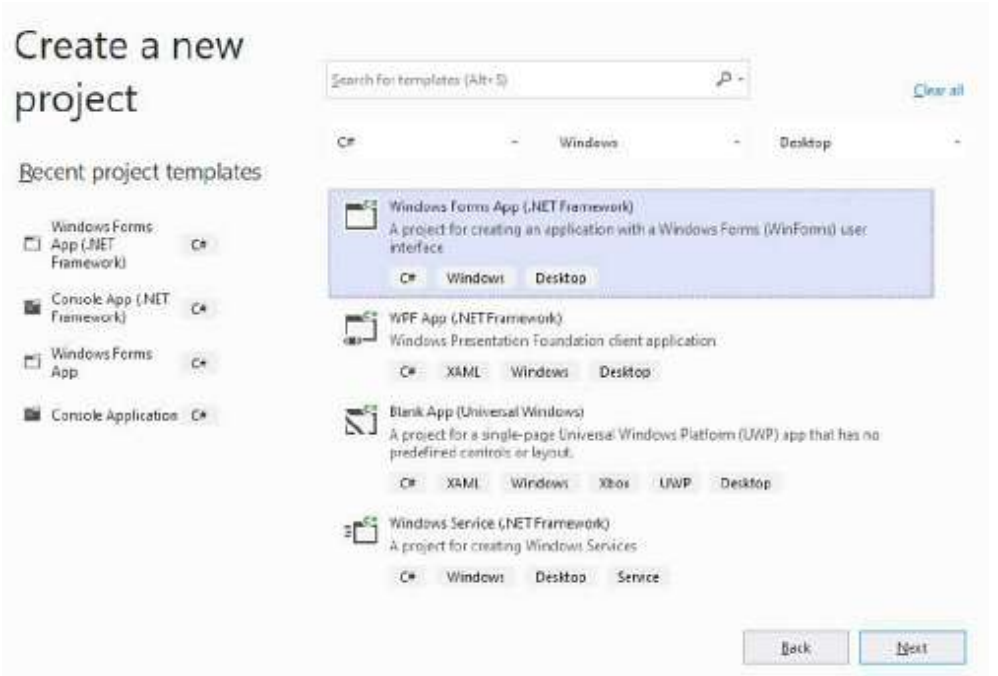
Visual Studio ile ilgili temel bileşenleri ve projeler üzerinde gerçekleştirilebilecek temel işlemleri bu konuya kadar işledik. Bu konuda ise bir proje uygulamasının dizayn görünümü ve kod görünümünü inceleyerek temel bir proje uygulaması oluşturacağız.

Öncelikle yeni bir proje açalım. Bunun için;

Start Page ekranından **Get Started** başlığı altındaki **Create a new project** seçeneği, **File** menüsünden **New Project** seçeneği, **Ctrl+Shift+N** tuşları ya da **Araç** çubuğundan  simgesi tıklanabilir.

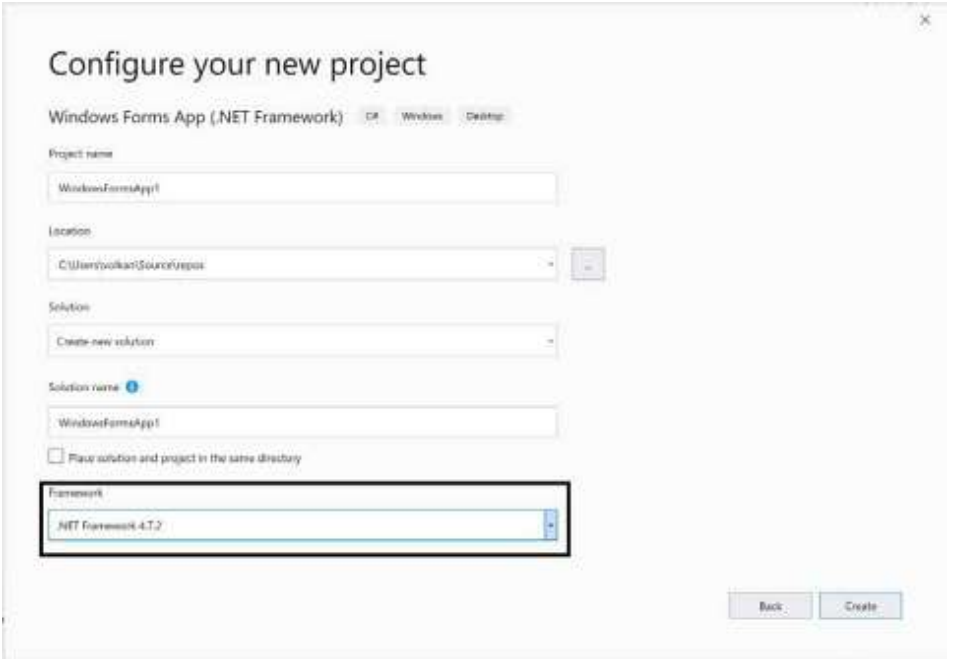


Karşımıza gelen **Create a new project** penceresinden mevcut olan proje şablonlarından birisi seçilerek yeni bir proje oluşturulabilir. NET 5.0 ile desktop ve console projelerini **.net framework** ve **.net core** uygulaması olmak üzere 2 farklı biçimde oluşturabilirsiniz. Yeni bir .NET Framework projesi açmak için Windows Forms App (.NET Framework) ya da Console App (.NET Framework) proje şablonu, .Net Core uygulaması içinse Windows Forms App ya da Console Application seçeneklerini seçebilirsiniz.

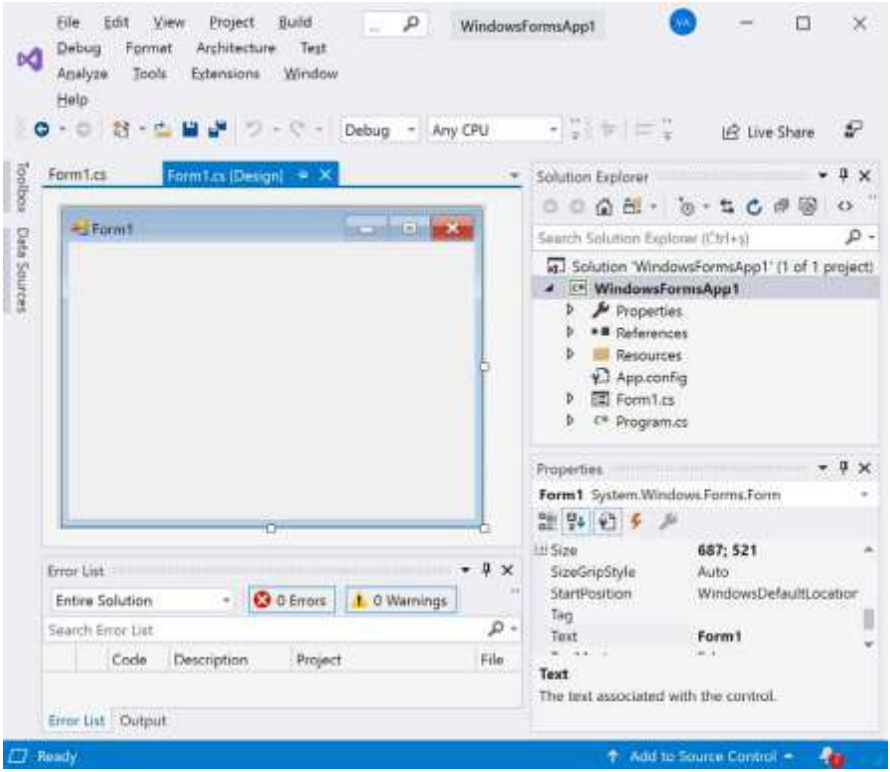


Konsol uygulamalarına daha sonra ayrıntılı olarak değineceğimizden dolayı, örnek uygulamamız için **Windows Forms App (.NET Framework)** şablonunu seçiyoruz.

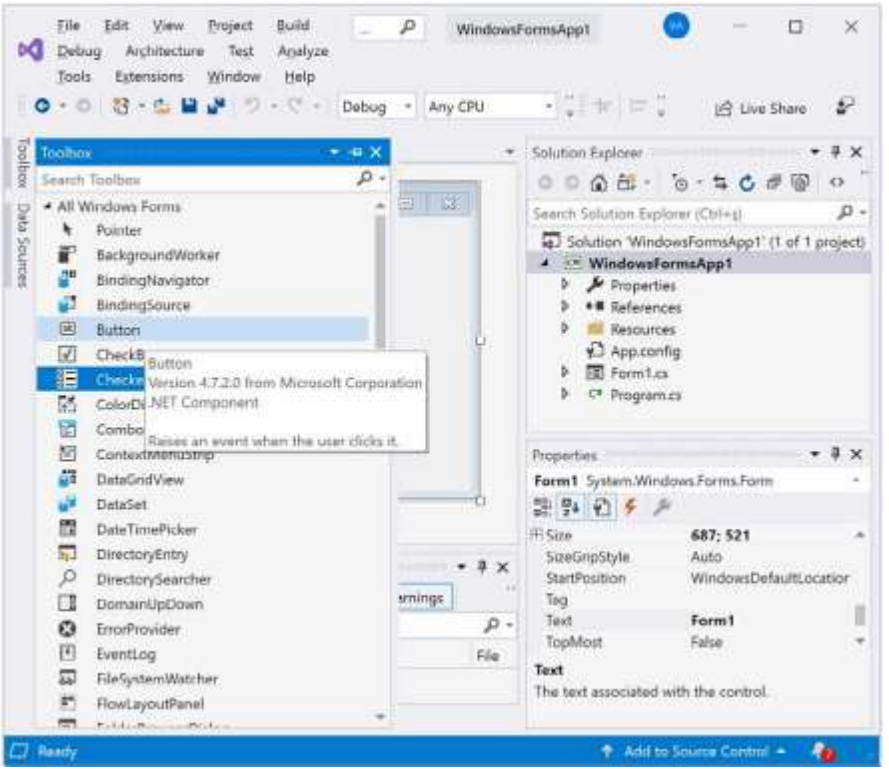
Bu pencerede Next butonuna tıklayarak gelen pencereden framework versiyonunu ayarlıyoruz.



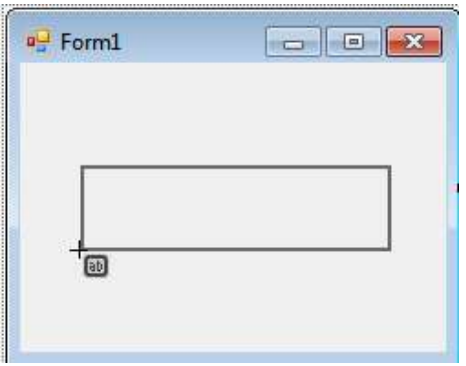
Bu işlemin ardından **Create** butonuna tıkladığımızda varsayılan olarak bir form eklenmiş olan uygulamamız açılır.



Form üzerine ekleyebileceğimiz kontroller pencerenin sol tarafında **ToolBox** içerisinde bulunmaktadır. Otomatik olarak gizlenen **Toolbox**'ı görüntülemek için fare ile üzerine ekranın sol üst kenarında yer alan **ToolBox** yazısına tıklamak yeterlidir. Eğer **Toolbox** görünmüyorsa **View/Toolbox** menüsü ya da **Ctrl+Alt+X** kısayol tuşları ile görüntülenebilir. Herhangi bir kontrolün üzerinde fare ile beklendiğinde o kontrolün versiyonu ve görevi ile ilgili bir açıklama görüntülenir.

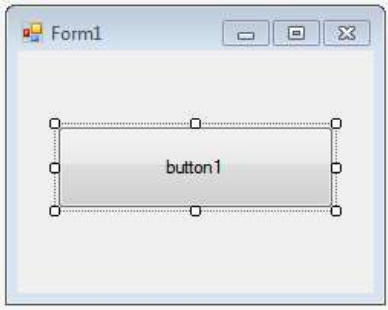


Form üzerine bir kontrolü eklemek için 2 yöntem kullanılabilir;

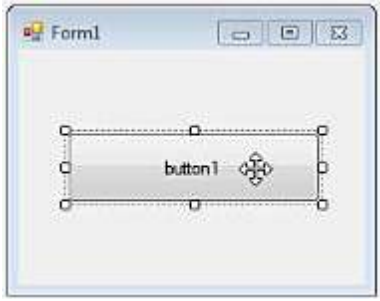


Birinci yöntemde fare ile kontrole çift tıklandığında form üzerine **otomatik** olarak eklenir.

İkinci yöntemde ise kontrol üzerine fare ile bir defa tıklanarak, form üzerine istenilen koordinata, istenilen boyutlarda eklenecek kontrolümüz çizilir.

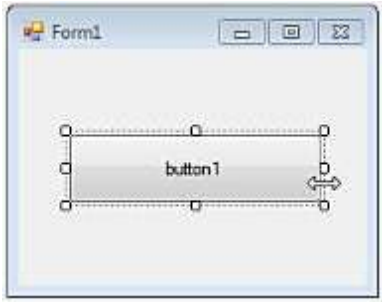


Farenin tuşu bırakıldığında kontrol form üzerine eklenmiş olur.



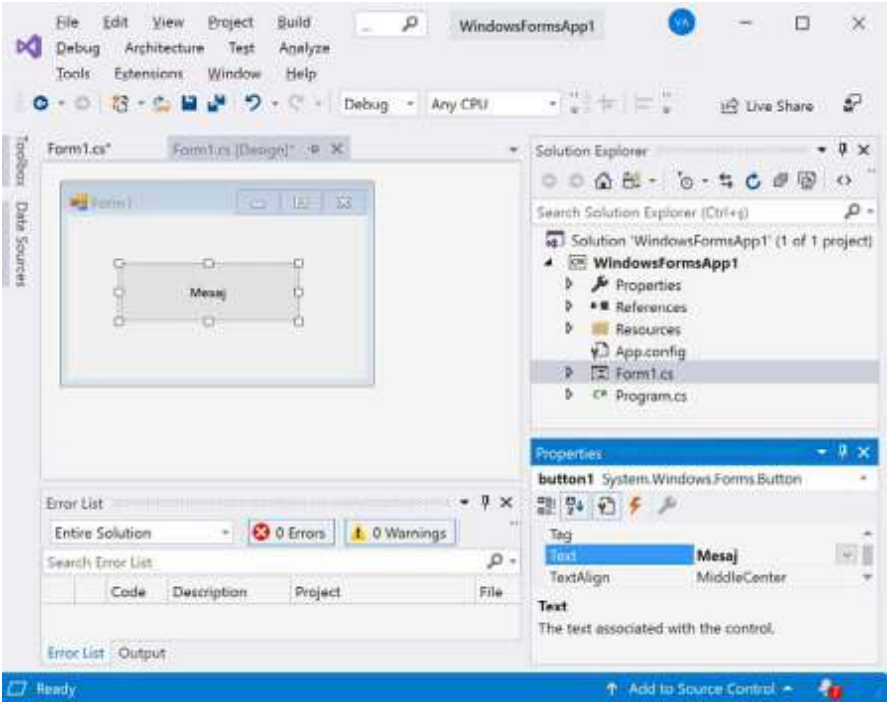
Form üzerindeki butonu istediğimiz bir konuma sürüklemek için fare ile tutup sürükleyebileceğimiz gibi, ok tuşlarını da kullanabiliriz. Daha geniş hareket aralığı için Ctrl tuşu basılı iken yön tuşları kullanılabilir.

Butonun boyutunu belirlemek için ise şu işlemler yapılabilir;



Farenin imleci ile butonun kenarlarına gelindiği anda imleç çift taraflı ok görünümünü alacaktır. Bu anda farenin sol tuşu basılı tutularak bırakılmadan butona istenilen boyut verilebilir. Ya da **Shift** tuşu basılı iken yön tuşları kullanılabilir.

Butonumuz seçili iken **Properties** penceresinden **Text** özelliğini seçerek **Mesaj** olarak değiştirelim.

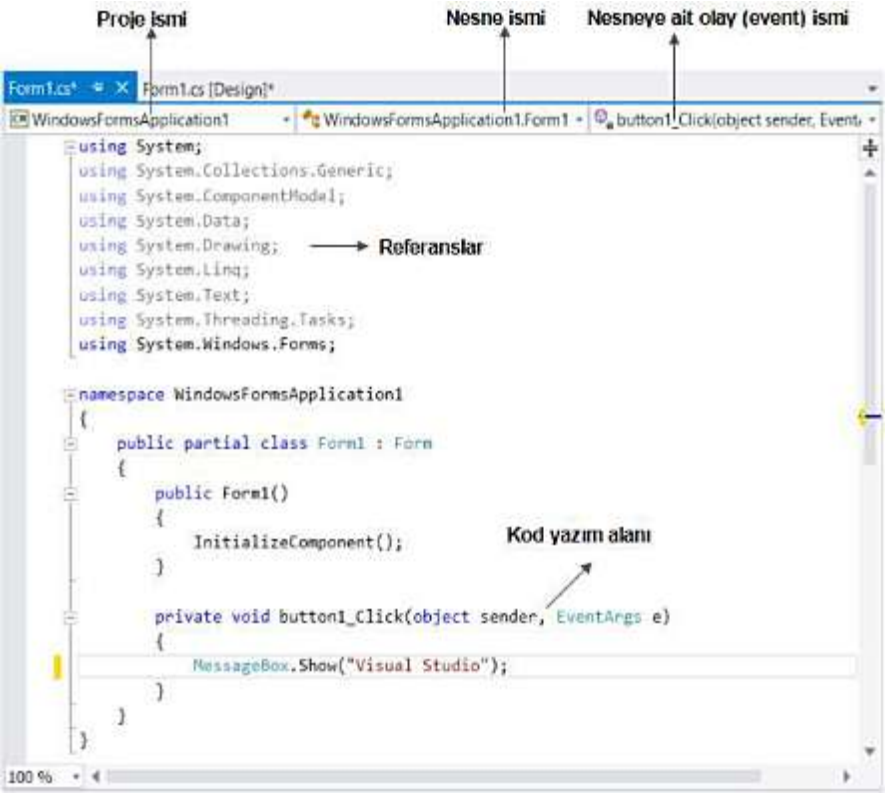


Kod görünümüne geçiş yaparak örnek uygulamamızın kodunu yazalım:

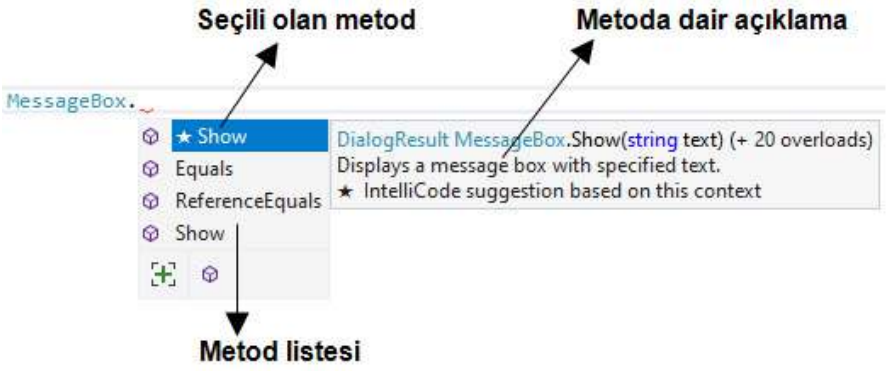
Kod görünümüne geçiş yapmak için **Solution Explorer** penceresindeki <> butonu tıklanabilir ya da **View** menüsünden **Code** seçeneği tıklanabilir.

Kod yazımı esnasında nesneler ve bu nesnelere ait olarak gerçekleşen olaylara dayalı olarak kodlarımızı yazarız. Örneğin; form üzerinde bulunan bir butona tıklandığında hangi işlemler gerçekleştirilecek ya da bir formun yüklenişinde ya da kapatılışında hangi olaylar gerçekleşecek gibi.

Kod yazma penceresinin üst tarafında sol tarafta nesneler, sağ tarafta ise bu nesnelere ait olaylar listelenir. Kullanıcı ilgili olayı seçtiği zaman olayın başlangıç kodu otomatik olarak yazılmaktadır. Elbette kullanıcı isterse bunları elle de yazabilir.

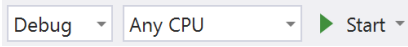


Kod yazarken Visual Studio IDE'si, bize kullanacağımız metodlar ile ilgili intellisense desteğini sağlamaktadır. Kullanabileceğimiz metodlar listelenmekte ve bu metodların kullanım şekilleri hakkında kullanıcı, bilgi pencereleri ile bilgilendirilmektedir.

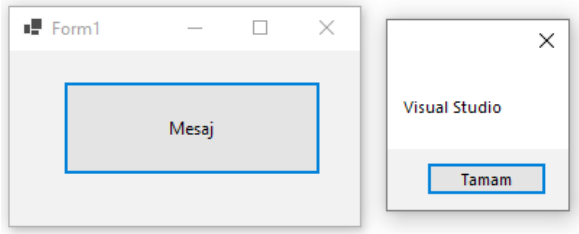


Kod yazımı esnasında kullandığımız metotların parametrelerinin kullanım şekilleri ve bu parametrelerin alabilecekleri değerlere dair açıklamalar hakkında da kullanıcı, bilgi pencereleri ile bilgilendirilmektedir. Örneğimizin kodunu, parametre kullanımını görebilmek için biraz değiştirelim.

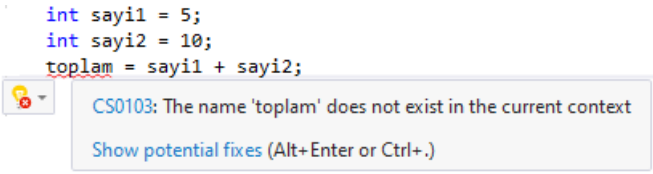


Örnek uygulamamızı çalıştırarak sonucunu görelim. Çalıştırmak için, **Debug** menüsünden **Start Debugging** seçeneği, **F5** tuşu ya da araç çubuğundan 

simgesi tıklanabilir.



Visual Studio editörü kod yazımı esnasında oluşan hataları ya da programın çalışması sırasında oluşabilecek muhtemel hataları düzeltmek amacıyla yol göstermektedir.



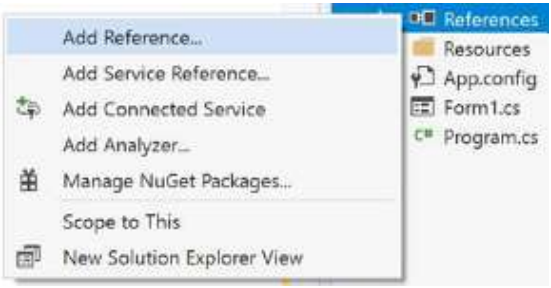
Yukarıdaki örnek kodda **`sayi1`** ve **`sayi2`** değişkenleri tanımlanmış, ancak bu iki sayının toplamının aktarılacağı toplam değişkeni tanımlanmamıştır. Visual Studio editöründe bir **lamba** simgesi ile oluşan hata durumu görüntülenmektedir.

"**Show potential fixes**" seçeneğine ya da **lamba** simgesinin alt tarafındaki **ok** işaretine tıklanarak ekrana gelen muhtemel hata düzeltmelerinden biri seçilebilir.

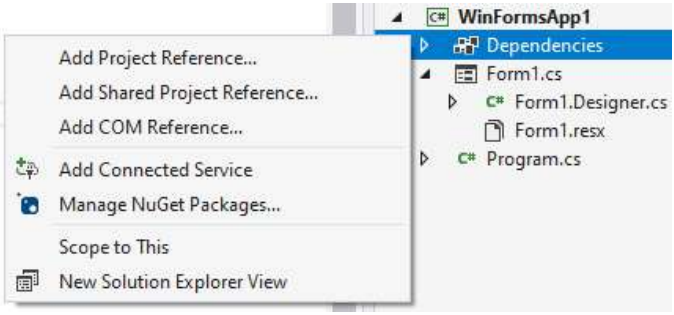


Yazmış olduğumuz uygulamaların kod satırlarını incelediğimizde en üstte **`using`** ifadelerinin yer aldığını görürüz. Projelerimizde kullanacağımız metot ve nesnelere ait sınıfları ***using*** ifadesi ile projemize dâhil ederiz. Örneğin konsol uygulamaları için standart olarak **`System`** ve **`System.Text`** sınıflarını projemize dâhil etmemiz gerekirken Windows Form uygulamaları için ise standart olarak **`System`** ve **`System.Windows.Forms`** sınıflarını projemize dâhil etmemiz gerekmektedir. Proje dâhilinde kullanılan referanslar **koyu renkli** olarak görüntülenmektedir. Ekleyeceğimiz sınıflar projelerimizde kullanacağımız metotlara ve nesnelere göre değişiklik gösterecektir.

Standart olarak eklenen sınıflar haricinde bir `dll` sınıfını proje içerisine referans olarak eklemek için öncelikle **Project** / **Add Reference** menü seçenekleri tıklanır ya da **Solution Explorer** içerisinde proje ismi farenin sağ tuşu ile tıklanarak gelen menüden **Add** / **Reference** seçeneği tıklanır.

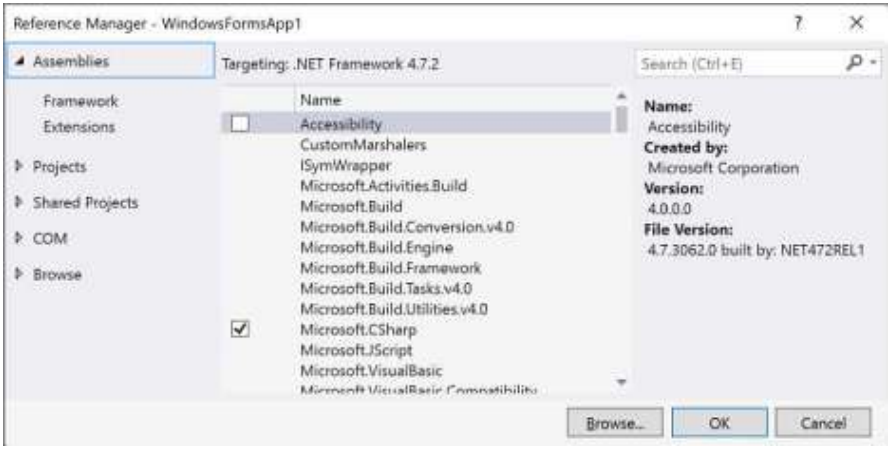


Solution Explorer içerisinde **References** başlığına sağ tıklayarak gelen menüden **Add Reference** seçeneğini kullanarak da referans ekleme penceresine ulaşabiliriz.

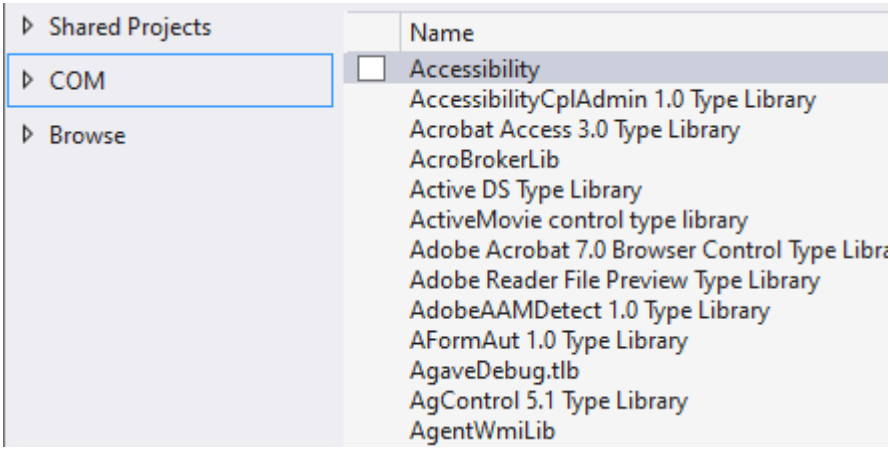


.NET Core projelerinde ise References yerine **Dependecies** kısmı yer almaktadır.

Gelen pencereden **referans** dosyası seçilerek eklenebilir.



.NET Core projelerinde aşağıda görüldüğü gibi **Assemblies sekmesi yer almaz**. Çünkü projeniz için gerekli olan tüm kütüphaneler seçmiş olduğunuz Framework sürümüne göre projenize otomatik olarak eklenecektir.



Burada yer alan başlıklardan;

- **Assemblies** başlığı altındaki,

  Framework kısmında, Visual Studio ile birlikte yüklenen sınıflar mevcuttur.

  Extensions kısmında, Visual Studio içerisinde yüklü olan eklentiler mevcuttur.

- **Projects** başlığı altında, mevcut projelere ait referans dosyalarını ekleyebilirsiniz.
- **Shared Projects** başlığı altında, Windows Phone ve Windows Store projeleri ile paylaşılan referans dosyalarını ekleyebilirsiniz.
- **COM** başlığı altında, bilgisayarda yüklü olan diğer programlara ait sınıflar mevcuttur.
- **Browse** başlığı ile belirtilen bir klasör içerisinde bulunan bileşen dosyalarından biri seçilerek o sınıf proje içerisine eklenebilir.

  Browse başlığı altındaki **Recent** kısmında da daha önceden kullanılmış olan sınıflardan biri seçilerek proje içerisine eklenebilir.

  Dll sınıflarına herhangi bir programlama dili içerisinden erişmek mümkündür. Yani dll dosyasının hangi dille yazıldığı önemli olmaksızın herhangi bir dille yazacağınız proje içerisinde `dll` sınıflarını kullanabilirsiniz.

  Kodlarımız varsayılan olarak proje ismi ile aynı olan bir **namespace** içerisinde yer almaktadır. Kullanıcı isterse kodlama esnasında bu namespace ismini değiştirebilir. Namespace, kullanacağımız metot ya da sınıfları daha iyi bir şekilde organize edebilmek amacıyla, **.NET** içerisinde oluşturulmuş olan bir yapıdır. Bir proje içerisinde yer alan birden fazla namespace'e ya da farklı sınıflarda yer alan namespace'lere `using` bildirimini kullanarak erişebiliriz.

  Örneğin `System.Collections.Generic` sınıfına ait bir sınıf olan List sınıfını aşağıdaki gibi tanımlayalım:

```
List<string> listemiz = new List<string>();
```

  Bu şekilde bir tanımlamayı 2 şekilde yapabiliriz. Birincisi,

```
System.Collections.Generic.List<string> listemiz =
            new System.Collections.Generic.List<string>();
```

  Şeklinde `List` metodunun ait olduğu `System.Collections.Generic` namespace'ini yazarak kullanmak ya da ikinci şekliyle;

```
using System.Collections.Generic;
namespace using_bildirimi
{
```

```
  class Program
  {
    static void Main(string[] args)
    {
      List<string> listemiz = new List<string>();
    }
}
```

using bildirimini kullanarak **System.Collections.Generic** namespace'ini proje kodlarımızın en üstünde tanımlamaktır.

Ayrıca **using** ifadesini kullanarak kodlarımızda kullandığımız nesnelerin, kullanımı bittikten sonra otomatik olarak hafızadan temizlenmesini sağlayabiliriz. Referans türünden değişkenlerin kullanımı bittikten sonra hafızadan temizlenmesi gerekmektedir. Bu işlem **.NET** içerisinde **Garbage Collector** isimli bir yapı tarafından gerçekleştirilir. Bu işlem .NET tarafından otomatik gerçekleştirilse de işlemin gerçekleştirilme zamanındaki belirsizlikten dolayı bu mekanizmayı **manuel** olarak başlatmak gerekebilir. Bunun için **IDisposable** arayüzünün **Dispose** metodu kullanılmaktadır. Örneğin aşağıdaki kod satırlarını inceleyelim.

```
Graphics formGraphics = this.CreateGraphics();
formGraphics.DrawString("DENEME", new Font("Arial", 20.0f),
new SolidBrush(Color.Black), 50, 30);
formGraphics.Dispose();
```

Kodumuzda form üzerine belirtilen yazı tipi özellikleri ile bir yazı çizdiriyoruz. **Dispose** metodunu kullanarak da form üzerine çizim yaparken kullandığımız hafıza kaynaklarının boşaltılmasını sağlıyoruz.

Aynı kodu **using** ifadesini kullanarak yeniden yazalım.

```
using (Graphics formGraphics = this.CreateGraphics())
{
  formGraphics.DrawString("DENEME", new Font("Arial", 20.0f),
    new SolidBrush(Color.Black), 50, 30);
}
```

Kodumuzda form üzerine çizim yapılırken **`using`** ifadesinin kullanılmasından dolayı, çizim işleminin bitmesinin ardından **Dispose** işlemi otomatik olarak gerçekleştirilmektedir.

C# 8 versiyonundan itibaren **`using`** ifadesini:

```
using (Graphics formGraphics = this.CreateGraphics());
formGraphics.DrawString("DENEME", new Font("Arial", 20.0f),
new SolidBrush(Color.Black), 50, 30);
```

Şeklinde kullanabiliriz. Yukarıdaki kod satırlarında **Dispose** işlemi **`formGraphics`** değişkeninin kullanımının bittiği son satırın hemen ardından gerçekleştirilecektir.

Örneğimizi inceleyecek olursak **class Program** biçiminde **Program** isimli class'ın (*sınıfın*) oluşturulduğunu görürüz. **C#** programlarımızın en azından **bir** sınıfı içermesi gereklidir.

Yazacağımız metotları ve kod bloklarını **`{ }`** işaretleri arasında yazarız.

`static void Main` ifadesi ile **`Main`** bir metot tanımladık. **`Main`** metot olmadan programlarımız çalışmaz. **`Void`** ifadesi genel olarak tek başına kullanıldığında değer göndermeyen metotları belirtir. **`Static`** ise bu metodun erişim şeklini belirtir.

Kod yazılırken derleyici kod satırının bitip bitmediğini **`;`** (*noktalı virgül*) işaretine bakarak belirler. Eğer noktalı virgül işareti konulmuşsa bu yeni bir kod satırı başlangıcı demektir.

```
MessageBox.Show("Güzel gören güzel düşünür");
```

ile

```
MessageBox.Show("Güzel gören güzel düşünür");
```

aynı sonucu verecektir.

NOT

Kitap içerisinde bir satıra sığmayıp da bölünmesi durumunda hata verecek olan satırları belirtmek amacı ile satır sonlarına ↓ işareti eklenecektir.

Örneğin;

```
openFileDialog1.Filter = "jpeg dosyaları (*.jpeg) ↓
        |*.jpg|bmp dosyaları (*.bmp)|*.bmp|gif ↓
        dosyaları(*.gif)|*.gif";
```

`openFileDialog` nesnesinin Filter özelliğine, diyalog penceresi içerisinde gösterilecek olan dosya türlerini atayan kod satırı bir satıra sığmamaktadır. Ancak bu satırlar bölündüğünde hata vermektedir. Bu satırların yukarıdaki satırların devamı olduğunu belirtmek amacı ile ↓ işaretini bilgilendirmek için kullanıyoruz.

Program içerisinde açıklama satırlarını satırın başına // işaretlerini ekleyerek yazabiliriz. Eğer birden fazla satırdan oluşan açıklama satırı yazacaksak her satırın başına // işaretini eklememiz gerekmektedir.

```
int i = 0;
private void button1_Click(object sender, EventArgs e)
{
    i = i + 1;
    // Butona tıklanma sayısını bulur
   // ve bir mesaj penceresi içerisinde görüntüler.
    MessageBox.Show(i + " defa tıklandı");
}
```

Birden fazla satırdan oluşacak açıklama satırlarını /* ve */ karakterleri arasına yazabiliriz.

```
int i = 0;
private void button1_Click(object sender, EventArgs e)
{
    i = i + 1;
    /* Butona tıklanma sayısını bulur
        ve bir mesaj penceresi içerisinde görüntüler. */
    MessageBox.Show(i + " defa tıklandı");
}
```

KONSOL EKRANI

3

C# içerisinde görsel ortamda Windows form uygulamaları vasıtası ile kod yazabileceğimiz gibi görsel özellikleri olmayan uygulamaları da konsol ekranında yazabiliriz.

Konsol ekranında kullanılan fonksiyonlar **`System.Console`** sınıfına ait olan fonksiyonlardır.

Konsol ekranında kod satırları...

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            //Kod satırları
        }
    }
}
```

...blokları arasına yazılır.

Bir C# programı **`Main`** metodu çağırarak başlar. **`Main`** metodu **`void`** şeklinde tanımlandığı için geriye herhangi bir değer döndürmeyecektir.



.NET Framework 5.0 ve C# 9 sürümünden itibaren .NET Core konsol uygulamaları için temel kodlarımızı names-pace, class ve main metot yazmaya gerek kalmadan doğrudan yazabiliyoruz.

Örneğin aşağıdaki kod satırlarını

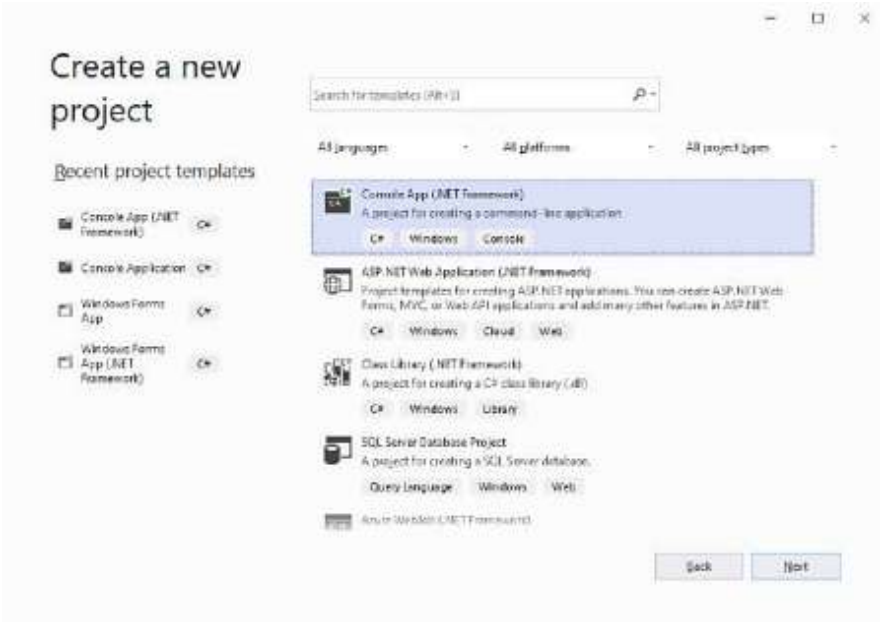
```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("C Sharp");
        }
    }
}
```

.NET Framework 5.0 ve C# 9 sürümünden itibaren

```
using System;
Console.WriteLine("C Sharp");
```

Şeklinde yazabilirsiniz.

.NET Core konsol uygulamalarınızı yeni proje açma penceresinden "**Console Application**" olarak seçip açabilirsiniz. Standart konsol uygulamalarını ise "**Console App (.NET Framework)**" şeklinde seçiyoruz.



Konsol Ekranında Değer Yazma ve Okuma

Konsol ekranında yazdırma işlemleri için **`Write`** ve **`WriteLine`** metotları kullanılırken, okuma işlemleri için ise **`Read`** ve **`ReadLine`** metotları kullanılır.

Write ve WriteLine Metotları

`Write` metodu, kendisine gönderilen değeri konsol ekranında aynı satıra yazdırmak için kullanılan bir metoddur.

`WriteLine` metodu ise, kendisine gönderilen değeri konsol ekranında satır sonu karakterini de ekleyerek bir alt satırdan itibaren yazdırmak için kullanılan bir metottur.

```
Console.Write("Volkan ");
Console.Write("Aktaş");
```

Şeklinde yazdığımızda…

```
Volkan Aktaş
```

… çıktısını verecek ve imleç cümlenin sonunda bekleyecektir.

```
Console.WriteLine("Volkan ");
Console.Write("Aktaş");
```

Şeklinde yazdığımızda…

```
Volkan
Aktaş
```

… çıktısını verecek ve imleç ikinci kelimenin sonunda bekleyecektir.

```
Console.WriteLine("Volkan ");
Console.WriteLine("Aktaş");
```

Şeklinde yazdığımızda ise…

```
Volkan
Aktaş
```

… çıktısını verecek ve imleç "*Aktaş*" kelimesinin bir alt satırında bekleyecektir.

C# dili içerisinde \ karakterinin ardından kullanılarak belli bir eylem yerine getiren karakterlere **Escape** (*kaçış*) karakterleri ismi verilir. Başlıca kaçış karakterleri şunlardır:

Karakter	Açıklama
\a	Bip sesi çıkartır
\b	Backspace
\f	Sonraki sayfa
\n	Alt satıra iner
\r	Satır başı
\t	Yatay Tab
\v	Dikey Tab
\'	Tek tırnak
\"	Çift tırnak
****	\ (Ters bölü işareti)
\?	? (Soru işareti)

Örneğin;

```
Console.Write("Kodlab" + "\t" + "Yayınevi");
```

Satırı **\t** karakteri kullanıldığından dolayı, "*Kodlab*" ve "*Yayınevi*" kelimeleri arasında bir Tab miktarı kadar boşluk bırakarak…

Kodlab Yayınevi

… şeklinde çıktı verecektir.

```
Console.Write("İlk satır \n");
Console.Write("İkinci satır ");
```

Kod satırlarında **\n** karakterinin kullanılmasından dolayı, ilk satırın yazımının ardından bir alt satıra inip, ikinci satırı yazarak şu çıktıyı verecektir.

```
İlk satır
İkinci satır
```



\n karakteri yerine alternatif olarak şu yöntemler de kullanılabilir;

(char)13 ve **(char)10** karakterleri kullanılabilir. **(char)** metodu belirtilen ASCII karakter kodunu karaktere dönüştürür. 13 nolu karakter **Enter** tuşu yerine geçerken, 10 nolu karakter ise imleci bir satır aşağı taşır. Bu iki karakter birlikte kullanılarak, yeni bir satır eklenerek imleç bir aşağı satıra geçer. **\n** escape karakteri ile yaptığımız örneği şu şekilde de yazabiliriz.

```
Console.Write("İlk satır" + (char)13 + (char)10);
Console.Write("İkinci satır ");
```

Environment.NewLine ifadesi kullanılabilir. Örneğimizi şu şekilde yazabiliriz.

```
Console.Write("İlk satır" +Environment.NewLine);
Console.Write("İkinci satır ");
```

Eğer birden fazla değişkenin içeriğini istediğimiz sıralama ile yazdırmak istersek, bu değişkenleri aralarına virgül işareti koyarak kullanabiliriz. Bu değişkenlerin içeriklerini ise, **{0} {1}** şeklinde yazdırabiliriz.

```
Console.WriteLine("{0}, {1}, {2}", 10, 20, 30);
```

Satırlarının ekran çıktısı **10**, **20**, **30** şeklinde olacaktır. Örneğimizdeki **{0} {1} {2}** değişkenlerin sıra numarasını belirtmektedir. Kodumuzu…

```
Console.WriteLine("{1}, {0}, {2}", 10, 20, 30);
```

… şeklinde düzenleyecek olursak; ekran çıktısı **20**, **10**, **30** şeklinde olacaktır.

Read ve ReadLine Metotları

ReadLine metodu; konsol ekranından değer okumak için kullanılan bir metottur. Geriye **string** değişken türünde değer döndürdüğünden dolayı, **string** değişken tiplerindeki değerleri okuma amacı ile kullanılır.

Read metodu ise, **ReadLine** metodu gibi konsol ekranından değer okumak için kullanılan bir metottur. Geriye **int** türünden değer döndürdüğünden dolayı, **string** değişken tipleri dışındaki değerleri okuma amacı ile kullanılır. **Read** metodu girilen değerlerden ilk karakterin ASCII karakter kodunu geriye döndürür.

Klavyeden girilen ismi konsol ekranına yazdıran bir uygulama yapalım.

```
using System;
using System.Text;

namespace Console_Read
{
 class Program
 {
  static void Main(string[] args)
  {
    String ad;
    Console.Write("Adınızı Girin: ");
    ad = Console.ReadLine();
    Console.Clear();
    Console.WriteLine(ad);
    Console.ReadKey();
  }
 }
}
```

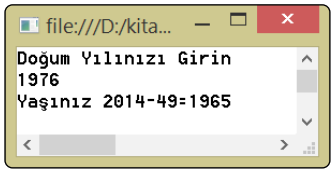
Örneğimizde girilen isim, **Console.ReadLine** metodu ile okunarak, **string** tipte tanımlanmış olan **ad** değişkenine aktarılıp, konsol ekranına yazdırılmaktadır.

Girilen değer metin türünde olduğu için değişkenimizi **`string`** değişken türü ile tanımladık. Kodumuzun sonucunda girilen isim değeri ekrana yazdırılmaktadır.

`Console.Clear()` metodu ise, konsol ekranını temizlemek için kullanılır.

Girilen doğum yılına göre yaşı hesaplayıp ekrana yazdıracak bir uygulama yapalım.

```
using System;
using System.Text;
namespace Console_Read
{
 class Program
 {
  static void Main(string[] args)
  {
    Console.WriteLine("Doğum Yılınızı Girin");
    int yil = Console.Read();
    Console.WriteLine("Yaşınız " + DateTime.Now.Year + "-" + yil +
  "=" + (DateTime.Now.Year-yil));
    Console.ReadKey();
  }
 }
}
```



Kodumuzu çalıştırdığımızda **`Console.WriteLine`** metodunu kullanarak öncelikle "*Doğum Yılınızı Girin*" mesajını görüntülüyoruz. **`Read`** metodu **`int`** geriye **`int`** türden bir değer döndürdüğü için, **`yil`** isminde bir değişkeni **`int`** türünden tanımlayarak değer okutuyoruz. Yaşı hesaplatmak için **`DateTime.Now.Year`** metodu ile o anki yılı bularak girilmiş olan doğum yılını çıkartıyoruz. Ancak ekran çıktısında görüldüğü gibi 1976 değerini girmiş olmamıza rağmen, 49 değerinin okunduğunu görüyoruz. **`Read`** metodu girilen değerlerden ilk karakterin ASCII karakter kodunu döndürdüğü için ilk rakam olan 1 rakamının ASCII karakter kodu olan 49'u geriye döndürdü.

Kodumuzda…

```
int yil = Console.Read();
```

… satırını **`Console.ReadLine`** metodunu kullanarak…

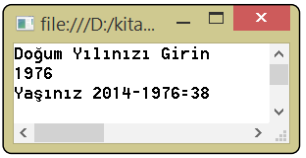
```
int yil = Console.ReadLine();
```

… şeklinde yeniden düzenleyelim. Uygulamamızı çalıştırmaya kalktığımızda, **Cannot implicitly convert type int to string** şeklinde dönüşüm hatası verecektir. Çünkü **`ReadLine`** metodu geriye **`string`** tipte bir değer döndürürken, bu değeri **`int`** türünden **`yil`** değişkenine aktarmaya çalıştık.

Dönüşüm hatasını gidermek için **`string`** türden girilen değeri **`int`** türüne dönüştüreceğiz. Bunun için **`Convert.ToInt32`** metodunu kullanabiliriz. İlgili satırı…

```
int yil = Convert.ToInt32(Console.ReadLine());
```

… şeklinde yeniden düzenleyelim ve uygulamamızı çalıştıralım.



`string` türde girilen doğum yılı değeri **`int`** türüne dönüştürüldüğü için uygulamamız artık doğru şekilde sonuç verecektir.

Kodumuzun son satırında…

```
Console.ReadKey();
```

… satırı dikkatinizi çekmiş olmalı. Konsol uygulamalarında gerekli metotlar kullanılmadığı sürece, uygulamamız çalıştırıldığında konsol ekranında sonuçlar görüntülenip, uygulama penceresi hızlıca kapanmaktadır. Sonuçların kullanıcılar tarafından görünmesi ve uygulama penceresini istenildiği zaman kapatılabilmesinin sağlanması gerekmektedir. Bunun için **`Read`**, **`ReadLine`** ve **`ReadKey`** metotlarını kullanabiliriz.

Read metodu, konsol ekranında girilen karakteri okuyup, herhangi bir tuşa basılmasını bekler. Ardından bir sonraki kodu çalıştırır.

ReadLine metodu, konsol ekranında girilen **string** ifadeyi okuyup, **Enter** tuşuna basılmasını bekler. Ardından bir sonraki kodu çalıştırır.

ReadKey metodu, konsol ekranında basılan herhangi bir tuşu okumak amacıyla kullanılır. Herhangi bir tuşa basılıncaya kadar konsol ekranını bekletir. Ardından bir sonraki kodu çalıştırır.

C# içerisinde **string** değerleri formatlamada kullanılan **yer tutucu** (*placeholder*) kullanarak değişkenleri istediğimiz sıralama ile yazdırabiliriz. Yer tutucu karakteri olarak **{ }** karakterlerini kullanmaktayız. Değişkenin **index** (*sıra*) numarasını **{0}{1}** gibi yazarak, değişken içeriklerini görüntületiriz. Örneğin; klavyeden ayrı ayrı girilen iki adet numarayı, belirtilen sırada ekrana yazdıran bir uygulama yapalım.

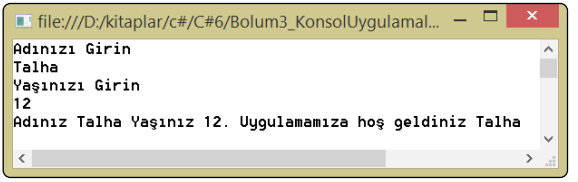
```
using System;
using System.Text;
namespace Console_PlaceHolder
{
 class Program
 {
  static void Main(string[] args)
  {
    string numara1, numara2;
    Console.Write("1. Numarayı Girin: ");
    numara1 = Console.ReadLine();
    Console.Write("2. Numarayı Girin: ");
    numara2 = Console.ReadLine();
    Console.Clear();
    Console.WriteLine("Girilen Numaralar {0} ve {1}", numara1, numara2);
    Console.ReadKey();
  }
 }
}
```

Kodumuzu çalıştırdığımızda örnek olarak `5` ve `10` sayılarını girdiğimizde, bu girilen değerler **`numara1`** ve **`numara2`** değişkenlerine aktarılır. Bu değişkenlerin içeriğini `{0} {1}` sıra numarası şeklinde konsol ekranında **`WriteLine`** komutu ile yazdırıyoruz. Buradaki `{0}` ve `{1}` değişkenlerin sırasını temsil etmektedir.

Bir yer tutucuyu birden fazla kullanmak mümkündür.

```
using System;
using System.Text;
namespace Console_PlaceHolder2
{
 class Program
 {
  static void Main(string[] args)
  {
    Console.WriteLine("Adınızı Girin");
    string ad = Console.ReadLine();
    Console.WriteLine("Yaşınızı Girin");
    int yas = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine ("Adınız {0} Yaşınız {1}. Uygulamamıza hoş
geldiniz {0}" ,ad, yas );
    Console.ReadKey();
  }
 }
}
```

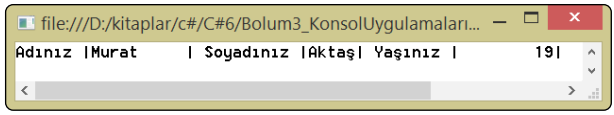
Örneğimizde index numarası ilk sırada olan **`ad`** değerini, `{0}` şeklinde belirterek iki defa yazdırdık.



Yer tutucuyu kullanarak konsol ekranına yazdırılan yazıyı, sütunlar halinde hizalayabilirsiniz. `{0,-10}` veya `{0,10}` şeklinde ikinci parametreye karakter adedi olarak hizalama değeri verilerek kullanılır. Eksi değerler sola dayalı olarak yazılırken, artı değerler sağa dayalı olarak yazılır.

```
using System;
using System.Text;
namespace Console_PlaceHolderHizalama
{
 class Program
 {
  static void Main(string[] args)
  {
    Console.WriteLine("Adınızı Girin");
    string ad = Console.ReadLine();
    Console.WriteLine("Soyadınızı Girin");
    string soyad = Console.ReadLine();
    Console.WriteLine("Yaşınızı Girin");
    int yas = Convert.ToInt32(Console.ReadLine());
    Console.Clear();
    Console.WriteLine("Adınız |{0,-10}| Soyadınız |{1,-4}| Yaşınız|
{2,10}|", ad, soyad, yas);
    Console.ReadKey();
  }
 }
}
```

Kodumuzda hizalamaları daha iyi görebilmek için değerleri `||` karakterleri arasına yazdırdık.



Ekran çıktısını inceleyecek olursak; ad değeri `{0,-10}` şeklinde belirtildiğinden, **`ad`** için `10` karakterlik yer ayrıldı ve sola dayalı olarak yazdırıldı. `10` karakterden daha kısa bir isim girildiği için kalan karakter sayısı kadar boşluk bırakıldı. **`soyad`** değeri `{1,-4}` şeklinde belirtildiği için **`soyad`** için 4 karakterlik yer ayrıldı ve sola dayalı olarak yazdırıldı. 4 karakter yer ayrılıp, girilen **`soyad`** 5 karakter olmasına rağmen herhangi bir karakter silinmeden girilen değer aynen ekrana yazdırıldı. Yer tutucu içerisinde belirtilen karakter sayısının girilen değere eşit ya da daha küçük olduğu durumlarda girilen değer, boşluk bırakılmadan aynen ekrana yazdırılır. Son olarak **`yas`** değeri `{2, 10}` şeklinde belirtildiği için **`yas`** için 10 karakterlik yer ayrıldı ve sağa dayalı olarak yazdırıldı. 10 karakterden daha kısa bir isim girildiği için kalan karakter sayısı kadar boşluk bırakıldı.

BackgroundColor ve ForegroundColor Metotları

`BackgroundColor` metodu, konsol ekranında yazılan yazıların arkaplan renklerini değiştirirken **`ForegroundColor`** metodu yazı rengini değiştirmek için kullanılır.

Renk değiştirmek için **`ConsoleColor`** renk sabitlerini kullanabilirsiniz. Kullanabileceğiniz renk değerleri şunlardır:

Renk Sabiti	Açıklama
`Black`	Siyah
`DarkBlue`	Koyu Mavi
`DarkGreen`	Koyu Yeşil
`DarkCyan`	Koyu Camgöbeği
`DarkRed`	Koyu Kırmızı
`DarkMagenta`	Koyu Eflatun
`DarkYellow`	Koyu Sarı
`Gray`	Gri

`DarkGray`	Koyu Gri
`Blue`	Mavi
`Green`	Yeşil
`Cyan`	Camgöbeği
`Red`	Kırmızı
`Magenta`	Eflatun
`Yellow`	Sarı
`White`	Beyaz

Örnek olarak konsol ekranında zemin ve yazı rengini değiştiren bir uygulama yapalım.

```
static void Main(string[] args)
{
 Console.BackgroundColor = ConsoleColor.Gray;
 Console.ForegroundColor = ConsoleColor.Black;
 Console.Clear();
 Console.Write("Adınızı Girin: ");
 Console.ReadKey();
}
```

Kodumuzda zemin rengini **gray** (*gri*), yazı rengini **black** (*siyah*) olarak belirledik. Zemin rengini tüm konsol ekranında geçerli kılmak için **`Console.Clear`** metodunu kullanarak ekranı temizledik.



ResetColor metodunu kullanarak yazı ve zemin renklerini, konsol ekranının varsayılan renklerine döndürebiliriz.

```
static void Main(string[] args)
{
 Console.BackgroundColor = ConsoleColor.Gray;
 Console.ForegroundColor = ConsoleColor.Black;
 Console.WriteLine("Adınızı Girin: ");
 Console.ResetColor();
 Console.WriteLine("Soyadınızı Girin: ");
 Console.ReadKey();
}
```

Uygulamamızı çalıştırdığımızda ilk satırda zemin rengi gri, yazı rengi siyah olarak "*Adınızı Girin:*" metni görüntülendi. Ardından **ResetColor** metodu ile konsol ekranı, varsayılan renkleri olan zemin rengi siyah, yazı rengi beyaz olarak değiştirilerek "*Soyadınızı Girin:*" metni görüntülendi.



ConsoleKeyInfo Yapısı

Konsol ekranında **Read** metodunu kullanarak basılan karakteri okuyabilir ve **char** metodu ile dönüşüm yaptırarak hangi karaktere basıldığı bilgisine erişebiliriz.

```
static void Main(string[] args)
{
 char karakter;
 Console.Write("Bir tuşa basarak ardından enter'a basınız: ");
 karakter = (char)Console.Read();
 Console.WriteLine("Basılan tuş: " + karakter);
 Console.ReadKey();
}
```

Kodumuzda **karakter** isimli değişkeni **char** türünden tanımlayarak, **Read** metodu ile klavyeden girilen karakter bilgisini okuyarak karakter isimli değişkene aktardık. **char** metodu, ASCII kodu belirtilen karakteri üretir. Klavyeden **a** harfine basıldığı anda, **Read** metodu ile **a** harfinin ASCII karakter kodu karşılığı olan **97** değeri üretilecektir. **char** metodu ile de **97** değeri **a** karakterine dönüştürülecektir.



Ancak her ne kadar **Read** metodu ile konsol ekranında basılan karakterleri okuyabilsek de **Enter**, **F1**, **Spacebar** gibi tuşları okuyamayız. Bunun için **ReadKey** metodunu ve **ConsoleKeyInfo** yapısını kullanacağız.

ReadKey metodu, konsol ekranında basılan herhangi bir tuşu okumak amacıyla kullanılırken, **ConsoleKeyInfo** yapısı ise konsol ekranında basılan tuş ile ilgili bilgi almak amacı ile kullanılır. **ConsoleKeyInfo** yapısı, **System** sınıfı altında bulunmaktadır.

```
static void Main(string[] args)
{
 Console.WriteLine("Bir Tuşa Basınız");
 System.ConsoleKeyInfo KeyInfo = Console.ReadKey(true);
 Console.WriteLine(KeyInfo.Key.ToString() + " tuşuna bastınız");
 Console.ReadLine();
}
```



Örneğimizde basılan tuşu **ReadKey** metodu ile okuyarak, tuş bilgisini **KeyInfo** isimli değişkene aktarıp, **KeyInfo** nesnesinin **Key** özelliği ile basılan tuş bilgisini ekrana yazdırıyoruz. Basılan tuş bilgisi **ConsoleKeyInfo** türünden olduğu için **ToString** metodu ile **string** türüne dönüştürerek ekrana yazdırıyoruz.

Basılan tuşa göre ekrana yazdırma ya da programdan çıkış işlemi yapacak bir örnek uygulama yapalım.

```
static void Main(string[] args)
{
  string ad, soyad;
  Console.Write("Adınızı Girin=");
  ad = Console.ReadLine();
  Console.Write("Soyadınızı Girin=");
  soyad = Console.ReadLine();
  Console.Clear();
  Console.WriteLine
 ("Ekrana yazdırmak için F2, Çıkmak için ESC tuşuna basınız");
  System.ConsoleKeyInfo KeyInfo = Console.ReadKey(true);
  if (KeyInfo.Key == ConsoleKey.Escape)
  {
   System.Environment.Exit(0);
  }
  else if (KeyInfo.Key == ConsoleKey.F2)
  {
   Console.Write(ad + " " + soyad);
   Console.ReadKey();
  }
}
```

Örneğimizde…

```
KeyInfo.Key ==ConsoleKey.Escape
```

… satırı ile konsol ekranındayken, **Escape** tuşuna basılıp basılmadığını tespit edip, eğer basılmışsa…

```
System.Environment.Exit(0);
```

… satırı ile konsol ekranının kapatılmasını sağlıyoruz. **`ConsoleKey`** sabiti, klavyedeki tuşları belirtir. **`ConsoleKey.TuşAdı`** şeklinde kullanılır.

```
KeyInfo.Key == ConsoleKey.F2
```

… satırı ile ise konsol ekranındayken, **F2** tuşuna basılıp basılmadığını tespit eder. Eğer basılmışsa…

```
Console.Write(ad + " " + soyad);
```

… satırı ile konsol ekranına **ad** ve **soyad** değişkenlerinin içeriklerini yazdırıyoruz.

ConsoleKeyInfo yapısı tek başına, konsol ekranında **Ctrl**, **Alt** ve **Shift** tuşlarına basılıp basılmadığını algılamak için yeterli değildir. Bu amaçla **ConsoleKeyInfo** yapısının **Modifiers** özelliğinden faydalanabiliriz.

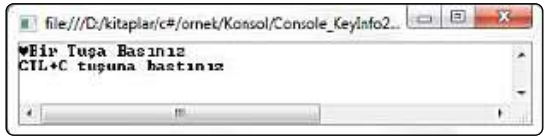
```
static void Main(string[] args)
{
 Console.TreatControlCAsInput = true;
 System.ConsoleKeyInfo KeyInfo;
 Console.WriteLine("Bir Tuşa Basınız");
 KeyInfo = Console.ReadKey();

 if ((KeyInfo.Modifiers & ConsoleModifiers.Alt) != 0)
Console.Write("ALT+");
 if ((KeyInfo.Modifiers & ConsoleModifiers.Shift) != 0)
Console.Write("SHIFT+");
 if ((KeyInfo.Modifiers & ConsoleModifiers.Control) != 0)
Console.Write("CTRL+");
 Console.WriteLine(KeyInfo.Key.ToString() + " tuşuna bastınız");
 Console.ReadLine();
}
```

Örneğimizde **ConsoleModifiers** özelliğinin aldığı değerlere göre **Ctrl**, **Alt** ya da **Shift** tuşları ile birlikte özel tuşlara basılıp basılmadığını anlıyoruz. Ayrıca program esnasında **Ctrl+C** tuşlarına basıldığında, varsayılan olarak programın çalışması kesilirken…

```
Console.TreatControlCAsInput = true;
```

… satırı ile programın çalışmasının kesilmeyip, **Ctrl+C** tuşlarına basıldığının da algılanmasını sağlıyoruz.



Konsol ekranında girilen metnin, şifre karakterleri ile görülmesini engelleyen bir uygulama yapalım.

```
static void Main(string[] args)
{
 ConsoleKeyInfo KeyInfo = default(ConsoleKeyInfo);
 string password = "";
 do
 {
  KeyInfo = Console.ReadKey(true);

  if (KeyInfo.Key != ConsoleKey.Backspace)
  {
   password += KeyInfo.KeyChar;
   Console.Write("*");
  }
 } while (!(KeyInfo.Key == ConsoleKey.Enter));

 Console.WriteLine();
 Console.WriteLine("Girilen Şifre : " + password);
 Console.ReadLine();
}
```

Örneğimizde…

```
do
{
} while (!(KeyInfo.Key == ConsoleKey.Enter));
```

… döngüsü ile konsol ekranında **Enter** tuşuna basılana kadar sürekli çalışan bir döngü kuruyoruz. Bu döngü içerisinde öncelikle **Backspace** tuşuna basılıp basılmadığını kontrol ediyoruz. **Backspace** tuşuna basılmadıkça işlem yapılmasını sağlıyoruz. Aksi takdirde **Backspace** tuşuna basıldığında da * karakteri konsol ekranında yazdırılacaktır. **Backspace** tuşuna basılmadı ise konsol ekranına girilen yazıyı...

```
password += KeyInfo.KeyChar;
```

… satırı ile **password** değişkenine aktararak, bunun yerine…

```
Console.Write("*");
```

… satırı ile konsol ekranına * karakterini yazdırıyoruz. Programın sonunda girilen metni **password** değişkeni içeriğinden okuyarak yazdırıyoruz.



SetCursorPosition Metodu

İmleci konsol ekranında belirtilen koordinata konumlandırmak amacıyla kullanılır.

```
SetCursorPosition( int yatay koordinat, int dikey koordinat)
```

Örneğin; konsol ekranının ortasına konumlanarak, belirtilen mesajı yazan bir uygulama yapalım.

```
static void Main(string[] args)
{
 Console.SetWindowSize(30, 15);
 Console.SetCursorPosition(Console.WindowWidth / 2, Console.WindowHeight / 2);
 Console.WriteLine("C Sharp");
 Console.ReadLine();
}
```

Kodumuzda **SetWindowSize** metodu ile konsol ekranının genişlik ve yükseklik değerlerini değiştiriyoruz.

SetWindowSize metodu şu şekilde kullanılır:

```
SetWindowSize(int genişlik, int yükseklik)
```



Console.WindowWidth ve **Console.WindowHeight** ile ekranın genişlik ve yükseklik değerlerini elde ederek, bu değerleri ikiye bölüyor ve **Console.SetCursorPosition** metodu ile de bu koordinatlara konumlanıyoruz.

Son olarak da bu koordinata mesajımızı yazdırıyoruz.

Konsol Uygulamalarından Çıkış

Her ne kadar konsol uygulamalarında yapılan işlemler bittiğinde, aksi belirtilmedikçe, uygulamalarımız kendiliğinden sonlanmakta ise de; uygulamanın istediğimiz bir yerinde uygulamayı sonlandırıp, konsol ekranından çıkmak mümkündür. Bunun için **Environment** sınıfının **Exit** metodunu kullanırız.

```
static void Main(string[] args)
{
 Console.WriteLine("Çıkmak İstediğinize Emin misiniz? [E/H]");
 string cevap = Console.ReadLine();
 if (cevap == "E")
  Environment.Exit(0);
 else
  Console.WriteLine("Devam");
 Console.ReadLine();
}
```

Örneğimizde eğer konsol ekranında sorulan soruya cevap olarak **E** değeri girilirse, konsol uygulamamız sonlandırılmaktadır.

SABİTLER VE DEĞİŞKENLER

4

SABİTLER

Sabitler, içeriği sabit olan değer ve ifadelerin saklanması amacı ile kullanılırlar. **C#** dilinde sabitleri **`const`** ya da **`readonly`** ifadesi kullanarak iki şekilde tanımlamak mümkündür.

- **const :**

  Sabit değerleri tanımlamada kullanılır. Değerleri tanımlandıkları anda verilmelidir. Örneğin:

```
const int rakam=45;
```

  Satırında, **`rakam`** isimli sabiti **`int`** türünden tanımlayarak değerini **`45`** olarak belirledik.

  `const` ifadesi ile tanımlanmış bir sabitin değeri, derleme esnasında değiştirilemez. Değiştirilmeye çalışılırsa **derleme zamanı hatası** verecektir.

Örneğin;

```
const int rakam = 45;
rakam = 12;
```

(local constant) int rakam = 45

The left-hand side of an assignment must be a variable, property or indexer

Yukarıdaki kod satırlarında görüldüğü gibi **rakam** isimli sabitin değerini ilk tanımlandığı anda **45** olarak belirleyip daha sonra **12** olarak değiştirmeye kalktığımızda, **derleme zamanı hatası** vererek uygulamamız çalışmamıştır.

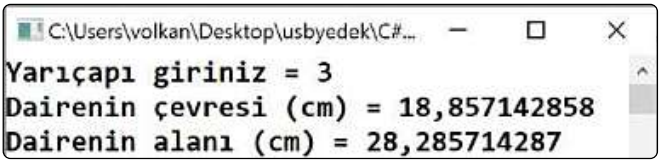
Örneğin dairenin çevresi ve alanını bulan bir program yapalım. Çevre ve alan formüllerinde kullanacağımız **pi** değeri sabit olup **3,14** değerine karşılık gelmektedir. Bu nedenle örneğimizde **pi** değerini sabit olarak tanımlayacağız.

```
static void Main(string[] args)
{
    const double PI = 3.142857143;
    Double alan,çevre,yarıçap;

    Console.Write("Yarıçapı giriniz = ");
    yarıçap = Convert.ToInt32(Console.ReadLine());

    çevre = 2 * PI * yarıçap;
    alan = PI * (Math.Pow(yarıçap, 2));

    Console.WriteLine("Dairenin çevresi (cm) = " + çevre );
    Console.WriteLine("Dairenin alanı (cm) = " + alan);
    Console.ReadLine();
 }
```



Örneğimizde **pi** değeri **3.142857143** şeklinde sabit olup değeri değişmediği için **const** ifadesi ile **sabit** olarak tanımladık.

Ondalıklı değer olması nedeniyle de **double** türünden tanımladık. Yine alan, çevre ve yarıçap değişkenlerini de ondalıklı değer içerebileceğinden dolayı **double** veri türü ile tanımladık.

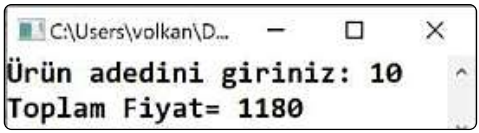
Dairenin çevresini bulmada kullanılan **2 x Pi x r** formülünü kodumuzda **2 * PI *** yarıçap şeklinde ifade ettik.

Dairenin alanını bulmada kullanılan **Pi x r2** formülünü ise kodumuzda **PI * (Math.Pow(yarıçap, 2))** şeklinde ifade ettik. **Math** sınıfına ait olan **Pow** metodu üs almada kullanılır. **Math.Pow(değer, üs)** şeklinde kullanılır.

Klavyeden girilen yarıçap değerini **Console.ReadLine** metodu ile okuttuk. **Console.ReadLine** metodu geriye **string** değer döndürdüğünden dolayı, okunan değeri **Convert.ToInt32** metodunu kullanarak **int** sayısal değişken türüne dönüştürdük.

const ifadesi ile bir sabiti tanımlarken başka bir sabitin değerini kullanarak tanımlamak da mümkündür.

```
static void Main(string[] args)
{
    const double urunfiyatı = 100;
    const double kdvlifiyat = urunfiyatı + (urunfiyatı * 0.18);
    byte adet = 0;
    Console.Write("Ürün adedini giriniz: ");
    adet = Convert.ToByte (Console.ReadLine()) ;
    Console.WriteLine("Toplam Fiyat= " + kdvlifiyat * adet);
    Console.ReadKey();
}
```



Örneğimizde **kdvlifiyat** adlı sabitimizi tanımlarken **urunfiyati** adlı sabitin değeri ile işleme tabi tutarak **urunfiyatı + (urunfiyatı * 0.18)** şeklinde yeni bir değer elde ettik.

- **readonly :**

  Sabit değerleri tanımlamada kullanılır. Tanımlandığı anda değeri verilebileceği gibi varsayılan değer kullanarak tanımlanıp değeri daha sonra da verilebilir. Değerleri **Class Constructor**'ında (*sınıfın yapıcı metodunda*) ya da çalışma zamanında verilebilir.

  Örnek olarak yaş hesaplayan bir uygulama yapalım.

```
readonly int yil = 2014;
static void Main(string[] args)
{
    Console.WriteLine("Doğum yılınızı giriniz");
    int dogum_yili = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Yaşınız= " +(yil-dogum_yili));
    Console.ReadKey();
}
```

Kodumuzu yukarıdaki gibi yazarak **`yil`** değişkenini **`readonly`** olarak tanımlayalım. Kodumuz doğru gibi görünse de **`yil`** değişkenini kod içerisinde kullanmaya kalktığımızda aşağıda görüldüğü gibi **erişim hatası** vermektedir.

```
(yil - dogum_yili));
```

(field) int Program.yil

An object reference is required for the non-static field, method, or property 'ReadOnly_YasHesap.Program.yil'

Bunun için,

```
readonly int yil = 2014;
```

olarak yaptığımız **`yil`** değişkeninin tanımlamasını,

```
public static readonly int yil = 2014;
```

şeklinde değiştirelim. **`public`** ve **`static`** ifadeleri erişim belirleyici olarak adlandırılmaktadır. Bir metot ya da sınıf içerisinden bir değişkene ya da nesneye erişim biçimini belirlemek amacıyla kullanılırlar. Kodumuzda, **`Main`** metodu içerisinden **`yil`** değişkenine serbestçe erişebilmek amacıyla, **`yil`** değişkenini **`public static`** olarak tanımladık.

readonly ifadesi ile yapılan sabit tanımlamalarında, değerin çalışma zamanında verilebileceğini belirtmiştik. Örneğimizi buna göre değiştirerek **yil** sabitinin değerinin **DateTime.Now.Year** metodu ile çalışma zamanında **otomatik** olarak verilmesini sağlayalım.

Kodumuz aşağıdaki gibi olacak:

```
public static readonly int yil=DateTime.Now.Year;
static void Main(string[] args)
{
    Console.WriteLine("Doğum yılınızı giriniz");
    int dogum_yili = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Yaşınız= " +(yil-dogum_yili));
    Console.ReadKey();
}
```

Aynı sabit tanımlamasını **readonly** yerine, **const** ifadesi ile:

```
const int yil=DateTime.Now.Year;
```

şeklinde yapmaya kalksaydık aşağıda görüldüğü gibi:



hata verecekti. Çünkü, **const** ifadesi ile yapılan sabit tanımlamalarında, çalışma zamanında değer aktarımı yapılamaz. Ancak derleme zamanında değer aktarımı yapılabilir.

`readonly` sabit değerlerini **Class Constructor**'ında (*sınıfın yapıcı metodunda*) yapabileceğimizi söylemiştik. Geçerli tarihi bularak konsol ekranına yazdıran bir uygulama yapalım.

```
class Program
{
    static void Main(string[] args)
    {
        Program nsn_tarih = new Program();
        Console.ReadLine();
    }

    readonly DateTime tarih;
    public Program()
    {
        tarih = DateTime.Now;
        Console.WriteLine(tarih);
    }
}
```

Kodumuzda:

```
readonly DateTime tarih;
```

şeklinde **`tarih`** isimli sabitimizi tanımladık, ancak değerini vermedik.

Değerini vermek için ise **`Program`** isimli class (sınıf) ile aynı isimde bir **Constructor Method** (*yapıcı metot*) oluşturduk. Yapıcı metodun içerisinde;

```
tarih = DateTime.Now;
```

satırında **`tarih`** isimli sabite, **`DateTime.Now`** metodu ile geçerli zamanı **değer** olarak aktardık.

```
Program nsn_yil = new Program();
```

Satırında ise oluşturmuş olduğumuz **`Program`** isimli sınıfımız ile aynı ismi taşıyan **`Program()`** isimli **yapıcı metodu**muzu çalıştırdık.

Değişkenler

Program içerisinde üretilen değerleri bellekte geçici olarak saklamak amacı ile kullanılırlar. Bir değişken tanımlanırken değişken ismi ve değişken tipi belirtilerek tanımlanırlar.

```
[Değişken tipi] [Değişken adı]  = [İlk Değer]
```

Şeklinde tanımlanırlar. İlk değer kullanımı zorunlu değildir. Program içerisinde istenirse daha sonra değişkene değer aktarımı yapılabilir.

Örneğin:

```
string adsoyad;
```

şeklinde ilk değer ataması yapılmadan tanımlanabileceği gibi:

```
string adsoyad =  "Volkan AKTAŞ";
```

şeklinde ilk değer ataması yapılarak da kullanılabilir. Bu tanımlamada **`string`** değişken türünü belirtirken **`adsoyad`** değişken ismini belirtir. Değişkenin ilk değeri ise "**`Volkan AKTAŞ`**" şeklindedir.

Eğer bir değişkene değer aktarımı yapılmazsa değişken varsayılan değeri ile kullanılır.

Aynı satırda birden fazla değişkeni tanımlamak mümkündür. Ancak aynı satırda tanımlanacak değişkenler aynı veri türünden olmalıdır. Değişken isimleri birbirinden **virgül**le ayrılarak yazılır.

```
string ad, soyad;
```

Yukarıdaki satırda ad ve soyad değişkenlerini **`string`** veri türü ile tanımladık.

Aynı satırda birden fazla değişken tanımlaması yapılabileceği gibi değişkenlere değer ataması da yapılabilir.

```
string ad="Volkan", soyad="AKTAŞ";
```

Yukarıdaki satırda **ad** ve **soyad** değişkenlerini **string** veri türü ile tanımlayarak **ad** değişkenine ilk değer olarak "**Volkan**", **soyad** değişkenine ise ilk değer olarak "**AKTAŞ**" değerini aktardık.

Aynı satırda değişkenleri tanımlarken bazı değişkenlere değer aktarması yaparak bazılarına ise değer aktarması yapmadan tanımlayabilirsiniz.

```
int a = 5, b = 10, c;
```

Satırında tüm değişkenleri **int** türünde tanımlarken **a** değişkenine ilk değer olarak **5**, **b** değişkenine de **10** değerini atadık. **c** değişkenine ise ilk değer ataması yapmadık.

Değişkenlere değer ataması yaparken değerler üzerinde çeşitli işlemler yapabilirsiniz.

```
int a = 5+10;
```

Satırında **a** değişkeninin değeri 15 olacaktır.

Bir başka değişkenden faydalanarak değişken tanımlaması yapabiliriz.

```
int a = 5;
int b = a + 3;
```

Satırlarında **b** değişkeninin değeri **8** olacaktır.

Değişkenler tanımlanırken aşağıdaki kurallara uyulmalıdır:

- Değişken ismi **harf** ile başlamalıdır. Diğer karakterler harf, rakam ya da özel karakter olabilir.
- Değişken ismi en fazla **255** karakter olabilir.
- Değişken ismi tanımlanırken **boşluk** bırakılmamalıdır. Bunun yerine _ (*alt çizgi*) gibi özel karakterler kullanılabilir.
- C#.Net içerisinde büyük küçük harf ayrımı olduğundan dolayı değişkenler hangi şekilde tanımlandılarsa program içerisinde de o şekilde kullanılmalıdırlar.
- C# .Net içerisinde kullanılan komutlar değişken ismi olarak kullanılamaz. (*AND, OR, NOT gibi*)

C# içerisinde metotları isimlendirirken küçük ya da büyük harf ile metot isimlerini biçimlendirmede üç çeşit isimlendirme biçimi kullanılır. Bizler de kendi tanımladığımız metotlarda ve değişkenlerde aynı isimlendirme biçimlerini kullanabiliriz. Bu isimlendirme biçimleri aşağıdaki gibidir;

- **Pascal Case:** Metodu ya da değişkeni oluşturan bitişik sözcüklerin ilk harfleri **büyük harfle** yazılır.

  `ForeColor`, **`MessageBox`** gibi.

- **Camel Case:** Metodu ya da değişkeni oluşturan bitişik sözcüklerden ilk sözcüğün baş harfleri küçük harfle yazılırken diğer sözcüklerin ilk harfleri büyük harfle yazılır.

  `foreColor`, **`messageBox`** gibi.

- **UpperCase:** Metodu ya da değişkeni oluşturan sözcüklerden ilk sözcüklerin baş harfleri büyük harfle yazılırken son sözcüğün tamamı büyük harfle yazılır. Genellikle iki, üç harften oluşan kısa sözcüklerde kullanılır.

  `System.IO`, **`System.Web.UI`** gibi.

Değişken Tipleri

Değişkenler değer tipi ve referans tipi değişkenler olmak üzere iki gruba ayrılırlar. Değer tipi değişkenler bellekte **stack** adı verilen alanlarda saklanırken referans tipi değişkenler ise **heap** adı verilen alanlarda saklanır.

Stack içerisinde saklanacak verinin boyutu bilinmelidir. Heap içerisinde saklanacak verinin ise boyutu dinamik olarak belirlenip çalışma zamanında kullanıma göre belirlenir.

Heap bölgesi, stack bölgesine göre **daha büyük** boyuttaki verileri saklayabilir.

Stack içerisinde saklanan değişkenlerin kullanımı bittikten sonra otomatik olarak bellekten düşer. Heap içerisinde tanımlanan değişkenlerin kullanımı bittikten sonra **garbage collector** yardımı ile bellekten düşümü yapılır.

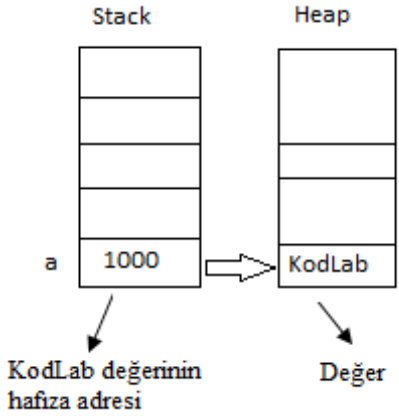
Heap içerisinde gerçekleştirilen işlemler, stack içerisinde gerçekleştirilen işlemlere göre **daha yavaş** gerçekleştirilir.

Referans tipi değişkenlerde değişkenin değeri heap içerisinde, heap içerisinde saklanan bu değere erişim için kullanılan referans hafıza adresi ise stack bölümünde saklanır.

Örneğin `string` değişken tipi **referans** türde bir değişkendir.

```
string a = "KodLab";
```

Tanımlaması sonucunda bellekte aşağıdaki işlemler gerçekleşmektedir:



a değişkeni tanımlandığında heap içerisinde, **a** değişkeninin değeri olan "**KodLab**" değeri sığacak kadar bir alan ayrıldı. Heap içerisinde ayrılan alanı gösteren referans hafıza adresi de stack içerisinde saklandı.

Başlıca değişken tipleri aşağıdaki gibidir:

Değer Tipleri:

Değişken tipi	CTS tipi	Kapladığı Alan	Değer Aralığı
byte	System.Byte	8 bit işaretsiz tam sayı	0 : 255
sbyte	System.Byte	8 bit işaretli tam sayı	-128 : 127
short	System.Int16	16 bit işaretli tam sayı	-32.768 : 32.767
ushort	System.UInt16	16 bit işaretsiz tam sayı	0 : 65.535
int	System.Int32	32 bit işaretli tam sayı	-2.147.483.648 : 2.147.483.647
uint	System.UInt32	32 bit işaretsiz tam sayı	0 : 4.294.967.295
long	System.Int64	64 bit işaretli tam sayı	-9.223.372.036.854.775.808 ile 9.223.372.036.854.775.807 arası
ulong	System.UInt64	64 bit işaretsiz tam sayı	0 : 18.446.744.073.709.551.615
float	System.Single	32 bit tek kayan sayı	+ ya da - $1,5*10^{-45}$ ile + ya da - $3,4*10^{38}$ arası
double	System.Double	64 bit çift kayan sayı	+ya da – $5,0*10^{-324}$ ile + ya da - $1,7*10^{308}$ arası
decimal	System.Decimal	128 bit ondalıklı sayı	+ya da - $1,0*10^{-28}$ ile + ya da - $7,9*10^{28}$ arası
bool	System.Boolean		**true** ya da **false**
char	System.Char	16 bit Unicode karakter	Tek bir karakteri içerir

Referans Tipleri:

Değişken tipi	
string	Unicode karakterlerinden oluşan string.
object	Genel veri tipi. Tip dönüşümlerinde oluşan hatalar derleme esnasında göz ardı edilerek çalışma zamanında hata üretilir.
dynamic	Genel veri tipi. Tip dönüşümlerinde oluşan hatalar üzerinde derleme esnasında herhangi bir kontrol yapılmadığı için hata üretilmez.

Değişkenlerin listelendiği tabloya dikkat edecek olursak değişken tipi sütununun yanında CTS tipi sütunu görülmektedir. Herhangi bir **.NET** dili ile yazılan program kodu derleme anında **IL** (*Microsoft Intermediate Language*) denilen bir ara dile çevrilir. Oluşturulan kod, **CLR** (*Common Language Runtime*) içerisinde makine diline çevrilir. Bu dönüştürme işlemi **CTS** (*Common Type System*) kurallarına göre yapılır. Bu sayede farklı dillerden, ortak bir standart altında dönüşüm yapılmış olur. Örneğin; **C#** içerisinde **`float`** olarak isimlendirilen değişken tipi, **VB.Net** içerisinde **`single`** olarak isimlendirilmektedir. **IL** koduna dönüştürme işlemi yapılırken her iki dildeki bu değişken tipi **`System.Single`** olarak dönüştürülür.

Değişkenleri tanımlarken ister o dile ait değişken tipi tanımlamasıyla, isterse CTS tipi tanımlamasıyla tanımlayabilirsiniz.

Örneğin:

```
int rakam = 10;
```

tanımlaması ile;

```
System.Int32 sayi = 10;
```

tanımlaması aynı anlama gelmektedir.

Değişkenlere **`null`** (*boş değer*) ataması yapılırken soru işaretinden faydalanarak değer ataması yapılır. Örneğin:

```
int sayi;
```

tanımlaması değer türünden değişkenin **`null`** olmayan bir şekilde tanımlanmasını sağlarken:

```
int? sayi;
```

tanımlaması ile değer türünden değişkenin **`null`** atama yapılarak tanımlanmasını sağlar.

Referans türünden tanımlama da:

```
string yazi;
```

şeklinde **null** olmayan biçimde yapılırken;

```
string? yazi;
```

tanımlaması da **null** atama yapılarak tanımlanabilir. (*Referans değişkenlere null atama yapılabilmesi C# 8 versiyonundan itibaren geçerli olan bir özelliktir*.)



Değişkenler tanımlanırken değişkenleri, tiplerini belirtmeden **var** ifadesini kullanarak tanımlayabiliriz. **Var** ifadesi ile tanımlanan değişkenlere ilk değer ataması yapılması gereklidir. Aksi takdirde derleyici, hata verecektir. Değişkenlerin tipleri değişken içerisine aktarılan değerlere göre otomatik olarak C# tarafından belirlenir.

```
var  değişken adı  = ilk değer;
```

Şeklinde kullanılırlar. Örneğin;

```
var adsoyad="talha";
```

Şeklinde kullanım ile **adsoyad** değişkenini herhangi bir tür belirtmeden tanımlayıp **string** değer ataması yaparak **adsoyad** değişkeninin veri türünün **string** olarak belirlenmesini sağladık.

NOT

C# 7 versiyonundan itibaren kullanılmaya başlanan Digit Seperator özelliği ile rakamları basamaklarına ayırarak daha düzenli bir şekilde yazabiliriz.

Örneğin;

```
int a = 33_000_000;
```

```
int b = 33_2_1;
```

şeklinde tanımladığımız değişken içeriğini konsol ekranına yazdırdığımızda sırasıyla `33000000` ve `3321` şeklinde çıktı verecektir.

Benzer şekilde digit separator özelliğini kullanarak hexadesimal ve binary sayıları daha düzenli bir şekilde tanımlayabiliriz. Örneğin;

```
int d = 0b0000_1100_1111_1001;
```

şeklinde düzenli bir biçimde tanımladığımız binary değişken içeriğini konsol ekranına yazdırdığımızda `3321` şeklinde sonuç verecektir.

```
int h = 0xff_ff_1a;
```

şeklinde yine digit seperator kullanarak düzenli bir biçimde tanımladığımız hexadesimal (16'lık sayı) ise onluk sayı sistemi karşılığı olan `16776986` sonucunu verecektir.

C# 7.2 versiyonundan itibaren bu özelliği:

```
int d = 0b_0000_1100_1111_1001;
```

```
int h = 0x_ff_ff_1a;
```

şeklinde daha gelişmiş bir biçimde başlangıç kısmından itibaren kullanmaya başlayabiliyor ve yazdığımız ifadelerin daha okunabilir olmasını sağlayabiliyoruz.

`Object` ve **`dynamic`** veri tiplerinin farkını daha iyi anlamak için bir **örnek** yapalım.

```
static void Main(string[] args)
{
object rakam = 10;
      Console.WriteLine(rakam.GetType());
      Console.ReadKey();
}
```

Örneğimizi çalıştırdığımızda **`System.Int32`** sonucunu verecektir. **`Object`** veri türü, tanımlandığı değişkene atanan değere göre değişken tipini belirlemektedir.

Aynı işlemi:

```
static void Main(string[] args)
{
dynamic rakam = 10;
      Console.WriteLine(rakam.GetType());
      Console.ReadKey();
}
```

Biçiminde **`dynamic`** veri türü ile yaptığımızda da aynı şekilde **`System.Int32`** sonucunu verecektir.

```
static void Main(string[] args)
{
object rakam = 10;
rakam = rakam + 10;
      Console.WriteLine(rakam);
      Console.ReadKey();
}
```

Şeklinde yazacağımız kod ise (**`Operator '+' cannot be applied to operands of type 'object' and 'int'`**) biçiminde **hata** üretecektir. **`Object`** ile **`int32`** türüne dönüşüm yapmamıza rağmen **`rakam`** değişkeni hâlâ **`object`** olarak kabul edildiğinden dolayı **hata** meydana gelecektir.

```
static void Main(string[] args)
{
object rakam = 10;
rakam = (int)rakam + 10;
      Console.WriteLine(rakam);
      Console.ReadKey();
}
```

Rakam değişkenini **int** türüne dönüştürerek toplama işlemini gerçekleştirdiğimizde bu defa hata meydana gelmeden işlem yapılacak ve **20** değeri konsol ekranında görüntülenecektir.

```
static void Main(string[] args)
{
object rakam = 10;
rakam= (double)rakam + 10;
      Console.WriteLine(rakam);
      Console.ReadKey();
}
```

Ancak yukarıda görüldüğü gibi başka bir veri türüne dönüşüm işlemi yapıp toplama işlemini gerçekleştirdiğimizde derleme esnasında herhangi bir hata mesajı görüntülenmeyip çalışma esnasında "**Specified cast is not valid**." biçiminde dönüşüm hatası meydana gelecektir.

```
static void Main(string[] args)
{
dynamic rakam = 10;
rakam= (double)rakam + 10;
      Console.WriteLine(rakam);
      Console.ReadKey();
}
```

Aynı işlemi **dynamic** veri tipi ile gerçekleştirdiğimizde ise oluşan hatalar göz ardı edilerek konsol ekranında **20** değeri görüntülenecektir.

Tanımlandıkları Yerlere Göre Değişken Türleri

Değişkenleri tanımlandıkları yerlere göre **global** (*genel*) ve **local** (*yerel*) olmak üzere ikiye ayırabiliriz.

1) Local (*yerel*) **değişken**ler, bir metot ya da kod bloğunun içerisinde tanımlanarak sadece tanımlandıkları metot ya da kod bloğunun içerisinde geçerli olur. Bir değişkeni **yerel** olarak tanımlama işlemi aşağıda görüldüğü gibi yapılır;

Metot, ya da kod bloğu:

```
{
 [Değişken tipi]  [Değişken adı] ;
}
```

Bu biçimde tanımlanan değişkenler içerisindeki değerleri yeniden tanımlanıncaya içerisindeki değer sıfırlanıncaya ya da proje sonlandırılıncaya kadar muhafaza ederler.

2) Global (*genel*) olarak tanımlanan değişkenler, tanımlandıkları yere göre bir kod bloğunun, metodun ya da sınıfın tamamında geçerli olur. Bir değişkeni bir kod bloğunun tamamında geçerli yapmak için:

```
[Değişken tipi]  [Değişken adı] ;
{
 Kod satırları;
}
```

Şeklinde değişkeni kod bloğunun dışında tanımlarız.

Tanımlanan değişkenin bir metot ya da sınıfın tamamında geçerli olması için:

```
class OrnekClass
{
        public static [Değişken tipi]  [Değişken adı] ;
        Metod1( )
        {
    Kod satırları;
        }
        Metod2( )
        {
```

```
    Kod satırları;
        }
}
```

Şeklinde değişkenin başına **public static** ifadesini ekleyerek **class** (*sınıf*) içerisinde **class OrnekClass** satırının altında tanımlama işlemi yapılır.

Tanımladığımız değişken, **OrnekClass** isimli **class** içerisindeki **Metod1** ve **Metod2** isimli metotlarda geçerli olacaktır. Bu şekilde bir tanımlama yapıldığında değişken nerede tanımlanmış olursa olsun değişken ismini yazarak herhangi bir kod bölümünden değişkene erişim sağlanabilir. **Main** metodu içerisinden değişkene serbestçe erişebilmek amacıyla, değişkeni **public static** olarak tanımladık.



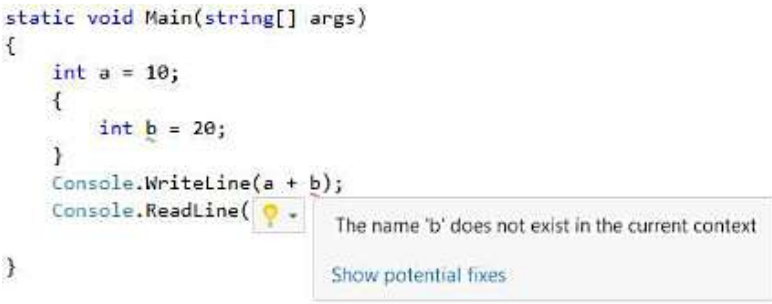
Public ve **static** ifadeleri **attribute** (*nitelik*) olarak adlandırılmaktadır. Nitelikler konusu **class** (*sınıf*) bölümünde ayrıntısıyla işleneceği için burada ayrıntıya inilmemiştir.

Örneğin:

```
static void Main(string[] args)
{
int a = 10;
      {
          int b = 20;
          Console.WriteLine(a + b);
      }
      Console.ReadLine();
}
```

Yukarıdaki kod satırlarında **b** değişkeni bir kod bloğu içerisinde tanımlanmış olup yalnızca tanımlanmış olduğu kod bloğu içerisinde geçerlidir. **a** değişkeni ise kod bloğunun dışında tanımlanmış olduğu için kod bloğunun içerisinde de geçerlidir. Dolayısı ile kod bloğu içerisinde yapılan **a** ve **b** değişkenlerinin değerlerini, toplama işlemi doğru bir şekilde çalışarak **30** sonucunu konsol ekranına yazdıracaktır.

Toplama işlemini aşağıda görüldüğü gibi kod bloğu dışına taşıyalım.



Kodumuzda **b** değişkeni yalnızca tanımlandığı kod bloğu içerisinde geçerli olduğu için, kod bloğu dışında yapacağımız toplama işleminde, **b** değişkeni derleme esnasında erişim hatası verecektir.

Değişkenimizi, başına **public static** ifadesini ekleyerek **class** içerisinde tanımlayıp erişim hatasını giderebiliriz.

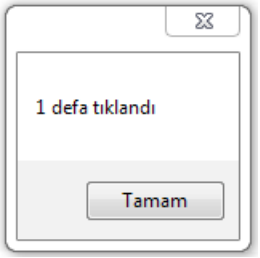
```
class Program
{
    public static int b;
    static void Main(string[] args)
    {
        int a = 10;
        {
            b = 20;
        }
        Console.WriteLine(a + b);
        Console.ReadLine();
    }
}
```

Kodumuzda **b** değişkenini genel olarak tanımlayıp değerini kod bloğu içerisinden verdik. Kodumuz doğru bir şekilde çalışarak **30** sonucunu üretecektir.

Genel ve **yerel** değişken kullanımına örnek olmak üzere, form üzerindeki bir butona tıklanma sayısını bulan örnek bir **windows form** uygulaması yapalım. Form üzerine bir adet **buton** ekleyelim. Öncelikle yerel olarak değişken tanımlaması ile ilgili kodumuzu yazalım.

```
using System;
using System.Windows.Forms;
namespace global_local
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            int i = 0;
            i = i + 1;
            MessageBox.Show (i + " defa tıklandı");
        }
    }
}
```



Örneğimizi çalıştırdığımızda hep "**1 defa tıklandı**" şeklinde bir mesaj penceresi görüntüleyecektir.

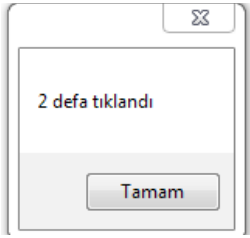
Örneğimizde butonun tıklanma olayı içerisinde yerel değişken tanımlaması yapılmıştır. Tanımlanan değişken yalnızca `button1` nesnesinin `Click` olayı içerisinde geçerlidir. Değişken tanımlandığında ilk değeri `0`'dır.

`i=i+1` komut satırı ile `i` değişkeninin değeri bir artarak `1` olmaktadır. Butona yeniden tıklandığında değişken yeniden tanımlanarak ilk değer ataması yapılır ve `0` değerini alır. Değişkenin değeri yeniden bir arttırılarak `1` yapılmaktadır. Bu süreç butona her tıklandığında tekrarlanmakta ve her defasında mesaj penceresi ile "**1 defa tıklandı**" mesajı görüntülenmektedir.

Değişkenimizi butona tıklanma olayının dışında form kodlarının içerisinde genel olarak tanımlayarak bu sorunu aşabiliriz.

```
using System;
using System.Windows.Forms;
namespace global_local
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
       int i = 0;
        private void button1_Click(object sender, EventArgs e)
        {
            i = i + 1;
            MessageBox.Show (i + " defa tıklandı");
        }
    }
}
```

Kodumuzu çalıştırdığımızda butona tıklandıkça tıklama sayısı mesaj penceresi içerisinde görüntülenir.



Tanımladığımız **i** değişkeni global olmakla beraber, yalnızca tanımlandığı form içerisinde geçerlidir.

Aynı örneğimizi "**i**" değişkenimizi **public static** olarak tanımlayarak düzenleyelim. Projemizin içerisine **Form2** adı ile yeni bir form daha ekleyelim. "**i**" değişkenini **Form2** içerisinde tanımlayarak **Form1** içerisinden çağıracağız.

Form2'nin kodu:

```
using System;
using System.Windows.Forms;

namespace global_local
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }

        public static int i;
        //Bütün proje içerisinde geçerli global değişken
    }
}
```

Görüldüğü gibi **i** değişkenini **Form2** içerisinde **public static** olarak tanımladık.

Form1'in kodu:

```
using System;
using System.Windows.Forms;

namespace global_local
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
        private void button1_Click(object sender, EventArgs e)
        {
            Form2.i = Form2.i + 1;
            MessageBox.Show(Form2.i + " defa tıklandı");
        }
    }
}
```

Değişkenimizi **Form2** içerisinde **public static** olarak tanımladığımız için **Form1** içerisinden, **Form2** içerisinde tanımlanmış olan **i** değişkenine, **Form2.i** şeklinde kullanarak erişebilmekteyiz. **Form1** içerisindeki butona her tıklandığında **Form2** değişkeni içerisinde tanımlanmış olan **i** değişkeninin değerini 1 arttırarak bu değişkenin içeriğini bir mesaj penceresi içerisinde görüntülüyoruz.

Değişkenleri tanımlarken değer aktarımı yapmadan tanımlayarak varsayılan değerleri ile kullanabileceğimizi, **değişkenler** konusunun başında belirtmiştik.

Değişkenlerin varsayılan değerleri aşağıdaki tabloda görüldüğü gibidir:

Değişken Türü	Varsayılan Değeri
bool	False
byte	0
char	'\0'
decimal	0.0M
double	0.0D
float	0.0F
int	0
long	0L
sbyte	0
short	0
uint	0
ulong	0
ushort	0
string	Null

Değişkenlerin tiplerinin varsayılan değerlerini **default** ifadesini kullanarak görüntüleyebiliriz. Örneğin:

```
static void Main(string[] args)
{
    Console.WriteLine(default(int));
    Console.WriteLine(default(bool));
    Console.ReadLine();
}
```

Şeklinde ekran çıktısı verecektir. **int** değişken tipinin varsayılan değeri **0**, **bool** değişken tipinin varsayılan değeri ise **False** olarak konsol ekranına yazdırılacaktır.

NOT

default ifadesini C#7.1 öncesi versiyonlara kadar

```
int sayi = default(int);
```

şeklinde kullanırken C# 7.1 versiyonundan itibaren

```
int sayi = default;
```

şeklinde de kullanabiliriz. **default** ifadesinin aktarılacağı değişkenin türü belli olduğu için artık **default** ifadesi ile birlikte değişken türünü yazmaya gerek kalmadan kullanabiliriz.

Genel değişkenlerde, değişkene değer aktarımı yapmadan varsayılan değeri ile kullanabiliriz. Ancak yerel değişkenlerde, önce değer aktarımı yapıp daha sonra kullanabiliriz.

Örneğin;

```
static void Main(string[] args)
{
    string ad;
    Console.WriteLine(ad);
    Console.ReadLine()
}
```

(local variable) string ad

Use of unassigned local variable 'ad'

Yukarıda görüldüğü gibi **ad** değişkenini **Main** metot içerisinde yerel değişken olarak tanımladık.

Ancak tanımlarken ilk değer aktarımı yapmadığımız gibi daha sonra da bir değer aktarımı yapmadık. Kodumuz bu nedenle **derleme hatası** verdi. Aynı değişkeni genel olarak tanımlayarak kodumuzu yeniden yazalım.

```
public static string ad;
static void Main(string[] args)
{
    Console.WriteLine(ad);
    Console.ReadLine();
}
```

Kodumuzda **ad** değişkenini genel olarak tanımladık ve ilk değer ataması yapmadık. Kod içerisinde de daha sonra herhangi bir değer ataması yapmadık.. Değişkenimizi **public static** olarak tanımladığımız için **Main** metodu içerisinden erişim sağlayabildik. Kodumuz ekranda herhangi bir çıktı üretmeden ve hata vermeden çalışacaktır.

Değer ve Referans Tipi Değişken Tanımlama

Değer tipleri, değişkenleri içerdikleri verinin bir kopyasını oluşturarak yalnızca kendisine gönderilen veriyi saklarlar. Diğer değişkenleri etkilemezler. **C#** içerisinde herhangi bir nitelendirici belirtilmeden tanımlanan parametreler **value** (*değer tipi*) olarak kabul edilirler.

Referans tipi değişkenler ise direkt olarak veriyi saklamak yerine verinin hafızada saklanacağı adresi saklarlar. Dolayısı ile aynı hafıza bölgesine adresleme yapan diğer referans değişkenlerini etkileyebilirler. **C#** içerisinde önüne **ref** ifadesi eklenerek tanımlanan parametreler, **reference** (referans tipi) olarak kabul edilirler. Örnek bir uygulama yapalım:

```
private void button1_Click(object sender, EventArgs e)
{
    string mesaj = "Ana Program tarafından oluşturulan mesaj";
    MessageBox.Show (mesaj);
    mesajyaz(mesaj);
    MessageBox.Show(mesaj);
}
```

```
private void mesajyaz(string mesaj)
{
    mesaj = "Metod tarafından oluşturulan mesaj";
 }
```

Örneğimizi çalıştırdığımızda her iki defasında da "**Ana Program tarafından oluşturulan mesaj**" mesajını görüntülemektedir. Çünkü metodumuzu **değer** tipinde tanımladığımız için, **mesaj** değişkeninin içeriğinde yapılan değişiklik ana programdaki **mesaj** değişkeninin içeriğini etkilememektedir.

Metot içerisinde tanımladığımız **mesaj** parametresini, referans türden olacak şekilde yeniden düzenleyelim.

```
private void button1_Click(object sender, EventArgs e)
{
    string mesaj = "Ana Program tarafından oluşturulan mesaj";
    MessageBox.Show (mesaj);
    mesajyaz(ref mesaj);
    MessageBox.Show(mesaj);
}
private void mesajyaz(ref string mesaj)
{
    mesaj = "Metod tarafından oluşturulan mesaj";
}
```

Örneğimizde görüldüğü gibi hem metot içerisinde değişkenimizi tanımlarken hem de **button1** içerisinden metodu çağırırken **ref** ifadesini değişkenimizin önüne ekliyoruz. Metot içerisinde tanımladığımız **mesaj** değişkenini **ref** ifadesi ile referans tipte tanımlayıp kodumuzu çalıştırdığımızda, önce "**Ana Program tarafından oluşturulan mesaj**" ifadesi, daha sonra ise "**Metod tarafından oluşturulan mesaj**" ifadesi görüntülenmektedir. Çünkü metodumuzu referans tipinde tanımladığımız için **mesaj** değişkeninin içeriğinde yapılan değişiklik, ana programdaki **mesaj** değişkeninin içeriğini de etkilemekte ve yeni içerik görüntülenmektedir.

C# 7.3 versiyonundan itibaren referans bir değişkene yapılmış olan bir atamayı yeniden yaparak başka bir değer ataması yapabiliriz.

```
int sayi1 = 5;
int sayi2 = 3;
ref int min = ref sayi2;
if (sayi1 < sayi2)
    min = ref sayi1;
Console.WriteLine(min);
Console.ReadLine();
```

Yukarıdaki örneğimizde **sayi1 sayi2**'den küçük olmadığı için **min** değişkenine atanmış olan **sayi2** değişkeninin değeri olan **3**, konsol ekranında görüntülenecektir. Değişken değerlerini değiştirip **sayi1=3, sayi2=5** şeklinde atama yaptığımızda ise **sayi1 sayi2**'den küçük olacağı için **min** değişkenine **sayi1**'in değeri aktarılacak ve konsol ekranında yine **3** değeri görüntülenecektir.

Boxing ve Unboxing

Boxing, değer türünden bir değişkeni referans türünden bir değişkene aktarmaktır. **Unboxing** ise bunun tam tersi, yani **referans** türünden bir değişkeni **değer** türünden bir değişkene aktarmaktır.

Boxing işlemi için aşağıdaki gibi bir örnek uygulama yapalım:

```
static void Main(string[] args)
{
   string mesaj;
   object boxing;

   mesaj = "İlk mesajımız";

   boxing = mesaj;

   Console.WriteLine(mesaj);
   Console.WriteLine(boxing);

   mesaj = "Yeni Mesajımız";

   Console.WriteLine(mesaj);
```

```
    Console.WriteLine(boxing);

    Console.ReadLine();
}
```

Kodumuzu çalıştırdığımızda;

İlk Mesajımız
İlk Mesajımız
Yeni Mesajımız
İlk Mesajımız

Şeklinde **çıktı** verdiğini görürüz.

Kodumuzu adım adım açıklayalım;

Kodumuzda görüldüğü gibi **boxing** değişkenini **object** tipinde tanımlayarak referans tipinde bir değişken elde ettik. **mesaj** değişkenini ise **string** türünde tanımlayarak **değer** tipinde bir değişken elde ettik.

```
mesaj = "İlk mesajımız";
```

Satırı ile **mesaj** değişkenine "**İlk mesajımız**" değeri aktarılmıştır. **Mesaj** değişkeni **değer** türünden bir değişken olduğu için "**İlk mesajımız**" değerinin bir kopyası hafızada saklanacaktır.

```
boxing=mesaj;
```

Satırı ile değer türünden olan **mesaj** değişkenini, referans türünden olan **boxing** değişkenine aktardık.

```
Console.WriteLine(mesaj);
```

Komut satırı ile **mesaj** değişkeninin içeriği olan "**İlk mesajımız**" yazısını ekrana yazdırdık.

```
Console.WriteLine(boxing);
```

Satırı ile **boxing** değişkenini **mesaj** değişkenine eşitlediğimiz için **boxing** değişkeninin içeriğini yazdırdığımızda da "**İlk mesajımız**" yazısı ekrana geldi.

```
mesaj = "Yeni Mesajımız" ;
```

Satırı ile **mesaj** değişkeninin içeriğini değiştirerek "**Yeni Mesajımız**" değerini aktardık.

```
Console.WriteLine(mesaj);
```

Satırında **mesaj** değişkeninin içeriğini değiştirdiğimiz için "**Yeni Mesajımız**" değerini ekrana yazmaktadır.

```
Console.WriteLine(boxing);
```

Satırında ise **mesaj** değişkeninin içeriğinde yapılan değişiklik **object** türünden olan **boxing** değişkenimize yansımamıştır. Çünkü **boxing** işlemi ile **boxing** değişkeni, **mesaj** değişkeninin hafızadaki kopyasını görüntülemektedir. **Mesaj** değişkeninin içeriğinde yapılan değişiklik ise **mesaj** değişkeninin hafızadaki gerçek değerini değiştirdiği için yapılan bu değişiklik **boxing** değişkeninin içeriğini etkilememektedir. Dolayısı ile ekranda **mesaj** değişkeninin ilk kopya değeri olan "**İlk mesajımız**" değeri görüntülenmektedir.

UnBoxing işlemi için aşağıdaki gibi bir örnek uygulama yapalım:

```
static void Main(string[] args)
{
string mesaj = null;
     object boxing = null;

      mesaj = "İlk mesajımız";

      boxing = mesaj;

      Console.WriteLine(mesaj);
      Console.WriteLine(boxing);

      mesaj = "Yeni Mesajımız";

      Console.WriteLine(mesaj);
      Console.WriteLine(boxing);

      string digermesaj = (string)boxing;

      Console.WriteLine(digermesaj);

      Console.ReadLine();
 }
```

Kodumuzu çalıştırdığımızda;

İlk Mesajımız
İlk Mesajımız
Yeni Mesajımız
İlk Mesajımız
İlk Mesajımız

Şeklinde **çıktı** verdiğini görürüz.

Kodumuzda yaptığımız değişiklikleri açıklayalım;

```
string digermesaj = (string)boxing;
```

Satırı ile **`digermesaj`** adlı bir değişken tanımlayarak **`boxing`** değişkeninin içeriğini ona aktarıyoruz. **`Boxing`** işleminin tersine olarak **`boxing`** değişkeninin hafızada belirttiği veri, **referans** türü olarak alınıp **`digermesaj`** değişkeninin içeriğine değer tipi olarak kopyalanır. Burada oluşacak tip uyuşmazlığını aşmak için de referans tipteki verimizin türünü, değer tipteki verimizin türüne, yani **`string`** veri tipine dönüştürüyoruz.

```
Console.WriteLine(digermesaj);
```

Satırı ile **`digermesaj`** değişkenimizin içeriğini görüntüleriz. **`digermesaj`** değişkeninin içeriğinde artık **`boxing`** değişkeninin içerdiği ilk mesaj değişkeninin hafızaya alınmış ilk kopyası olduğu için "**`İlk mesajımız`**" şeklinde **çıktı** üretecektir.

Değişken Tiplerini Öğrenmek

Değişkenlerin tiplerini öğrenmek için **GetType** ve bu metotla birlikte **TypeOf** metodu kullanılabilir. **GetType** metodu ile direkt olarak değişkenin tipi öğrenilebilecekken **TypeOf** metodu ile bir değişkenin belirtilen veri türüne ait olup olmadığı belirlenebilir.

1) GetType Metodu

Herhangi bir değişkenin tipini öğrenebilmek için **GetType** metodundan faydalanılır. **GetType** metodunun ürettiği değer, **Type** türünden olduğu için herhangi bir **string** değişkene aktarılamaz. **ToString( )** metodu kullanılarak **string** türüne dönüştürüldükten sonra kullanılabilir.

Konu ile ilgili örnek bir uygulama yapalım. Örneğimizde değişkenlerin tipini belirtmeden **var** türünden tanımlayarak değer ataması yapalım. Değişkenlerin içerdikleri değerlere göre **C#** tarafından **otomatik** olarak tip ataması yapılmaktadır. **GetType** metodu ile bu değişkenin tipini öğrenerek **ToString( )** metodu üretilen değeri **string** değere dönüştürüp mesaj penceresi içerisinde görüntüleyeceğiz.

Örneğimizin kodu aşağıdaki gibi olacak:

```
private void btnDegiskenTipi_Click(object sender, EventArgs e)
{
   var isim = "Sümeyye AKTAŞ";
   MessageBox.Show (isim.GetType().ToString() );
   var yas = 5;
   MessageBox.Show(yas.GetType().ToString());

   var ondalık_sayı = 12.4;
   MessageBox.Show(ondalık_sayı.GetType().ToString());

   var mantıksal_degisken = true;
   MessageBox.Show(mantıksal_degisken.GetType().ToString());
}
```

Kodumuzu çalıştırdığımızda mesaj penceresi içerisinde sıra ile;

System.String
System.Int32
System.Double
System.Boolean

İfadelerinin görüntülendiğini görürüz.

2) TypeOf Metodu

Herhangi bir değişken tipinin, bir değişken türüne ait olup olmadığını anlamak amacı ile **`TypeOf`** metodundan faydalanılır.

Konu ile ilgili bir örnek uygulama yapalım:

```
private void Form1_Load(object sender, EventArgs e)
{
    var ondalık_sayı = 12.4;
    if (ondalık_sayı.GetType() == typeof(byte))
    {
        MessageBox.Show ("Değişken türü byte");
    }
        else
    {
        MessageBox.Show("Değişken türü byte değil");
    }
}
```

Örneğimizde **`ondalık_sayı`** değişkeninin tipini belirtmeden **var** türünden tanımlayarak **`12.4`** ondalık değerini atadık. **`ondalık_sayı`** isimli değişkenin içeriğinin byte türü olup olmadığını **`typeof(byte)`** metodu ile kontrol etmekteyiz. Atanan değer ondalık sayı olduğundan dolayı değişkenin türü **`double`** olacaktır. Dolayısı ile kodumuzu çalıştırdığımızda "**`Değişken türü byte değil`**" sonucunu görüntüleyecektir.

Değişken Tipleri Arasında Dönüşüm Yapmak

.Net framework tip dönüştürme işlemlerinde, başlıca iki tür dönüşümü destekler. Bunlar, **kapalı dönüşüm** (*implicit conversion*) ve **açık dönüşüm** (*explicit conversion*) işlemleridir.

Implicit conversion (*kapalı dönüşüm*) işleminde, Net framework, değişken tipleri üzerinde işlem yaparken dönüşüm işlemlerini **otomatik** olarak yapmaktadır.

Dönüşüm işlemleri, aşağıdaki tablodaki veri türleri üzerinde **implicit conversion** (*kapalı dönüşüm*) biçiminde otomatik olarak gerçekleştirilir.

Dönüşen	Dönüştürülebilecek Değişken Tipi
sbyte	short, int, long, float, double, decimal
byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
char	ushort, int, uint, long, ulong, float, double, decimal
float	double
ulong	float, double, decimal



Ayrıca açık dönüşüm (explicit conversion) işlemlerinde kullanılan metotlarla da dönüşüm işlemi bu tablodaki biçimde gerçekleştirilmelidir. Aksi takdirde hata meydana gelecektir.

```
static void Main(string[] args)
{
   int sayi1 = 2;
   double sayi2 = 2.5;
   double sonuc=sayi1+sayi2 ;
   Console.WriteLine(sonuc);
   Console.ReadLine();
}
```

Örneğin yukarıdaki kod satırlarında **sayi1** değişkeni **integer** tipinde tanımlanmış olmasına rağmen, **double** türünden **sayi2** ile toplanmaya

kalkıldığında, **C#** tarafından otomatik olarak **`double`** veri türüne dönüştürüldüğünden dolayı, kodumuzu çalıştırdığımızda hata vermeyerek **`4,5`** değerini üretir.

```
static void Main(string[] args)
{
   byte a=1;
   string b=2.5 +"";
   Console.WriteLine(a + b);
   Console.ReadLine();
}
```

Sayısal değişkenleri **`string`** değişkenlere dönüştürmek için ise yukarıdaki örneğimizde görüldüğü gibi sayısal değerin yanına "" karakterlerini ekleyerek dönüşüm işlemi gerçekleştirilebilir. Örneğimizde **`b`** değişkenine değer aktarırken **`2.5`** değerinin yanına "" karakterlerini ekleyerek bu değerin **`string`** olarak kabul edilmesini sağladık. Kodumuzu çalıştırdığımızda **`a`** ve **`b`** değişkenlerini toplamayarak bu değişkenlerin toplamı olan **`3,5`** değeri yerine, **`a`** ve **`b`** değişkenlerini birleştirerek **`12,5`** değerini üretecektir.

Açık dönüşüm (*explicit conversion*) türünde ise dönüştürme işleminin hangi türe yapılacağı kullanıcı tarafından çeşitli metotlar ile gerçekleştirilir.

Örneğin:

```
static void Main(string[] args)
{
   int sayi1 = 2;
   double sayi2 = 2.5;
   double sonuc=sayi1+sayi2 ;
   Console.WriteLine(sonuc);
   Console.ReadLine();
}
```

Biçiminde **Implicit conversion** (*kapalı dönüşüm*) ile otomatik olarak gerçekleştirdiğimiz dönüşüm işlemini, **açık dönüşüm** (*explicit conversion*) ile gerçekleştirelim.

```
static void Main(string[] args)
{
   int sayi1 = 2;
   double sayi2 = 2.5;
   double sonuc = (double)sayi1 + sayi2;
   Console.WriteLine(sonuc);
   Console.ReadLine();
}
```

Örneğimiz yine bir önceki ile aynı değer olan **4,5** değerini üretecektir.

Açık dönüşüm işleminde, dönüşüm esnasında veri türleri arasında uyumsuzluk olduğu zaman **InvalidCastException** hatası oluşacaktır.

Örneğin:

```
int sayi = 123;
string yazi1 = sayi;
```

Şeklinde yazacağımız kod "**Cannot convert type 'int' to 'string'**" hatasını verecektir. Dönüşüm esnasında hangi veri tiplerinin, hangilerine dönüştürülebileceğini yukarıdaki tabloda belirtmiştik.

```
int sayi = 123;
object yazi1 = sayi;
```

Şeklinde yazacağımız kod ise **object** veri türünün genel bir veri türü olması nedeniyle hata vermeden çalışacaktır.

Dönüşüm esnasında oluşacak bir hata durumunda, geriye hata dönmemesi için **is** ve **as** operatörlerini kullanabiliriz.

is operatörü bir değişkenin tipini belirtilen bir değişken tipi ile karşılaştırarak değişken tipinin, karşılaştırılan veri tipi ile aynı olup olmadığını geriye **boolean** bir değer döndürerek kontrol etmek amacıyla kullanılır.

```
static void Main(string[] args)
{
   object a = 21.5;
```

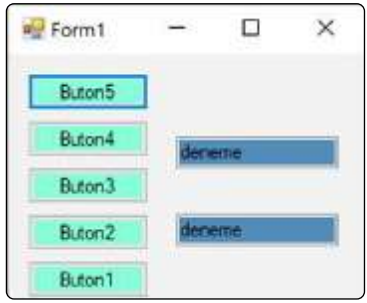
```
    if (a is int)
        Console.WriteLine("Değer int türündendir.");
    else
        Console.WriteLine("Değer int türünden değildir.");
    Console.ReadLine();
}
```

object türünden tanımladığımız **a** isimli değişkene **21.5** değerini aktardığımız için **double** veri türünü almaktadır. Dolayısıyla **is** operatörü ile karşılaştırma yaptığımızda geriye **false** değeri döneceği için konsol ekranında "**Değer int türünden değildir.**" mesajı görüntülenecektir.

as operatörü dönüşüm esnasında geriye hata döndürmek yerine, boş değer (*null*) döndürür. Referans tipi değişkenler üzerinde kullanılır.

Örnek olarak bir form üzerindeki kontrollere erişerek onların özelliklerini değiştirecek bir uygulamayı **as** operatörünü kullanarak yapalım. Form üzerine beş adet buton ve iki adet **textBox** nesnesi yerleştirelim.

```
private void Form1_Load(object sender, EventArgs e)
{
    int sayi = 0;
    foreach (Control nesne in this.Controls) {
        Button btn = nesne as Button;
        if (btn == null) {
            nesne.BackColor = Color.SteelBlue;
            nesne.Text = "deneme";
        }
        else {
            nesne.BackColor = Color.Aquamarine;
            sayi = sayi + 1;
            nesne.Text = "Buton" + sayi;
        }
    }
}
```



Kodumuzu çalıştırdığımızda form üzerindeki butonların rengi değiştirilerek üzerlerine **`Buton1..Buton5`** şeklinde yazılar yazılacak, **`textBox`** nesnelerinin de rengi değiştirilerek üzerlerine **`deneme`** şeklinde yazı yazılacak.

Kodumuzda, **`as`** metodu ile form üzerindeki kontrolleri **`button`** nesnesine dönüştürmeye çalıştık. Eğer geriye **`null`** değer dönmüyorsa o nesnenin **`button`** nesnesi olduğuna karar vererek **`button`** dizaynı ile ilgili kodları çalıştırıyoruz. Eğer dönüşüm sırasında dönüştürülmek istenen nesne **`button`** nesnesi değilse geriye **`null`** değeri döndürecektir. Bu durumda **`textBox`** dizaynı ile ilgili kodları çalıştırıyoruz.

Açık dönüşüm (*explicit conversion*) işlemleri için ayrıca aşağıdaki yöntemleri kullanabiliriz.

1) Parse Metodu

`String` türündeki bir veriyi, belirtilen bir veri türüne dönüştürmek amacı ile kullanılır. **`String`** ve **`object`** veri türleri haricindeki veri türleri ile birlikte kullanılabilmektedir. **`System`** sınıfına ait olan bir metoddur.

Kullanımı şu şekildedir:

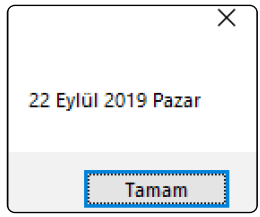
```
DeğişkenTürü.Parse(Dönüştürülecek Değer)
```

`String` tipteki bir veriyi `Parse` metodu ile **tarih** veri türüne değiştiren bir örnek yapalım.

```
private void button1_Click(object sender, EventArgs e)
{
    string deger = "22/09/2019";
    MessageBox.Show((System.DateTime.Parse(deger)).
    ToLongDateString());
}
```

Kodumuzda `string` türde olan "`22/09/2019`" ifadesi `DateTime.Parse` metodu ile `DateTime` türüne dönüştürülmektedir.

Kodumuzu çalıştırdığımızda şekildeki gibi sonuç verecektir:



Dönüşüm esnasında;

- Değişken `null` değere sahipse `ArgumentNullException` hatası **geriye** döndürülür.
- Değişken farklı bir değişken türüne sahipse `FormatException` hatası **geriye** döndürülür.
- Değişkenin değeri `MaxValue` değerinin üstünde ya da `MinValue` değerinin altında ise `OverflowException` hatası `geriye` döndürülür.

NOT `MaxValue`, değişkenin alabileceği en üst değer iken `MinValue` ise değişkenin alabileceği en alt değerdir.

Bu değerler değişken tiplerine göre şu şekildedir:

Değişken Türü	Minimum	Maximum
byte	0	255
sbyte	-128	127
short	-32768	32767
ushort	0	65535
int	-2147483648	2147483647
uint	0	4294967295
long	-9223372036854775808	9223372036854775807
uint64	0	18446744073709551615
float	-3.402823E+38	3.402823E+38
double	-1.79769313486232E+308	1.79769313486232E+308
decimal	-79228162514264337593543950335	79228162514264337593543950335

Örneğin:

```
static void Main(string[] args)
{
    string sayi1 = "123";
    string sayi2 = null;
    string sayi3 = "123.45";
    string sayi4 = "12345678910111213141516171819 20";
    Console.WriteLine(Int32.Parse(sayi1));
    Console.WriteLine(Int32.Parse(sayi2));
    Console.WriteLine(Int32.Parse(sayi3));
    Console.WriteLine(Int32.Parse(sayi4));
    Console.ReadLine() ;
}
```

Yukarıdaki kodumuzu çalıştırdığımızda:

```
Console.WriteLine(Int32.Parse(sayi1));
```

Satırı için **123** sonucunu üretecektir.

```
Console.WriteLine(Int32.Parse(sayi2));
```

Satırında **sayi2** değişkeninin içeriği **null** olduğundan dolayı **ArgumentNullException** hatasını geriye döndürecektir.

```
Console.WriteLine(Int32.Parse(sayi3));
```

Satırında **sayi3** değeri ondalıklı değer olmasından dolayı **int32** değişken türünden farklıdır. Dolayısı ile **FormatException** hatasını geriye döndürecektir.

```
Console.WriteLine(Int32.Parse(sayi4));
```

Satırında **sayi4** değeri **int32** değişken türünün sınırlarını aştığından dolayı, **OverflowException** hatasını **geriye** döndürecektir.

2) TryParse Metodu

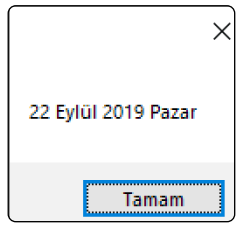
Parse metodunda olduğu gibi **string** türündeki bir veriyi belirtilen bir veri türüne dönüştürmek amacı ile kullanılır. **Parse** metodundan farklı olarak dönüşüm esnasında değişken **null** değere sahipse ve farklı bir değişken türüne sahipse **MaxValue** değerinin üstünde veya **MinValue** değerinin altında ise yani oluşabilecek bütün hata durumlarında, geriye herhangi bir hata döndürmeyip **sıfır** değerini döndürür. Eğer herhangi bir hata durumu oluşmazsa başında **out** ifadesi bulunan parametreye **sonuç** değeri aktarılır. **System** sınıfına ait olan bir metoddur.

Kullanımı şu şekildedir:

```
DeğişkenTürü.TryParse(Dönüştürülecek Değer, out Sonuç Değer)
```

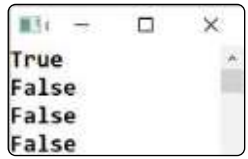
String tipteki bir veriyi **tarih** veri türüne değiştiren örneği **TryParse** metodu ile yapalım:

```
private void button1_Click(object sender, EventArgs e)
{
    string deger = "22/09/2019";
    DateTime tarih;
    if (System.DateTime.TryParse(deger, out tarih))
    {
          MessageBox.Show(tarih.ToLongDateString());
    }
}
```



Örneğimizde eğer, **`TryParse`** metodu ile dönüşüm işlemini gerçekleştirdiğimizde işlem başarılı biçimde gerçekleştirilirse dönüştürülen değer **`tarih`** değişkenine aktarılmakta ve geriye **`true`** değeri döndürülmektedir. Sonuç **`true`** olduğunda ise **`tarih`** değişkeninin içeriği **mesaj penceresi** içerisinde görüntülenmektedir.

```
static void Main(string[] args)
{
    string sayi1 = "123";
    string sayi2 = null;
    string sayi3 = "123.45";
    string sayi4 = "12345678910111213141516171819 20";
    int s1;
    int s2;
    int s3;
    int s4;
    Console.WriteLine(Int32.TryParse(sayi1,out s1));
    Console.WriteLine(Int32.TryParse(sayi2,out s2));
    Console.WriteLine(Int32.TryParse(sayi3,out s3));
    Console.WriteLine(Int32.TryParse(sayi4,out s4));
    Console.ReadLine();
}
```



Kodumuzu çalıştırdığımızda oluşabilecek tüm hata durumlarında geriye hata döndürülmeyip **`false`** değerinin döndüğünü görürüz. Bu da program akışının kesilmeden çalışmasını netice vermektedir.

Decleration Expressions: **`TryParse`** metodunu kullanırken ikinci parametre olan işlem sonucunun aktarıldığı değişkeni her defasında:

```
int s1;
Console.WriteLine(Int32.TryParse(sayi1,out s1));
```

Satırlarında olduğu gibi ayrı ayrı tanımlamak zorunda kalıyorduk. **C# 6** sürümünden itibaren gelen Decleration Expressions ile **`TryParse`** metodunu artık:

```
Console.WriteLine(Int32.TryParse(sayi1,out int s1));
```

Biçiminde, işlem sonucu parametresini yazarken aynı yerde değişken olarak tanımlayarak kullanabilmekteyiz. Buna göre **`Int32.TryParse`** örneğindeki kodumuz aşağıdaki gibi olacak:

```
static void Main(string[] args)
{
    string sayi1 = "123";
    string sayi2 = null;
    string sayi3 = "123.45";
    string sayi4 = "12345678910111213141516171819 20";
    Console.WriteLine(Int32.TryParse(sayi1,out int s1));
    Console.WriteLine(Int32.TryParse(sayi2,out int s2));
    Console.WriteLine(Int32.TryParse(sayi3,out int s3));
    Console.WriteLine(Int32.TryParse(sayi4,out int s4));
    Console.ReadLine();
}
```

C# 7 versiyonundan itibaren **`(_)`** karakterini kullanarak dönüş parametresi kullanmak istemediğimiz durumlarda, **`out`** parametresi ile yapılan atama işlemini **iptal** edebiliyoruz. Bu duruma **discard** adı verilmektedir. Bu şekilde **`out`** parametresi için bir değişken tanımlanmaz ve hafızada yer ayrılmaz.

Örneğin, yalnızca girilen bir tarihin geçerli olup olmadığını kontrol edecek bir kod yazalım:

```
static void Main(string[] args)
{
    Console.WriteLine("Bir tarih giriniz");
    if (DateTime.TryParse(Console.ReadLine(), out _))
        Console.WriteLine("Geçerli tarih");
    else
        Console.WriteLine("Geçersiz tarih");
    Console.ReadLine();
}
```

Kodumuzda yalnızca **`TryParse()`** metodunun sonucuna göre işlem yapıyoruz. Dönen değer **`True`** ise tarih **geçerli**, **`False`** ise **geçersiz**dir. **`out`** parametresini kullanmadığımız için girilen tarih üzerinde bir işlem gerçekleştirmiyoruz.

3) Convert Metodu

Değişken tiplerini dönüştürmek amacıyla kullanılan, **`System`** sınıfına ait bir metot olan **`Convert`** metodu, şu alt metotlara sahiptir;

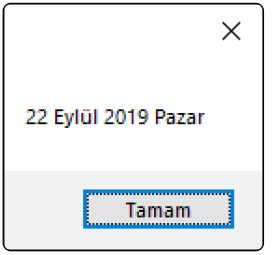
`Convert.ToBase64String()`	Verilen bir string'i Base64 şifreleme tipine dönüştürür.
`Convert.FromBase64String()`	Base64 şifreleme tipinden string tipine dönüştürme yapar.
`Convert.ToBoolean()`	Booelan tipine dönüştürme yapar.
`Convert.ToByte()`	Byte tipine dönüştürme yapar.
`Convert.ToChar()`	Char tipine dönüştürme yapar.
`Convert.ToDateTime()`	Tarih ve zaman tiplerine dönüştürme yapar.
`Convert.ToDecimal()`	Decimal tipine dönüştürme yapar.
`Convert.ToDouble()`	Double tipine dönüştürme yapar.
`Convert.ToInt16()`	Int16 tipine dönüştürme yapar.
`Convert.ToInt32()`	Int32 tipine dönüştürme yapar.
`Convert.ToInt64()`	Int64 tipine dönüştürme yapar.
`Convert.ToSByte()`	Sbyte tipine dönüştürme yapar.
`Convert.ToSingle()`	Single tipine dönüştürme yapar.
`Convert.ToString()`	String tipine dönüştürme yapar.
`Convert.ToUInt16()`	UInt16 tipine dönüştürme yapar.
`Convert.ToUInt32()`	UInt32 tipine dönüştürme yapar.
`Convert.ToUInt64()`	UInt64 tipine dönüştürme yapar.

String tipteki bir veriyi **Parse** ve **TryParse** metotları ile tarih veri türüne dönüştürme işlemini yapmıştık. Şimdi de **Convert.ToDateTime( )** metodu ile yapalım.

```
private void button1_Click(object sender, EventArgs e)
{
    string deger = "22/09/2014";
    MessageBox.Show(Convert.ToDateTime(deger).ToLongDateString());
}
```

Kodumuzda **string** türde olan "**22/09/2014**" ifadesi **Convert.ToDateTime** metodu ile **DateTime** türüne dönüştürülmektedir.

Kodumuzu çalıştırdığımızda görüldüğü gibi yine aynı sonucu verecektir.



Dönüşüm esnasında;

Değişken **null** değere sahipse **0** değeri geriye döndürülür.

Değişken farklı bir değişken türüne sahipse **FormatException** hatası geriye döndürülür.

Değişkenin değeri **MaxValue** değerinin üstünde ya da **MinValue** değerinin altında ise **OverflowException** hatası geriye döndürülür.

Örneğin:

```
static void Main(string[] args)
{
    string sayi1 = "123";
    string sayi2 = null;
    string sayi3 = "123.45";
    string sayi4 = "12345678910111213141516171819 20";
    Console.WriteLine(Convert.ToInt32(sayi1));
    Console.WriteLine(Convert.ToInt32(sayi2));
    Console.WriteLine(Convert.ToInt32(sayi3));
    Console.WriteLine(Convert.ToInt32(sayi4));
    Console.ReadLine();
}
```

Yukarıdaki kodumuzu çalıştırdığımızda;

```
Console.WriteLine(Convert.ToInt32(sayi1));
```

Satırı için **123** sonucunu üretecektir.

```
Console.WriteLine(Convert.ToInt32(sayi2));
```

Satırında **sayi2** değişkeninin içeriği **null** olduğundan dolayı **0** değerini geriye döndürecektir.

```
Console.WriteLine(Convert.ToInt32(sayi3));
```

Satırında **sayi3** değeri ondalıklı değer olmasından dolayı **int32** değişken türünden farklıdır. Dolayısı ile **FormatException** hatasını geriye döndürecektir.

```
Console.WriteLine(Convert.ToInt32(sayi4));
```

Satırında **sayi4** değeri **int32** değişken türünün sınırlarını aştığından dolayı **OverflowException** hatasını geriye döndürecektir.

Kullanıcı Tanımlı Veri Tipleri

1) Struct (Yapı) Oluşturmak

`struct` (*structure [yapı*]) veri türü, farklı tiplerden değişken tiplerini, özellikleri ve metotları bir çatı altında toplayarak gruplandırmak amacı ile kullanılan veri türü yapısıdır. Değer türünden oldukları için küçük miktarda verilerin saklanmasında daha yüksek performans sağlarlar. **`struct`** içerisindeki her eleman tek tek tanımlanmalıdır. Yapının boyutu, içerisinde tanımlanmış olan elemanların toplam boyutu kadardır. Tanımlaması aşağıdaki gibi yapılır:

```
struct structAdı
{
    public değişken tipi değişkenadı;
}
```

Yukarıda da görüldüğü gibi **`struct`** içerisindeki elemanlar **`public`** **erişim belirteci** ile tanımlanırlar.

`struct` içerisindeki elemanlara:

```
struct nesne = new struct();
```

şeklinde **`struct`** türünden bir nesne tanımlayarak erişebiliriz.

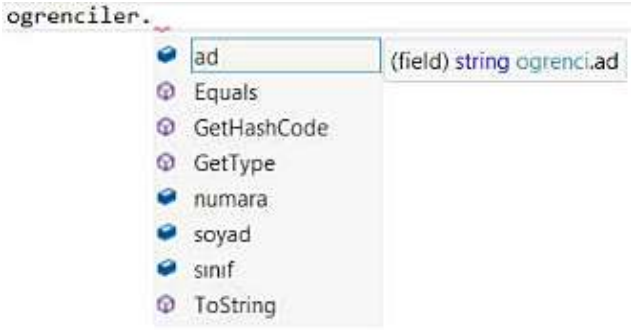
Örnek olarak bir **`struct`** oluşturup içerisindeki elemanları **`listBox`** içerisinde listeleyen bir uygulama yapalım. Öncelikle **`ogrenci`** adında bir **`struct`** oluşturup içerisinde elemanlar tanımlayalım:

```
struct ogrenci
{
    public string ad;
    public string soyad;
    public int numara;
    public string sınıf;
}
```

Yeni bir nesne tanımlayarak nesnenin tipini oluşturduğumuz **ogrenci** adlı **struct** tipinden verelim.

```
ogrenci ogrenciler = new ogrenci();
```

Daha sonra formun yüklenme olayı olan **Load** olayı içerisinde sıra ile **struct** içerisindeki elemanlara değer aktaralım.



Değer aktarmak üzere **struct** ismini yazdığımızda, **ogrenciler** adlı **struct** türünden nesnemizin elemanlarının listelendiğini görebiliriz.

Böylece:

```
ogrenciler.ad = "Talha Osman";
```

Şeklinde **struct** içerisindeki elemanlara erişip değer aktarabiliriz.

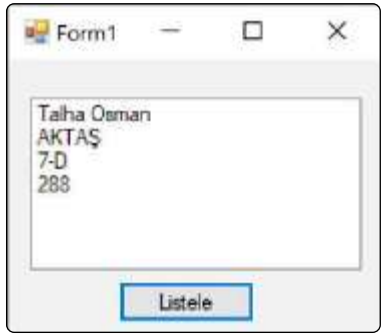
Daha sonra form üzerine **btn_Listele** isimli bir adet buton ekleyerek **struct** içerisindeki elemanların değerlerini:

```
listBox1.Items.Add(ogrenciler.ad);
```

Şeklinde **listBox**'ın **Items.Add** metodunu kullanarak **listBox** içerisine ekliyoruz.

Sonuç olarak kodumuz aşağıdaki gibi olacak:

```
using System;
using System.Windows.Forms;
namespace struct_to_listbox
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        struct ogrenci
        {
            public string ad;
            public string soyad;
            public int numara;
            public string sınıf;
        }
        ogrenci ogrenciler = new ogrenci();
        private void btn_Listele_Click(object sender, EventArgs e)
        {
            listBox1.Items.Add(ogrenciler.ad);
            listBox1.Items.Add(ogrenciler.soyad);
            listBox1.Items.Add(ogrenciler.sınıf);
            listBox1.Items.Add(ogrenciler.numara);
        }
        private void Form1_Load(object sender, EventArgs e)
       {
      ogrenciler.ad = "Talha Osman";
      ogrenciler.soyad = "AKTAŞ";
      ogrenciler.numara = 288;
      ogrenciler.sınıf = "7-D";
  }
    }
}
```



Kodumuzu çalıştırdığımızda struct içerisine aktardığımız değerlerin **`listBox`** içerisinde listelendiğini görebiliriz.

C# 7.2 versiyonundan itibaren bir **`struct`**'ı **read only** (*yalnız okunabilir*) olarak tanımlayarak, **`struct`** içerisindeki değişkenlerin yalnızca okunabilir olup güvenli bir şekilde içeriğinin değiştirilmeden kullanılmasını sağlayabiliriz. Örneğin yukarıdaki **`ogrenci`** adlı **`struct`**'ı yalnız okunabilir olacak şekilde aşağıdaki gibi düzenleyelim:

```
readonly struct ogrenci
{
  public string ad { get; }
  public string soyad { get; }
  public int numara { get; }
  public string sınıf { get; }
}
```

`struct` içerisindeki elemanlara erişebilmek için yeni bir nesne tanımlayarak nesnenin tipini oluşturduğumuz **`ogrenci`** adlı **`struct`** tipinden verelim.

```
ogrenci ogrenciler = new ogrenci();
```

`struct` içerisindeki elemanlara:

```
ogrenciler.ad = "Talha";
```

satırında olduğu gibi değer aktarmaya kalktığımızda "**`ad`**" elemanı yalnız okunabilir olduğundan dolayı **hata** durumu oluşacaktır.

Eğer **struct** içerisindeki elemanlara değer ataması yapmak istiyorsak bunu yalnızca **struct**'ın **constructor** metodu üzerinden gerçekleştirebiliriz. **Constructor** (*yapıcı*) metotlar, sınıflardan nesne türetilirken **new** ifadesi ile çalıştırılan ve sınıf ile aynı ismi taşıyan metotlardır. Bu konu ile ilgili ayrıntılı bilgi için kitabımızın **21. bölümü** olan "**Sınıflar**" bölümüne başvurabilirsiniz. Kodumuzu buna göre yeniden düzenleyelim.

```
readonly struct ogrenci
{
  public string ad { get; }
  public string soyad { get; }
  public int numara { get; }
  public string sınıf { get; }
  public ogrenci(string Ad, string Soyad, int Numara, string Sınıf)
  {
    this.ad = Ad;
    this.soyad = Soyad;
    this.numara = Numara;
    this.sınıf = Sınıf;
  }
}
```

Artık **struct** içerisindeki elemanlara aşağıdaki gibi **constructor** metot üzerinden değer aktarabiliriz.

```
ogrenci ogrenciler = new ogrenci();
ogrenciler = new ogrenci("Talha Osman","Aktas",288,"7-D");
```

Ayrıca **C# 7.2** versiyonundan itibaren bir **struct**'ı referans türde de tanımlayabiliriz. Normal kullanımda **struct**'ın bir değer tipi olması nedeniyle bir kopyası alınıp kullanılmakta ve kopyası üzerinde yapılan değişiklikler asıl **struct** üzerinde bir değişikliğe neden olmamaktadır. Ancak **referans** türden tanımlama yapıldığında, kopya **struct** üzerinde yapılan değişiklikler asıl **struct** üzerinde değişikliğe neden olmayacaktır.

Referans türden **`struct`** tanımlamanın bazı kısıtlamaları vardır. Bunlar;

- Lambda ya da asenkron bir metoda parametre olarak tanımlanamaz.
- **`Boxing`** işlemine tabi tutulamaz.
- Generic type parametre ya da dizi olarak tanımlanamaz.
- struct içerisindeki elemanlara erişebilmek için daha önce yaptığımız gibi yeni bir nesne tanımlayarak nesnenin tipini oluşturduğumuz **`ogrenci`** adlı **`struct`** tipinden veremiyoruz. Bunun yerine local bir değişken tanımlaması yapmamız gerekiyor.

  Örneğin normal **`struct`** tanımlamasında struct içerisindeki elemanlara erişmek için:

```
ogrenci ogrenciler = new ogrenci();
```

Satırında olduğu gibi yeni bir nesne tanımlayarak nesnenin tipini, oluşturduğumuz **`ogrenci`** adlı **`struct`** tipinden veriyorduk.

Referans türden **`struct`** için ise:

```
var ogrenciler = new ogrenci();
```

Şeklinde **local** (*yerel*) bir değişken tanımlaması yapmamız gerekmektedir.

Buna göre kodumuzun son hâli aşağıdaki gibi olacak:

```
ref readonly struct ogrenci
{
  public string ad { get; set; }
  public string soyad { get; set; }
  public int numara { get; set; }
  public string sınıf { get; set; }
  public ogrenci(string Ad, string Soyad, int Numara, string Sınıf)
  {
    this.ad = Ad;
    this.soyad = Soyad;
    this.numara = Numara;
    this.sınıf = Sınıf;
  }
}
```

```
private void btnListele_Click(object sender, EventArgs e)
{
  var ogrenciler = new ogrenci();
  ogrenciler.ad = "Talha Osman";
  ogrenciler.soyad = "Aktaş";
  ogrenciler.numara = 288;
  ogrenciler.sınıf = "7-D";

  listBox1.Items.Add(ogrenciler.ad);
  listBox1.Items.Add(ogrenciler.soyad);
  listBox1.Items.Add(ogrenciler.sınıf);
  listBox1.Items.Add(ogrenciler.numara);
}
```

Kodumuzda **struct**'ı referans olarak tanımlamak için önüne **ref** ifadesini eklerken **struct** içerisindeki elemanlara değer aktarabilmek için **get** özelliği ile beraber **set** özelliğini de ekledik.

struct'ı bir diziye aktararak diziler üzerinde uyguladığımız işlemleri **struct**'a da uygulayabiliriz.

```
struct[] nesne = new struct[eleman sayısı];
```

Şeklinde **struct**'ı dizi olarak tanımlayıp kullanabiliriz.

```
using System;
using System.Windows.Forms;
using Microsoft.VisualBasic;
namespace struct_To_Array
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        struct ogrenci
        {
            public string ad;
            public string soyad;
```

```
            public int numara;
            public string sınıf;
        }
         ogrenci[] ogrenciler = new ogrenci[3];
         byte i;

        private void btn_Listele_Click(object sender, EventArgs e)
        {
            for (i = 0; i <= ogrenciler.Length   - 1; i++)
            {
                listBox1.Items.Add(ogrenciler[i].ad);
                listBox1.Items.Add(ogrenciler[i].soyad);
                listBox1.Items.Add(ogrenciler[i].sınıf);
                listBox1.Items.Add(ogrenciler[i].numara);
            }
        }

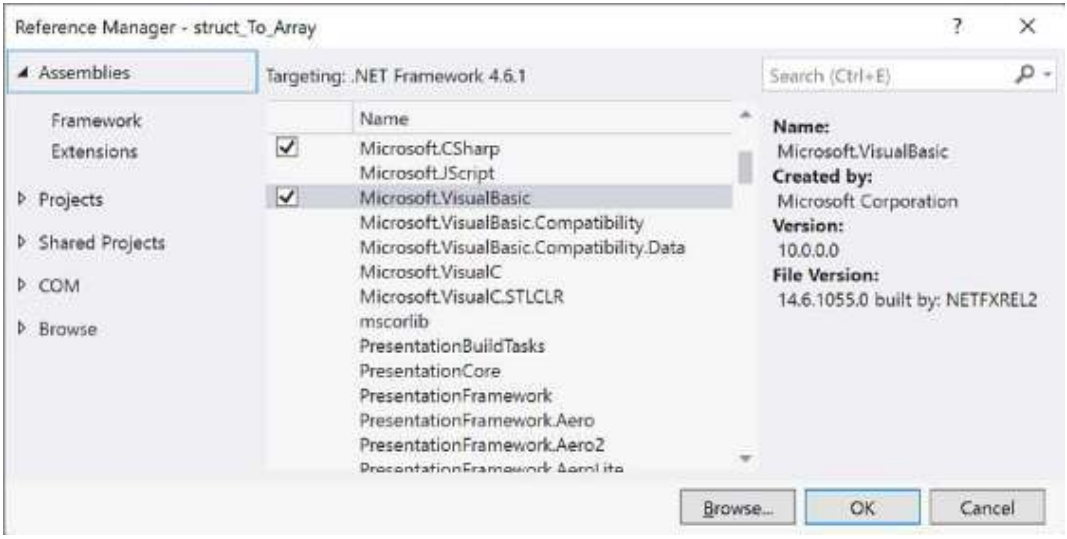
        private void btn_Ekle_Click(object sender, EventArgs e)
        {
            for (i = 0; i <= 2; i++)
            {
                ogrenciler[i].ad = Interaction.InputBox
(i + ". öğrencinin adını giriniz");
                ogrenciler[i].soyad = Interaction.InputBox
(i + ". öğrencinin soyadını giriniz");
                ogrenciler[i].sınıf = Interaction.InputBox
(i + ". öğrencinin sınıfını giriniz");
                ogrenciler[i].numara = Convert.ToInt32
(Interaction.InputBox(i +
". öğrencinin numarasını giriniz"));
            }
        }
    }
}
```

Kodumuz içerisinde öncelikle **ogrenci** adlı **struct**'ı oluşturarak içerisinde **ad**, **soyad**, **numara** ve **sınıf** elemanlarını tanımladık. Daha sonra:

```
ogrenci[] ogrenciler = new ogrenci[3];
```

satırında **öğrenci** adlı diziyi 3 elemanlı olarak **öğrenciler** adlı `struct` türünden tanımladık.

`btn_Ekle` içerisinde `for` döngüsü ile öğrenciler dizisinin elemanlarına `InputBox` yardımı ile kullanıcı tarafından değer girişi yapılmaktadır. **C#** içerisinde `InputBox` olmadığı için **Visual Basic .NET** içerisinde bulunan `InputBox`'ı kullanacağız. Bunun için öncelikle **Project** / **Add Reference** seçeneği ile ya da Solution Explorer içerisinde farenin sağ tuşu ile **solution** ismine tıklayıp **Add Reference** seçeneğinde **Reference Manager** penceresini açıyoruz. Karşımıza gelen pencerede Framework sekmesinden **Microsoft.VisualBasic** sınıfını seçip işaretleyerek **OK** butonuna tıklayıp projemize referans olarak ekliyoruz.



Kodumuzun en üstüne:

```
using Microsoft.VisualBasic;
```

Satırını ekleyerek referansı kodumuza dâhil ediyoruz. Artık `InputBox`'ı kodumuzda kullanabiliriz.

`btn_Listele` içerisinde yine `for` döngüsü ile `listBox`'ın `Items.Add` metodunu kullanarak öğrenciler dizisi içerisindeki değerleri `listBox` içerisine ekliyoruz. `for` döngüsünü kurarken kullandığımız `ogrenciler.Length` ifadesi dizinin eleman sayısını bulmak için kullanılmaktadır. `-1` yazmamızın nedeni dizilerin index numarasının 1'den değil de **0**'dan başlamasıdır.



Kodumuzu çalıştırıp diziye **Ekle** butonuna basarak dizimize üç adet eleman ekleyerek **Listele** butonuna basalım ve diziye eklediğimiz elemanları `listBox` içerisine aktaralım.

`Ogrenci` adlı yapımızı `Ogrenciler` adlı dizi değişkenine aktarmamız nedeni ile artık yapımızda diziye ait özellik ve metotları kullanabilmekteyiz. Örneğimizde dizi elemanlarına index numaralarını kullanarak erişmemiz ya da dizinin eleman sayısını bulmakta kullandığımız `Length` metodunda olduğu gibi.

STRUCT İÇERİSİNDE STRUCT TANIMLAMAK

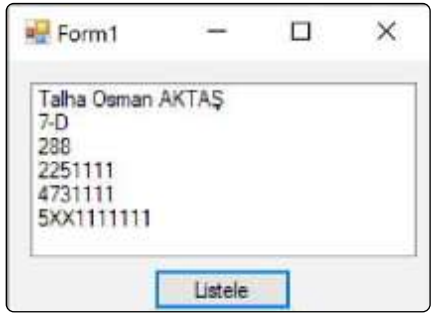
Bir `struct` içerisinde bir ya da birden fazla `struct` tanımlamak mümkündür. Aşağıda görüldüğü gibi tanımlama yapabiliriz.

```
struct structAdı
{
    public değişken türü değişken adı;
    public struct structAdı
    {
          public değişken türü değişken adı;
    }
}
```

İçeride tanımladığımız **struct**'a erişebilmemiz için **public** olarak tanımlamamız gerekmektedir.

```
using System;
using System.Windows.Forms;
namespace Structure_İcinde_Structure
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        struct ogrenci
        {
            public string adsoyad;
            public int numara;
            public string sınıf;
            public struct veli_bilgi
            {
                public string adı;
                public string ev_telefonu;
                public string is_telefonu;
                public string cep_telefonu;
            }
        }
        ogrenci ogrenciler = new ogrenci();
        ogrenci.veli_bilgi veli = new ogrenci.veli_bilgi();
        private void Form1_Load(object sender, EventArgs e)
        {
            ogrenciler.adsoyad = "Talha Osman AKTAŞ";
            ogrenciler.numara = 288;
            ogrenciler.sınıf = "7-D";
            veli.adı = "Volkan AKTAŞ";
            veli.ev_telefonu = "2251111";
            veli.is_telefonu = "4731111";
            veli.cep_telefonu = "5XX1111111";
        }
        private void btnListele_Click(object sender, EventArgs e)
        {
```

```
            listBox1.Items.Add(ogrenciler.adsoyad);
            listBox1.Items.Add(ogrenciler.sınıf);
            listBox1.Items.Add(ogrenciler.numara);
            listBox1.Items.Add(veli.ev_telefonu);
            listBox1.Items.Add(veli.is_telefonu);
            listBox1.Items.Add(veli.cep_telefonu);
        }
    }
}
```



Örneğimizde **ogrenci** adlı yapı içerisinde **veli_bilgi** adlı başka bir yapı tanımladık. Oluşturduğumuz **veli_bilgi** adlı yapıyı **ogrenci.veli_bilgi veli = new ogrenci.veli_bilgi()** şeklindeki tanımlama ile **veli** adlı nesneye aktardık. Daha sonra **veli.adı**, **veli.ev_telefonu** şeklinde bu yapının elemanlarına erişim sağladık.

struct içerisinde değişkenlerin yanında özellikler, metotlar da tanımlanabilir.

Örnek olarak üçgenin çevresini bulan bir uygulama yapalım.

```
class Program
{
    struct ucgen
    {
        private int r;
        public int yaricap
        {
            get
            {
                return r;
```

```
            }
            set
            {
                r = value;
            }
        }
    }
    static void Main(string[] args)
    {
        ucgen ucgen1 = new ucgen();
        Console.WriteLine("Üçgenin yarı çapını giriniz=");
        ucgen1.yaricap = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Üçgenin çevresi= " +
(2 * 3.14 * ucgen1.yaricap));
        Console.ReadLine();
    }
}
```

Kodumuzda **yaricap** isimli bir özellik (*property*) oluşturduk. Oluşturduğumuz **property** içerisinde **set** ifadesi ile değer gönderme, **get** ifadesi ile de değer okuma işlemi yapabiliriz. Kodumuzda **r** yani yarıçap değeri üzerinde işlem yapıyoruz.

```
ucgen ucgen1 = new ucgen();
```

Satırında **ucgen1** adlı bir nesneyi **ucgen** isimli **struct** türünden tanımlıyoruz.

```
ucgen1.yaricap = Convert.ToInt32(Console.ReadLine());
```

Üçgenin yarıçapını klavyeden **ReadLine** metodu ile okuyarak **ucgen1** nesnesinin **yaricap** özelliğine aktarıyoruz.Daha sonra çevreyi hesaplatıp konsol ekranına yazdırıyoruz.

2) Enum Oluşturmak

Program içerisinde kullanılan sabitlerin anlamlandırılması amacı ile bu sabitlere isimler vererek bir grup altında toplayabiliriz. Bu şekilde oluşturulan gruplara **enum** (*enumeration [numaralandırma]*) adı verilir.

```
enum enumAdı
{
    Sabit1,
    Sabit2,
    …
    Sabit n
}
```

Şeklinde tanımlanırlar.

Örneğin **Gunler** adında bir **enum** tanımlayalım:

```
enum Gunler
{
   Pazartesi,
   Salı,
   Çarşamba,
   Perşembe,
   Cuma,
   Cumartesi,
   Pazar
}
```

IsDefined Metodu

Enum içerisinde bir elemanın tanımlanıp tanımlanmadığını bulmak amacı ile kullanılır.

Örneğin **Gunler** adlı **Enum** içerisinde **Pazartesi** gününün tanımlanıp tanımlanmadığına bakalım:

```
if (Gunler.IsDefined(typeof(Gunler), "Pazartesi"))
{
    MessageBox.Show ("Pazartesi değeri mevcut");
}
```

```
    else
{
    MessageBox.Show("Pazartesi değeri mevcut değil");
}
```

Aynı işlemi eleman ismi yerine sıra numarasını kullanarak da gerçekleştirebiliriz.

```
if (Gunler.IsDefined(typeof(Gunler), 0))
{
    MessageBox.Show ("Pazartesi değeri mevcut");
}
    else
{
    MessageBox.Show("Pazartesi değeri mevcut değil");
}
```

Enum İçerisindeki Elemanlara Erişim

Enum içerisindeki elemanlar **0** sıra numarasından başlamak üzere sıralanırlar. Sıra numaralarını kullanarak **enum** içerisindeki elemanlara erişim sağlanabilir.

Örneğin **Gunler** adında bir **enum** oluşturup elemanlarına sıra numaralarını kullanarak erişelim:

```
using System;
using Microsoft.VisualBasic;
using System.Windows.Forms;
namespace enum_Erisim
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        public enum Gunler
        {
            Pazartesi,
            Salı,
            Çarşamba,
```

```
            Perşembe,
            Cuma,
            Cumartesi,
            Pazar
        }
        private void button1_Click(object sender, EventArgs e)
        {
            byte i = Convert.ToByte(Interaction.InputBox
("Gün numarasını giriniz"));
      Gunler gunadi;
      gunadi = (Gunler)i;
            MessageBox.Show(gunadi.ToString());
        }
    }
}
```

Örneğimizde **gunadi** adlı bir değişkeni **Gunler** adındaki **enum** tipinden tanımlayarak ve **inputbox** ile girilen gün numarasına **Gunler** adlı **enum** içerisinde karşılık gelen gün ismini **gunadi** isimli değişkene aktararak bu değişkenin içeriğini mesaj penceresi içerisinde görüntüledik. Gün numarası olarak **0** girdiğimizde **Pazartesi**, **1** girdiğimizde ise **Salı** mesajı görüntülenecektir. Çünkü **enum** içerisindeki ilk elemanın sıra numarası 0'dan başlar. Kodumuzda **InputBox** kullandığımız için **Microsoft.VisualBasic** sınıfını referans olarak ekliyoruz.

enum içerisindeki elemanlara erişirken kullanacağımız, **ToString** metodu ile birlikte kullanılan bazı format karakterleri ile **enum** elemanlarını çeşitli biçimlerde elde edebiliriz. Bu karakterler ve anlamları şu şekildedir:

G veya g (General): **Enum** değerini genel biçimde görüntüler.

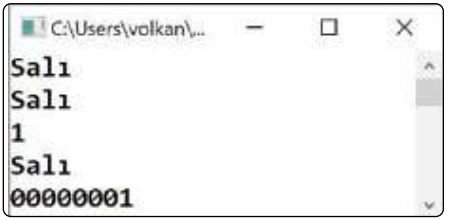
D veya d (Decimal): **Enum** değerinin sayısal karşılığını görüntüler.

F veya f (Flag): **Enum** değerinin **string** karşılığını görüntüler. Eğer birden çok eleman, mantıksal operatör kullanılarak gruplandırıldı ise bu gruplandırılmış eleman görüntülendiğinde aralarına virgül konularak görüntülenir.

X veya x (HexaDecimal): **Enum** değerinin hexadecimal (on altılık) karşılığını görüntüler.

Örneğin daha önce görüntülediğimiz **Gunler** isimli **enum** içerisindeki elemanları **ToString** metodunda biçimlendirme karakterleri kullanarak görüntüleyelim.

```
using System;
using System.Text;
namespace enum_Formats
{
    class Program
    {
        enum Gunler
        {
            Pazartesi,
            Salı,
            Çarşamba,
            Perşembe,
            Cuma,
            Cumartesi,
            Pazar
        }
        static void Main(string[] args)
        {
            Gunler gunadi;
            gunadi = (Gunler)1;
            Console.WriteLine(gunadi.ToString());
            Console.WriteLine(gunadi.ToString("G"));
            Console.WriteLine(gunadi.ToString("d"));
            Console.WriteLine(gunadi.ToString("F"));
            Console.WriteLine(gunadi.ToString("X"));
            Console.ReadLine();
        }
    }
}
```



Kodumuz içerisinde **1** sıra numarasına sahip olan **Salı** değerini sırasıyla aşağıdaki biçimlerde görüntüledik.

Herhangi bir format karakteri kullanmadan yalnızca değeri yani **Salı** sonucu görüntülendi.

G karakteri ile genel format biçimi görüntüledik. **Salı** sonucu görüntülendi.

d karakteri ile **Salı** değerinin sayısal karşılığı olan **1** rakamı görüntülendi. Burada küçük harfin de kullanılabildiğini göstermek için d karakteri şeklinde yazdık.

F karakteri ile **Salı** değerinin ***string*** karşılığı olan **Salı** değeri görüntülendi.

X karakteri ile **Salı** değerinin **hexadecimal** karşılığı olan **00000001** değeri görüntülendi.

Mantıksal operatörler kullanarak **enum** içerisindeki değerler üzerinde işlemler yapabiliriz. Örneğin **uzmanlık** adlı bir **enum** tanımlayarak içerisinde masaüstü programlama, web programlama, script programlama dillerini eleman olarak ekleyerek bu diller üzerinde | (or - veya) mantıksal operatörü ile gruplandırma yapalım.

```
using System;
using System.Text;
namespace enum_MantıksalOperator
{
    class Program
    {
        enum uzmanlık
        {
            VBNet = 1,
            CSharp = 2,
            ASPNet = 4,
            PHP = 8,
            Python = 10,
            Perl = 20
        }
        static void Main(string[] args)
        {
            uzmanlık masaustuprogramlama = uzmanlık.
VBNet | uzmanlık.CSharp;
```

```
            Console.WriteLine(masaustuprogramlama.ToString());
            Console.WriteLine(masaustuprogramlama.ToString("G"));
            Console.WriteLine(masaustuprogramlama.ToString("d"));
            Console.WriteLine(masaustuprogramlama.ToString("F"));
            Console.WriteLine(masaustuprogramlama.ToString("X"));
            Console.ReadLine();
        }
    }
}
```



Örneğimizde;

`uzmanlık.VBNet | uzmanlık.CSharp`; cümlesi ile **`VB.Net`** ve **`CSharp`** elemanlarını gruplandırarak **`masaustuprogramlama`** adlı değişkene aktardık. Burada görüldüğü gibi **`ToString`** metodunu **`F`** (*Flag*) biçimlendirmesi ile birlikte kullandığımızda **`masaustuprogramlama`** değişkeni içerisine **`|`** (or) mantıksal operatörü kullanılarak eklenmiş olan **`VB.Net`** ve **`CSharp`** değişkenleri, aralarına **virgül** konularak listelenir. Diğer biçimlendirmelerde ise **`3`** rakamının görüntülendiğini görmekteyiz. Bu değer VB.Net elemanının değeri olan **`1`** ve **`CSharp`** elemanının değeri olan 2 rakamlarının toplanması ile elde edilmiştir. Örneğin **`webprogramlama`** adında bir değişken tanımlayalım ve içerisine **ASP. Net** ve **PHP** elemanları **`|`** (or - veya) mantıksal operatörü kullanarak ekleyelim:

```
using System;
using System.Text;
namespace enum_or
{
    class Program
    {
        enum uzmanlık
        {
            VBNet = 1,
```

```
            CSharp = 2,
            ASPNet = 4,
            PHP = 8,
            Python = 10,
            Perl = 20,
            ButunDiller = (VBNet | CSharp | ASPNet | PHP | Python |
Perl)
        }
        static void Main(string[] args)
        {
            uzmanlık webprogramlama = uzmanlık.ASPNet | uzmanlık.
PHP;
            Console.WriteLine(webprogramlama.ToString());
            Console.WriteLine(webprogramlama.ToString("G"));
            Console.WriteLine(webprogramlama.ToString("d"));
            Console.WriteLine(webprogramlama.ToString("F"));
            Console.WriteLine(webprogramlama.ToString("X"));
            Console.ReadLine();
        }
    }
}
```



ASPNet ve **PHP** elemanlarının değerleri **8** ile **4** olduğundan bunların toplamları olan **12** değeri **G** ve **d** biçimlendirme karakterleri ile görüntülendi. **F** karakteri kullanıldığında ise **ASPNet** ve **PHP** elemanları aralarına **virgül** konularak listelendi. **X** karakterinde ise **12** rakamının **hexadecimal** karşılığı olan **0000000C** değeri görüntülendi.

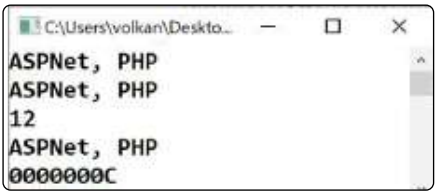
[Flags( )] niteliğini kullanarak **F** parametresiyle elde ettiğimiz karşılıkları diğer parametreler için de elde edebiliriz. Yukarıdaki örneğimizde **ToString( )** ya da **ToString ("G")** metotları ile de **ASP.NET, PHP** şeklinde **webprogramlama**

değişkeni içerisindeki elemanları elde etmek istersek **enum** tanımlamamızı:

```
[Flags()]
enum uzmanlık
{
    VBNet = 1,
    CSharp = 2,
    ASPNet = 4,
    PHP = 8,
    Python = 10,
    Perl = 20
}
```

Şeklinde düzenlememiz gerekmektedir.

Programımızı bu düzenlemeyi yapıp çalıştırdığımızda:



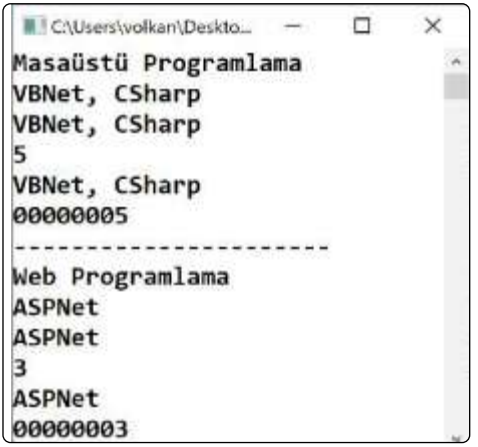
Şeklinde **ekran çıktısı** elde ederiz.

Enum içerisindeki elemanlara verdiğimiz değerler rastgele şekilde verilmemiştir. Örneğin ardışık rakamlardan oluşacak şekilde değerler verelim ve sonucuna bakalım:

```
using System;
using System.Text;
namespace enum_flags
{
    class Program
    {
        [Flags()]
        enum uzmanlık
        {
            VBNet = 1,
            PHP = 2,
```

```
            ASPNet = 3,
            CSharp = 4
        }
        static void Main(string[] args)
        {
            uzmanlık masaustuprogramlama = uzmanlık.
VBNet | uzmanlık.CSharp;
            uzmanlık webprogramlama = uzmanlık.PHP | uzmanlık.
ASPNet;
            Console.WriteLine("Masaüstü Programlama");
            Console.WriteLine(masaustuprogramlama.ToString());
            Console.WriteLine(masaustuprogramlama.ToString("G"));
            Console.WriteLine(masaustuprogramlama.ToString("d"));
            Console.WriteLine(masaustuprogramlama.ToString("F"));
            Console.WriteLine(masaustuprogramlama.ToString("X"));
            Console.WriteLine("----------------------");
            Console.WriteLine("Web Programlama");
            Console.WriteLine(webprogramlama.ToString());
            Console.WriteLine(webprogramlama.ToString("G"));
            Console.WriteLine(webprogramlama.ToString("d"));
            Console.WriteLine(webprogramlama.ToString("F"));
            Console.WriteLine(webprogramlama.ToString("X"));
            Console.ReadLine();
        }
    }
}
```

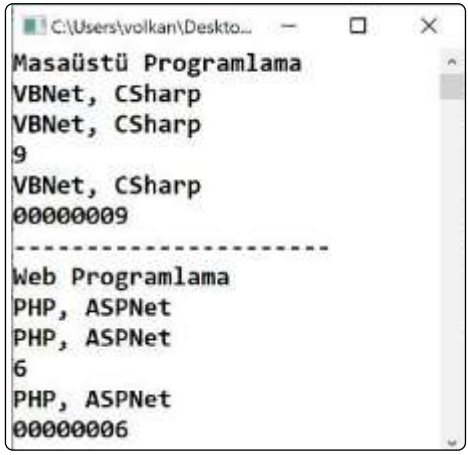
Kodumuzu çalıştırdığımızda **masaustuprogramlama** adlı değişkenin elemanları olan **VBNet** ve **CSharp** elemanlarının ikisinin de doğru olarak listelendiğini ve bu elemanların değerleri olan **1** ve **4** 'ün toplanarak **5** sonucunun doğru vaziyette görüntülendiğini görüyoruz.



Ancak **webprogramlama** adlı değişkenin elemanları olan **ASPNet** ve **PHP** elemanlarından sadece **ASPNet** elemanının görüntülendiğini ve bu elemanın değeri olan **3** değerinin görüntülendiğini görüyoruz. **enum** içerisindeki sabitlerde kullanılan sayısal değerler, bu tip hatalara neden olabilmektedir. **enum** içerisindeki sabitlere verilen değerlerin toplamının başka bir sabitin değeri ya da değerlerinin toplamına eşit olmamasına dikkat etmeliyiz. Bunu engellemek için bir sabitin sayısal değeri en az kendinden önceki değerlerin toplamının bir fazlası olacak biçimde ya da sabitlerin sayısal değerleri artan katlar şeklinde olacak biçimde yazabilirsiniz.

enum içerisindeki değerlerin numaralandırmasını artan katlar şeklinde düzenleyerek numara verelim:

```
[Flags()]
enum uzmanlık
{
    VBNet = 1,
    PHP = 2,
    ASPNet = 4,
    CSharp = 8
}
```



Kodumuzu bu düzenlemeyi yaparak çalıştırdığımızda, **masaustuprogramlama** ve **webprogramlama** değişkenlerinin elemanlarının doğru olarak görüntülendiğini ve bu elamanlara karşılık gelen değerlerin toplamlarının doğru olarak görüntülendiğini görürüz.

System.Enum Sınıfı

enum içerisindeki elemanlar üzerinde işlem yapmak için **System.Enum** sınıfından faydalanılabilir. Bu sınıfa ait metotlardan **enum** içerisindeki elemanlara erişim için kullanılan başlıcalarını inceleyelim.

GetName Metodu

enum içerisinde mevcut olan bir elemana, sayısal değerini kullanarak erişmek için **GetName** metodunu kullanabiliriz. mesela oluşturmuş olduğumuz **uzmanlık** adlı **enum** içerisindeki bir elemana ulaşabilmek için **GetName** metodundan faydalanabiliriz.

Örneğin **uzmanlık** adlı **enum** içerisindeki **CSharp** adlı elemana erişelim:

```
class Program
{
    [Flags()]
    enum uzmanlık
    {
        VBNet = 1,
        PHP = 2,
```

```
        ASPNet = 4,
        CSharp = 8
    }
    static void Main(string[] args)
    {
        Console.WriteLine(Enum.GetName(typeof(uzmanlık), 8));
        Console.ReadLine();
    }
}
```

Örneğimizde **uzmanlık** adlı **enum** içerisindeki elemanlara değer ataması yapmamız dolayısı ile burada **CSharp** elemanının değeri olan **8** değerini girerek:

```
GetName(typeof(uzmanlık), 8)
```

Şeklinde **CSharp** elemanını görüntüledik.

Eğer elemanlara aşağıda görüldüğü gibi değer ataması yapmadan tanımlama yapsaydık:

```
[Flags()]
enum uzmanlık
{
   VBNet,
   PHP,
   ASPNet,
   CSharp
}
```

Kodumuzda geçen **TypeOf** metodu, **uzmanlık** adlı değişkenin **enum** türünden olduğunu belirtmek amacıyla kullanılmaktadır.

GetName metodunu:

```
GetName(typeof(uzmanlık), 3)
```

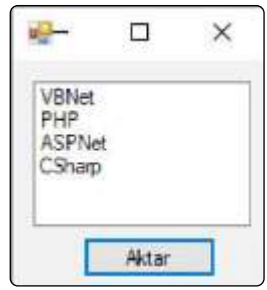
Şeklinde son elemanın sıra numarasını belirterek kullanarak **CSharp** değerini görüntüleyebilirdik.

GetNames Metodu

enum içerisindeki elemanları **string** formatta elde ederek bir dizi içerisinde saklamak için **GetNames** metodundan faydalanılır. Örneğin **uzmanlık** adlı **enum** içerisindeki elemanları bir **listbox**'a aktarmak için:

```
foreach (string uzmanlıkalanı in Enum.GetNames(typeof(uzmanlık)))
{
    listBox1.Items.Add(uzmanlıkalanı);
}
```

Şeklinde **GetNames** metodundan faydalanarak aktarım yapabiliriz. **foreach** döngüsü eleman sayısının belli olmadığı döngülerde kullanılır.



Yukarıdaki kod satırlarında **uzmanlık** adlı **enum** içerisindeki elemanların sayısı kadar bir döngü kurarak elemanları **uzmanlık** alanı isimli değişkene aktarıp **listBox** içerisine ekledik.

GetValues Metodu

enum içerisindeki sabitlerin sayısal değerlerini **int** formatta elde ederek bir dizi içerisinde saklamak amacıyla kullanılır. Örneğin **uzmanlık** adlı **enum** içerisindeki elemanların sayısal değerlerini bir **listBox**'a aktarmak için:

```
foreach (int deger in Enum.GetValues(typeof(uzmanlık)))
{
    listBox1.Items.Add(deger);
}
```

Şeklinde **GetValues** metodundan faydalanarak **enum** içerisindeki sabitlerin sayısal değerlerini aktarabiliriz.



Parse Metodu

Enum içerisindeki elemanın ismi ya da aldığı değere erişmek için **Parse** metodu kullanılabilir.

Örneğin **uzmanlık** adlı **enum** içerisindeki **CSharp** elemanının değerini, **Parse** metodu ile görüntüleyelim:

```
using System;
using System.Text;
namespace Enum_Parse
{
    class Program
    {
        [Flags()]
        enum uzmanlık
        {
            VBNet = 1,
            PHP = 2,
            ASPNet = 4,
            CSharp = 8
        }
        static void Main(string[] args)
        {
          uzmanlık uzmanlıkalanı = (uzmanlık) Enum.Parse
(typeof(uzmanlık), "CSharp",true);
  Console.WriteLine("CSharp uzmanlık alanının değeri = {0}" ,
uzmanlıkalanı.ToString("D") );
         Console.ReadLine();
        }
    }
}
```

Kodumuzu çalıştırdığımızda **"CSharp uzmanlık alanının değeri =8"** şeklinde sonuç üretecektir.

OPERATÖRLER

5

C# içerisinde kendi başlarına bir anlam ifade etmeyip, çeşitli işlemleri gerçekleştirmede kullanılan sembollere operatör adı verilir. C# .NET dilinde aritmetiksel, aktarma, mantıksal, karşılaştırma ve birleştirme operatörü olarak 5 çeşit operatör kullanılmaktadır.

C# içerisinde aritmetiksel operatörler kullanılırken işlem önceliği dikkate alınarak işlem yapılır. İşlem önceliği yukarıdan aşağıya doğru olmak üzere aşağıdaki tablodaki gibidir;

Operatörler
( )
! + - ++ --
* / %
+ -
< <= > >= is as
== !=
&
^
|
&&
||
?:
= *= /= %= += -= &= ^= |=

Aritmetik Operatörler

Matematiksel işlemlerin gerçekleştirilmesi amacı ile kullanılan operatörlerdir.

Aritmetik operatörler başlıca şu şekildedir:

+	Toplama
-	Çıkarma
*	Çarpma
^	Üs alma
/	Bölme
%	Mod

Örneğin; bölme işlemi ile ilgili örnek bir Windows Form uygulaması yapalım.

```
private void button1_Click(object sender, EventArgs e)
{
 double sayi1, sayi2;
 sayi1 = 10.15;
 sayi2 = 3.5;
 MessageBox.Show((sayi1 / sayi2).ToString());
}
```

Kodumuzu çalıştırdığımızda bölme işlemi (**sayi1/sayi2**) şeklinde gerçekleştirilip, sonuç sayısal olduğundan dolayı, mesaj penceresi içerisinde görüntülemek için **ToString()** metodu ile **string** veri türüne dönüştürülerek işlem sonucu olan **2.9** değeri görüntülenmektedir.

% operatörünü kullanarak mod alma işlemi yapalım.

```
private void button1_Click(object sender, EventArgs e)
{
 int sayi1, sayi2;
 sayi1 = 10;
 sayi2 = 3;
 MessageBox.Show((sayi1 % sayi2).ToString());
}
```

Kodumuzu çalıştırdığımızda **sayi1** değişkeninin değeri olan **10** sayısı, **sayi2**

değişkeninin değeri olan 3 sayısı ile mod işlemine tabi tutularak (10 sayısının 3 ile bölümünden kalan alınarak) konsol ekranında 1 sonucu yazdırılır.

Parantez içlerinin işlem önceliğinin daha yüksek olduğunu daha önce belirtmiştik. Örneğin…

```
int sonuc = 2 + 3 * 5;
Console.WriteLine(sonuc);
```

… işleminin sonucu 17 iken…

```
int sonuc = (2 + 3) * 5;
Console.WriteLine(sonuc);
```

… işleminin sonucu ise 25 olmaktadır. Parantez içinin işlem önceliğinin daha yüksek olması nedeniyle, öncelikle parantez içindeki işlem yapılmaktadır.

Aktarma Operatörleri

Aktarma işlemlerinin gerçekleştirilmesi amacı ile kullanılan operatörlerdir.

- **= Atama**

+= Solundaki değişken ile sağındakini toplayarak, sonucu solundaki değişkene aktarır.

```
byte a = 3;
byte b= 2;
a += b;
```

a, 5 değerini alır.

-= Solundaki değişkenden sağındakini çıkartarak, sonucu solundaki değişkene aktarır.

```
byte a = 3;
 byte b = 2;
 a -= b;
```

a, 1 değerini alır.

***=** Solundaki değişken ile sağındakini çarparak, sonucu solundaki değişkene aktarır.

```
byte a = 3;
 byte b = 2;
 a *= b;
```

a, 6 değerini alır.

/= Solundaki değişkeni sağındaki değişkene bölerek, sonucu solundaki değişkene aktarır.

```
double a = 5;
 double b = 2;
 a /= b;
```

a, 2,5 değerini alır.

%= Solundaki değişkeni sağındaki değişkene bölerek, kalanı solundaki değişkene aktarır.

```
byte a = 7;
 byte b = 4;
 a %= b;
```

a, 3 değerini alır.

Mantıksal Operatörler

Aynı zamanda **boolean** operatörler adını da almaktadır. Mantıksal operatörlerle **true** ve **false** olmak üzere iki değer üzerinde işlem yapılır. Mantıksal operatörler ve görevleri aşağıdaki gibidir.

& (Ve) Operatörü

true değerini alması için iki değişken de **true** olmalıdır.

```
true & true = true
true & false = false
false & true = false
false & false = false
```

| (VEYA) OPERATÖRÜ

Değişkenlerin ikisi birden false olduğunda, false sonucunu üretir.

```
true | true = true
true | false = true
false | true = true
false | false = false
```

&& (VEDEĞİL) OPERATÖRÜ

Eğer ilk değişken **`true`** ise **`and`** işlemi gerçekleştirilir. **`False`** ise ikinci değişkene bakılmaksızın **`false`** sonucu üretilir.

```
true &&  true = true
true && false = false

false && işlem yapılmayan değişken = false
```

|| (VEYADEĞİL) OPERATÖRÜ

Eğer ilk değişken **`false`** ise, **`or`** işlemi gerçekleşir. **`true`** ise ikinci değişkene bakılmaksızın true sonucu üretilir.

```
true || işlem yapılmayan değişken = true
false || true = true
false || false = false
```

^ (ÖZEL VEYA) OPERATÖRÜ

İki değişkenden biri diğerinden farklı olduğunda true sonucunu üretir.

```
true ^ false = true
true ^ true = false
false ^ false = false
false ^ true = true
```

! (DEĞİL) OPERATÖRÜ

Değişkenin değerini tersine çevirir.

```
! true = false
! false = true
```

?? (Null Coalescing) Operatörü

X ?? y

Eğer ilk değişken **null** ise, ikinci değer hesaplanır. Aksi takdirde ikinci değişken işleme konulmaz.

```
static void Main(string[] args)
{
 string adı=null;
 string durum = adı ?? "Adı girilmedi";
 Console.WriteLine(durum );
 Console.ReadLine();
}
```

Örneğimizi çalıştırdığımızda **adı** değişkeninin değeri **null** olduğundan dolayı, **durum** değişkenine "*Adı girilmedi*" değerini aktaracak ve konsol ekranında görüntülenecektir.

Eğer örneğimizde…

```
static void Main(string[] args)
{
 string adı="Azra";
 string durum = adı ?? "Adı girilmedi";
 Console.WriteLine(durum );
 Console.ReadLine();
}
```

… şeklinde adı değişkenine bir değer aktarılırsa, "*Azra*" değeri durum değişkenine aktarılacak ve konsol ekranında görüntülenecektir.

C# 8 versiyonundan itibaren **null coalescing** operatörünü aktarma operatörüyle birlikte **??=** biçiminde kullanabiliriz. Örneğin yukarıda yazdığımız

```
string durum = adı ?? "Adı girilmedi";
```

satırını artık;

```
string durum = adı ??= "Adı girilmedi";
```

biçiminde yazabiliriz.

?: (Koşul) Operatörü

x ?: y : z

3 adet operanda sahiptir. Eğer **x** değeri true ise, **y** değişkeni hesaplanır, **false** ise **z** değişkeni hesaplanır.

```
static void Main(string[] args)
{
 string durum;
 Console.WriteLine("Çıkmak İstediğinize Emin misiniz? (E, H)");
 durum = Console.ReadLine();
 Console.Write(durum == "E" ? "Güle Güle" : "Çalışmaya Devam");
 Console.ReadLine();
}
```

Örneğimizde **E** harfine basıldığında "*Güle Güle*", **H** harfine basıldığında "*Çalışmaya Devam*" sonucu görüntülenecektir.

Karşılaştırma Operatörleri

>	Büyüktür.
<	Küçüktür.
>=	Büyük ya da eşittir.
<=	Küçük ya da eşittir.
==	Eşittir.
!	Değil.
!=	Farklıdır, eşit değildir.
-->	Uzaklaşma operatörü.
<--	Yakınlaşma operatörü.

Örnek olarak uzaklaşma ve yakınlaşma operatörlerini bir konsol uygulaması içerisinde gösterelim.

Öncelikle (-->) uzaklaşma operatörü ile yapalım.

```
static void Main(string[] args)
{
 int i = 5;
 while (i --> 0)
 {
  Console.WriteLine("i = {0}", i);
```

```
 }
 Console.ReadLine();
}
```



Örneğimiz 4'den 0'a doğru sayıların azalarak yazılmasını netice verecektir.

Burada (`i-->0`) ifadesi "`i` değerinin `0`'a doğru giderken" anlamını ifade etmektedir.

Aynı örneği (`<--`) yakınlaşma operatörü ile yapalım.

```
static void Main(string[] args)
{
 int i = 5;
 while (-1 <-- i)
 {
  Console.WriteLine("i = {0}", i);
 }
 Console.ReadLine();
}
```

Aynı sonucu vermesi için bu defa döngü ifadesini `-1 <-- i` şeklinde yazdık. Burada `-1 <-- i` ifadesi; `i` değerinin `0`'a doğru gelirken anlamını ifade etmektedir.

Birleştirme Operatörleri

+	İki string değişkeni birleştirir
+=	Solundaki değişken ile sağındakini birleştirerek sonucu solundaki değişkene aktarır.

```
String a= "3";
String b= "2";
a += b;
```

Her iki değişkenin de string olması nedeniyle, **a** ile **b** değişkenlerinin içerikleri birleştirilerek **a** değişkenine aktarılır ve **a** "**32**" değerini alır.

KONTROL YAPILARI

6

IF YAPISI

if komutu, verilen bir şartın gerçekleşmesi ya da gerçekleşmemesi durumunda belirtilen kodların çalıştırılmasını sağlar. Kullanımı şu şekildedir:

```
if (Şart)
{
   Komutlar;
}
else if (Şart)
{
   Komutlar;
}
else
{
   Komutlar;
}
```

Şartın sağlanması durumunda **if**'den sonraki komutlar, sağlanmaması durumunda **else if** ya da **else** komutundan sonraki komutlar çalıştırılır.

if komutunu **eğer**, **else** komutunu **değilse**, **else if** komutunu ise **değilse eğer** biçiminde yorumlayabiliriz. **else**, artık gerçekleşebilecek herhangi bir ihtimal kalmadığında kullanılır. Dolayısıyla **else** komutundan sonra herhangi bir şart cümlesi yazılmaz.

if ve **else if** komutundan sonra yazılan şart ifadesi **parantez içinde** yazılmalıdır.

Örneğin **else** komutunu kullanan bir konsol uygulaması yapalım. Örneğimizde girilen not **45**'den küçükse **kaldınız**, değilse **geçtiniz** şeklinde bir uyarı verilsin.

```
static void Main(string[] args)
{
    Console.WriteLine("Notu Giriniz");
    byte not_degeri = Convert.ToByte
(Console.ReadLine());
    if (not_degeri >= 45)
    {
        Console.WriteLine("Geçtiniz");
    }
    else
    {
        Console.WriteLine("Kaldınız");
    }
    Console.ReadKey();
}
```



Örneğimizde eğer not değeri **45**'den büyük ve eşitse ekrana "**Geçtiniz**" yazısı, değilse "**Kaldınız**" yazısı yazdırılıyor. **else** ifadesini kullandığımızdan dolayı **45**'den büyük ve eşit olmayan her not için "**Kaldınız**" yazısı görüntülenecektir. **Convert.ToByte** metodu ile **string** veri türünde girilen not değeri **byte** türüne dönüştürülmektedir.

Şart ifadesinden sonra kullanacağımız komutlar, tek satırdan oluşuyorsa parantez kullanmadan da komut satırını yazabiliriz.

Örneğin yukarıdaki örneğimizde:

```
if (not_degeri >= 45)
{
    Console.WriteLine("Geçtiniz");
}
else
{
    Console.WriteLine("Kaldınız");
}
```

Şeklinde şart ifadesinden sonra parantez kullanarak yazdığımız kodu:

```
if (not_degeri >= 45)
    Console.WriteLine("Geçtiniz");
else
    Console.WriteLine("Kaldınız");
```

Şeklinde parantez kullanmadan da yazabiliriz. Her iki kod da aynı sonucu verecektir.

`else if` komutunu kullanan bir **windows form** uygulaması yapalım. Örneğimiz **`45`**'den büyük değer girilirse **`geçtiniz`**, küçük değer girilirse **`kaldınız`** şeklinde mesaj görüntülüyor. Ayrıca 100'den büyük değer girişi de kontrol ediliyor. Örneğimiz için form üzerine bir adet buton eklememiz yeterli. **`InputBox`** nesnesini kullanarak not girişi yapacağımız için **`Microsoft.VisualBasic`** sınıfını referans olarak eklememiz gerekiyor.

```
using System;
using Microsoft.VisualBasic;
using System.Windows.Forms;
namespace not_else_if
{
    public partial class Form1 : Form
    {
        public Form1()
        {
```

```
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            byte notu;
            notu = Convert.ToByte (Interaction.InputBox("Notunuzu
girin"));
            if (notu <= 100)
            {
                if (notu < 45)
                {
                    MessageBox.Show("Kaldınız");
                }
                else if (notu >= 45)
                {
                    MessageBox.Show("Geçtiniz");
                }
            }
            else
            {
                MessageBox.Show("Not 100'den küçük olmalı");
            }
        }
    }
}
```

Kodumuzda öncelikle:

```
if (notu <= 100)
```

Satırında **100**'den küçük ve eşit değer girilmesi durumunda kod bloğu içerisindeki kod satırlarının çalıştırılmasını sağlıyoruz. **else** ifadesi ile de **100**'den küçük ve eşit olmayan değer girilmesi durumunda "**Not 100'den küçük olmalı**" mesajı görüntülenecektir.

```
if (notu<45)
```

Satırında **45**'den küçük değer girildiğinde "**Kaldınız**" mesajı görüntülenirken:

```
else if  (notu >= 45)
```

Satırı ile de **45**'den büyük ve eşit not girilmesi durumunda "**Geçtiniz**" mesajı görüntülenmektedir.

Ayrıca kodumuzda görüldüğü üzere iç içe birden fazla **if** kullanmak da mümkündür.

Örneğimizi bir adım ilerletip girilen notu **5**'lik not sistemine çevirelim.

```
private void button1_Click(object sender, EventArgs e)
{
        byte notu;
        notu = Convert.ToByte(Interaction.InputBox("Notunuzu
girin"));
        if (notu < 25) {
            MessageBox.Show("Notunuz 0"); }
        else if (notu < 45) {
            MessageBox.Show("Notunuz 1"); }
        else if (notu < 55) {
            MessageBox.Show("Notunuz 2"); }
        else if (notu < 70) {
            MessageBox.Show("Notunuz 3"); }
        else if (notu < 85) {
            MessageBox.Show("Notunuz 4"); }
        else if (notu <= 100) {
            MessageBox.Show("Notunuz 5"); }
        else {
            MessageBox.Show("Yanlış Not Girdiniz"); }
}
```

Örneğimizde girilen notun hangi puan aralığında olduğu tespit edilerek nota karşılık gelen puan, mesaj penceresi içerisinde görüntülenmektedir. Eğer **0-100** aralığı dışında bir not girişi yapılmışsa **else** ile bunun kontrolü yapılarak **"Yanlış Not Girdiniz"** şeklinde bir mesaj görüntülenmektedir.

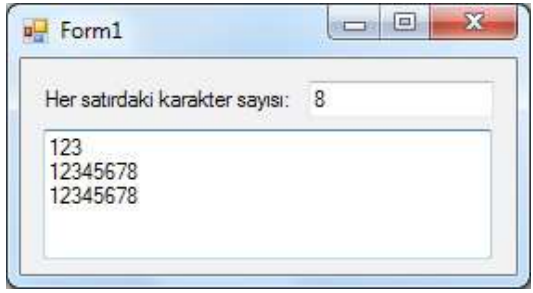
Konsol ekranında girilen bir sayının tek mi çift mi olduğunu bulup konsol ekranı içerisinde yazan bir uygulama yapalım.

```
static void Main(string[] args)
{
    int sayi;
    Console.WriteLine("Sayıyı girin");
    sayi = Console.Read();
    if (sayi % 2 ==0)
        Console.WriteLine("Sayı Çift");
    else
        Console.WriteLine("Sayı Tek");
    Console.ReadKey();
}
```

Bir sayının tek mi çift mi olduğunu bulmak için `2` ile bölümünden kalana bakabiliriz. 2 ile bölümünden kalan `0` ise sayı çift, değilse sayı tektir. Sayının `2` ile bölümünden kalanı bulmak için örneğimizde `%` (*mod*) operatörünü kullandık.

Bir **`textBox`** içerisine belli sayıda karakter girildikçe **otomatik** olarak bir alt satıra geçen örnek bir uygulama yapalım. Form üzerine iki adet **`textBox`** ekliyoruz. **`txtKarakter`** isimli **`textBox`**'a karakter adedi, **`textBox1`** isimli **`textBox`** içerisine ise metnimizi giriyoruz.

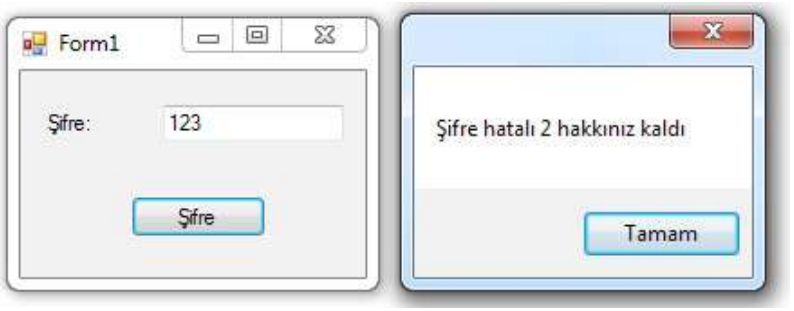
```
Int32 karakteradedi;
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (karakteradedi >0)
    {
      if (karakteradedi % Convert.ToInt32(txtKarakter.Text) == 0)
          textBox1.AppendText(Environment.NewLine);
    }
    karakteradedi = karakteradedi + 1;
}
```



Kodumuzda öncelikle karakter adedinin `0`'dan büyük olup olmadığını kontrol ederek karakter adedi `0`'dan büyükse kod bloğunun çalıştırılmasını sağlıyoruz. Kod bloğu içerisinde **`karakteradedi`** değişkeninin içeriği ile **`txtKarakter`** isimli **`textBox`** içerisindeki rakam `%` operatörü ile mod işlemine tabi tutularak eğer dönen sonuç `0` ise **`textBox`** içerisine **`AppendText`** metodu ile yeni bir satır (**`Environment.NewLine`**) ekliyoruz.

Şimdi de şifre sorma ile alakalı bir örnek uygulama yapalım:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    int hak;
    int toplamhak;
    private void button1_Click(object sender, EventArgs e)
    {
        toplamhak = 3;
        if (textBox1.Text == "deneme") {
            MessageBox.Show("Şifre doğru");
        }
        else {
           hak = hak + 1;
           MessageBox.Show("Şifre hatalı " +
     (toplamhak-hak) + " hakkınız kaldı");
           if (hak == 3) Application.Exit();
        }
    }
}
```



Örneğimizde bir **text** kutusu içerisine girilen şifrenin doğru olup olmadığı `if` ile kontrol edilmekte, `else` ifadesi ile de eğer girilen şifre yanlış ise her defasında `hak` isimli değişkenin değeri bir arttırılmakta ve kaç hakkının kaldığı kullanıcıya bir mesaj penceresi içerisinde bildirilmektedir. Eğer `hak` değişkeninin değeri `3` ise `Application.Exit` metodu ile programın çalışması sona erdirilmektedir.

Konsol ekranında girilen `3` adet sayıyı karşılaştırıp içlerinden en büyüğünü bulan örnek bir konsol uygulaması yapalım.

```
static void Main(string[] args)
{
    int sayi1, sayi2, sayi3;
    Console.WriteLine("1. Notu Giriniz: ");
    sayi1 = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("2. Notu Giriniz: ");
    sayi2 = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("3. Notu Giriniz: ");
    sayi3 = Convert.ToInt32((Console.ReadLine()));
    if ((sayi1 > sayi2) & (sayi1 > sayi3))
        Console.WriteLine("Sayılardan en büyüğü Sayı1= " + sayi1);
    else if ((sayi2 > sayi1) & (sayi2 > sayi3))
        Console.WriteLine("Sayılardan en büyüğü Sayı2= " + sayi2);
    else if ((sayi1 < 0) | (sayi2 < 0) | (sayi3 < 0))
        Console.WriteLine("Sayılar 0'dan küçük olmamalı!");
    else
        Console.WriteLine("Sayılardan en büyüğü Sayı3= " + sayi3);
    Console.ReadKey();
}
```

Örneğimizde, her bir sayıyı ayrı ayrı diğer iki sayı ile karşılaştırmak için **&** **(AND-VE)** operatörünü kullandık. Her bir sayının da 0'dan küçük olup olmadığını kontrol etmek için **|** **(OR-VEYA)** operatörünü kullandık. Bu şekilde birden fazla şartın sağlanıp sağlanmadığını kontrol ettik.

if yapısını kullanarak sayı tahmin oyunu yapalım. Örneğimizde üretilen rastgele bir sayıyı, kullanıcı tahmin etmeye çalışmaktadır. Kullanıcının 5 defa tahmin hakkı bulunmaktadır. Kullanıcı 5 tahmin hakkı dolmadan önce, **"Daha Küçük!"** veya **"Daha Büyük"** şeklinde yönlendirilmektedir. Bu 5 hakkı dolunca **"Hakkınız Doldu"** şeklinde uyarılmaktadır. Hakkı dolmadan bildiğinde ise **"Tebrik ederim!Bildiniz"** şeklinde bilgilendirilmektedir.

Formumuzun üzerine **btnTahmin** ve **btnBasla** ismi ile iki buton, **lblMesaj** adıyla da bir adet **label** nesnesi ekliyoruz.

Kodumuz ise şu şekilde olacak:

```
int hak, tahmininiz, sayi;
private void btnTahmin_Click(object sender, EventArgs e)
{
    if (hak == 0) btnTahmin.Enabled = false;
    tahmininiz = Convert.ToInt32(Interaction.InputBox
        ("Tahmininizi Giriniz"));
    if ((Convert.ToInt32(tahmininiz) <= 100) &
        (Convert.ToInt32(tahmininiz) >= 0))
    {
        if (sayi < 100 | sayi >= 0)
        {
          if (tahmininiz < sayi)
              lblMesaj.Text =
                "Daha Büyük!" + hak + " hakkiniz kaldi";
          if (tahmininiz > sayi)
              lblMesaj.Text =
                "Daha Küçük!" + hak + " hakkiniz kaldi";
          if (tahmininiz == sayi)
              lblMesaj.Text =
                "Tebrik ederim! Bildiniz...";
          if (hak > 0)       {
              hak = hak - 1; }
          else {
```

```
                lblMesaj.Text =
                  "Sayıyı Bulamadınız. Tutulan sayı = " + sayi; }
            }
        }
    }
    private void uret()
    {
        sayi = 0;
        Random rastgele = new Random();
        int tutulansayi = rastgele.Next(1, 100);
        sayi = tutulansayi;
    }
    private void btnBasla_Click(object sender, EventArgs e)
    {
        hak = 5;
        lblMesaj.Text = "...";
        btnTahmin.Enabled = true;
        uret();
    }
```

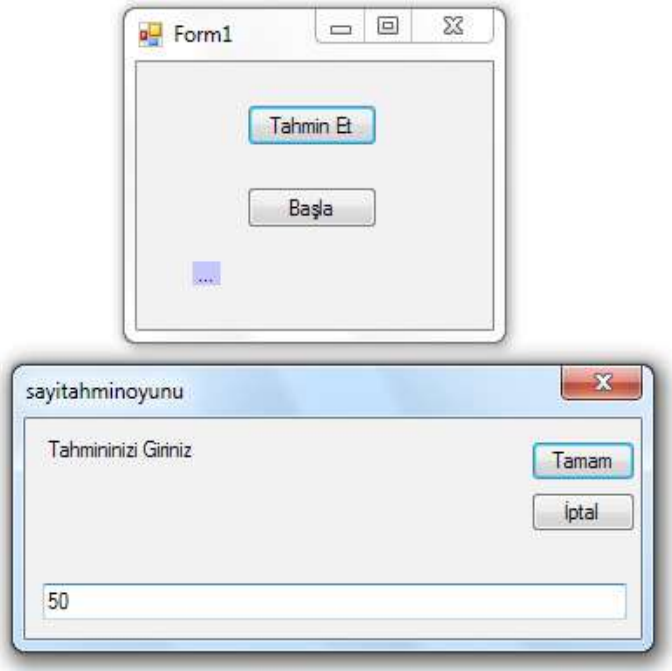
Programımızda **uret** alt programı ile rastgele sayı üretme işlemi yapılmaktadır.

```
Random rastgele = new Random();
```

Satırında rastgele adında bir değişken **random** olarak yani rastgele değişken tipi olarak tanımlanmaktadır.

```
int tutulansayi = rastgele.Next(1, 100);
```

tutulansayi değişkenine ise **1** ile **100** arasında üretilen rastgele tam sayı atanmaktadır.

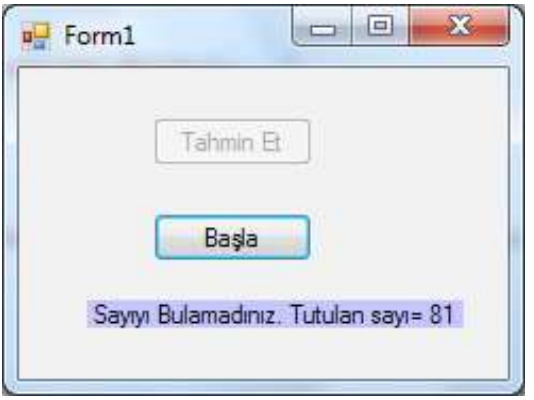


Başla düğmesine basıldığında `uret` alt programı çağrılarak programdan çıkmadan yeni bir rastgele sayı oluşturulması sağlanmaktadır. **Tahmin et** butonuna tıklandığında ekrana gelen giriş penceresi (`InputBox`) yardımıyla kullanıcı tahminini girmektedir.

`String` türde girilen değer `Convert.ToInt32` metodu ile `int32` türüne dönüştürülmektedir. Eğer girilen sayı `0` ile `100` aralığında ise girilen sayıyı kontrol eden kod bloğu çalıştırılmaktadır.



Eğer girilen tahmin yanlışsa daha küçük ya da daha büyük sayı girilmesi hakkında kullanıcı uyarılarak kaç tahmin hakkı kaldığı kendisine bildirilecektir.



Kullanıcı, sayıyı doğru tahmin etmişse **"Tebrik ederim!Bildiniz"** şeklinde mesaj görüntülenmektedir. Eğer kullanıcı 5 tahmin hakkında da bilemezse **"Sayıyı Bulamadınız."** mesajı görüntülenip program tarafından oluşturulan rastgele sayı görüntülenecek ve **btnTahmin** isimli buton **pasif** yapılacaktır.

SWITCH CASE YAPISI

switch case yapısı, verilen bir değişkenin aldığı değer ya da değer aralığına göre seçim yapmak için kullanılır. Kullanımı şu şekildedir:

```
switch (değişken)
{
   case değer:
   {
      Komutlar;
      break;
   }
   case değer:
   {
      Komutlar;
      break;
   }
   default
   {
```

```
        komutlar;
        break;
    }
}
```

switch komutundan sonra yazılan değişken **parantez** içerisinde yazılmalıdır. **case** ifadesinden sonra **switch** ile belirtilen değişkenin alabileceği değer yazılır. Her **case** bloğunda komut satırları icra edildikten sonra, çıkılmasını sağlamak için **break** ifadesi kullanılır. **default**'un görevi **if** yapısında kullanılan **else** ile aynıdır. Yani artık gerçekleşebilecek herhangi bir ihtimal kalmadığında kullanılır. Dolayısıyla **default** komutundan sonra herhangi bir değer yazılmaz.

Örnek olarak girilen puana göre **geçti** ve **kaldı** yazacak olan örneği **switch case** yapısı ile yazalım.

```
private void button1_Click(object sender, EventArgs e)
{
    byte notu;
    notu = Convert.ToByte(Interaction.InputBox("Notunuzu girin"));
    switch (notu)
    {
        case 0:
        {
            MessageBox.Show("Kaldınız.");
            break;
        }
  case 1:
        {
            MessageBox.Show("Kaldınız.");
            break;
        }
  case 2:
        {
            MessageBox.Show("Geçtiniz.");
            break;
        }
  case 3:
        {
            MessageBox.Show("Geçtiniz.");
            break;
        }
```

```
    case 4:
          {
              MessageBox.Show("Geçtiniz.");
              break;
          }
    case 5:
          {
              MessageBox.Show("Geçtiniz.");
              break;
          }
          default :
          {
              MessageBox.Show("Yanlış Not Girdiniz.");
              break;
          }
      }
}
```

Örneğimizde girilen nota karşılık gelen sonuç, mesaj penceresi içerisinde görüntülenmektedir. Eğer belirtilen not değerleri dışında bir not girişi yapılmışsa **default** ile bunun kontrolü yapılarak **"Yanlış Not Girdiniz"** şeklinde bir mesaj görüntülenmektedir.

Aynı işlemleri yapan **case** blokları arasında yönlendirme yapmak için **goto case** metodunu kullanabiliriz. Yukarıdaki örneğimiz **goto case** metodunu kullanarak yeniden düzenleyelim.

```
private void button1_Click(object sender, EventArgs e)
{
    byte notu;
    notu = Convert.ToByte(Interaction.InputBox("Notunuzu girin"));
    switch (notu)
    {
        case 0: MessageBox.Show("Kaldınız."); break;
        case 1: goto case 0;
        case 2: MessageBox.Show("Geçtiniz."); break;
        case 3: goto case 2;
        case 4: goto case 2;
        case 5: goto case 2;
        default:
```

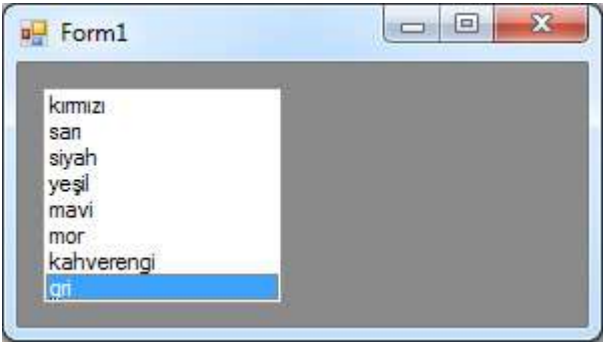
```
        {
            MessageBox.Show("Yanlış Not Girdiniz.");
            break;
        }
    }
}
```

Kodumuzda "**Kaldınız**" mesajı **case 0**, "**Geçtiniz**" mesajı ise **case 2** bloğu içerisinde kullanılmaktadır. "**Kaldınız**" sonucu verecek **case** blokları **goto case 0** ile "**Geçtiniz**" sonucu verecek **case** blokları ise **goto case 2** ile yönlendirilerek aynı kod satırlarının birden fazla tekrarlanması engellenmiş olur.

Bir **listBox** içerisinde seçilen renk ismine göre formun rengini değiştiren örnek bir uygulama yapalım.

```
private void listBox1_DoubleClick(object sender, EventArgs e)
{
    switch (listBox1.Text)
    {
        case "mavi": this.BackColor = Color.Blue; break;
        case "sarı": this.BackColor = Color.Yellow; break;
        case "siyah": this.BackColor = Color.Black; break;
        case "kırmızı": this.BackColor = Color.Red; break;
        case "yeşil": this.BackColor = Color.Green; break;
        case "kahverengi": this.BackColor = Color.Brown; break;
        case "mor": this.BackColor = Color.Purple; break;
        case "gri": this.BackColor = Color.Gray; break;
    }
}
```

listBox1 içerisinde belirtilen renk ismine göre **case** ile seçim yapıyoruz. **listBox**'ın **DoubleClick** olayına kodumuzu yazdığımız için **listBox** içerisinde **çift tıklandıkça** seçili olan renk değeri **Color.RenkİsmI** şeklinde ifade edilir. **this** ifadesi formu temsil etmektedir. this.BackColor ise formun zemin rengini belirtir. Seçili olan renk değeri formun **zemin rengi** olarak değiştirilir.



Pattern Matching

Pattern matching ile **if** ve **switch** yapılarını çok daha etkili bir biçimde kullanmak mümkündür. Daha gelişmiş karşılaştırma seçenekleri sağlarken karşılaştırılan nesneleri analiz etme, daha fazla ve doğru sonuçlar elde etmemizi sağlar. **C# 7** versiyonu ile birlikte **type pattern**, **constant pattern**, **var pattern** gibi temel **pattern matching** özellikleri kullanılmaya başlanılmışken **C# 8** versiyonundan itibaren **recursive pattern** ve bunun alt **pattern**'leri olan **positional pattern**, **property pattern** gibi gelişmiş **pattern** yapıları kullanılmaya başlanmıştır.

NOT Bu konu **if** ve **switch** kontrol yapıları konuları ile ilgili olduğu için bu bölümde ele alınmıştır. Bu konuyu daha iyi anlayabilmek için öncelikle kitabımızın 19. Bölümünde yer alan "Metotlar" konusunu okumanızı tavsiye ederiz.

- **Type pattern:** Bir değişkenin, bir tür ile karşılaştırılarak sonuç olumlu ise değişken değerinin bir başka değişkene aktarılması şeklinde kullanılır. Örneğin, **C#7** öncesi sürümlerde aşağıdaki biçimde yazdığımız form üzerindeki bir kontrolün türünü tespit edip ona göre özelliklerine atama yaptığımız kodu;

```
if (this.Controls[0] is Button)
{
    Control btn = this.Controls[0];
    btn.BackColor = Color.Aquamarine;
    btn.Text = "Buton";
}
else if (this.Controls[0] is TextBox)
{
```

```
    Control txt = this.Controls[0];
    txt.BackColor = Color.SteelBlue;
    txt.Text = "TextBox";
}
```

C# 7 sürümünden itibaren aşağıdaki gibi yazabiliriz:

```
if (this.Controls[0] is Button btn)
{
    btn.BackColor = Color.Aquamarine;
    btn.Text = "Buton";
}
else if (this.Controls[0] is TextBox txt)
{
    txt.BackColor = Color.SteelBlue;
    txt.Text = "TextBox";
}
```

Kodumuzda:

```
this.Controls[0] is TextBox txt
```

Satırında görüldüğü gibi form üzerinde `0` indeks numaralı kontrolün **`TextBox`** olup olmadığını kontrol edip eğer sonuç olumlu ise **`txt`** isimli değişkene bu kontrolü aktardık.

- **Constant pattern:** Karşılaştırma işleminin bir **sabit** ile yapılmasıdır.

```
object sayi = 2;
if (sayi is int i)
    Console.WriteLine(i + 3);
else if (sayi is null)
    Console.WriteLine("Boş değer");
else
    Console.WriteLine("Sayi int değil");
```

Örneğimizde değişkeni bir tür ile değil de bir sabit ile karşılaştırıyoruz.

```
if (sayi is int i)
    Console.WriteLine(i + 3);
```

Satırında görüldüğü gibi **sayi** değişkeninin türünü int ile karşılaştırarak eğer türü **int** ise **sayi** değişkeninin değerini **i** değişkenine aktarıyoruz. **i** değişkeninin değerine **3** ekleyerek konsol ekranında yazdırıyoruz.

Ayrıca kodumuzda görüldüğü gibi değişken değerini **null** sabiti ile de karşılaştırabiliriz.

Var pattern: Değişken değeri **var** ile elde edilerek değeri bir başka değişkene aktarılır.

```
object obj = "deneme";
if (obj is var str)
{
    Console.WriteLine(str);
}
```

Örneğimizde **obj** değişkeninin değerini **var** ile elde ederek **str** değişkenine aktarıyor ve **str** değişkenin içeriğini konsol ekranında yazdırıyoruz.

Pattern Matching'in switch-case Yapisi İçerisinde Kullanimi

Pattern matching özelliğini kullanarak **switch** – **case** yapısını daha esnek ve daha etkili bir biçimde kullanabiliriz.

```
object deger = null;
Console.WriteLine("Notunuzu Giriniz:");
deger = Convert.ToInt32(Console.ReadLine());
switch (deger)
{
    case int a when a<=0 :
        Console.WriteLine("0'dan büyük bir değer giriniz!");
        break;
    case int b when b<50:
        Console.WriteLine("Kaldınız!");
        break;
    case int c when c>=50:
        Console.WriteLine("Geçtiniz!");
        break;
}
Console.ReadLine();
```

Örnek uygulamamızda "**deger**" isimli değişkenin içeriğine göre seçim yaparak, değişkenin türünü **int** ile karşılaştırarak eğer türü **int** ise değişkenin içeriğini bir başka değişkene aktarıp sonucuna göre işlemleri gerçekleştiriyoruz.

```
case int a when a<=0 :
Console.WriteLine("0'dan büyük bir değer giriniz!");
```

Satırında görüldüğü gibi girilen değerin türünü **int** ile karşılaştırarak **eğer** türü **int** ise değerin içeriğini **a** değişkenine aktararak eğer değer 0'dan küçükse "**0'dan büyük bir değer giriniz**" ifadesi konsol ekranında yazdırılacaktır.

Recursive pattern: Recursive pattern'i kısaca bir **pattern** ifadesinin başka bir **pattern** ifadesinin sonucuna uygulanması şeklinde tanımlayabiliriz. **Positional pattern** ve **property pattern** şeklinde iki farklı uygulaması mevcuttur.

- **Positional pattern:** Giriş değerinin boş olup olmadığını kontrol ederek eğer boş değilse kod içerisinde tanımlanmış olan bir **Deconstruct** metodunu çağırarak elde edilen sonuç değerleri içerisinde, daha fazla **pattern** karşılaştırmayı sağlar.

  Örnek uygulamamız aşağıdaki gibi olacak:

```
class Program
{
  public class Personel
  {
    public string Ad { get; set; }
    public string Dal { get; set; }
    public int Yas { get; set; }
    public bool Ogretmen { get; set; }
    public void Deconstruct(out string ad, out string dal,
out int yas, out bool ogretmen)
    {
      ad = Ad;
      dal = Dal;
      yas = Yas;
      ogretmen= Ogretmen;
    }
  }
  static void Main(string[] args)
  {
```

```
    var p1 = new Personel {Ad = "Volkan AKTAŞ",
Dal = "Veri Tabanı Programcılığı",
      Yas = 43, Ogretmen = true};

    if (p1 is Personel(_, _, _, true) p)
    {
        Console.WriteLine($"Personel: {p.Ad}");
    }
    Console.ReadLine();
  }
}
```

Kodumuzda **`Personel`** adında bir sınıf tanımladık. Bu sınıfın içerisinde **`Deconstruct`** bir metot tanımladık.

```
if (p1 is Personel(_, _, _, true) p)
```

Satırında öncelikle:

```
p1 is Personel
```

İfadesi, **`p1`** değişkeni ile tanımlanmış olan personel bilgilerini **`Personel`** türü karşılaştırılarak ilk pattern eşlemesi yapılır. Eğer başarılı olursa **`p1`** değişkeni Personel türüne dönüştürülür ve **`out`** parametrelerini almak için **`Deconstruction`** metot ile oluşturulan değişkenlerle çağrılır. Alınan her değer:

```
(_, _, _, true)
```

Kısmında karşılık gelen **alt pattern** ile karşılaştırılır. Eğer hepsi başarılı olursa karşılaştırma sonucu başarılı olacaktır.

Uygulamamızı çalıştırdığımızda **sonuç** olarak:

Personel: Volkan AKTAŞ

Şeklinde **çıktı** verecektir.

- **Property pattern:** Giriş değerinin boş olup olmadığını kontrol ederek **rekursif** olarak erişilebilen özellik ve alanları karşılaştırmak amacıyla kullanılır.

Örnek uygulamamız aşağıdaki gibi olacak:

```
class Program
{
  public class Personel
  {
    public string Ad { get; set; }
    public string Dal { get; set; }
    public int Yas { get; set; }
    public bool Ogretmen { get; set; }
    public void Deconstruct(out string ad, out string dal,
out int yas, out bool ogretmen)
    {
      ad = Ad;
      dal = Dal;
      yas = Yas;
      ogretmen= Ogretmen;
    }
  }
  static void Main(string[] args)
  {
    var p1 = new Personel {Ad = "Volkan AKTAŞ",
Dal = "Veri Tabanı Programcılığı",
      Yas = 43, Ogretmen = true};

    if (p1 is Personel { Ad: "Volkan AKTAŞ", Ogretmen: true })
    {
        Console.WriteLine($"Personel: {p1.Dal}");
    }
    Console.ReadLine();
  }
}
```

Kodumuzda daha önceki positional pattern örneğimizde olduğu gibi **`Personel`** adında bir sınıf tanımladık. Bu sınıfın içerisinde **`Deconstruct`** bir metot tanımladık.

```
if (p1 is Personel { Ad: "Volkan AKTAŞ", Ogretmen: true })
```

Satırında öncelikle:

```
p1 is Personel
```

İfadesi ile **`p1`** değişkeni ile tanımlanmış olan personel bilgilerini Personel türü karşılaştırılarak ilk pattern eşlemesi yapılır. Eğer başarılı olursa **`p1`** değişkeni **`Personel`** türüne dönüştürülür ve **`out`** parametrelerini almak için **`Deconstruction`** metot ile oluşturulan değişkenlerle çağrılır.

Alınan her değer:

```
{ Ad: "Volkan AKTAŞ", Ogretmen: true }
```

Kısmında karşılık gelen alt pattern ile karşılaştırılır. Burada parametrelere aktarılan değerler ile karşılaştırma yapılarak karşılaştırma işleminde eğer belirtilen parametre değerlerinin karşılaştırma işlemi başarılı olursa karşılaştırma sonucu **başarılı** olacaktır.

Uygulamamızı çalıştırdığımızda **sonuç** olarak:

Personel: Veri Tabanı Programcılığı

Şeklinde **çıktı** verecektir.

NOT: Yukarıdaki kod satırlarında

```
var p1 = new Personel {Ad = "Volkan AKTAŞ",
Dal = "Veri Tabanı Programcılığı",
      Yas = 43, Ogretmen = true};
```

Şeklinde yaptığımız tanımlamayı C# 9 sürümünden itibaren yeni bir **.Net Core Console Application** projesi oluşturup

```
Personel p1 = new() {Ad = "Volkan AKTAŞ",
Dal = "Veri Tabanı Programcılığı",
      Yas = 43, Ogretmen = true};
```

Şeklinde yapabiliriz. Bu şekilde tür bildirimlerini sol tarafta güzelce hizalanmış biçimde yazabilir ve **constructor metot** parametrelerine değer geçebilirsiniz.

C# 9 sürümünden itibaren ?? (**Null Coalescing**) operatörünü farklı veri türleri ile kullanabilirsiniz. Örnek olarak yukarıdaki kod satırlarını aşağıdaki gibi değiştirelim:

```
class Program
{
  public class Personel
  {
    public string Ad { get; set; }
    public string Dal { get; set; }
    public int Yas { get; set; }
    public bool Ogretmen { get; set; }
    public void Deconstruct(out string ad, out string dal,
out int yas, out bool ogretmen)
    {
      ad = Ad;
      dal = Dal;
      yas = Yas;
      ogretmen= Ogretmen;
    }
  }
  static void Main(string[] args)
  {
    Personel p1 = new()
    {
        Ad = "Volkan AKTAŞ",
        Dal = "Veri Tabanı Programcılığı",
        Yas = 43,
        Ogretmen = true
    };
    Personel p2 = new()
    {
        Ad = "Bünyamin KARAHAN",
        Dal = "Teknik Servis",
        Yas = 38,
        Ogretmen = true
    };
```

```
    Personel p = p1 ?? p2;

    Console.WriteLine($"Personel: {p.Dal}");
    Console.ReadLine();
  }
}
```

`??` (**Null Coalescing**) operatörü kullanımında ilk değişken `null` ise ikinci değer hesaplanır. Aksi takdirde ikinci değişken işleme konulmaz. Kodumuzu çalıştırdığımızda `p1` değişkeninin değeri `null` olmadığından dolayı `p` değişkenine `p1` değişkenini aktaracak ve `p1` değişkeninin `Dal` parametresinin değeri olan "`Veri Tabanı Programcılığı`" değeri görüntülenecektir.

Generic Pattern Matching

Generic metotları, `pattern matching` özelliği ile birlikte kullanarak `type pattern`'i daha etkili bir biçimde kullanabiliriz. Örnek olarak konsol ekranında girilen bir değerin tek mi çift mi olduğunu `generic` metot kullanarak bulan bir uygulama yapalım.

> **NOT** Bu konu Pattern Matching konusu ile ilgili olduğu için bu bölümde ele alınmıştır. Bu konuyu daha iyi anlayabilmek için öncelikle kitabımızın 19. Bölümünde yer alan "Generic Metotlar" konusunu okumanızı tavsiye ederiz.

```
static void Main(string[] args)
{
  Console.WriteLine("Sayıyı Giriniz:");
  int? sayi = int.TryParse(Console.ReadLine(),
      out int s) ? s : default(int?);
  Console.WriteLine(sayikontrol(sayi));
  Console.ReadLine();
}
public static string sayikontrol<T>(T sayi)
{
  string mesaj="";
  switch (sayi)
  {
    case int s when s <= 0 :
      mesaj = "0'dan büyük bir değer giriniz!";
      break;
```

```
    case int cift when (cift % 2) == 0:
      mesaj = "Sayı Çift";
      break;
    case int tek when (tek % 2) == 1:
      mesaj = "Sayı Tek";
      break;
    default:
      mesaj = "Boş değer!";
      break;
  }
  return mesaj;
}
```

Kodumuzda "**sayikontrol**" isimli metodumuz **Type** türünde kendisine gönderilen değerleri kabul etmektedir. Metot içerisinde:

```
case int cift when (cift % 2) == 0:
case int tek when (tek % 2) == 1:
```

Satırlarında **sayi** isimli değişkenin içeriğine göre seçim yaparak değişkenin içeriğinin **2** ile bölümünden kalan sonucuna göre **mesaj** değişkeninin içeriğine "**Sayı Tek**" ya da "**Sayı Çift**" değerini göndermektedir. Ayrıca:

```
case int s when s <= 0 :
```

Satırında sayının değeri **0**'dan büyükse **mesaj** değişkeninin içeriğine "**0'dan büyük bir değer giriniz!**" değerini göndermektedir. **default** ifadesiyle de geriye başka bir ihtimal kalmadığı zaman **mesaj** değişkenine "**Boş Değer!**" ifadesini gönderiyoruz. "**mesaj**" değişkeninin içeriğini metodumuzda:

```
return mesaj;
```

Satırı ile geriye döndürüyoruz. **Main** metot içerisinde öncelikle:

```
int? sayi = int.TryParse(Console.ReadLine(), out int s) ? s :
default(int?);
```

Satırında konsol ekranında **Console.ReadLine()** metodu ile okuttuğumuz değeri **int.TryParse()** metodu ile **int** türüne dönüştürdük. **TryParse()**

metodu dönüşüm esnasında oluşabilecek bütün hata durumlarında geriye herhangi bir hata döndürmeyip **sıfır** değerini döndürür. Eğer herhangi bir hata durumu oluşmazsa başında **out** ifadesi bulunan parametreye sonuç değeri aktarılır. **?** (Conditional - Koşul) operatörü kullandığımız için konsol ekranında **int** türünde değer girilir ve dönüşüm işlemi başarılı olursa işlem sonucu **s** değişkenine, eğer **null** değer girilirse; **default(int?)** ifadesi ile **int** türünün **default** değeri olan 0 geriye döner.

Böylece dönüşüm işleminde boş değer girilirse uygulamamızın hata vermesini önlemiş oluruz. "**sayikontrol**" isimli metodumuzu **Main** metot içerisinde:

```
sayikontrol(sayi)
```

Şeklinde çağırarak işlem sonucunu konsol ekranına yazdırıyoruz.

Switch Expressions

C# 7 sürümünden itibaren gelen **pattern matching** ile **switch** yapılarını daha etkili bir biçimde kullanabildiğimizi yukarıda belirtmiştik. **C# 8** sürümünden itibaren de **switch expressions** yapısını kullanarak **switch** yapılarını daha kısa ve etkili bir biçimde kullanabiliriz.

```
switch yapısını
switch (değişken)
{
   case değer:
   {
      Komutlar;
      break;
   }
   default
   {
        komutlar;
        break;
   }
}
```

Aşağıdaki gibi **switch expressions** yapısını kullanarak:

```
değişken türü metod adı = değişken adı switch
{
   _ when değişken adı koşul => komutlar,
   _ => default durumundaki komutlar
};
```

Şeklinde kısaltabiliriz. switch expressions yapısını kullanırken **local function** (*yerel metot*) yapısından faydalanıyoruz. Local Function, **C# 7** versiyonundan itibaren gelen yeniliklerden biridir. Haricî bir metot ya da sınıf içerisinden erişilmeyecek metotlarımızı herhangi bir metodun içerisinde tanımlayabiliriz. Bu şekildeki bir metot, ancak tanımlandığı metodun içerisinde kullanılacaktır. Bu konu hakkında detaylı bilgi, kitabımızın **19. Bölümü** olan Metotlar bölümü **Local Function** konusunda mevcuttur. Bu bölüme başvurabilirsiniz. Burada yalnızca konu ile ilgili olması nedeniyle kısaca değinilmiştir.

Normal **`switch`** ifadesi kullanımı ile karşılaştırdığımızda:

```
case ifade:
```

yerine,

```
_ when değişken adı koşul =>
```

yazımının,

`default` ifadesi yerine ise **`_ =>`** yazımının kullanıldığını görüyoruz.

Örneğin **pattern matching** konusunda:

```
object deger = null;
Console.WriteLine("Notunuzu Giriniz:");
deger = Convert.ToInt32(Console.ReadLine());
switch (deger)
{
    case int a when a<=0 :
        Console.WriteLine("0'dan büyük bir değer giriniz!");
        break;
    case int b when b<50:
```

```
        Console.WriteLine("Kaldınız!");
        break;
    case int c when c>=50:
        Console.WriteLine("Geçtiniz!");
        break;
}
Console.ReadLine();
```

Biçiminde yapmış olduğumuz uygulamayı:

```
var notu =0;
Console.WriteLine("Notunuzu Giriniz:");
notu = Convert.ToInt32(Console.ReadLine());
string notdurumu = notu switch
{
    _ when notu < 50 & notu>=0 => "Kaldınız!",
    _ when notu >= 50 & notu<=100 => "Geçtiniz!",
    _ => "Geçersiz not!"
};
Console.WriteLine("Not Durumu:"+notdurumu);
Console.ReadLine();
```

Biçiminde **switch expressions** ile yazıyoruz. Kodumuzda her bir koşul durumundan sonra **=>** ifadesini eklerken her bir koşul diğerinden , (*virgül*) işareti ile ayrılıyor. Normal **`switch`** yapısındaki **`default`** durumunu ise kodumuzda görüldüğü gibi:

```
_ => "Geçersiz not"
```

Biçiminde herhangi bir şart ifadesi yazmadan yazıyoruz ve yukarıda belirtilen şartlardan farklı bir durum oluştuğunda **`notdurumu`** değişkenine **`"Geçersiz not"`** metninin aktarılmasını sağlıyoruz.



C# 9 sürümünden itibaren not, and, or gibi mantıksal operatörleri ve <, >, <=, >= gibi ilişkisel operatörleri **is** operatörü ile birlikte **pattern** ifadeleri içerisinde kullanabilirsiniz.

Örnek olarak yeni bir **.Net Core Console Application** projesi oluşturup bir önceki kodumuzda değişiklik yaparak girilen notu 5'lik puan sistemine çeviren bir uygulama yapalım.

```
Console.WriteLine("Notunuzu Giriniz:");
int notu = Convert.ToInt32(Console.ReadLine());
string notdurumu = notu switch
{
    _ when notu is > 0 and < 25 => "Sıfır",
    >= 25 and < 45 => "Bir",
    >= 45 and < 55 => "İki",
    >= 55 and < 70 => "Üç",
    >= 70 and < 85 => "Dört",
    >= 85 and <= 100 => "Beş",
    _ => "Geçersiz not!"
};
Console.WriteLine("Not Durumu:" + notdurumu);
Console.ReadLine();
```

Kodumuzda **and** mantıksal operatörünü kullanarak girilen not belirtilen aralıklarda olduğunda **=>** ilişkisel operatörü ile sonucun **notdurumu** değişkenine aktarılarak **ekrana yazdırılmasını sağlıyoruz**.

DÖNGÜLER

7

Programlama dillerinde birden fazla tekrarlanması gereken işlemleri döngüler vasıtası ile gerçekleştiririz. Döngüler gerçekleştirilmesi gereken işlemleri hem daha hızlı, hem de daha az kod satırları ile gerçekleştirmemizi sağlar.

C# içerisinde

- **for** döngüsü
- **foreach** döngüsü
- **while** döngüsü olmak üzere 3 çeşit döngü kullanabiliriz.

FOR DÖNGÜSÜ

Bir ya da birden fazla kod satırının belirtilen şarta göre, belirtilen artış ya da azalış değeri kadar tekrarlanmasını sağlayan döngüdür.

Kullanımı şu şekildedir:

```
for (sayaç değişkeni ; şart ; artış ya da azalış değeri)
{
 Komutlar;
}
```

Şart sağlandığı sürece **for** döngüsü içerisindeki komut satırları, artış ya da azalış değeri adedince tekrarlanır.

Örneğin; 0 ile 10 arasındaki sayıları ekranda listeleyen bir konsol uygulaması yazalım.

```
static void Main(string[] args)
{
 int i;
 for (i = 0; i <= 10; i++)
 {
  Console.WriteLine(i);
 }
 Console.ReadLine();
}
```



Döngümüz 0'dan 10'a kadar kurulduğu için i değişkeninin değeri ekrana 10 defa artan şekilde yazılacaktır. **Console.Readline** ifadesi ile de ekran çıktısını **Enter** tuşuna basılana kadar bekletiyoruz.

Döngü ifadesinden sonra kullanacağımız komutlar, tek satırdan oluşuyorsa parantez kullanmadan da komut satırını yazabiliriz.

Örneğin yukarıdaki örneğimizde…

```
for (i = 0; i <= 10; i++)
{
 Console.WriteLine(i);
}
```

… şeklinde döngü ifadesinden sonra parantez kullanarak yazdığımız kodu,

```
for (i = 0; i <= 10; i++)
 Console.WriteLine(i);
```

... şeklinde parantez kullanmadan da yazabiliriz. Her iki kod da aynı sonucu verecektir.

0'dan 10'a kadar tüm sayıları yazan uygulamamızı, yalnızca çift sayıları görüntüleyecek şekilde değiştirelim.

```
static void Main(string[] args)
{
 for (int i = 0; i <= 10; i+=2)
 {
  Console.WriteLine(i);
 }
 Console.ReadLine();
}
```



Görüldüğü gibi `i++` şeklindeki artış değerimizi `i+=2` yaptık. Ekran çıktımız aşağıdaki gibi oldu:

Döngü ifadesi içerisinde kullandığımız değişkeni, yukarıdaki örneğimizde görüldüğü gibi döngüden önce tanımlamak yerine, döngü içerisinde de tanımlayabiliriz (`for (int i = 0; i <= 10; i+=2`) şeklinde olduğu gibi).

`10`'dan geriye doğru `0`'a kadar azalarak, konsol ekranına yazdıracak olan kodu yazalım.

```
static void Main(string[] args)
{
 for (int i = 10; i >= 0; i--)
 {
  Console.WriteLine(i);
 }
 Console.ReadLine();
}
```



Kodumuzda döngümüzün başlangıç değerini `10` olarak belirttik. Döngü değişkenimiz `10`'dan geriye doğru, döngü değişkeni `0`'dan büyük ve eşit olduğu müddetçe azalacaktır.

Yine `0` ile `10` arasındaki sayıları yazacak olan ama 5 sayısını atlayarak yazmayacak bir kod yazalım. Bunu iki şekilde yapabiliriz.

İlkinde…

```
static void Main(string[] args)
{
 for (int i = 0; i <= 10; i++)
 {
  if (i == 5 ) i ++;
  Console.WriteLine(i);
 }
 Console.ReadLine();
}
```

… döngümüzü `0`'dan `10`'a kadar birer birer artacak şekilde kurduk. `i` değişkeninin değeri `5` olduğunda `i` değişkeninin değerini bir arttırarak, değişkenin içeriğinin yazılmadan bir arttırılmasını sağladık.



Diğer yöntemde ise **`continue`** metodunu kullanarak, döngünün bir sonraki adımdan ilerletilmesini sağlarız.

```
static void Main(string[] args)
{
 for (int i = 0; i <= 10; i++)
 {
  if (i == 5 ) continue ;
  Console.WriteLine(i);
 }
 Console.ReadLine();
}
```

Kodumuzda `i`, `5` değerine ulaştığında ekrana yazdırma işlemi yapılmadan, döngünün bir sonraki adıma atlanması sağlanmıştır.

Klavyeden girilen bir sayının faktöriyelini bulacak bir örnek uygulama yapalım.

Örneğin; `4` sayısının faktöriyeli matematikte şu şekilde bulunur:

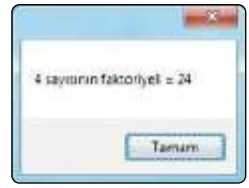
`4*3*2*1=24`

`4` sayısının faktöriyeli `24`'tür. Buna göre programımızda `1`'den klavyeden girilen

sayıya kadar bir döngü oluşturacağız ve döngü değeri ile çarpım değerini her defasında yeniden çarpacağız. Kodumuz şu şekilde olacak:

```
private void button1_Click(object sender, EventArgs e)
{
 int i,sayi,fak=1;
 sayi = Convert.ToInt32(Interaction.InputBox("Sayıyı Girin"));
 for (i = 1; i <= sayi; i++)
 {
  fak = fak * i;
 }
 MessageBox.Show (sayi + " sayısının faktöriyeli = " + fak);
}
```

Kodumuzu çalıştırıp **InputBox** ile **4** sayısını girdiğimizde, yandaki gibi çıktı alırız.



Çalışmakta olan bir döngüden döngü sonuna gelmeden çıkmak için **break** metodu kullanılabilir. Örneğin; konsol ekranında çalışmakta olan bir döngüden bir tuşa basılınca çıkacak olan bir uygulama yapalım.

```
static void Main(string[] args)
{
 int i;
 for (i = 1; i <= 10000; i++) {
  if (Console.KeyAvailable == false)
  {
   Console.WriteLine(i);
  }
  else
  {
   break;
  }
  if (i == 10000) i = 1;
 }
}
```

Kodumuz **1** ile **10000** arasında sürekli olarak saymaktadır.

Çünkü…

```
if (i == 10000) i = 1;
```

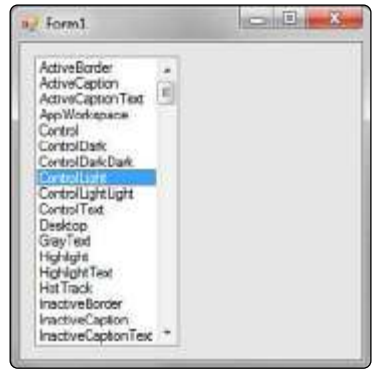
… satırı ile döngü içerisinde iken, **i** değeri **10000** olduğunda **1** yaparak tekrar başa dönülmesini sağladık.

```
if (Console.KeyAvailable == false)
{
 Console.WriteLine(i);
}
else
{
 break;
}
```

Satırları ile de konsol ekranında basılan bir tuş yoksa, konsol ekranına **i** değişkeninin değerinin yazılmasını, basılan bir tuş varsa da **break** metodu ile döngüden çıkılmasını sağladık.

Bir **listBox** içerisinde listelenen renk isimlerine tıklandıkça, formun zemin rengini değiştirecek örnek bir uygulama yapalım. **Form** üzerine 1 adet **listBox** ekleyerek kodumuzu yazıyoruz:

```
private void listBox1_DoubleClick(object sender, EventArgs e)
{
 this.BackColor =
  System.Drawing.ColorTranslator.FromHtml(listBox1.Text);
}
private void Form1_Load(object sender, EventArgs e)
{
 KnownColor renk = default(KnownColor);
 for (renk = KnownColor.ActiveBorder; renk <= KnownColor.YellowGreen;
  renk++)
 {
  listBox1.Items.Add(renk);
 }
}
```



Formun **Load** olayında, form yüklenirken **KnownColor** özelliği ile sistem renklerini elde ederek **for** döngüsü kuruyoruz. Döngümüzü **KnownColor** özelliği elemanlarının ilk elemanı olan **ActiveBorder** elemanından son elemanı olan **YellowGreen** elemanına kadar kuruyoruz. **listBox** içerine döngü içerisinde sistem renklerini tek tek ekliyoruz.

listBox nesnesinin **DoubleClick** olayında **listBox** içerisinde seçili olan renk ismini **ColorTranslator.FromHTML** metodu ile sistem rengine dönüştürerek formun zemin rengine atıyoruz.

İç içe birden fazla **for** kullanarak döngü kurabilirsiniz. Böyle bir durumda öncelikle içteki döngü bitecek, ardından dıştaki döngü çalışacaktır.

```
static void Main(string[] args)
{
 for (int i = 1; i <= 3; i++)
 {
  for (int j = 1; j <= 2; j++)
  {
   Console.WriteLine("Komut {0}", j);
  }
 }
 Console.ReadLine();
}
```

Kodumuzda döngümüz toplam 6 defa (`i` döngüsü 3 defa * `j` döngüsü 2 defa) çalışacaktır. Kodumuzun çalışması sırasıyla şu şekilde olacak:

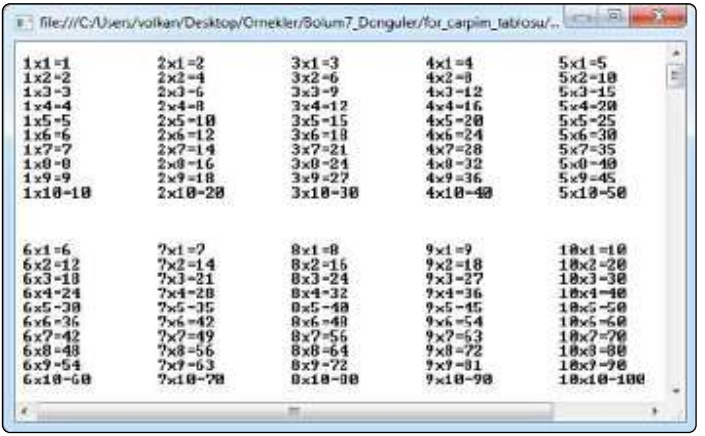


- `i` döngüsü 1 değerini alacak.
- İçteki `j` döngüsü 2 defa çalışacak ve `j` döngüsünden çıkılıp `i` döngüsüne dönülecek.
- `i` döngüsü 1 artarak 2 olacak.
- İçteki `j` döngüsü 2 defa çalışacak ve çıkılıp `i` döngüsüne dönülecek.
- `i` döngüsü 3 olacak.
- İçteki `j` döngüsü tekrar 2 defa çalışacak ve döngü sonlanacaktır.

Konsol ekranına 1'den 10'a kadar olan çarpım tablosunu yazdıracak bir uygulama yapalım.

```
static void Main(string[] args)
{
 int sutun=1;
 int satir=0;
 for (int i=1; i<=10;i++)
 {
  for (int j=1;j<=10;j++)
  {
   Console.SetCursorPosition(sutun,satir+j);
   Console.Write(i +"x" + j +'=' + (i*j));
  }
  sutun=sutun+14;
  if (i==5)
  {
   sutun=1;
   satir=13;
  }
 }
 Console.ReadLine();
}
```

Kodumuzda iç içe iki adet **for** döngüsü kuruyoruz. **j** döngüsü bir sütunda sırasıyla 1'den 10'a kadar çarpma işleminin yapılmasını sağlarken, **i** döngüsü ise sırasıyla 1'ler, 2'ler şeklinde 10'a kadar 10 ayrı sütun halinde işlem yapılmasını sağlar.



1'den 5'e kadar olan çarpım işlemlerinde j döngüsünün her bitiminde sütun değeri 14 arttırılarak, sırasıyla sütunlar halinde yazdırılır. i döngüsü 5 olduğunda, sütun 1, satır ise 13 olarak 6'dan 10'a kadar olan çarpım işlemleri alt sırada yazdırılır.

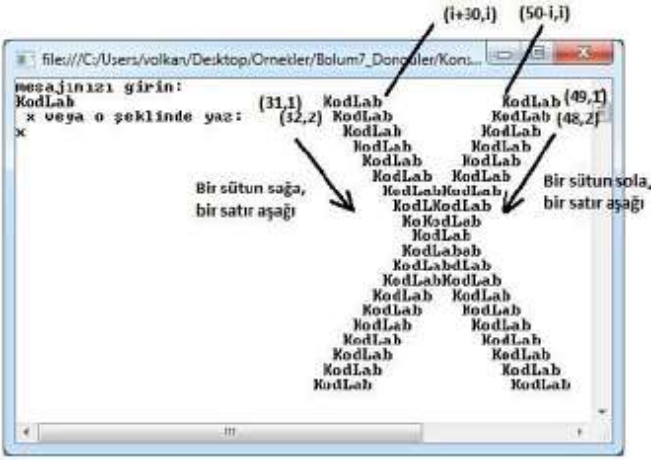
Kodumuzda imlecin belirtilen koordinata konumlanarak satır ve sütunların kaydırılması işlemini **SetCursorPosition** metodu ile yapıyoruz.

Konsol ekranından girilen bir mesajı o veya **x** şeklinde yazan bir uygulama yapalım.

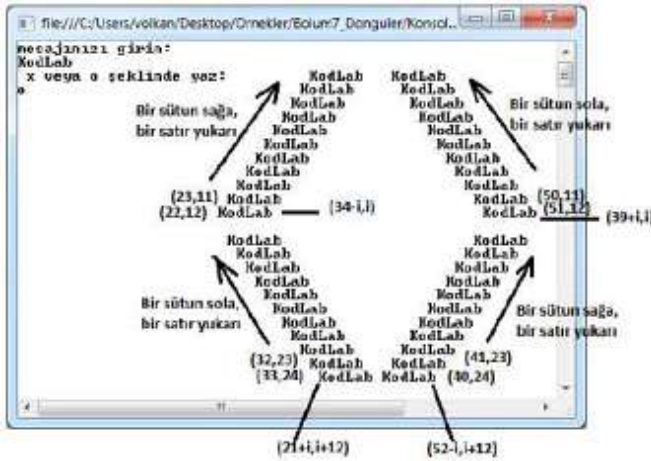
```
static void Main(string[] args)
{
 string m, a;
 int i;
 Console.WriteLine("mesajınızı girin:");
 m = Console.ReadLine();
 Console.WriteLine(" x veya o şeklinde yaz:");
 a = Console.ReadLine();
 if (a == "x")
 {
  for (i = 1; i <= 20; i++)
  {
   Console.SetCursorPosition(i + 30, i);
   Console.Write(m);
   Console.SetCursorPosition(50 - i, i);
   Console.Write(m);
  }
 }
 else if (a == "o")
 {
  for (i = 12; i > 1; i--)
  {
   Console.SetCursorPosition(34 - i, i);
   Console.Write(m);
   Console.SetCursorPosition(39 + i, i);
   Console.Write(m);
   Console.SetCursorPosition(21 + i, i + 12);
   Console.Write(m);
   Console.SetCursorPosition(52 - i, i + 12);
```

```
    Console.Write(m);
   }
  }
  Console.ReadLine();
}
```

Uygulamamızda kullanıcının klavyeden girdiği mesaj, yine kullanıcının yaptığı tercihe göre **x** veya **o** şeklinde görüntülenmektedir.



Uygulamamızda **x** şeklindeki yazımda, mesaj en üst satırdan yazdırılmaya başlanır. Aynı anda bir satıra iki sütun halinde yazdırma işlemi yapılır. Her seferinde bir satır aşağı inilirken, **x**'in sol üst köşesinden itibaren sağa doğru birer birer kaydırılır, **x**'in sağ üst köşesinden itibaren de sola doğru birer birer kaydırılır.



Uygulamamızda `o` şeklindeki yazımda ise, `o` şeklinin üst yarısı ortadan itibaren yazdırılmaya başlanır. Yukarıya doğru birer satır azaltılırken, sol taraf sağa doğru, sağ tarafta sola doğru kaydırılır. Alt yarısı ise en alt taraftan itibaren yazdırılmaya başlanır. Yukarıya doğru birer satır azaltılırken, sol taraf sola doğru, sağ tarafta sağa doğru artarak yazdırılır. Aynı anda iki satıra ikişer sütun halinde yazdırma işlemi yapılır.

`for` döngüsünde kullanılan bütün ifadelerin kullanımı isteğe bağlıdır. İfadeleri aşağıdaki şekillerde kullanabilirsiniz.

NOT **`for`** döngüsünü **`for( ; ; )`** şeklinde, herhangi bir ifade yazmadan kullanırsanız, döngümüz sürekli olarak tekrarlanacaktır.

```
for ( ; ; )
{
 Console.WriteLine("C#");
}
```

Yukarıdaki döngümüz, konsol ekranında sürekli olarak "*C#*" mesajını tekrarlayacaktır.

Şart ifadesi yazılmadığında varsayılan değeri olan, **`true`** kabul edilecektir.

```
for ( ; ; )
```

… döngüsü ile…

```
for ( ;true ; )
```

… döngüsü, aynı anlama gelmekte olup, her ikisi de sürekli olarak çalışacaktır.



Döngü tanımlamasındaki herhangi bir ifadeyi, tanımlama satırı dışında yazarak da kullanabiliriz.

```
for (int i=0;i<10; )
{
 i++;
 Console.WriteLine("C#");
}
```

Bu kullanımda görüldüğü gibi döngünün artım değerini döngü tanımlanırken değil de, daha sonra döngü içerisinde verdiğimizi zaman döngümüz doğru bir şekilde çalışarak, ekrana 10 defa mesajımızı yazacaktır.

```
int i;
i=0;
for ( ;i<10; )
{
 i++;
 Console.WriteLine("C#");
}
```

Yukarıdaki kullanımda görüldüğü gibi sayaç değişkenini döngü dışında tanımlayıp döngümüzü doğru bir biçimde çalıştırabiliriz. Ya da…

```
int i;
for (i=0;i<10; )
{
 i++;
 Console.WriteLine("C#");
}
```

… kullanımında olduğu gibi sayaç değişkenini döngü dışında tanımlayıp, ilk değerini döngü tanımlama satırında yapabiliriz.



Döngü tanımlamasını, birden fazla değişken ya da şart ifadesi kullanarak yapabilirsiniz.

```
int i;
int j;
for (i = 0, j=0; (i < 10 & j < 5);i++,j++ )
{
 Console.WriteLine("C#");
}
```

Yukarıdaki kodumuzda **i** ve **j** olmak üzere iki adet sayaç değişkeni kullandık. Sayaç değişkenlerini birbirinden virgül ile ayırıyoruz. Şart ifadesi olarak da (**i<10**) ve (**j<5**) olmak üzere iki adet şart ifadesini aralarına **&** (*and*) operatörünü ekleyerek kullandık. Döngü artım değerlerini de yine döngü tanımlama satırında aralarını virgül ile ayırarak yazdık. Uygulamamızda **&** operatörünü kullanmamız nedeniyle şartlardan herhangi biri sağlandığında döngümüz sonlanacaktır. **j** döngüsü **5** değerine daha çabuk ulaşarak, **i** döngüsünden önce sonlanacağı için, uygulamamızın sonucunda konsol ekranına 5 defa "*C#*" mesajı yazdırılacaktır.

FOREACH DÖNGÜSÜ

Listeler ya da diziler üzerinde işlem yapmak için kullanılan döngü yapısıdır. Özellikle eleman sayısının bilinmediği durumlarda kullanılabilmesi nedeni ile çok kullanışlıdır.

Kullanımı şu şekildedir:

```
foreach (Değişken Tipi Değişken Adı in Liste Ya da Dizi)
{
 Komutlar;
}
```

Örnek olarak bir dizinin elemanlarını listeleyen bir uygulama yapalım.

```
static void Main(string[] args)
{
 string[] adsoyad = { "Volkan AKTAŞ", "Ayşe AKTAŞ", "Talha Osman
AKTAŞ", "Sümeyye AKTAŞ" };

 foreach (string eleman in adsoyad)
 {
 Console.WriteLine(eleman);
 }
 Console.ReadKey();
}
```

Örneğimizde **foreach** döngüsü ile **adsoyad** dizisinin elemanlarını alarak **eleman** adlı değişkene atayıp, döngü içerisinde eleman değişkeninin içeriği yazdırılmaktadır.

foreach döngüsünden sıklıkla bir Form üzerindeki kontrollere erişmek amacı ile faydalanılmaktadır. Örneğin; bir Form üzerinde bulunan **textBox** nesnelerine erişerek onların içeriğini temizleyen, zemin ve yazı renklerini değiştiren bir uygulama yapalım.

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace foreach_kontroller
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
  }

  private void Form1_Load(object sender, EventArgs e)
  {
   foreach (Control nesne in this.Controls)
   {
    if ((nesne) is TextBox)
    {
```

```
      nesne.Text = "";
      nesne.BackColor = Color.SteelBlue;
      nesne.ForeColor = Color.Red;
     }
    }
   }
  }
}
```

Görüldüğü gibi **foreach** döngümüzde **this.Controls** sınıfı içerisinde yer alan nesnelere erişerek, bu nesneleri **nesne** adlı değişken içerisinde saklıyoruz.

```
if ((nesne) is TextBox)
```

Komutu ile eğer erişilen nesne bir **TextBox** nesnesi ise, üzerinde gerekli değişikliklerin yapılmasını sağlıyoruz.

WHILE DÖNGÜSÜ

Döngü, koşul cümlesi sonucu **true** olduğu sürece çalıştırılır ve sonuç **false** olduğunda döngünün bitmesi sağlanır. Döngünün artım ya da azalım değeri döngü içerisinde belirtilmelidir.

Kullanımı şu şekildedir:

```
while (Şart)
{
  Komutlar;
  Artım Değeri;
}
```

Örneğin daha önce **for** döngüsü ile yaptığımız 0 ile 10 arasındaki sayıları ekranda listeleyen bir konsol uygulamasını, **while** döngüsü ile yazalım.

```
static void Main(string[] args)
{
 byte i=0;
 while (i <= 10)
 {
  Console.WriteLine(i);
```

```
  i +=1;
 }
 Console.ReadLine();
}
```



Kodumuzda **i** değeri **10**'dan küçük ve eşit olduğu sürece, **i** değişkeninin değeri **1** arttırılarak değeri konsol ekranında yazdırılmaktadır.

while döngüsünü koşul başta değil de sonda kontrol edilecek şekilde **do while** şeklinde de kullanabiliriz. Bu şekildeki kullanımda, şartın doğruluğunun kontrolü, döngü bloğunun sonunda yapıldığı için şart sağlanmıyor olsa bile döngü içindeki komutlar mutlaka bir defa icra edilir.

Kullanımı şu şekildedir:

```
do
{
 Komutlar;
 Artım Değeri;
}
while (Şart);
```

while ile yaptığımız örneği, **do while** şeklinde değiştirelim:

```
static void Main(string[] args)
{
 byte i=0;
 do
 {
  Console.WriteLine(i);
  i ++;
 }
 while (i <= 10);
 Console.ReadLine();
}
```

Kodumuzu çalıştırdığımızda **while** ile aynı sonucu ürettiğini görürüz.



Yukarıdaki örneğimizdeki **while** döngüsündeki şartı…

```
static void Main(string[] args)
{
 byte i=0;
 do
 {
  Console.WriteLine(i);
  i ++;
 }
 while (i > 10);
 Console.ReadLine();
}
```

… şeklinde değiştirelim. `i` değişkeninin ilk değeri `0` olduğu için `i > 10` şartı sağlanmayacaktır. Ancak bu kontrol döngü bloğunun sonunda yapıldığı için döngü en az bir defa çalışacak ve uygulamamızın çalışması sonucunda konsol ekranında `0` değeri görüntülenecektir.

Klavyeden `0` girilene kadar konsol ekranında girilen sayıları toplayıp, `0` girildiğinde toplama sonucunu ekrana yazacak bir uygulama yapalım.

```
static void Main(string[] args)
{
 int toplam=0, sayi=0;
 do
 {
  Console.Write("Bir sayı giriniz= ");
  sayi = Convert.ToInt32(Console.ReadLine());
  toplam += sayi;
 }
 while (sayi != 0);
 Console.WriteLine("Sayıların toplamı= " + toplam);
 Console.ReadLine();
}
```



Kodumuzda **sayi** değişkeninin ilk değeri 0 olduğu için eğer şart kontrolünü döngü bloğunun başında yapsaydık, döngümüz (**sayi != 0**) şartının false sonucu üretmesi nedeniyle çalışmayacaktı. **sayi** değişkeni, ancak bir değer girildikten sonra değer içerecektir. Bu nedenle şart kontrolünü döngü bloğunun sonunda yaptık. Döngümüz ilk çalışmasından sonra değer alacak ve bu değerin kontrolü döngü bloğunun sonunda yapılacaktır. Klavyeden 0 girilene kadar sayıların toplanarak **toplam** adlı değişkende saklanacak ve 0 girildiğinde döngü sonlanarak konsol ekranına **toplam** adlı değişkenin içeriği yazdırılacaktır.

Klavyeden girilen bir sayının basamaklarındaki rakamları toplayan bir uygulama yapalım.

```
static void Main(string[] args)
{
    Console.Write("Sayıyı Giriniz:");
    int sayi = Convert.ToInt32(Console.ReadLine());
    int a = 0;
    while (sayi > 0)
    {
        a += sayi % 10;
        sayi /= 10;
    }
    Console.WriteLine("Sayının rakamları toplamı=" + a);
    Console.ReadLine();
}
```

Döngümüz girilen sayı 0'dan büyük olduğu sürece çalışacaktır. Döngünün her adımında önce sayının 10 ile bölümünden kalan bulunarak **a** değişkeninin değerine eklenmekte, daha sonra da sayı 10'a bölünmektedir. Bu işlem sayı değeri 0 olana kadar sürmektedir. Örneğin 321 sayısını girdiğimizi düşünelim ve kodumuzu adım adım çalıştıralım.

- **1. Adım :** a+= 321 % 10 (321'in 10 ile bölümünden kalan)

  a =1

  sayi =321 /10

  sayi= 32

- **2. Adım :** a+= 32 % 10 (32'nin 10 ile bölümünden kalan)

  a =1 +2

  a=3

  sayi =32 /10

  sayi= 3

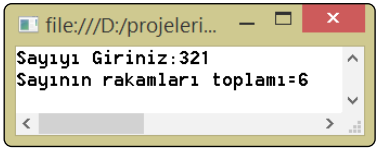
- **3. Adım :** a+= 3 % 10 (3'ün 10 ile bölümünden kalan)

  a =1 +2 +3

  a=6

  sayi =3 /10

  sayi= 0



Kodumuzu çalıştırıp örnek değer olarak 321 girdiğimizde rakamları toplamı olan 6 değeri ekrana yazdırılmaktadır.

while döngüsünü sürekli çalışacak biçimde kullanmak istersek…

```
while (true)
```

… şeklinde kullanabiliriz.

Örneğin…

```
while (true)
{
 Console.WriteLine("C#");
}
```

…döngüsü ya da…

```
do
{
 Console.WriteLine("C#");
} while (true);
```

… döngüsü ekrana sürekli olarak tekrarlanacak biçimde "*C#*" yazar.

DİZİLER VE KOLEKSİYONLAR

8

Diziler ve koleksiyonlar .Net içerisinde, verileri gruplayarak saklamak için kullanılan yapılardır.

Diziler ve koleksiyonlar amaç olarak birbirlerine benzemelerine rağmen, aralarında bazı temel farklılıklar vardır. Bu farklar şunlardır:

- Diziler, aynı türdeki verilerin sıralanarak gruplandırılmasında kullanılan yapılardır. Koleksiyonlar içerisinde ise, aynı türden veriler bir arada saklanabileceği gibi, farklı türden veriler de bir arada saklanabilir.
- Dizilerin eleman sayıları sabit olarak belirlenmelidir. Tanımlanmış olan eleman sayısı daha sonra değiştirilmek istenirse **`Array.Resize`** metodu ile değiştirilebilir. Koleksiyonlar ise, eleman sayısını belirlemeden, herhangi bir eleman sınırı konulmadan tanımlanır. Bellekte kapladığı alan, içerisine eklenen eleman sayısına göre otomatik olarak belirlenir.
- Diziler, **`System.Array`** namespace'i (*isim uzayı*) altında yer alırken, koleksiyonlar ise **`System.Collections`** namespace'i altında yer alır.
- Dizilerin eleman sayıları sabit ve dizi içerisine eklenen elemanlar herhangi bir dönüştürme işlemine tabi tutulmadığı, koleksiyonlarda ise eleman sayıları dinamik ve içerisine aktarılan değerler, **`boxing`** ve **`unboxing`** ile dönüşüme tabi tutulduğu için diziler koleksiyonlara göre daha hızlıdır.

Diziler (Arrays)

Diziler, aynı türdeki verilerin sıralanarak gruplandırılmasında kullanılan yapılardır. Diziler de aslında bir koleksiyondur. Ancak aynı türdeki verileri saklamada kullanılan bir koleksiyondur. Diziler referans türünden olmakla beraber, hem değer tipinde hem de referans tipinden veriler dizi içerisinde saklanabilir. Dizi içerisinde veriler kaydedilirken, `0` numaralı sıra değerinden itibaren kaydedilmeye başlanılır. Dizi içerisindeki elemanlara bu index numaraları ile erişim sağlanabilir. Dizi oluşturulurken dizinin eleman sayısı belirtilmelidir.

```
Değişken türü [ ] Dizi değişkeni adı = new Değişken türü [Eleman sayısı ];
```

Şeklinde tanımlanırlar.

Örneğin…

```
int[ ] numara = new int[10];
```

… şeklindeki bir tanımlamada **`numara`** adında 10 elemanlı integer türünde bir dizi tanımlanmaktadır.

Dizi elemanlarına değer atama işlemi yapılırken, dizi elemanının index değeri köşeli parantez içerisinde yazılarak istenen dizi elemanına atama yapılabilir.

```
string[ ] adsoyad = new string[3];
adsoyad[0] = "Talha Aktaş";
adsoyad[1] = "Sümeyye Aktaş";
adsoyad[2] = "Azra Uncu";
```

Yukarıdaki örnekte görüldüğü gibi 3 elemanlı bir **`adsoyad`** dizisi tanımlayarak, daha sonra dizinin elemanlarına atama yapılmıştır. Tanımladığımız dizi içerisindeki elemanlara index numaralarını belirterek erişiyoruz. Örneğin; yukarıda tanımladığımız **`adsoyad`** dizisi içerisindeki "*Sümeyye Aktaş*" elemanına değer aktarırken **`adsoyad[1]`** şeklinde erişiyoruz. Dizilerin index numaraları 0'dan başladığı için ikinci sıradaki elemana **`adsoyad[1]`** şeklinde erişiyoruz.

Dizilere erişimde dizi elemanlarının index numaralarını tek tek yazmak her zaman kullanışlı olmayabilir. Örneğin; `0` ile `100` arasındaki çift sayıları bir dizi

içerisine aktarıp, daha sonrada konsol ekranına yazdırmak için **for** döngüsünü kullanalım.

Eğer **for** döngüsünü kullanmayacak olsaydık kodumuz…

```
int[] dizi = new int[51];
dizi[0] = 0;
dizi[1] = 2;
dizi[2] = 4;
dizi[3] = 6;
...
dizi[51]=100
```

… şeklinde uzun kod satırları şeklinde yazılacaktı. Ancak **for** döngüsünü ile dizilere index numaralarını kullanarak, aşağıda görüldüğü gibi erişebiliriz.

```
static void Main(string[] args)
{
 int i;
 int[] dizi = new int[51];
 int j=0;
 for (i=0; i <=100; i+=2)
 {
  dizi[j] = i;
  j++;
 }
 for (i = 0; i <= 50; i++)
 {
  Console.WriteLine(dizi[i]);
 }
 Console.ReadLine();
}
```

Kodumuzda 0'da dahil olmak üzere 0 ile 100 arasında 51 adet çift sayı olduğu için dizimizi 51 elemanlı olarak tanımladık. Daha sonra ayrı bir döngü içerisinde dizi içerisindeki elemanları konsol ekranında yazdırdık.

Diziler üzerinde işlem yapmak için **foreach** döngüsü sıklıkla kullanılmaktadır. **foreach** döngüsünün, eleman sayısının belirtilmediği ve dizinin bir bölümü değil de tümü üzerinde işlem yapmak gerektiği durumlarda kullanılması faydalı olacaktır.

Örneğin yukarıda yaptığımız `0` ile `100` arasındaki çift sayıları konsol ekranına yazdıran kodumuzda dizi içerisindeki değerleri konsol ekranına yazdırma işlemini **`foreach`** döngüsü ile yapacak olursak döngümüzü…

```
foreach (int k in dizi)
{
 Console.WriteLine(k);
}
```

… biçiminde yazarız. Döngümüzde dizi içerisindeki elemanlara erişilerek **`k`** değişkeni içerisinde saklanmakta ve **`k`** değişkeni içerisinde saklanan dizi değişkenleri konsol ekranına yazdırılmaktadır.

Dizilere tanımlama esnasında da değer ataması yapılabilir. Örneğin…

```
string[ ] adsoyad = { "Volkan", "Ayşe", "Talha", "Sümeyye" };
```

… şeklinde bir diziye ilk değer ataması yapılabilir. Ancak burada da görüldüğü gibi ilk değer ataması sırasında dizi boyutu belirtilmeden tanımlama yapılmalıdır.

Karakter dizilerini tanımlarken ise diziyi oluşturan elemanları…

```
char[] karakterler = { '!', '?'};
```

… örneğinde olduğu gibi, `""` (*çift tırnak*) karakterleri arasına değil de, `''` (*tek tırnak*) karakterleri arasına yazarız.

Boyutlarına Göre Diziler

Dizileri boyutlarına göre tek boyutlu ve çok boyutlu olmak üzere başlıca iki gruba ayırabiliriz.

Tek Boyutlu Diziler

Tek boyutlu diziler yalnızca eleman sayısı belirtilerek tanımlanan dizilerdir. Dizilerin boyutları belirtilmediği sürece tek boyutlu olarak kabul edilirler. Diziler konusunun başından beri anlattığımız dizi örnekleri tek boyutlu dizidir.

Çok Boyutlu Diziler

Çok boyutlu dizilerde tek boyutlu dizilere benzer şekilde tanımlanır ve kullanılırlar. Farklı olarak virgül karakteri kullanılarak dizi boyutu belirlenir ve kullanılır.

```
Değişken türü [,] Değişken ismi = new Değişken türü[Eleman sayısı, Eleman sayısı];
```

Virgül sayısı arttıkça, boyut da artacaktır. Burada bir tane virgül kullanılması dizinin iki boyutlu olduğunu belirtmektedir.

Örneğin…

```
string[,] adsoyad = new string[3, 3];
```

… satırında 2 boyutlu 3'er elemanlı **`string`** türde **`adsoyad`** isimli bir dizi tanımladık. Dizinin **toplam eleman sayısını boyut x eleman sayısı** şeklinde bulabiliriz. Örneğimizdeki **`adsoyad`** dizisinin toplam eleman sayısı **3x3=9**'dur.

Çok boyutlu dizilere de tanımlama esnasında değer ataması yapılabilir.

```
string[,] adsoyad = { { "Volkan", "Aktaş" }, { "Mustafa", "Aksu" },
{ "Enver", "Arslan" } };
```

Örneğin; yukarıdaki **`adsoyad`** dizisi iki boyutlu bir dizi olarak tanımlanmıştır. Eleman sayısı ise 3 adet olduğu için toplam eleman sayısı **2x3=6** adettir.

Bu dizi içerisindeki "*Aksu*" elemanına **`adsoyad[1,1]`** şeklinde ulaşılabilir.

Çok boyutlu dizileri **dikdörtgensel** (*rectangular*) ve **düzensiz** (*jagged*) olmak üzere iki gruba ayırabiliriz.

Dikdörtgensel Diziler (Rectangular Array)

Dikdörtgensel diziler her satırında eşit sayıda elemanın bulunduğu dizilerdir. Şekli dikdörtgene benzediği için bu isimle adlandırılırlar.

Örneğin…

```
int[,] dizi = new int[3, 4] { { 1, 2, 3, 4 }, { 1, 2, 3, 4 }, { 1, 2, 3, 4 } };
```

… biçiminde **3x4** boyutunda tanımladığımız dizi dikdörtgensel bir dizidir.

Her satırında **4**'er eleman bulunmaktadır.

Yukarıda tanımladığımız örnek dizi elemanlarını, konsol ekranında tek tek listeleyen örnek bir uygulama yapalım.

```
static void Main(string[] args)
{
 int[,] dizi = new int[3, 4]
  { { 1, 2, 3, 4 }, { 1, 2, 3, 4 }, { 1, 2, 3, 4 } };
 for (int i = 0; i < dizi.GetLength(0); i++)
 {
  for (int j = 0; j < dizi.GetLength(1); j++)
  {
   Console.Write(dizi[i, j]);
   if (j == dizi.GetLength(1)-1)
    Console.WriteLine();
  }
 }
 Console.ReadLine();
}
```

Kodumuzda dizinin elemanlarını okumak için **`i`** döngüsü, boyutunu okumak için ise, **`j`** döngüsünü kullanıyoruz. **`Array`** sınıfına ait olan **`GetLength`** metodu ile belirtilen boyuttaki eleman sayısı elde edilir. **`GetLength(0)`** ile satır içerisindeki eleman sayısını, **`Getlength(1)`** ile de sütun içerisindeki eleman sayısını elde ediyoruz.




```
Console.Write(dizi[i, j]);
```

Satırında dizi içerisindeki belirtilen satır ve sütuna ait olan elemanı konsol ekranına yazdırıyoruz.

```
if (j == dizi.GetLength(1)-1)
 Console.WriteLine();
```

Satırlarında konsol ekranına bir satırın yazdırılması bitmiş ise, konsol ekranına satır sonu karakterini ekleyerek yazdırma işleminin dizinin bir sonraki satırından itibaren devam etmesini sağlıyoruz.

Düzensiz Diziler (Jagged Array)

Her bir satırı farklı uzunlukta olan diziler düzensiz diziler olarak adlandırılır. İç içe diziler şeklinde tanımlanırlar. Yani bir dizinin elemanları tanımlanırken her bir elemanın içerisinde başka bir dizinin elemanları tanımlanabilir.

```
Değişken Türü [ ] [ ] Dizi Adı
```

Örneğin…

```
int[][] düzensiz_dizi = { new int[] { 1 },
                          new int[] { 1, 2, 3 },
                          new int[] { 1, 2 } };
```

… şeklinde tanımlanan dizimiz tek boyutlu olup, her satırında farklı uzunlukta elemanın olması dolayısıyla düzensiz bir dizi olarak adlandırılmaktadır. Her satırın tanımlanması **`new int [ ]`** şeklinde yapılmaktadır.

Düzensiz dizilerin her satırı farklı uzunlukta olabildiği için dizi üzerinde yapılacak işlemlerde **`foreach`** döngüsünü kullanmak daha mantıklı olacaktır.

Düzensiz bir dizi oluşturarak içerisindeki elemanları listeleyen örnek bir uygulama yapalım.

```
static void Main(string[] args)
{
 int[][] duzensiz_dizi = { new int[] { 1},
                           new int[] { 1, 2, 3 },
                           new int[] { 1, 2 } };
 foreach (var satir in duzensiz_dizi)
 {
  foreach (var eleman in satir)
   Console.Write("{0}  ", eleman);
  Console.WriteLine();
 }
 Console.ReadLine();
}
```



Kodumuzda dizi içerisindeki her satırı **`new int[ ]`** şeklinde tanımlayarak elemanları aktarıyoruz. Daha sonra iç içe iki adet döngü kuruyoruz. En dışta olan,

```
foreach (var satir in duzensiz_dizi)
```

döngüsünü satırları okumak için kullanırken, içte olan…

```
foreach (var eleman in satir)
```

… döngüsünü ise, geçerli satırdaki her elemanı tek tek okumak amacıyla kullanıyoruz.

Ayrıca içteki döngüde kullandığımız **`Console.WriteLine`** metodu ile de her bir satırın yazılmasının ardından, satır sonu karakteri ekleterek yazdırma işleminin bir alt satırdan devam etmesini sağlıyoruz.

Düzensiz dizilerin tanımlamasını, her bir satırdaki dizi elamanlarını tek tek tanımlayarak ve elemanları tek tek aktararak da yapabilirsiniz. Yukarıda yaptığımız örneği, buna göre yeniden düzenleyelim.

```
static void Main(string[] args)
{
 int[][] duzensiz_dizi = new int[3][];

 duzensiz_dizi[0] = new int[1];
 duzensiz_dizi[1] = new int[3];
 duzensiz_dizi[2] = new int[2];

 duzensiz_dizi[0][0] = 1;
 duzensiz_dizi[1][0] = 1;
 duzensiz_dizi[1][1] = 2;
 duzensiz_dizi[1][2] = 3;
 duzensiz_dizi[2][0] = 1;
 duzensiz_dizi[2][1] = 2;

 foreach (var satir in duzensiz_dizi)
 {
```

```
  foreach (var eleman in satir)
   Console.Write("{0}  ", eleman);
  Console.WriteLine();
 }
 Console.ReadLine();
}
```

Kodumuzda öncelikle…

```
int[][] duzensiz_dizi = new int[3][];
```

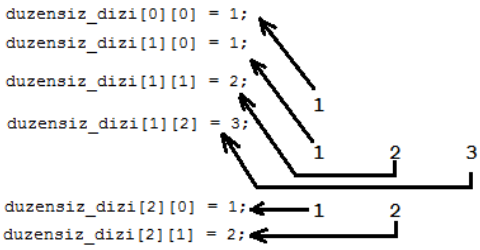
… satırında dizimizi tek boyutlu ve üç satırdan oluşacak şekilde tanımladık.

Daha sonra ise…

```
duzensiz_dizi[0] = new int[1];
duzensiz_dizi[1] = new int[3];
duzensiz_dizi[2] = new int[2];
```

… satırlarında dizinin sırasıyla her satırında kaç eleman olacağını her satırda yeni bir dizi tanımlayarak belirledik. İlk satırda (0 nolu satır) 1 eleman, 2. satırda 3 eleman, 3. satırda 2 eleman olacak şekilde tanımladık.

Daha sonra ise aşağıda görüldüğü gibi dizinin her satırındaki elemanlara sırasıyla değer aktardık.



Dizileri Yeniden Boyutlandırma

Bir dizi boyut belirtilerek tanımlanabildiği gibi parantez içi boş bırakılarak da boyutu belirtilmeden tanımlanabilir. Gerek boyutu belirtilerek tanımlanmış olsun gerekse, boyutu belirtilmeden tanımlanmış olsun program içerisinde dizi boyutu yeniden belirlenebilir.

Örneğin;

```
string[ ] adsoyad = new string[2];
adsoyad[0] = "Talha";
adsoyad[1] = "Sümeyye";
```

… yukarıda tanımladığımız 2 elemanlı **adsoyad** dizisindeki ilk elemanın içeriğine…

```
adsoyad[0]
```

… şeklinde eriştiğimizde içeriğinin "*Talha*" olduğunu görürüz.

Bu diziyi…

```
adsoyad = new string[3];
adsoyad[2] = "Ömer";
```

… şeklinde yeniden boyutlandırarak 3 elemanlı bir dizi haline çevirdik. Kodumuzda daha sonra dizinin 3. elemanına da "*Ömer*" değerini aktardık.

Adsoyad[2] ifadesi ile dizinin son elemanı olan 3. elemanını görüntülemek istediğimizde **"Ömer"** değerini görebiliriz. Ancak yukarıda görüntülediğimiz gibi **adsoyad[0]** ile dizinin ilk elemanını görüntülemeye kalktığımızda dizinin ilk elemanı boş olarak görüntülenecektir. Çünkü yeniden boyutlanan bir dizinin içeriğindeki değerler silinerek boyutlandırma işlemi yapılır.

Boyutlandırma sonrasında dizi içerisindeki değerlerin silinmesi istenmiyorsa, **Array.Resize** metodu kullanılarak dizi boyutu belirlenebilir. **Array.Resize** metodu kullanılarak yeniden boyutlanan bir dizinin içeriğindeki değerler muhafaza edilerek boyutlandırma işlemi yapılır.

Array.Resize metodunun kullanımı şu şekildedir:

```
Array.Resize(ref dizi değişkeni adı, yeni dizi boyutu)
```

ref parametresi ile referans alınacak dizi değişkeni adını belirtirken, ikinci parametremiz ile de dizinin yeni boyutunu belirliyoruz.

Yukarıda yaptığımız boyutlandırma işlemini…

```
Array.Resize(ref adsoyad, 3);
```

… şeklinde **`Array.Resize`** metodu ile yeniden yapacak olursak, **`adsoyad[0]`** ile dizinin ilk elemanını görüntülemeye kalktığımızda dizinin ilk elemanını "*Talha*" olarak görüntüleriz. Çünkü **`Array.Resize`** metodu ile yeniden boyutlanan bir dizinin içeriğindeki değerler muhafaza edilerek boyutlandırma işlemi yapılır.

Anlattıklarımızı daha iyi anlaşılması için koda dökecek olursak…

```
static void Main(string[] args)
{
  string[] adsoyad = new string[2];
  adsoyad[0] = "Talha";
  adsoyad[1] = "Sümeyye";
  Console.WriteLine("Dizinin İlk Elemanının Değeri = " +adsoyad[0]);
  adsoyad = new string[3];
  adsoyad[2] = "Ömer";
  Console.WriteLine("Dizinin İlk Elemanının Değeri = " +adsoyad[0]);
  Console.WriteLine("Dizinin Son Elemanının Değeri = " +adsoyad[2]);
  Console.ReadKey();
}
```

… şeklinde **`Array.Resize`** metodunu kullanmadan yeniden boyutlandırma yapıp uygulamayı çalıştırdığımızda, aşağıdaki çıktıyı elde ederiz.

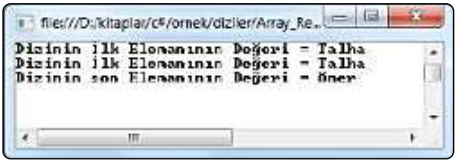


Görüldüğü gibi yeniden boyutlandırma işleminin ardından, dizinin ilk elemanının değerini yeniden okutmaya kalktığımızda boş olarak görüntülenmiştir.

Boyutlandırma işlemini…

```
Array.Resize(ref adsoyad, 3);
```

… şeklinde **`Array.Resize`** metodunu kullanarak aynı uygulamamızı bir daha çalıştıralım.



Ekran çıktısında da görüldüğü gibi dizimizi yeniden boyutlandırdıktan sonra, dizi içerisindeki elemanların değerleri muhafaza edildiği için dizinin ilk elemanının değeri "*Talha*" olarak görüntülenmiştir.

Dizi Değişkenlerinde Kullanılan Metodlar

Diziler üzerinde kullanılan metodlar **`System.Array`** sınıfına ait metodlardır. Bu metodlardan sıklıkla kullanılanlarını inceleyelim.

CreateInstance Metodu

Yeni bir diziyi, değişken tipini ve eleman sayısını belirleyerek tanımlamak için kullanılan metoddur.

Kullanımı şu şekildedir:

```
Array.CreateInstance(Değişken Türü, Eleman Sayısı, Boyut)
```

Örnek olarak tek boyutlu, 3 elemanlı bir dizi oluşturalım ve dizi elemanlarından birine erişelim.

```
static void Main(string[] args)
{
  Array okulturu = Array.CreateInstance(typeof(string), 3);
  string[] okulturleri = (string[])okulturu;
  okulturleri[0] = ("Endüstri Meslek");
  okulturleri[1] = ("Teknik");
  okulturleri[2] = ("Anadolu Teknik");
  Console.Write(okulturleri[1]);
  Console.ReadKey();
}
```

Kodumuzda…

```
Array okulturu = Array.CreateInstance(typeof(string), 3);
```

… satırında **okulturu** adlı string türde, tek boyutlu 3 elemanlı bir dizi oluşturduk.

okulturu değişkeni object olarak geriye döndüğünden dolayı…

```
string[] okulturleri = (string[])okulturu;
```

… satırı ile **okulturleri** dizisine dönüştürüyoruz.

Kodumuzu çalıştırdığımızda "*Teknik*" sonucunu üretecektir.

3x3 boyutunda bir dizi tanımlamasını ise…

```
Array diller = Array.CreateInstance(typeof(string), 3, 3);
```

… şeklinde yapabiliriz.

IsArray Metodu

Bir değişkenin dizi değişkeni olup olmadığını anlamak amacı ile kullanılan metoddur. Geriye **true** ya da **false** olmak üzere boolean değer döndürür.

```
static void Main(string[] args)
{
   string[] okulturu = {"Anadolu", "Meslek", "Teknik" };
   string ad = "Volkan";
   Type tip;
   tip = okulturu.GetType();
   Console.WriteLine(tip.IsArray );
   tip = ad.GetType();
   Console.WriteLine(tip.IsArray );
   Console.ReadKey();
}
```

Kodumuzda **GetType** metodu ile **okulturu** ve **ad** isimli değişkenlerin tipini elde edip, **tip** isimli değişkene aktarıyoruz. Daha sonra ise **IsArray** metodunu kullanarak ilgili değişkenin dizi türünden olup olmadığını öğreniyoruz.

Kodumuzu çalıştırdığımızda **okulturu** değişkeni dizi olduğundan dolayı **true** sonucunu üretirken, **ad** değişkeni dizi olmadığından dolayı **false** sonucunu üretecektir.

SetValue Metodu

Dizi elemanlarına değer atamak amacı ile kullanılan metoddur.

Kullanımı şu şekildedir:

```
SetValue("Atanacak değer",atanacak index numarası)
```

Örnek olarak daha önce…

```
okulturleri[0] = ("Endüstri Meslek");
okulturleri[1] = ("Teknik");
okulturleri[2] = ("Anadolu Teknik");
```

… şeklinde yaptığımız değer atamalarını, **SetValue** metodu ile yapalım.

```
static void Main(string[] args)
{
   Array okulturleri = Array.CreateInstance(typeof(string), 3);
   okulturleri.SetValue("Endüstri Meslek", 0);
   okulturleri.SetValue("Teknik", 1);
   okulturleri.SetValue("Anadolu Teknik", 2);
   Console.Write(okulturleri.GetValue(2));
   Console.ReadKey();
}
```

Kodumuzda **SetValue** metodu ile…

```
okulturleri.SetValue("Endüstri Meslek", 0);
```

… satırında olduğu gibi belirtilen index numarasına, **string** türdeki değerimizi aktarıyoruz.

Kodumuzu çalıştırdığımızda dizinin 2. ve son elemanı olan "*Anadolu Teknik*" değeri, konsol ekranına yazdırılacaktır.

GetValue Metodu

Dizi içerisindeki elemanlara erişmek amacı ile kullanılır.

Kullanımı şu şekildedir:

```
GetValue(İndex değeri)
```

Örnek olarak çok boyutlu bir diziye değer ataması yaparak aynı zamanda değer okuyalım.

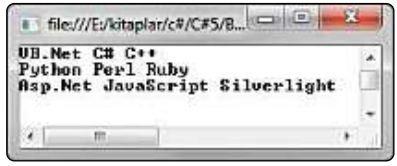
```
static void Main(string[] args)
{
   Array diller = Array.CreateInstance(typeof(string), 3, 3);
   diller.SetValue("VB.Net",0,0);
   diller.SetValue("C#",0,1);
   diller.SetValue("C++",0,2);
   diller.SetValue("Python",1,0);
   diller.SetValue("Perl",1,1);
   diller.SetValue("Ruby",1,2);
   diller.SetValue("Asp.Net",2,0);
   diller.SetValue("JavaScript",2,1);
   diller.SetValue("Silverlight",2,2);
   Console.WriteLine(diller.GetValue(0,1));
   Console.ReadKey();
}
```

Örneğimizde **`Array.CreateInstance`** metodu ile **3x3** boyutlu bir dizi oluşturarak, **`SetValue`** metodu ile dizinin elemanlarına değer atayıp, **`GetValue`** metodu ile dizinin (**0**,**1**) indeks değerli elamanına erişerek bu değeri, yani "*C#*" değerini konsol ekranında görüntüledik.

Kodumuzu, dizi içerisindeki tüm elemanları listeleyecek şekilde düzenleyelim.

```
static void Main(string[] args)
{
   Array diller = Array.CreateInstance(typeof(string), 3, 3);
   diller.SetValue("VB.Net",0,0);
   diller.SetValue("C#",0,1);
   diller.SetValue("C++",0,2);
```

```
    diller.SetValue("Python",1,0);
    diller.SetValue("Perl",1,1);
    diller.SetValue("Ruby",1,2);
    diller.SetValue("Asp.Net",2,0);
    diller.SetValue("JavaScript",2,1);
    diller.SetValue("Silverlight",2,2);
    for (int i = 0; i < diller.GetLength(0); i++)
    {
     for (int j = 0; j < diller.GetLength(1); j++)
     {
      Console.Write(diller.GetValue(i, j) +" ");
      if (j == diller.GetLength(1) - 1)
       Console.WriteLine();
     }
    }
    Console.ReadKey();
}
```



Kodumuzda dizinin elemanlarını okumak için **i** döngüsü, boyutunu okumak için ise **j** döngüsünü kullanıyoruz. **Array** sınıfına ait olan **GetLength** metodu ile belirtilen boyuttaki eleman sayısı elde edilir. **GetLength(0)** ile satır içerisindeki eleman sayısını, **Getlength(1)** ile de sütun içerisindeki eleman sayısını elde ediyoruz.

```
 Console.Write(diller.GetValue(i, j) +" ");
```

Satırında dizi içerisindeki belirtilen satır ve sütuna ait olan elemanı **GetValue** metodu ile okuyarak, konsol ekranına yazdırıyoruz.

```
if (j == dizi.GetLength(1)-1)
    Console.WriteLine();
```

Satırlarında eğer konsol ekranına bir satırın yazdırılması bitmiş ise, konsol ekranına satır sonu karakterini ekleyerek, yazdırma işleminin dizinin bir sonraki satırından itibaren devam etmesini sağlıyoruz.

Join Metodu

Dizi içerisindeki değerlerin hepsini bir **`string`** değişkene, belirtilen ayırıcı karakter kullanılarak aktaran metoddur.

Kullanımı şu şekildedir:

```
String.Join(ayırıcı karakter, dizi değişkeni adı)
```

Örnek olarak bir **listBox** içerisindeki değerleri **`join`** metodu ile, karakterini kullanarak bir **textBox** içerisine aktaran bir örnek yapalım.

```
private void btnislem_Click(object sender, EventArgs e)
{
   string[] diller;
   diller = new string[listBox1.Items.Count];
   byte i;
   for (i = 0; i <= listBox1.Items.Count - 1; i++)
   {
    diller[i] = listBox1.Items[i].ToString() ;
   }
   textBox1.Text = string.Join(",", diller);
}
```



Uygulamamızda formun tasarım görünümünde **Properties Window** içerisinde **listBox**'ın **`items`** özelliğine, şekilde görüldüğü gibi değerleri önceden girmemiz gerekiyor. Kodumuzda….

```
string[] diller;
diller = new
string[listBox1.Items.Count];
```

…. satırlarında diller adında bir diziyi, **listBox** içerisindeki elemanların sayısı kadar elemanla tanımlıyoruz.

Daha sonra `for` döngüsü içerisinde, bu dizi içerisine **listBox** içerisindeki elemanları sırasıyla okuyarak aktardık. Bu dizinin elemanları da **textBox** içerisine, `Join` metodu yardımıyla elemanlar arasına , karakteri kullanılarak aktarıldı.

Copy Metodu

Bir dizinin içeriğini başka bir diziye kopyalamak amacı ile kullanılan metoddur.

Kullanımı şu şekildedir:

```
Array.Copy(Kaynak dizi adı, Hedef dizi adı, Eleman sayısı)
```

Eleman sayısı parametresi, kaynak diziden kaç adet elemanın hedef diziye kopyalanacağını belirtir.

Örnek olarak oluşturduğumuz bir diziyi `Copy` metodu ile bir başka dizinin içerisine kopyalayalım.

```
static void Main(string[] args)
{
   string[] diller = new string[3];
   diller[0] = "VB.Net";
   diller[1] = "C#";
   diller[2] = "ASP.Net";
   string[] digerdiller = new string[3];
   digerdiller[0] = "Python";
   digerdiller[1] = "F#";
   digerdiller[2] = "C++";
   Array.Copy(digerdiller, diller, 3);
   Console.WriteLine(diller[0]);
   Console.WriteLine(diller[1]);
   Console.WriteLine(diller[2]);
   Console.ReadKey();
}
```

Kodumuzda…

```
Array.Copy(digerdiller, diller, 3);
```

… satırında **Copy** metodu ile digerdiller dizisinin 3 adet elemanı **diller** dizisi içerisine kopyalanmaktadır.



Daha sonra ise **diller** dizisinin yeni içeriği yazdırılmaktadır. **diller** dizisi önce "*VB.Net*, *C#*, *ASP.Net*" elemanlarına sahipken, **Copy** işlemi sonucunda "*Python, F#, C++*" elemanlarına sahip olmaktadır. Dolayısı ile de kodumuzu çalıştırdığımızda "*Python, F#, C++*" elemanları yazdırılmaktadır.

Eğer ki…

```
Array.Copy(digerdiller, diller, 3);
```

… yerine…

```
Array.Copy(digerdiller, diller, 2);
```

… şeklinde **digerdiller** dizisinin yalnızca ilk 2 elemanını kopyalasaydık, **diller** dizisinin son elemanı olan "*ASP.Net*", **diller** dizisinin içerisinde mevcut olmaya devam edecekti. Çünkü kopyalama işlemi hedef dizinin ilk index'inden itibaren yapılacağı için ilk 2 eleman olan "*Python*" ve "*ASP.Net*" elemanları hedef diziye eklenecek, hedef dizinin son elemanı olan "*ASP.Net*" hedef dizi içerisinde var olmaya devam edecektir. Bu şekilde kodumuzu çalıştırdığımızda ise; "*Python, F#, ASP.Net*" şeklinde sonuç verecektir.

Copy metodunu…

```
Array.Copy(Kaynak dizi adı, Kaynak index, Hedef dizi adı, Hedef
index, eleman sayısı)
```

… şeklinde de kullanabilirsiniz.

Kaynak index; kopyalanacak kaynak dizi elemanlarının kaçıncı elemandan itibaren alınacağını belirtir.

Hedef index; hedef dizinin kaçıncı elemanından itibaren kopyalama yapılacağını belirtir.

Yukarıda yaptığımız örneği bu yöntemi kullanacak şekilde düzenleyelim.

```
static void Main(string[] args)
{
   string[] diller = new string[5];
   diller[0] = "VB.Net";
   diller[1] = "C#";
   diller[2] = "ASP.Net";
   string[] digerdiller = new string[3];
   digerdiller[0] = "Python";
   digerdiller[1] = "F#";
   digerdiller[2] = "C++";
   Array.Copy(digerdiller, 1, diller, 3, 2);
   Console.WriteLine(diller[0]);
   Console.WriteLine(diller[1]);
   Console.WriteLine(diller[2]);
   Console.WriteLine(diller[3]);
   Console.WriteLine(diller[4]);
   Console.ReadLine();
}
```

Kodumuzdaki…

```
string[] diller = new string[5];
```

…. satırında görüldüğü gibi **`diller`** dizisini diğer örnekten farklı olarak 5 elemanlı olarak tanımladık. İlk 3 index değerine "*VB.Net, C#, ASP.Net*" elemanlarını ekledik. Son iki elemanı ise, **`digerdiller`** dizisinden kopyalayacağımız elemanlar için boş bıraktık.

```
Array.Copy(digerdiller, 1, diller, 3, 2);
```

Burada **Copy** metodu ile **digerdiller** dizisinin 1 nolu indexinden itibaren 2 adet elemanı alıyoruz. (*F#, C++*). Bu elemanları **diller** dizisinin 3 nolu index değerinden itibaren (boş bırakıp değer aktarmadığımız son iki eleman) ekliyoruz.



diller dizisi önce "*VB.Net, C#, ASP.Net*" elemanlarına sahipken, **Copy** işlemi sonucunda sırasıyla "*VB.Net, C#, ASP.Net, F#, C++*" elemanlarına sahip olmaktadır.

Diziler İçerisinde Arama İşlemleri

Contains Metodu ile Arama Yapma

Bir dizi içerisinde bir elemanın var olup olmadığını bulma amacı ile kullanılır. Geriye **boolean** bir değer döndürür. Eğer aranan değer bulundu ise **true**, bulunamadı ise, **false** değeri dönecektir.

Kullanımı şu şekildedir:

```
Contains(aranan değer)
```

```
static void Main(string[] args)
{
   string aranan;
   string[] isimler = new string[4];
   isimler[0] = "Hasan";
   isimler[1] = "Murat";
   isimler[2] = "Talha";
   isimler[2] = "Onur";

   Console.Write("Aranan değeri giriniz:");
   aranan = Console.ReadLine();
   if (isimler.Contains(aranan) == false)
```

```
    {
     Console.Write("Aranan değer bulunamadı");
    }
    else
    {
     Console.Write("Aranan eleman bulundu");
    }

  Console.ReadKey();
}
```

Kodumuzda aranan değer konsol ekranında girilerek **aranan** adlı değişkene aktarılıyor.

```
if (isimler.Contains(aranan) == false)
```

Bu komutta **isimler** dizisi içeriğinde **aranan** değerin mevcut olup olmadığı **Contains** metodu ile kontrol edilerek, dönen değere göre sonuç konsol ekranına yazdırılıyor.

IndexOf Metodu ile Arama Yapma

Dizi içerisinde aranan elemanın index numarasını geriye döndüren metoddur.

Kullanımı şu şekildedir:

```
IndexOf(Arama yapılacak dizi değişkeninin adı, aranacak değer,
aramanın başlayacağı index değeri)
```

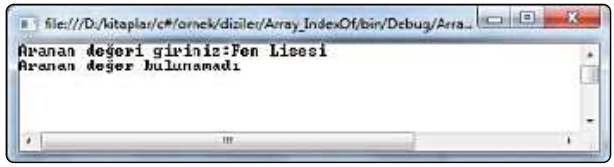
IndexOf metodu geriye, aranan değer bulunursa o değerin index değerini, bulunamazsa **-1** değerini döndürür.

Örnek olarak bir dizi içerisinde arama yaparak bulunduğunda index değerini, bulunamadığında "*Aranan değer bulunamadı*" mesajını görüntüleyecek bir konsol uygulaması yapalım.

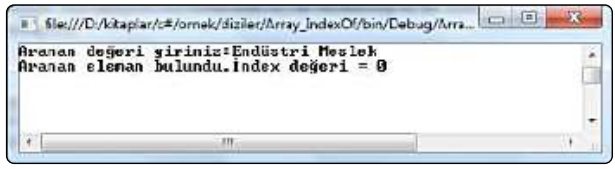
```
static void Main(string[] args)
{
   int indexno;
   string aranan;
   Array okulturu = Array.CreateInstance(typeof(string), 3);
```

```
    okulturu.SetValue("Endüstri Meslek", 0);
    okulturu.SetValue("Teknik", 1);
    okulturu.SetValue("Anadolu Teknik", 2);
    Console.Write("Aranan değeri giriniz:");
    aranan = Console.ReadLine();
    indexno = Array.IndexOf(okulturu, aranan.ToString());
    if (indexno == -1)
    {
     Console.Write("Aranan değer bulunamadı");
    }
    else
    {
     Console.Write("Aranan eleman bulundu.İndex değeri = " + indexno);
    }
    Console.ReadKey();
}
```

Örnek olarak "*Fen Lisesi*" değerini girdiğimizde **`IndexOf`** metodu, geriye **`-1`** değerini döndüreceği için aşağıdaki gibi "*Aranan değer bulunamadı*" mesajı yazılacaktır.



"*Endüstri Meslek*" değerini verdiğimizde ise, eleman dizi içerisinde var olduğundan **`index`** numarası olan `0` görüntülenir.



Örneğimizde arama işlemine `1` index numaralı elemandan başlanılmasını isteseydik arama işlemini…

```
indexno = Array.IndexOf(okulturu, aranan.ToString(),1);
```

… şeklinde gerçekleştirirdik.

LastIndexOf Metodu ile Arama Yapma

IndexOf metodunda olduğu gibi diziler içerisinde arama yapma amaçlı olarak kullanılır. Farklı olarak eğer aranan elemandan birden fazla mevcut ise, bu elemanlardan sonuncusunun index değerini geriye döndürür.

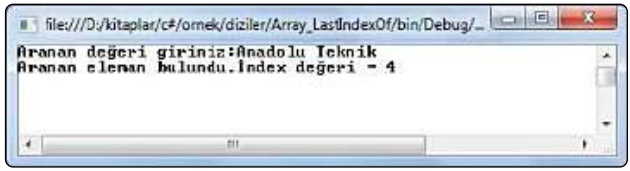
Kullanımı şu şekildedir:

```
LastIndexOf(Arama yapılacak dizi değişkeninin adı, aranacak değer,
aramanın başlayacağı index değeri)
```

Örnek olarak içerisinde bazı elamanlarının birden fazla sayıda, tekrarlandığı bir dizi içerisinde **LastIndexOf** metodu ile arama yapalım.

```
static void Main(string[] args)
{
   int indexno;
   string aranan;
   Array okulturu = Array.CreateInstance(typeof(string), 5);
   okulturu.SetValue("Endüstri Meslek", 0);
   okulturu.SetValue("Teknik", 1);
   okulturu.SetValue("Teknik", 2);
   okulturu.SetValue("Anadolu Teknik", 3);
   okulturu.SetValue("Anadolu Teknik", 4);
   Console.Write("Aranan değeri giriniz:");
   aranan = Console.ReadLine();
   indexno = Array.LastIndexOf(okulturu, aranan.ToString());
   if (indexno == -1)
   {
    Console.Write("Aranan değer bulunamadı");
   }
   else
   {
    Console.Write("Aranan eleman bulundu.İndex değeri = " + indexno);
   }
   Console.ReadKey();
}
```

"*Teknik*" ve "*Anadolu Teknik*" elemanlarını dizi içerisine ikişer defa ekledik. Örneğin; "*Anadolu Teknik*" elemanını arayalım. Görüldüğü gibi "*Anadolu Teknik*" elamanı 3 ve 4 nolu index numaralarına sahipken, son index değeri olan 4 ekrana getirilmiştir.



BinarySearch Metodu İle Arama Yapma

Dizi içerisinde aranan elemanın index numarasını geriye döndüren metoddur.

Kullanımı şu şekildedir:

```
BinarySearch(Arama yapılacak dizi değişkeninin adı, aranacak değer,
aramanın başlayacağı index değeri)
```

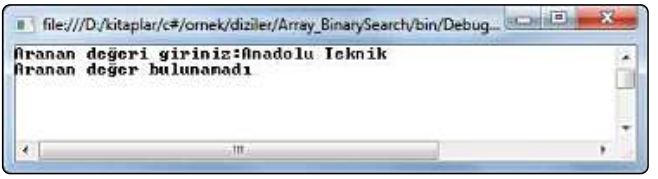
`IndexOf` metodu geriye eğer aranan değer bulunursa o değerin index değerini, bulunamazsa negatif bir değer döndürür.

Bu metod ile arama yapabilmek için üzerinde arama işlemi yapılacak dizinin sıralı olması gerekir.

```
static void Main(string[] args)
{
   int indexno;
   string aranan;
   Array okulturu = Array.CreateInstance(typeof(string), 3);
   okulturu.SetValue("Endüstri Meslek", 0);
   okulturu.SetValue("Teknik", 1);
   okulturu.SetValue("Anadolu Teknik", 2);
   Console.Write("Aranan değeri giriniz:");
   aranan = Console.ReadLine();
   indexno = Array.BinarySearch(okulturu, aranan.ToString());
   if (indexno < 0)
   {
    Console.Write("Aranan değer bulunamadı");
```

```
    }
    else
    {
     Console.Write("Aranan eleman bulundu.İndex değeri = " + indexno);
    }
    Console.ReadKey();
}
```

Kodumuzu çalıştıralım ve arama yapalım.



Görüldüğü gibi "*Anadolu Teknik*" elemanı dizi içerisinde bulunmasına rağmen "*Aranan değer bulunamadı*" mesajı döndürüldü.

Dizi değerlerini aşağıdaki şekilde düzenleyerek yeniden oluşturalım.

```
okulturu.SetValue("Anadolu Teknik", 0);
okulturu.SetValue("Endüstri Meslek", 1);
okulturu.SetValue("Teknik", 2);
```

Aynı arama işlemini tekrarlayalım.



Aranan eleman bulunarak **index** değeri doğru bir şekilde yazdırıldı.

Koleksiyonlar

Koleksiyonlar, **System.Collections** namespace'i altında yer alırlar. Koleksiyonlar içerisinde aynı türden veriler bir arada saklanabileceği gibi, farklı türden veriler de bir arada saklanabilir. Eleman sayısını belirlemeden, herhangi

bir eleman sınırı konulmadan tanımlayabiliriz. Bellekte kapladığı alan, içerisine eklenen eleman sayısına göre otomatik olarak belirlenir. Koleksiyon içerisine aktarılan değerler, **boxing** ve **unboxing** ile dönüşüme tabi tutulurlar. Koleksiyonları **Generic**, **Non Generic** (*Generic olmayan*) ve **Specialized** (*Özelleştirilmiş*) koleksiyonlar başlıkları altında üç gruba ayırabiliriz.

Non Generic (Generic Olmayan) Koleksiyonlar

Non Generic koleksiyonlar; farklı türlerdeki verilerin bir arada saklanabildiği, içerisine aktarılan değerlerin, **boxing** işlemi ile **object** türüne dönüştürülerek tanımlandığı koleksiyon türleridir. **System.Collections** namespace'i altında yer alırlar.

Non-Generic koleksiyon başlığı altında **ArrayList**, **Hashtable** ve **SortedList** sınıflarını inceleyeceğiz.

ArrayList Sınıfı

ArrayList sınıfı, içerisine aktarılan değerlerin, **boxing** işlemi ile object türüne dönüştürülerek tanımlandığı, eleman sayısı belirlemeden tanımlama yapılabildiği koleksiyon türüdür. **System.Collections** sınıfı içerisinde yer alır.

```
ArrayList isim = new ArrayList();
```

… şeklinde tanımlanırlar.

Add Metodu

ArrayList içerisine eleman eklemek amacı ile kullanılır.

```
static void Main(string[] args)
{
   ArrayList isimler = new ArrayList();
   isimler.Add("Hasan");
   isimler.Add("Murat");
   isimler.Add("Talha");
   isimler.Add("Onur");
   Console.Write(isimler[2]);
```

```
    Console.ReadKey();
}
```

Kodumuzu çalıştırdığımızda **isimler** dizisinin 2 index numaralı değeri olan "*Talha*" değerini geriye döndürür.

ArrayList içerisinde farklı türden değişken tiplerinden oluşan elemanlar aynı anda eklenebilir.

```
static void Main(string[] args)
{
    ArrayList bilgi = new ArrayList();
    bilgi.Add("Talha");
    bilgi.Add(123);
    bilgi.Add("a");
    Console.Write(bilgi[1]);
    Console.ReadKey();
}
```

Kodumuzda görüldüğü gibi bilgi isimli **ArrayList** içerisine hem **string** türde hem de **int** türünde değerler ekledik. Kodumuzu çalıştırdığımızda 1 index numaralı değer olan **123** değeri konsol ekranında görüntülenecektir.

ArrayList içerisine eklediğimiz bir değeri, indeks numarasını belirterek daha sonra değiştirebiliriz.

```
ArrayList diller = new ArrayList();
diller.Add("VB.Net");
diller[0] = "C#";
Console.WriteLine(diller[0]);
Console.ReadLine();
```

Örneğimizde **diller** isimli **ArrayList** içerisine öncelikle "*VB.Net*" değerini ekliyoruz. 0 nolu index değerinde "*VB.Net*" değeri mevcutken, bu değeri daha sonra "*C#*" değeri ile değiştiriyoruz. Dolayısı ile kodumuzu çalıştırdığımızda "*C#*" değeri görüntülenecektir.

Insert Metodu

Add metodu kullanılarak dizi içerisine yapılan eklemelerde, eklenen eleman dizinin sonuna eklenir. Dizi içerisinde istenilen bir yere eleman eklemek için ise **Insert** metodu kullanılır.

Kullanımı şu şekildedir:

```
Insert(index değeri, eklenecek değer)
```

Add ile **Insert** metodları arasındaki farkı görebilmek için her ikisini birlikte kullanarak bir örnek yapalım.

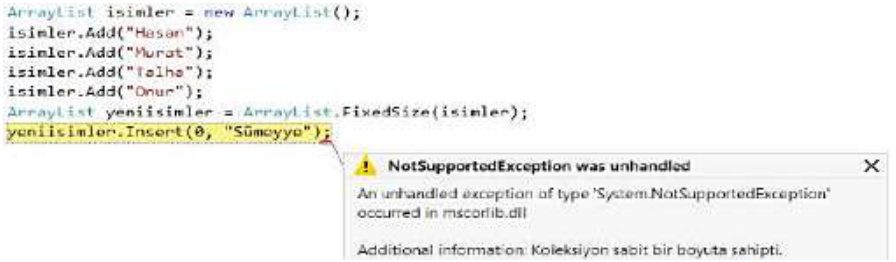
```
static void Main(string[] args)
{
   ArrayList isimler = new ArrayList();
   isimler.Add("Hasan");
   isimler.Add("Murat");
   isimler.Add("Talha");
   isimler.Add("Onur");
   isimler.Insert(0, "Sümeyye");
   Console.WriteLine(isimler[0]);
   Console.WriteLine(isimler[1]);
   Console.ReadKey();
}
```

Normalde dizinin ilk değeri "*Hasan*" iken, **Insert** metodunu kullanarak dizinin ilk sırasına "*Sümeyye*" değerini eklediğimiz için kodumuzu çalıştırdığımızda **isimler[0]** değeri olarak "*Sümeyye*", **isimler[1]** değeri olarak da "*Hasan*" değerinin yazıldığını görürüz. Daha önce **0** nolu index değerinde olan "*Hasan*" değeri **Insert** işleminin ardından bir sonraki index değerine kaydırılmıştır.

Sabit kapasiteli bir **ArrayList** tanımlamak amacı ile **FixedSize** metodunu kullanabiliriz. Aşağıdaki gibi kullanılır:

```
ArrayList Yeni ArrayList İsmi = ArrayList.FixedSize(Kaynak ArrayList İsmi)
```

Ancak sabit kapasiteli olarak oluşturulan bir **ArrayList** içerisine **Add** ve **Insert** metodlarını kullanarak ekleme yapılamaz. Örneğin; yukarıdaki örnekte tanımladığımız **ArrayList**'i sabit kapasiteli olarak yeniden tanımlayalım.



Görüldüğü gibi **`Insert`** metodunu kullanmamız sonucunda hata üretilmiştir.

`FixedSize` metodu ile sabit kapasiteli olarak tanımlanan bir **`ArrayList`** içerisinde **`Add`** ve **`Insert`** metodları ile ekleme yapılamasa da içerisindeki değerleri değiştirebiliriz.

```
static void Main(string[] args)
{
   ArrayList isimler = new ArrayList();
   isimler.Add("Hasan");
   isimler.Add("Murat");
   isimler.Add("Talha");
   isimler.Add("Onur");
   ArrayList yeniisimler = ArrayList.FixedSize(isimler);
   yeniisimler[0] = "Sümeyye";
   Console.WriteLine(yeniisimler[0]);
   Console.WriteLine(yeniisimler[1]);
   Console.ReadKey();
}
```

Kodumuzda **`FixedSize`** metodu ile **`yeniisimler`** adlı sabit kapasiteli bir **`ArrayList`**'i isimler adlı **`ArrayList`**'den oluşturduk. Artık **`yeniisimler`** adlı **`ArrayList`** içerisinde isimler adlı **`ArrayList`**'in elemanları vardır ve kapasitesi değiştirilemez. Ancak…

```
yeniisimler[0] = "Sümeyye";
```

… satırında yenişimler adlı **`ArrayList`**'in 0 nolu index'inde mevcut olan "*Hasan*" değerini "*Sümeyye*" olarak değiştirdik. Dolayısıyla kodumuzu çalıştırdığımızda sırasıyla "*Sümeyye*, Murat" değerleri görüntülenecektir.

Sort Metodu

Dizileri sıralama amacı ile kullanılan bir metoddur.

Kullanımı şu şekildedir:

```
Diziadı.Sort()
```

Bir **`string`** dizi tanımlayıp bu dizi içerisindeki elemanları harf sırasına göre küçükten büyüğe sıralayacak bir örnek uygulama yapalım.

```
static void Main(string[] args)
{
   ArrayList isimler = new ArrayList();
   isimler.Add("Murat");
   isimler.Add("Hasan");
   isimler.Add("Talha");
   isimler.Add("Onur");
   isimler.Sort();
   Console.WriteLine(isimler[0]);
   Console.WriteLine(isimler[1]);
   Console.WriteLine(isimler[2]);
   Console.WriteLine(isimler[3]);
   Console.ReadKey();
}
```

Kodumuzu çalıştırdığımızda dizi elemanlarının küçükten büyüğe doğru sıralanarak "*Hasan, Murat, Onur, Talha*" şeklinde sıralandığını görürüz.

Rakamlardan oluşan bir dizi tanımlayarak sıralama işlemi yapalım.

```
static void Main(string[] args)
{
 ArrayList rakamlar = new ArrayList();
 rakamlar.Add(8);
 rakamlar.Add(5);
 rakamlar.Add(3);
 rakamlar.Add(9);
 rakamlar.Add(7);
 rakamlar.Sort();
```

```
 Console.WriteLine(rakamlar[0]);
 Console.WriteLine(rakamlar[1]);
 Console.WriteLine(rakamlar[2]);
 Console.WriteLine(rakamlar[3]);
 Console.WriteLine(rakamlar[4]);
 Console.ReadKey();
}
```

Kodumuzu çalıştırdığımızda *3*, *5*, *7*, *8*, *9* şeklinde küçükten büyüğe doğru sıralandığını görürüz.

IComparer interface den türeteceğimiz kendi karşılaştırma sınıfımızı yazarak, sıralama işlemin yönünü tersine çevirebiliriz. Sayıları sıralayan kodumuzu buna göre düzenleyelim.

```
using System;
using System.Collections;
using System.Text;
namespace ArrayList_IComparer
{
 public class Compare : IComparer
 {
  int IComparer.Compare(Object x, Object y)
  {
   return ((new  CaseInsensitiveComparer()).Compare(y, x));
  }
 }
 class Program
 {
  static void Main(string[] args)
  {
   Compare CompareOptions = new Compare();
   ArrayList rakamlar = new ArrayList();
   rakamlar.Add(8);
   rakamlar.Add(5);
   rakamlar.Add(3);
   rakamlar.Add(9);
   rakamlar.Add(7);
   rakamlar.Sort(CompareOptions);
   Console.WriteLine(rakamlar[0]);
```

```
    Console.WriteLine(rakamlar[1]);
    Console.WriteLine(rakamlar[2]);
    Console.WriteLine(rakamlar[3]);
    Console.WriteLine(rakamlar[4]);
    Console.ReadKey();
   }
  }
}
```

Kodumuzda **Compare** isimli **IComparer** interface'den türettiğimiz bir karşılaştırma sınıfı yazıyoruz. Bu sınıfın içerisinde **CaseInsensitiveComparer.Compare** metodunu kullanıyoruz. **Compare** metodu iki adet parametre almaktadır. Bu parametrelerle **ArrayList** içerisindeki elemanlar karşılaştırılmaktadır. Eğer geriye dönen değer 0 ise, her iki eleman birbirine eşit, 0'dan küçükse ilk parametredeki eleman daha üst sıraya taşınır, 0'dan büyükse ikinci parametredeki eleman daha üst sıraya taşınır.



Kodumuzu bu hali ile çalıştırdığımızda sıralama işleminin yönünün tersine çevrilerek, büyükten küçüğe doğru sıralama yapıldığını görürüz.

Eğer sıralama işleminin yönünü küçükten büyüğe olacak şekilde değiştirmek istersek kodumuzu…

```
CaseInsensitiveComparer()).Compare(x, y)
```

… şeklinde **x** ve **y** parametrelerini yer değiştirerek yazmalıyız.

Count Özelliği

Bir dizinin eleman sayısını bulmak amacı ile kullanılan metoddur.

Örneğin; bir üstteki sort örneğinde yaptığımız gibi…

```
Console.WriteLine(rakamlar[0]);
Console.WriteLine(rakamlar[1]);
Console.WriteLine(rakamlar[2]);
Console.WriteLine(rakamlar[3]);
Console.WriteLine(rakamlar[4]);
```

… şeklinde dizi elemanlarını tek tek yazdırmak yerine **Count** metodundan yararlanıp bir döngü kurarak, dizi elemanlarını bu döngü içerisinde yazdıralım.

```
static void Main(string[] args)
{
 ArrayList rakamlar = new ArrayList();
 rakamlar.Add(8);
 rakamlar.Add(5);
 rakamlar.Add(3);
 rakamlar.Add(9);
 rakamlar.Add(7);
 rakamlar.Sort();
 for (int i = 0; i <= rakamlar.Count - 1; i++)
 {
  Console.WriteLine(rakamlar[i]);
 }
 Console.ReadKey();
}
```

Kodumuzda **rakamlar.Count-1** ile eleman sayısını elde ettik. **for** döngüsünü 0'dan rakamlar isimli **ArrayList**'in eleman sayısına kadar kurarak döngü içerisinde sırasıyla rakamlar isimli **ArrayList**'in elemanlarını konsol ekranına yazdırdık.

Remove Metodu

Dizi içerisindeki elemanları silmek amacı ile kullanılır. Sileceğimiz elemanın değerini belirtmemiz gerekmektedir.

```
static void Main(string[] args)
{
   ArrayList nickname = new ArrayList();
```

```
    nickname.Add("Talha");
    nickname.Add(123);
    nickname.Add("a");
    nickname.Remove(123);
    Console.Write(nickname[1]);
    Console.ReadKey();
}
```

Dizi içerisinden **Remove** metodunu kullanarak **123** elemanını sildiğimiz için **1** nolu index değerine sahip elemanı yazdırdığımızda artık "*123*" elemanı yerine, "*a*" değeri yazdırılır.

RemoveAt Metodu

Dizi içerisinden, indeks değeri belirtilen elemanları silmek amacı ile kullanılır.

```
static void Main(string[] args)
{
    ArrayList bilgi = new ArrayList();
    bilgi.Add("Talha");
    bilgi.Add(123);
    bilgi.Add("a");
    Console.WriteLine(bilgi[bilgi.Count - 1]);
    bilgi.RemoveAt(bilgi.Count - 1);
    Console.WriteLine(bilgi[bilgi.Count - 1]);
    Console.ReadKey();
}
```

Örneğimizde…

```
Console.WriteLine(bilgi[bilgi.Count - 1]);
```

… satırında **Count** metodunu kullanarak bilgi dizisinin son elemanı olan "*a*" değerini görüntüledik.

Ardından…

```
bilgi.RemoveAt(bilgi.Count - 1);
```

… satırında dizinin son elemanı olan "*a*" değerini sildik.

Son olarak…

```
Console.WriteLine(bilgi[bilgi.Count - 1]);
```

… satırında dizinin son elemanını yeniden yazdırdığımızda, "*a*" elemanı silindiği için artık dizinin son elemanı olarak "*123*" değerinin yazdırıldığını görürüz.

Clear Metodu

Dizi içerisindeki elemanların hepsini silmek amacı ile kullanılır. **`Diziadı.Clear()`** şeklinde kullanılır.

Reverse Metodu

Diziyi tersine çevirmek amacı ile kullanılan bir metoddur.

Kullanımı şu şekildedir:

```
Diziadı.Reverse()
```

```
static void Main(string[] args)
{
   ArrayList isimler = new ArrayList();
   isimler.Add("Hasan");
   isimler.Add("Murat");
   isimler.Add("Talha");
   isimler.Add("Onur");
   isimler.Reverse();
   Console.WriteLine(isimler[0]);
   Console.WriteLine(isimler[1]);
   Console.WriteLine(isimler[2]);
   Console.WriteLine(isimler[3]);
   Console.ReadKey();
}
```

Kodumuzu çalıştırdığımızda dizi sıralamasının tersine çevrilmiş olduğunu ve "*Onur, Talha, Murat, Hasan*" şeklinde yazıldığını görürüz.

Capacity Özelliği

Bir dizi için tahsis edilmiş eleman kapasitesini verir. **Count** değeri, **Capacity** değerine ulaştığında, **Capacity** değeri otomatik olarak iki kat arttırılmış yeni bir dizi oluşturularak, eski dizinin elemanları bu yeni diziye üzerlerinde bir değişiklik yapılmadan aktarılır. Bu arttırma işlemi 4'ün katları şeklinde gerçekleştirilmektedir.

```
static void Main(string[] args)
{
 ArrayList rakamlar = new ArrayList();
 rakamlar.Add(1);
 Console.WriteLine("Eleman Sayısı= " + rakamlar.Count);
 Console.WriteLine("Kapasite= " + rakamlar.Capacity);
 Console.ReadKey();
}
```

Kodumuzu çalıştırdığımızda "*Eleman Sayısı=1*", "*Kapasite=4*" sonucu görüntülenir.

Dizimize…

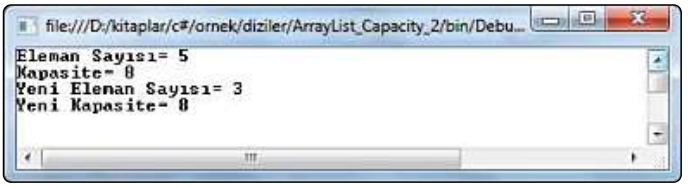
```
ArrayList rakamlar = new ArrayList();
rakamlar.Add(1);
rakamlar.Add(2);
rakamlar.Add(3);
rakamlar.Add(4);
rakamlar.Add(5);
```

… şeklinde yeni elemanlar ekleyerek çalıştırdığımızda ise, "*Eleman Sayısı=5*", "*Kapasite=8*" sonucu görüntülenir. Görüldüğü gibi **Count** değeri **Capacity** değerine ulaştığında otomatik olarak iki kat arttırıldı.

Dizi içerisinden değer silinse bile **Capacity** eski değerini koruyacaktır.

```
static void Main(string[] args)
{
 ArrayList rakamlar = new ArrayList();
 rakamlar.Add(1);
 rakamlar.Add(2);
```

```
 rakamlar.Add(3);
 rakamlar.Add(4);
 rakamlar.Add(5);
 Console.WriteLine("Eleman Sayısı= " + rakamlar.Count);
 Console.WriteLine("Kapasite= " + rakamlar.Capacity);
 rakamlar.Remove(5);
 rakamlar.Remove(4);
 Console.WriteLine("Yeni Eleman Sayısı= " + rakamlar.Count);
 Console.WriteLine("Yeni Kapasite= " + rakamlar.Capacity);
 Console.ReadKey();
}
```



Kodumuzu çalıştırdığımızda dizi içerisinden 2 adet eleman silmemize rağmen, **`Capacity`** değeri değişmeyerek **`8`** değeri görüntülenecektir.

GetRange Metodu

Bir dizinin belirtilen kısmını geriye döndüren bir metoddur.

Kullanımı şu şekildedir:

```
Dizi adı. GetRange(index, adet)
```

- **Indeks:** Dizinin başlangıç değerini belirtir.
- **Adet:** Dizinin başlangıç değerinden itibaren kaç adet elemanın alınacağını belirtir.

Örneğin; **`GetRange(2,3)`** ifadesi, dizinin 2. elemanından itibaren 3 adet elemanın sırasıyla alınacağını belirtir.

```
static void Main(string[] args)
{
 ArrayList diller = new ArrayList();
 diller.Add("VB.Net");
```

```
 diller.Add("C#");
 diller.Add("Visual C++");
 diller.Add("ASP.Net");
 ArrayList digerdiller = diller.GetRange(0, 2);
 foreach (string eleman in digerdiller)
  Console.WriteLine(eleman);
 Console.ReadKey();
}
```

Örneğimizi çalıştırdığımızda…

```
VB.Net
C#
```

… şeklinde çıktı verecektir.

`Diller.GetRange(0,2)` ifadesi ile **`diller`** dizisinin ilk elemanından itibaren 2 elemanını (VB.Net ve C#) alarak, **`digerdiller`** adı ile oluşturmuş olduğumuz yeni dizi içerisine aktarıyoruz.

`foreach` döngüsü ile **`digerdiller`** dizisi içerisinde mevcut değerleri ekrana yazdırdığımızda, dizi içerisinde sadece "*VB.Net*" ve "*C#*" değerleri mevcut olduğu için bu değerler ekrana yazdırıldı.

AddRange Metodu

Bir diziye belirtilen başka bir diziyi eklemek amacı ile kullanılan bir metoddur.

Kullanımı şu şekildedir:

```
Ekleme yapılacak dizi adı.AddRange(eklenecek dizi adı)
```

```
static void Main(string[] args)
{
 ArrayList diller = new ArrayList();
 diller.Add("VB.Net");
 diller.Add("C#");
 diller.Add("ASP.Net");

 ArrayList digerdiller = new ArrayList();
 digerdiller.Add("Python");
 digerdiller.Add("F#");
```

```
 digerdiller.Add("C++");

 diller.AddRange(digerdiller);
 foreach (string eleman in diller)
    Console.WriteLine(eleman);
 Console.ReadKey();
}
```

Örneğimizde **diller** dizisi, **digerdiller** dizisi elemanları ile birleştirilerek içeriği yazdırıldı. Kodumuzu çalıştırdığımızda **diller** dizisinin yeni elemanları olan "*VB.Net, C#, ASP.Net, Python, F#, C++*" değerlerinin ekrana yazdırıldığını görürüz.

InsertRange Metodu

Belirtilen bir diziyi, belirtilen index numarasından itibaren başka bir diziye ekleme amacı ile kullanılan metoddur.

Kullanımı şu şekildedir:

```
HedefDizi.InsertRange(index, KaynakDizi)
```

```
static void Main(string[] args)
{
 ArrayList diller = new ArrayList();
 diller.Add("VB.Net");
 diller.Add("C#");
 diller.Add("ASP.Net");

 ArrayList digerdiller = new ArrayList();
 digerdiller.Add("Python");
 digerdiller.Add("F#");
 digerdiller.Add("C++");

 diller.InsertRange(2, digerdiller);
 foreach (string eleman in diller)
           Console.WriteLine(eleman);
 Console.ReadKey();
}
```

diller dizisinin 2 nolu elemanından itibaren **digerdiller** dizisinin elamanlarını ekledik. Kodumuzu çalıştırdığımızda "VB.*Net, C#, Python, F#, C++, ASP.Net*" şeklindeki sıralama ile sonuç üretecektir. Görüldüğü gibi **diller** dizisinin ilk 2 elemanı olan "*VB.Net*" ve "*C#"* elemanlarından sonra, **digerdiller** dizisinin elemanları olan "*Python, F#, C++*" elemanları eklendi. **diller** dizisinin 3. elemanı olan "*ASP.Net*" elamanı ise, eklenen **digerdiller** dizisinin elemanlarından sonra yer aldı.

RemoveRange Metodu

Belirtilen index numarasından itibaren belirtilen sayıdaki elemanı diziden silme amacı ile kullanılan metoddur.

Kullanımı şu şekildedir:

```
Diziadı.Removerange(index,adet)
```

```
static void Main(string[] args)
{
 ArrayList diller = new ArrayList();
 diller.Add("VB.Net");
 diller.Add("C#");
 diller.Add("ASP.Net");
 diller.RemoveRange(0, 2);
 Console.Write(diller[0]);
 Console.ReadKey();
}
```

Örneğimizde **diller** dizisi içerisinde ilk elemandan itibaren 2 adet elemanı sildik. Kodumuzu çalıştırdığımızda "*ASP.Net*" sonucunu üretecektir. Çünkü ilk iki sıradaki eleman silindiğinden dolayı, ilk eleman olarak "*ASP.Net*" elemanı görüntülendi.

CopyTo Metodu

Bir dizinin içeriğini başka bir diziye kopyalamak amacı ile kullanılan bir metoddur. **CopyTo** metodu aynı zamanda **System.Array** sınıfı altında da mevcut olan bir metoddur.

Çeşitli kullanım şekilleri vardır.

```
Kaynak Dizi Adı.CopyTo(Hedef Dizi Adı)
```

Bu şekildeki kullanımda hedef dizi içerisindeki elemanlar kaynak diziye, `0` numaralı ilk index numarasından itibaren kopyalanır.

```
static void Main(string[] args)
{
 Array diller = Array.CreateInstance(typeof(string), 3);
 diller.SetValue("VB.Net", 0);
 diller.SetValue("C#", 1);
 diller.SetValue("ASP.Net", 2);

 ArrayList digerdiller = new ArrayList();
 digerdiller.Add("Python");
 digerdiller.Add("F#");
 digerdiller.Add("C++");
 digerdiller.CopyTo(diller);

 Console.WriteLine(diller.GetValue(0));
 Console.WriteLine(diller.GetValue(1));
 Console.WriteLine(diller.GetValue(2));
 Console.ReadKey();
}
```

Kodumuzu çalıştırdığımızda **`digerdiller`** dizisinin elemanları olan "*Python, F#, C++*" elemanlarının, **`diller`** dizisi içerisine ilk elemandan itibaren kopyalandığını görürüz. **`diller`** dizisi elemanlarını ilk eleman numarasından itibaren ekrana yazdırdığımızda "*Python, F#, C++*" şeklinde listelendiğini görürüz.

```
Kaynak Dizi Adı.CopyTo(Hedef Dizi Adı,İndex Numarası)
```

Bu kullanımda ise, kopyalama işlemi belirtilen index numarasından itibaren başlanır. Ancak hedef dizinin boyutunun eklenecek yeni elemanlarla birlikte, oluşacak dizinin boyutundan küçük olmaması gerekir.

```
static void Main(string[] args)
{
 Array diller = Array.CreateInstance(typeof(string), 4);
```

```
 diller.SetValue("VB.Net", 0);
 diller.SetValue("C#", 1);
 diller.SetValue("ASP.Net", 2);

 ArrayList digerdiller = new ArrayList();
 digerdiller.Add("Python");
 digerdiller.Add("F#");
 digerdiller.Add("C++");

 digerdiller.CopyTo(diller,1);
 foreach (string eleman in diller)
           Console.WriteLine(eleman);
 Console.ReadKey();
}
```

Kodumuzda görüldüğü gibi…

```
digerdiller.CopyTo(diller, 1);
```

… komut satırı ile kopyalama işlemi, **`diller`** dizisinin 1. elemanından itibaren yapılacaktır. Eklenen 3 elemanla birlikte **`diller`** dizisi 4 elemana sahip olacaktır. Bu nedenle…

```
Array diller = Array.CreateInstance(typeof(string), 4);
```

… satırı ile **`diller`** dizisinin boyutunu 4 elemanlı olarak tanımladık. Eğer daha önce olduğu gibi diller dizisini 3 elemanlı olarak tanımlamış olsaydık, kodumuzu çalıştırdığımızda "*Destination array was not long enough. Check destIndex and length, and the array's lower bounds.*" şeklinde bir hata alırdık. Kodumuzu yukarıdaki hali ile çalıştırdığımızda **`diller`** dizisinin elemanları "*VB.Net, Python, F#, C++*" şeklinde değerler listelenir. diller dizisinin ilk elemanı olan "*VB.Net*" değerinden sonra, **`digerdiller`** dizisinin elemanları olan "*Python, F#, C++*" değerleri eklendi.

```
Kaynak Dizi Adı.CopyTo(Kaynak dizi index numarası, Hedef Dizi Adı,
Hangi index numarasına kopyalanacak, Kaç adet eleman kopyalanacak)
```

Bu kullanım şekli ile kopyalama işleminin hangi elemandan başlayacağı, hangi index numarasına ekleneceği, kaç adet elemanın kopyalanacağı gibi parametreler kullanılarak kopyalama işlemi birçok seçenekle gerçekleştirilebilir.

Yukarıda yazdığımız kod üzerinde değişiklikler yaparak bu kullanım şeklini deneyelim.

```
digerdiller.CopyTo(0, diller, 0, 0);
```

Bu şekilde yazdığımızda **digerdiller** dizisinin ilk elemanından itibaren elemanlar alınıp, **diller** dizisinin ilk elemanından itibaren eklenecektir. Ancak kopyalanacak eleman parametresi **0** olarak belirtildiğinden dolayı, hiçbir eleman kopyalanmayıp sonuç olarak "*VB.Net, C#, ASP.Net*" şeklinde **diller** dizisi üzerinde hiçbir değişiklik olmamış halde listelenecektir.

```
digerdiller.CopyTo(0, diller, 0, 2);
```

digerdiller dizisinin ilk elemanından itibaren 2 adet eleman alınıp, **diller** dizisinin ilk elemanından itibaren eklenecektir. Sonuç olarak "*Python, F#, ASP.Net*" şeklinde **diller** dizisi yeni elemanlarıyla listelenecektir.

```
digerdiller.CopyTo(0, diller, 1, 2);
```

digerdiller dizisinin ilk elemanından itibaren 2 adet eleman alınıp, **diller** dizisinin ikinci elemanından itibaren eklenecektir. Sonuç olarak "*VB.Net, Python, F#*" şeklinde diller dizisi yeni elemanlarıyla listelenecektir.

```
digerdiller.CopyTo(1, diller, 0, 2);
```

digerdiller dizisinin ikinci elemanından itibaren 2 adet eleman alınıp, **diller** dizisinin ikinci elemanından itibaren eklenecektir. Sonuç olarak "*F#, C++, ASP.Net*" şeklinde **diller** dizisi yeni elemanlarıyla listelenecektir.

Clone Metodu

Belirtilen dizinin kopyasını object türünden almak için kullanılan bir metoddur.

Kullanımı şu şekildedir:

```
Atanacak değişken adı=KaynakDiziAdı.Clone()
```

Örnek olarak bir dizinin kopyalama işlemini iki farklı şekilde yaparak aralarındaki farkı görelim.

```
static void Main(string[] args)
{
 ArrayList diller = new ArrayList();
 diller.Add("VB.Net");
 diller.Add("C#");
 diller.Add("ASP.Net");

 ArrayList digerdiller = diller;
 diller.Remove("ASP.Net");

 Console.WriteLine(digerdiller[0]);
 Console.WriteLine(digerdiller[1]);
 Console.WriteLine(digerdiller[2]);
 Console.ReadKey();
}
```

Kodumuzda **diller** dizisinin içeriğini **digerdiller** adlı diziye aktarıyoruz. Bu aktarma işleminden sonra **diller** dizisinin elemanlarından "*ASP.Net*" adlı son elemanı siliyoruz. Hem **diller** değişkeni hem de **digerdiller** değişkeni referans türden değişken olmaları nedeni ile aynı bellek bölgesini paylaşmaktadırlar. Dolayısı ile **diller** dizisinde yaptığımız değişiklik **digerdiller** dizisini de etkilemektedir. Dolayısı ile kodumuz...

```
Console.WriteLine(digerdiller[2]);
```

… satırında ilgili elemanın silinmiş olması nedeni ile hata verecektir.

```
ArrayList digerdiller = diller;
```

… satırında yaptığımız aktarma işlemini **Clone** metodu ile yeniden yaparsak:

```
ArrayList digerdiller = new ArrayList();
digerdiller = (ArrayList)diller.Clone();
```

Kodumuzu çalıştırdığımızda artık **diller** değişkeninin object türünden olması dolayısı ile **diller** ve **digerdiller** değişkenleri farklı bellek bölgelerini paylaşmakta ve **diller** dizisinde yapılan değişiklik **digerdiller** dizisini etkilemeyecektir. Netice olarak kodumuz hata vermeden "*VB.Net, C#, ASP.Net*" şeklinde **digerdiller** dizisi elemanlarını listeleyecektir.

ToArray Metodu

ArrayList içerisindeki elemanları yeni bir dizi oluşturarak içerisine, istenilen veri türü ile kopyalamak amacıyla kullanılır.

Kullanımı şu şekildedir:

```
Veri Türü [ ]Atanacak Dizi Değişkeni Adı
= (Veri Türü[ ]) KaynakDiziAdı.ToArray(typeof(Veri Türü));
```

Örnek olarak yeni bir dizi oluşturarak içerisine bir **ArrayList** içerisindeki değerleri aktaralım.

```
static void Main(string[] args)
{
 ArrayList diller = new ArrayList();
 diller.Add("VB.Net");
 diller.Add("C#");
 diller.Add("ASP.Net");
 string[] yenidizi = (string[])diller.ToArray(typeof(string));
 foreach (string eleman in yenidizi)
  Console.WriteLine(eleman);
 Console.ReadKey();
}
```

Örneğimizde diller isimli yeni bir **ArrayList** oluşturup, içerisine **Add** metodu ile değerleri ekliyoruz.

```
string[] yenidizi = (string[])diller.ToArray(typeof(string));
```

Bu satırda ise, **String** tipte **yenidizi** isimli bir dizi oluşturarak, **diller** isimli **ArrayList** içerisindeki değerleri aktarıyoruz. **diller** isimli **ArrayList** içerisindeki değerler **object** türünden olduğu için, **string** türüne dönüştürerek değerleri aktarıyoruz.

Hashtable Sınıfı

System.Collections sınıfı altında yer alan **HashTable** sınıfı, içerisindeki değerlerin index numarasıyla değil de anahtar yardımıyla saklandığı bir koleksiyon türüdür. **Hashtable** içerisindeki anahtar ve değerlere **ICollections** arayüzünü kullanarak erişebiliriz.

Add metodu ile **Hashtable** içerisine değer eklenebilirken, **Remove** metodu ile de **Hashtable** içerisindeki değerler silinebilir.

Add metodu şu şekilde kullanılabilir.

```
Hashtable ismi.Add(anahtar, değer)
```

Örnek olarak bir **Hashtable** oluşturarak, **Add** metodunu kullanarak içerisine değer ekleyelim.

```
static void Main(string[] args)
{
 Hashtable sınıflar = new Hashtable();
 sınıflar.Add("E10A", "Endüstri Meslek");
 sınıflar.Add("E10B", "Endüstri Meslek");
 sınıflar.Add("T10A", "Teknik Lise");
 sınıflar.Add("A10A", "Anadolu Teknik");
 sınıflar.Add("A10B", "Anadolu Teknik");

 ICollection degerler = sınıflar.Values;

 foreach (string eleman in degerler)
    Console.WriteLine(eleman);
 Console.ReadLine();
}
```

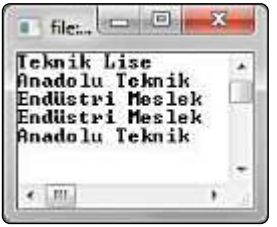
Kodumuzda **sınıflar** adlı bir **Hashtable** tanımladık. **Add** metodunu kullanarak içerisine…

```
sınıflar.Add("E10A", "Endüstri Meslek");
```

…şeklinde ilk parametrede anahtar, ikinci parametrede değer olacak şekilde ekleme yapıyoruz. Kodumuzda da görüldüğü gibi her anahtar birbirinden farklı olmak zorundadır. Değerler ise, birden fazla olacak biçimde kullanılabilir.

```
ICollection degerler = sınıflar.Values;
```

…satırında **ICollection** arayüzünden faydalanarak **degerler** isimli bir koleksiyon tanımlayıp, içerisine **Hashtable** içerisindeki değerleri aktardık.



foreach döngüsünü kullanarak da **degerler** isimli koleksiyon içerisine aktarılmış olan **sınıflar** adlı **Hashtable** içerisindeki değerleri konsol ekranına yazdırıyoruz.

Hashtable içerisindeki değerler **values** şeklinde ifade edilirken, anahtarlar ise **keys** şeklinde ifade edilmektedir. Uygulamamızı değerler yerine anahtarları listeleyecek şekilde düzenlemek için…

```
ICollection degerler = sınıflar.Values;
foreach (string eleman in degerler)
   Console.WriteLine(eleman);
```

… satırlarını…

```
ICollection anahtarlar = sınıflar.Keys;
   foreach (string eleman in anahtarlar)
    Console.WriteLine(eleman);
```

… şeklinde düzenleyelim.



Kodumuzu çalıştırdığımızda artık değerler yerine anahtarlar listelenecektir.

Remove metodu…

```
Hashtable ismi.Remove(anahtar)
```

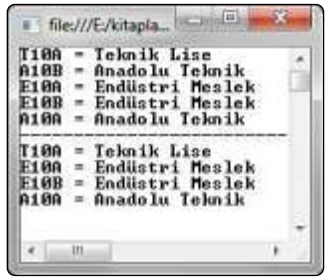
… şeklinde kullanılabilir. Görüldüğü gibi anahtar ismi belirtilerek silme işlemi yapılmaktadır.

Örnek olarak oluşturmuş olduğumuz bir **Hashtable** içerisinden, **Remove** metodunu kullanarak silme işlemi yapalım.

```
static void Main(string[] args)
{
 Hashtable sınıflar = new Hashtable();
 sınıflar.Add("E10A", "Endüstri Meslek");
 sınıflar.Add("E10B", "Endüstri Meslek");
 sınıflar.Add("T10A", "Teknik Lise");
 sınıflar.Add("A10A", "Anadolu Teknik");
 sınıflar.Add("A10B", "Anadolu Teknik");

 foreach (DictionaryEntry eleman in sınıflar)
    Console.WriteLine(eleman.Key + " = " + eleman.Value);

 sınıflar.Remove("A10B");
 Console.WriteLine("-------------------------");
 foreach (DictionaryEntry eleman in sınıflar)
    Console.WriteLine(eleman.Key + " = " + eleman.Value);
 Console.ReadLine();
}
```



Kodumuzda, tanımlayıp, içerisine **Add** metodu ile değer eklemiş olduğumuz **sınıflar** adlı **Hashtable** içerisindeki "*A10B*" anahtarını ve bu anahtara ait "*Anadolu Teknik*" değerini **Remove** metodu ile siliyoruz. Silme öncesi ve silme sonrası farkı görebilmek için iki defa listeleme işlemi yapıyoruz. Ancak listeleme işlemini yaparken, bir önceki örneğimizde yaptığımız gibi **ICollection** arayüzünü kullanmadık. Örneğimizde **DictionaryEntry** yapısını kullandık. **DictionaryEntry** yapısını kullanarak **Hashtable** içerisindeki hem anahtar hem de değere aynı anda erişebilirsiniz.

Hashtable içerisine değer aktarımı yaparken anahtar belirtmeden yalnızca değerleri girersek, anahtarlar varsayılan olarak oluşturulur.

Örnek bir **Hashtable** oluşturarak içerisindeki değerleri **listBox** içerisine aktaran bir **Windows Form** uygulaması yapalım. **Form** üzerine 1 adet **Button** nesnesi ve 1 adet de **listBox** ekleyerek, **Button** nesnesinin **Click** olayına aşağıdaki kodu yazıyoruz.

```
private void btnEkle_Click(object sender, EventArgs e)
{
 Hashtable isimler = new Hashtable();
 isimler[0] = ("Volkan AKTAŞ");
 isimler[1] = ("İrfan MERGAN");
 isimler[2] = ("Bünyamin KARAHAN");
 isimler[3] = ("Hale KATMER");
 isimler[4] = ("Tamer MATBAN");
 foreach (DictionaryEntry eleman in isimler)
    listBox1.Items.Add(eleman.Key + " = " + eleman.Value);
}
```



Örneğimizi çalıştırdığımızda görüldüğü gibi kodumuzda anahtarları belirtmeden…

```
isimler[0] = ("Volkan AKTAŞ");
```

… şeklinde yalnızca değerleri girdiğimiz halde anahtarlar varsayılan olarak eklendi.

Hashtable içerisinde bir anahtar ya da değerin mevcut olup olmadığını belirlemek için **ContainsKey** ve **ContainsValue** metodları kullanılabilir. **ContainsKey** metodu belirtilen anahtarın mevcut olup olmadığını bulmak amacıyla kullanılırken, **ContainsValue** metodu belirtilen değerin mevcut olup

olmadığını bulmak amacıyla kullanılır. Eğer dönen değer **`true`** ise aranan değer bulunmuş, **`false`** ise bulunamamıştır.

Örnek olarak oluşturacağımız bir **`Hashtable`** içerisinde bir anahtarın mevcut olup olmadığını kontrol eden bir uygulama yapalım.

```
static void Main(string[] args)
{
 string aranan;
 Hashtable sınıflar = new Hashtable();
 sınıflar.Add("E10A", "Endüstri Meslek");
 sınıflar.Add("E10B", "Endüstri Meslek");
 sınıflar.Add("T10A", "Teknik Lise");
 sınıflar.Add("A10A", "Anadolu Teknik");
 sınıflar.Add("A10B", "Anadolu Teknik");
 Console.Write("Aranan anahtarı giriniz:");

 aranan = Console.ReadLine();
 if (sınıflar.ContainsKey(aranan) == false)
 {
  Console.Write("Aranan anahtar bulunamadı");
 }
 else
 {
  Console.Write("Aranan anahtar bulundu");
 }
 Console.ReadKey();
}
```

Kodumuzda **`ContainsValue`** metodunu kullanarak konsol ekranında girilen anahtarın mevcut olup olmadığını konsol ekranına yazdırıyoruz. Eğer anahtar yerine değer kontrolü yapacak olsaydık;

```
if (sınıflar.ContainsKey(aranan) == false)
```

… satırını…

```
if (sınıflar.ContainsValue(aranan) == false)
```

… şeklinde değiştirecektik.

SortedList Sınıfı

Elemanlarına erişimin hem index numaraları hem de anahtar yardımıyla yapılabildiği bir koleksiyon türüdür. **`System.Collections`** sınıfı altında yer alan bir sınıftır. Adından da anlaşıldığı gibi içerisindeki değerler sıralı olarak saklanır.

`IndexOfKey` metodunu kullanarak, anahtar ismini verdiğimiz bir elemanın index numarasını elde edebiliriz.

```
static void Main(string[] args)
{
 SortedList sınıflar = new SortedList();
 sınıflar.Add("E10A", "Endüstri Meslek");
 sınıflar.Add("E10B", "Endüstri Meslek");
 sınıflar.Add("T10A", "Teknik Lise");
 sınıflar.Add("A10A", "Anadolu Teknik");
 sınıflar.Add("A10B", "Anadolu Teknik");
 Console.WriteLine(sınıflar.IndexOfKey("T10A"));
 Console.ReadLine();
}
```

Kodumuzu çalıştırdığımızda "*T10A*" anahtarının index'i olarak **`4`** rakamı görüntülenir.

`IndexOfValue` metodunu kullanarak da değer ismini verdiğimiz bir elemanın index numarasını elde edebiliriz.

```
Console.WriteLine(sınıflar.IndexOfKey("T10A"));
```

… satırını…

```
Console.WriteLine(sınıflar.IndexOfValue("Teknik Lise"));
```

… şeklinde değiştirip çalıştırdığımızda, uygulamamız "*Teknik Lise*" değerinin index numarası olan **`4`** rakamı görüntülenecektir.

`GetKey` metodunu kullanarak index değerini belirttiğimiz bir elemanın anahtarını elde edebiliriz. Örneğin; **`4`** nolu index değerine sahip anahtarı elde etmek için kodumuzu…

```
Console.WriteLine(sınıflar.GetKey(4));
```

… şeklinde yazarız. Kodumuz "*T10A*" sonucunu üretecektir.

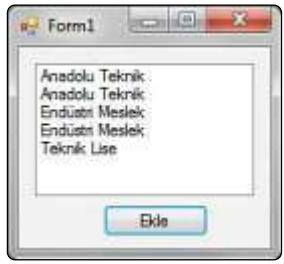
GetByIndex metodunu kullanarak index değerini belirttiğimiz bir elemanın değerini elde edebiliriz. Örneğin; **4** nolu index değerine sahip değeri elde etmek için kodumuzu…

```
Console.WriteLine(sınıflar.GetByIndex(4));
```

… şeklinde yazarız. Kodumuz "*Teknik Lise*" sonucunu üretecektir.

GetByIndex metodunu kullanarak **SortedList** içerisindeki bütün değerleri **listBox** içerisine aktaran bir uygulama yapalım.

```
private void btnEkle_Click(object sender, EventArgs e)
{
 int i;
 SortedList sınıflar = new SortedList();
 sınıflar.Add("E10A", "Endüstri Meslek");
 sınıflar.Add("E10B", "Endüstri Meslek");
 sınıflar.Add("T10A", "Teknik Lise");
 sınıflar.Add("A10A", "Anadolu Teknik");
 sınıflar.Add("A10B", "Anadolu Teknik");
 for (i=0; i < sınıflar.Count; i++)
   listBox1.Items.Add(sınıflar.GetByIndex(i));
}
```



Kodumuzda **sınıflar** adlı **SortedList**'in içeri-sindeki eleman sayısı kadar bir döngü kurarak, döngü içerisinde **GetByIndex** metodu ile sırasıyla index değerini vererek değerleri **listBox** içerisine aktarıyoruz.

DictionaryEntry yapısını kullanarak **SortedList** içerisindeki hem anahtara hem de değere aynı anda erişebilirsiniz.

Yukarıdaki kodumuzu buna göre düzenleyelim.

```
private void btnEkle_Click(object sender, EventArgs e)
{
 SortedList sınıflar = new SortedList();
 sınıflar.Add("E10A", "Endüstri Meslek");
 sınıflar.Add("E10B", "Endüstri Meslek");
 sınıflar.Add("T10A", "Teknik Lise");
 sınıflar.Add("A10A", "Anadolu Teknik");
 sınıflar.Add("A10B", "Anadolu Teknik");
 foreach (DictionaryEntry eleman in sınıflar)
      listBox1.Items.Add(eleman.Key + " = " + eleman.Value);
}
```



Uygulamamızı çalıştırdığımızda, **DictionaryEntry** yapısını kullanmamızdan dolayı, **SortedList** içerisindeki anahtarlara **Key** özelliği ile değerlere de **Value** özelliği ile erişerek **listBox** içerisine ekledik.

IDictionaryEnumarator arayüzünü kullanarak da aynı işlemi gerçekleştirebiliriz. Kodumuzu buna göre düzenleyelim.

```
private void btnEkle_Click(object sender, EventArgs e)
{
 SortedList sınıflar = new SortedList();
 sınıflar.Add("E10A", "Endüstri Meslek");
 sınıflar.Add("E10B", "Endüstri Meslek");
 sınıflar.Add("T10A", "Teknik Lise");
 sınıflar.Add("A10A", "Anadolu Teknik");
 sınıflar.Add("A10B", "Anadolu Teknik");
```

```
 IDictionaryEnumerator eleman = sınıflar.GetEnumerator();
 while (eleman.MoveNext())
 {
   listBox1.Items.Add(eleman.Key.ToString() + " = " + eleman.Value.ToString());
 }
}
```

Kodumuzda **`IDictionaryEnumerator`** arayüzüne ait **`GetEnumerator`** metodu ile **`SortedList`** içerisindeki elemanları aldık. **`MoveNext`** metodu ile **`while`** döngüsü içerisinde sırasıyla bu elemanları gezerek, anahtar ve değerleri **listBox** içerisine ekledik. Kodumuzu çalıştırdığımızda bir önceki örnekteki ile aynı sonucu verecektir.

`TryGetValue` metodunu kullanarak belirtilen anahtara ait değeri elde edebiliriz. Aşağıdaki biçimde kullanılır.

```
TryGetValue(Anahtar ismi, out Değerin aktarılacağı değişken ismi)
```

Verilen anahtara karşılık gelen değer, out ile belirtilen değişkene aktarılır.

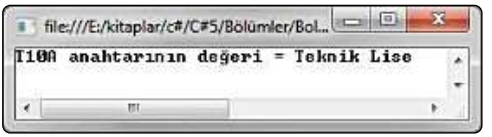
Ancak bunun için **`SortedList`**'i tanımlarken…

```
SortedList<Anahtar değişken türü, Değer değişken türü>
```

… şeklinde tanımlamamız gerekmektedir.

```
static void Main(string[] args)
{
  SortedList<string, string> sınıflar = new SortedList<string, string>();
  sınıflar.Add("E10A", "Endüstri Meslek");
  sınıflar.Add("E10B", "Endüstri Meslek");
  sınıflar.Add("T10A", "Teknik Lise");
  sınıflar.Add("A10A", "Anadolu Teknik");
  sınıflar.Add("A10B", "Anadolu Teknik");
  string deger;
  if (sınıflar.TryGetValue("T10A", out deger))
  {
   Console.WriteLine("T10A anahtarının değeri = " + deger);
  }
```

```
  Console.ReadLine();
}
```



Kodumuzda "*T10A*" anahtarına karşılık gelen değer, deger isimli değişkene aktarılarak konsol ekranına yazdırılmıştır.

IComparer interface den türeteceğimiz kendi karşılaştırma sınıfımızı yazarak, sıralama işlemin yönünü tersine çevirebiliriz. Örnek olarak…

```
private void btnEkle_Click(object sender, EventArgs e)
{
 SortedList sınıflar = new SortedList();
 sınıflar.Add("E10A", "Endüstri Meslek");
 sınıflar.Add("E10B", "Endüstri Meslek");
 sınıflar.Add("T10A", "Teknik Lise");
 sınıflar.Add("A10A", "Anadolu Teknik");
 sınıflar.Add("A10B", "Anadolu Teknik");
 foreach (DictionaryEntry eleman in sınıflar)
   listBox1.Items.Add(eleman.Key + " = " + eleman.Value);
}
```



… biçiminde yazdığımız kod küçükten büyüğe doğru sıralayacak şekilde sonuç üretiyordu.

IComparer interface'den türeteceğimiz kendi karşılaştırma sınıfımızı yazacak şekilde kodumuzu düzenleyelim.

```
namespace SortedList_IComparer
{
  public partial class Form1 : Form
  {
 public class Compare : IComparer
 {
   int IComparer.Compare(Object x, Object y)
   {
    return ((new CaseInsensitiveComparer()).Compare(y, x));
   }
 }
 public Form1()
 {
   InitializeComponent();
 }
 private void btnEkle_Click(object sender, EventArgs e)
 {
   SortedList sınıflar = new SortedList(new Compare());
   sınıflar.Add("E10A", "Endüstri Meslek");
   sınıflar.Add("E10B", "Endüstri Meslek");
   sınıflar.Add("T10A", "Teknik Lise");
   sınıflar.Add("A10A", "Anadolu Teknik");
   sınıflar.Add("A10B", "Anadolu Teknik");
   foreach (DictionaryEntry eleman in sınıflar)
    listBox1.Items.Add(eleman.Key + " = " + eleman.Value);
 }
}
}
```



Kodumuzda **Compare** isimli **IComparer** interface'den türettiğimiz bir karşılaştırma sınıfı yazıyoruz. Bu sınıfın içerisinde **CaseInsensitiveComparer.Compare** metodunu kullanıyoruz. Kodumuzu bu hali ile çalıştırdığımızda sıralama işleminin yönünün tersine çevrilerek büyükten küçüğe doğru sıralama yapıldığını görürüz.

Eğer sıralama işleminin yönünü küçükten büyüğe olacak şekilde değiştirmek istersek; kodumuzu…

```
CaseInsensitiveComparer()).Compare(x, y)
```

… şeklinde **x** ve **y** parametrelerini yer değiştirerek yazmalıyız.

Generic Koleksiyonlar

Generic koleksiyonlar, içerisinde saklanabilecek veri türüne karar verilebilen, aynı türdeki verilerin bir arada saklandığı koleksiyon türleridir. **System.Collections.Generic** namespace'i altında yer alırlar. **Non-Generic** koleksiyonlarda olduğu gibi veriler object türden olmadığı için **boxing** ve **unboxing** işlemleri yapılmaz. Dolayısıyla bu da **Non-Generic** koleksiyonlara göre daha performanslı çalışmasını sağlar.

Generic koleksiyon başlığı altında **List**, **Stack**, **Queue**, **LinkedList**, **Dictionary**, **SortedDictionary**, **SortedSet** ve **HashSet** sınıflarını inceleyeceğiz.

List Sınıfı

Veri türü belirtilerek dizi tanımlamak amacı ile kullanılır. Veri türü belirtildiğinden dolayı, içerisine aktarılan değerler, herhangi bir **boxing** işlemine tabi tutulmayacaktır. Dolayısı ile **boxing** işleminin gerçekleştirildiği **ArrayList** sınıfına göre daha performanslı çalışacaktır. Herhangi bir eleman sınırı konulmadan tanımlanır. **List**'in boyutu, içerisine eklenen eleman sayısına göre otomatik olarak belirlenir. **List** sınıfı **System.Collections.Generic** sınıfına ait bir sınıftır. Aşağıdaki gibi tanımlanır:

```
List<Değişken Türü> isim = new List<Değişken Türü>()
```



`List` sınıfı için **`ArrayList`** sınıfının generic karşılığıdır, şeklinde bir nitelendirme yapabiliriz. Hemen hemen tüm metodları ortaktır. Dolayısı ile bu metodları tekrar açıklamayacağız. Yalnızca **`List`** sınıfının kullanımına dair birkaç örnek yapacağız.

Örnek olarak basitçe bir list nesnesi oluşturup list içerisine değer ekleyip silelim.

```
static void Main(string[] args)
{
 List<string> listemiz = new List<string>();
 listemiz.Add("VB.NET");
 listemiz.Add("C#");
 listemiz.Add("ASP.NET");
 Console.WriteLine(listemiz[listemiz.Count-1]);
 listemiz.RemoveAt(listemiz.Count-1);
 Console.WriteLine(listemiz[listemiz.Count-1]);
 Console.ReadLine();
}
```

Örneğimizde…

```
List<string> listemiz = new List<string>();
```

… satırı ile listemiz adında **`string`** tipte yeni bir **`List`** nesnesi oluşturduk.

Oluşturduğumuz **`List`** nesnesi içerisine **`Add`** metodu ile 3 adet eleman ekledik.

```
Console.WriteLine(listemiz[listemiz.Count-1]);
```

… satırında **`Count`** metodunu kullanarak, **`listemiz`** adlı **`List`** nesnesinin son elemanı olan "*ASP.NET*" değerini görüntüledik.

Ardından…

```
listemiz.RemoveAt(listemiz.Count-1);
```

… satırında **`List`** nesnesinin son elemanı olan "*ASP.NET*" değerini sildik.

Son olarak…

```
Console.WriteLine(listemiz[listemiz.Count-1]);
```

… satırında dizinin son elemanını yeniden yazdırdığımızda, artık **`List`** nesnesinin son elemanı olarak "*C#*" değerinin yazdırıldığını görürüz.

list nesnesi içerisindeki değerleri **`ArrayList`** içerisine aktaran örnek bir uygulama yapalım.

```
static void Main(string[] args)
{
 List<string> listemiz = new List<string>();
 listemiz.Add("VB.NET");
 listemiz.Add("C#");
 listemiz.Add("ASP.NET");

 ArrayList yenidizi = new ArrayList(listemiz);

 foreach (string eleman in yenidizi)
    Console.WriteLine(eleman);
 Console.ReadKey();
}
```

Örneğimizde…

```
ArrayList yenidizi = new ArrayList(listemiz);
```

… satırında yenidizi isminde **`List`** nesnesi referansı ile yeni bir **`ArrayList`** oluşturduk. **`List`** nesnesi **`ICollection`** arayüzünü kullandığı ve **`ArrayList`** de **`ICollection`**'dan **implement** edildiği için ArrayList'i bu şekilde tanımlayabildik.

```
foreach (string eleman in yenidizi)
   Console.WriteLine(eleman);
```

… satırlarında ise **`foreach`** döngüsü ile yenidizi isimli **`ArrayList`** içerisindeki değerleri konsol ekranında yazdırdık.

`list` nesnesinin performansını gösteren örnek bir uygulama yapalım. Örneğimizde öncelikle 100 000 adet rastgele sayıyı **`listbox`** içerisine ekleyeceğiz. Sayılar rastgele olarak eklendiği için haliyle içerisinde tekrarlanan sayılar olacaktır. Örneğimizde öncelikle iç içe for döngüsü kullanarak tekrarlı

sayıları sileceğiz, diğer yöntemde ise list nesnesini kullanarak sayıların tekrarsız biçimde yazılmasını sağlayacağız. Kodumuz aşağıdaki gibi olacak:

```
DateTime başlangıçzamanı;
DateTime bitiş;
TimeSpan fark;
private void btnAktar_Click(object sender, EventArgs e)
{
    listBox1.Items.Clear();
    Random rasgele = new Random();
    int sayı;
    for (int i = 0; i < 100000; i++)
    {
        sayı = rasgele.Next(1, 100);
        listBox1.Items.Add(sayı.ToString());
    }
}
private void btnforTekrarsız_Click(object sender, EventArgs e)
{
    başlangıçzamanı = DateTime.Now;
    for (int i = 0; i < listBox1.Items.Count - 1; i++)
    {
        for (int j = listBox1.Items.Count - 1; j >= i + 1; j += -1)
        {
            if (listBox1.Items[i].ToString() ==
listBox1.Items[j].ToString())
                listBox1.Items.RemoveAt(j);
        }
    }
    bitiş = DateTime.Now;
    fark = bitiş.Subtract(başlangıçzamanı);
    lblForSonuc.Text= fark.TotalSeconds.ToString()+" sn";
}
private void btnlistTekrarsiz_Click(object sender, EventArgs e)
{
    başlangıçzamanı = DateTime.Now;
    List<string> liste = listBox1.Items.OfType<string>().ToList();
    listBox1.Items.Clear();
    listBox1.Items.AddRange(liste.Distinct().ToArray());
    bitiş = DateTime.Now;
```

```
    fark = bitiş.Subtract(başlangıçzamanı);
    lbListSonuc.Text=fark.TotalSeconds.ToString() + " sn";
}
```

Örneğimizde öncelikle;

```
Random rasgele = new Random();
sayı = rasgele.Next(1, 100);
```

satırlarında 1 ile 100 arasında rastgele sayı oluşturuyoruz. Döngümüz 100 000 defa çalıştığı için, 100 000 rastgele sayı üretilmekte ve;

```
listBox1.Items.Add(sayı.ToString());
```

satırında üretilen bu rastgele sayı **listBox** içerisine eklenmektedir.

Sayıları tekrarsız bir biçimde listelemek için öncelikle iç içe **`for`** döngüsünü kullanıyoruz. Döngü içerisinde **listBox** içerisinde okunan değer, bir sonraki değere eşit ise **`RemoveAt`** metodu ile sonraki değer silinmektedir.

list nesnesini kullanarak sayıları tekrarsız biçimde listelemek için ise öncelikle;

```
List<string> liste = listBox1.Items.OfType<string>().ToList();
```

satırında liste isminde **`string`** türde bir **`list`** nesnesi tanımlayarak, **listBox** içerisindeki değerleri tanımladığımız list nesnesine aktarıyoruz.

```
listBox1.Items.AddRange(liste.Distinct().ToArray());
```

satırında ise **`list`** nesnesinin **Distinct** metoduyla **`list`** nesnesi içerisindeki değerlerin tekrarsız biçimde oluşturulmasını, **`ToArray`** metoduyla da diziye dönüştürülmesini sağlıyoruz. **`AddRange`** metodu kullanarak da list nesnesinden dizi türüne dönüştürülmüş olan değerleri **listBox** içerisine aktarıyoruz.



Uygulamamızı çalıştırıp, işlem zamanlarını incelediğimizde list nesnesinin ne kadar performanslı bir şekilde çalıştığını görebiliriz.

Stack (Yığın) Sınıfı

`System.Collections.Generic` sınıfına ait bir sınıftır. **Last in first out** (*LIFO*) yöntemine göre çalışır. Yani yığın içerisine eklenen son elemana ilk olarak erişilebilirken, ilk olarak eklenen elemana ise en sonda erişilebilir.

`Push` metodu stack içerisine değer eklemek için kullanılmaktadır. İlk eklenen değer en sona eklenecektir. Örnek bir **`stack`** oluşturarak içerisine değer ekleyelim.

```
static void Main(string[] args)
{
 Stack<string> isimler = new Stack<string>();
 isimler.Push("Talha");
 isimler.Push("Sümeyye");
 isimler.Push("Azra");
 isimler.Push("Murat");
 foreach (string i in isimler)
  Console.WriteLine(i);
 Console.ReadLine();
}
```



Uygulamamızı çalıştırdığımızda görüldüğü gibi son eklenen "*Murat*" elemanı en başta listelenmektedir.

`ElementAt` özelliğini kullanarak stack içerisinde index numarası verilen bir elemanı görüntüleyebiliriz. Yukarıdaki örneğimizi buna göre düzenleyelim.

```
static void Main(string[] args)
{
 int i;
 Stack<string> isimler = new Stack<string>();
 isimler.Push("Talha");
 isimler.Push("Sümeyye");
 isimler.Push("Azra");
 isimler.Push("Murat");
 for (i = 0; i<isimler.Count ; i++)
  Console.WriteLine(isimler.ElementAt(i));
 Console.ReadLine();
}
```

Kodumuzda **`Count`** özelliğini kullanarak stack içerisindeki eleman sayısını elde ettik ve döngümüzü bu eleman sayısı kadar kurduk. Döngü içerisinde **`ElementAt`** özelliği ile sırasıyla index numarası belirtilen elemanları konsol ekranına yazdırdık. Uygulamamız bir önceki örnekle aynı sonucu vererek, stack içerisindeki elemanları index numarasına göre listeleyecektir.

`Peek` metodu stack içerisinde en üst sırada bulunan elemanı vermektedir.

Yukarıdaki örnek kodumuza göre…

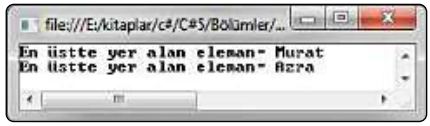
```
Console.WriteLine(isimler.Peek());
```

… satırını çalıştırdığımızda "*Murat*" değerini konsol ekranında görüntüleyecek-

tir. "*Murat*" elemanı stack içerisine en son girildiğinden dolayı en üstte yer almaktadır.

Pop metodu stack içerisinden eleman silmek amacıyla kullanılır. **Stack** içerisinden silinen eleman en üstte yer alana eleman olacaktır.

```
static void Main(string[] args)
{
 Stack<string> isimler = new Stack<string>();
 isimler.Push("Talha");
 isimler.Push("Sümeyye");
 isimler.Push("Azra");
 isimler.Push("Murat");
 Console.WriteLine("En üstte yer alan eleman= " + isimler.Peek());
 isimler.Pop();
 Console.WriteLine("En üstte yer alan eleman= " +isimler.Peek());
 Console.ReadLine();}
}
```



Kodumuzu çalıştırdığımızda ilk olarak **Peek** metodu ile stack içerisindeki en üstte olan eleman olan "*Murat*" listelenmektedir. Ardından **Pop** metodu ile en üstte yer alan eleman olan "*Murat*" elemanı silinecek ve artık en üstte yer alan eleman olan "*Azra*" elemanı konsol ekranında görüntülenecektir.

Queue (Kuyruk) Sınıfı

System.Collections.Generic sınıfına ait bir sınıftır. FIFO yöntemine göre çalışır. Yani kuyruk içerisine eklenen ilk elemana ilk olarak erişilebilirken, son olarak eklenen elemana en sonda erişilebilir.

Enqueue metodu queue içerisine değer eklemek için kullanılmaktadır. İlk eklenen değer en başa eklenecektir.

Stack sınıfında örnek olarak tanımladığımız değerleri bu sefer de **Queue** sınıfı için tanımlayalım.

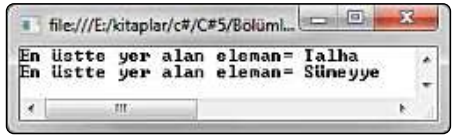
```
static void Main(string[] args)
{
 Queue<string> isimler = new Queue<string>();
 isimler.Enqueue("Talha");
 isimler.Enqueue("Sümeyye");
 isimler.Enqueue("Azra");
 isimler.Enqueue("Murat");
 foreach (string i in isimler)
  Console.WriteLine(i);
 Console.ReadLine();
}
```



Kodumuzu çalıştırdığımızda görüldüğü gibi **`Stack`** sınıfının tersine olarak ilk eklenen "*Talha*" değeri en başta, en son eklenen "*Murat*" değeri ise en sonda yer almaktadır.

`Peek` metodu, **`Stack`** sınıfında olduğu gibi queue içerisinde de en üst sırada olan elemanı bulmak amacıyla kullanılır. **`Queue`** içerisinden en üstteki elemanı silmek için ise, **`Dequeue`** metodu kullanılmaktadır.

```
static void Main(string[] args)
{
 Stack<string> isimler = new Stack<string>();
 isimler.Push("Talha");
 isimler.Push("Sümeyye");
 isimler.Push("Azra");
 isimler.Push("Murat");
 Console.WriteLine("En üstte yer alan eleman= " + isimler.Peek());
 isimler.Pop();
 Console.WriteLine("En üstte yer alan eleman= " +isimler.Peek());
 Console.ReadLine();}
}
```



Kodumuzu çalıştırdığımızda **Peek** metodu ile queue içerisindeki en üstte olan eleman olan "*Talha*" listelenmektedir. "*Talha*" değeri **Queue** içerisine ilk olarak girildiği için en üstte yer almaktadır. Ardından **Dequeue** metodu ile en üstte yer alan eleman olan "*Talha*" elemanı silinecek ve bir sonraki sırada yer alan "*Azra*" elemanı artık en üstte yer alan eleman olarak konsol ekranında görüntülenecektir.

LinkedList (Bağlı Liste) Sınıfı

System.Collections.Generic sınıfına ait bir sınıftır. Elemanlarının birbirlerine link şeklinde bağlı olduğu koleksiyon türüdür.

AddFirst metodu bağlı liste içerisinde ilk konuma değer eklemek için kullanılırken, **AddLast** metodu da bağlı liste içerisinde son konuma değer eklemek için kullanılmaktadır.

```
static void Main(string[] args)
{
 LinkedList<string> liste = new LinkedList<string>();
 liste.AddLast("Değer 2");
 liste.AddLast("Değer 3");
 liste.AddLast("Değer 4");
 liste.AddFirst("Değer 1");
 foreach (var eleman in liste)
  Console.WriteLine(eleman);
 Console.ReadLine();
}
```



Kodumuzda "*Değer 1*" isimli elemanımız **`AddFirst`** metodu kullanılarak eklendiği için diğer elemanlardan sonra eklenmesine rağmen, bağlı liste içerisinde en üst sırada yer almaktadır.

`First` özelliği ile bağlı liste içerisindeki ilk elemana erişilirken, **`Last`** özelliği ile de son elemana erişilir.

```
static void Main(string[] args)
{
 LinkedList<string> liste = new LinkedList<string>();
 liste.AddLast("Değer 2");
 liste.AddLast("Değer 3");
 liste.AddLast("Değer 4");
 liste.AddFirst("Değer 1");
 Console.WriteLine("Listenin ilk elemanı= " +liste.First.Value);
 Console.WriteLine("Listenin son elemanı= " + liste.Last.Value);
 Console.ReadLine();
}
```



Kodumuzda **`Value`** ifadesi, belirtilen özelliğin değerini görüntülemek için kullanılmaktadır.

Bağlı liste içerisinde belirtilen değeri silmek için **`Remove`** metodunu kullanırız.

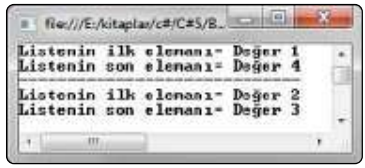
```
static void Main(string[] args)
{
 LinkedList<string> liste = new LinkedList<string>();
 liste.AddLast("Değer 2");
 liste.AddLast("Değer 3");
 liste.AddLast("Değer 4");
 liste.AddFirst("Değer 1");
 liste.Remove("Değer 3");
 foreach (var eleman in liste)
  Console.WriteLine(eleman);
 Console.ReadLine();
}
```



Kodumuzda **Remove** metodu ile "*Değer 3*" adlı elemanı bağlı liste içerisinden sildik.

RemoveFirst metodu ile bağlı liste içerisindeki ilk elemanı silerken, **RemoveLast** metodu ile de bağlı liste içerisindeki son elemanı silebiliriz.

```
static void Main(string[] args)
{
  LinkedList<string> liste = new LinkedList<string>();
  liste.AddLast("Değer 2");
  liste.AddLast("Değer 3");
  liste.AddLast("Değer 4");
  liste.AddFirst("Değer 1");
  Console.WriteLine("Listenin ilk elemanı= " +liste.First.Value);
  Console.WriteLine("Listenin son elemanı= " + liste.Last.Value);
  liste.RemoveFirst();
  liste.RemoveLast();
  Console.WriteLine("-----------------------------");
  Console.WriteLine("Listenin ilk elemanı= " + liste.First.Value);
  Console.WriteLine("Listenin son elemanı= " + liste.Last.Value);
  Console.ReadLine();
}
```

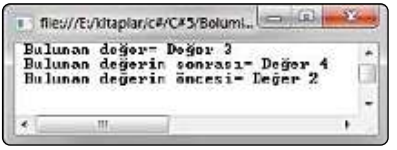


Uygulamamızda öncelikle bağlı listenin ilk ve son elemanlarını **First** ve **Last** özellikleri ile konsol ekranına yazdırdık. Ardından **RemoveFirst** ve **RemoveLast** metodları ile listenin ilk ve son elemanlarını sildik. Son olarak silme işleminin ardından listenin ilk ve son elemanlarını tekrar konsol ekranına yazdırdık.

Bağlı liste içerisinde arama yapmak amacıyla **Find** metodu kullanılmaktadır. **Find** metodu ile birlikte **Next** ve **Previous** özelliklerini kullanabiliriz. **Next** özelliği bulunan değerin bir sonraki değerini, **Previous** özelliği ise bir önceki değerini belirtir.

```
static void Main(string[] args)
{
  LinkedList<string> liste = new LinkedList<string>();
  liste.AddLast("Değer 2");
  liste.AddLast("Değer 3");
  liste.AddLast("Değer 4");
  liste.AddFirst("Değer 1");

  LinkedListNode<string> bulunan = liste.Find("Değer 3");
  Console.WriteLine(" Bulunan değer= " +bulunan.Value.ToString());
  Console.WriteLine(" Bulunan değerin sonrası= " +
  bulunan.Next.Value.ToString());
  Console.WriteLine(" Bulunan değerin öncesi= " +
  bulunan.Previous.Value.ToString());
  Console.ReadLine();
}
```



Kodumuzda **Find** metodu ile belirtilen değeri bulduk.

```
Console.WriteLine(" Bulunan değer= " +bulunan.Value.ToString());
```

… satırında bulunan değeri **Value** özelliği ile elde ederek konsol ekranına yazdırdık.

```
Console.WriteLine(" Bulunan değerin sonrası= " +
    bulunan.Next.Value.ToString());
```

… satırında bulunan değerin bir sonraki değerini elde ederek konsol ekranına yazdırdık.

```
Console.WriteLine(" Bulunan değerin öncesi= " +
     bulunan.Previous.Value.ToString());
```

… satırında ise bulunan değerin bir önceki değerini elde ederek konsol ekranına yazdırdık.

AddBefore ve **AddAfter** metodlarını kullanarak **Find** metodu ile bulunan değerin bir öncesi ve bir sonrası konumuna ekleme yapabiliriz. **AddBefore** bir önceki konuma ekleme yaparken, **AddAfter** bir sonraki konuma ekleme yapar. Aşağıdaki görüldüğü gibi kullanılır.

```
Bağlı liste ismi.AddBefore.(Bağlı liste ismi.Find("Aranan değer" ,
"Eklenecek değer"))
```

```
static void Main(string[] args)
{
  LinkedList<string> liste = new LinkedList<string>();
  liste.AddLast("Değer 2");
  liste.AddLast("Değer 3");
  liste.AddLast("Değer 4");
  liste.AddFirst("Değer 1");

  liste.AddBefore(liste.Find("Değer 3"), "Eklenen Değer");
  foreach (var eleman in liste)
   Console.WriteLine(eleman);
  Console.ReadLine();
}
```



Kodumuzda…

```
liste.AddBefore(liste.Find("Değer 3"), "Eklenen Değer");
```

…satırında **AddBefore** metodu içerisinde **Find** metodunu kullanarak "*Değer 3*" adlı elemanı aratıp, bulunan değerin bir önceki konumuna "*Eklenen Değer*" elemanını ekliyoruz.



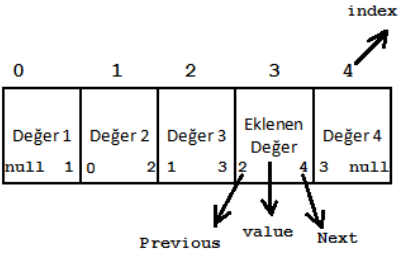
AddAfter metodunu kullanarak…

```
liste.AddBefore(liste.Find("Değer 3"), "Eklenen Değer");
```

… satırını…

```
liste.AddAfter(liste.Find("Değer 3"), "Eklenen Değer");
```

… biçiminde düzenleyip uygulamamızı çalıştırdığımızda "*Değer 3*" adlı elemanı bulunup, bir sonraki konumuna "*Eklenen Değer*" elemanını eklenmektedir.



LinkedList içerisinde her eleman 3 özellik ile birlikte saklanmaktadır. Bunlar **Previous**, **Next**, **Value** özellikleridir. **Value** elemanın değerini, **Previous** bir önceki elemanın index değerini, **next** ise bir sonraki elemanın index değerini saklar.

```
static void Main(string[] args)
{
  LinkedList<string> liste = new LinkedList<string>();
  liste.AddLast("Değer 2");
  liste.AddLast("Değer 3");
```

```
  liste.AddLast("Değer 4");
  liste.AddFirst("Değer 1");

  LinkedListNode<string> eleman = liste.First;
  while (eleman != null)
  {
    Console.WriteLine(eleman.Value);
    eleman = eleman.Next;
  }
  Console.ReadLine();
}
```

Kodumuzda…

```
LinkedListNode<string> eleman = liste.First;
```

… satırında **`First`** metodu ile listenin ilk elemanına hareket ettik.

`while` döngüsü içerisinde ilk elemandan itibaren, sırasıyla elemanların değerini yazdırdık. Döngü içerisinde bir sonraki elemana geçiş yapmak için **`Next`** metodunu kullandık.

Eğer baştan sona doğru değil de, son elemandan başa doğru bir yazdırma işlemi yapacak olsaydık, kodumuzda…

```
LinkedListNode<string> eleman = liste.First;
while (eleman != null)
{
  Console.WriteLine(eleman.Value);
  eleman = eleman.Next;
}
```

… satırlarını…

```
LinkedListNode<string> eleman = liste.Last;
while (eleman != null)
{
 Console.WriteLine(eleman.Value);
 eleman = eleman.Previous;
}
```

… biçiminde düzenlememiz gerekecekti. Burada **Last** metodu ile listenin son elemanına hareket ederek, **while** döngüsü içerisinde son elemandan itibaren, sırasıyla elemanların değerini yazdırdık. Döngü içerisinde bir önceki elemana geçiş yapmak için **Previous** metodunu kullandık.

Dictionary (Sözlük) Sınıfı

System.Collections.Generic sınıfına ait bir sınıftır. İçerisindeki değerlerin anahtar yardımıyla saklandığı bir koleksiyon türüdür. **Dictionary** içerisindeki anahtarların türü belirtilmelidir. Her anahtar birbirinden farklı olmak zorundadır. Değerler ise birden fazla olacak biçimde kullanılabilir.

Aşağıda görüldüğü gibi tanımlanır.

```
Dictionary<Anahtar değişken türü, Değer değişken türü > DictionaryİsmI
= new Dictionary< Anahtar değişken türü, Değer değişken türü >( );
```

Add metodunu kullanarak **Dictionary** içerisine değer ekleyebiliriz.

```
Dictionary ismi.Add(Anahtar, Değer)
```

… şeklinde kullanılır.

```
static void Main(string[] args)
{
  Dictionary<string, string> dallar = new Dictionary<string, string>();
  dallar.Add("Volkan AKTAŞ", "Programlama");
  dallar.Add("Bünyamin KARAHAN", "Programlama");
  dallar.Add("İrfan MERGAN", "Donanım");
  dallar.Add("Hale KATMER", "Web Tasarım");
  Console.WriteLine(dallar["Bünyamin KARAHAN"]);
  Console.ReadLine();
}
```

Kodumuzda **Add** metodunu kullanarak anahar ve değerleri ekledik.

```
Console.WriteLine(dallar["Bünyamin KARAHAN"]);
```

Bu satırda ise belirtilen anahtara karşılık gelen değer konsol ekranında yazdırılmaktadır. Uygulamamızı çalıştırdığımızda "*Bünyamin KARAHAN*"

anahtarına ait olan "*Programlama*" değeri konsol ekranında yazdırılmaktadır. Kodumuzdan da görüldüğü gibi anahtarlar birbirinden farklı iken, değerler birbirinin aynı olabilmektedir.

foreach döngüsünü kullanarak **Dictionary** içerisindeki hem anahtar hem de değerleri birlikte listeletebiliriz.

```
static void Main(string[] args)
{
  Dictionary<string, string> dallar = new Dictionary<string, string>();
  dallar.Add("Volkan AKTAŞ", "Programlama");
  dallar.Add("Bünyamin KARAHAN", "Programlama");
  dallar.Add("İrfan MERGAN", "Donanım");
  dallar.Add("Hale KATMER", "Web Tasarım");
  foreach (var eleman in dallar)
     Console.WriteLine(eleman);
  Console.ReadLine();
}
```



Kodumuzu çalıştırdığımızda görüldüğü gibi **dallar** isimli **Dictionary** içerisindeki hem anahtar hem de değerlere **foreach** döngüsü ile eriştik.

Dictionary içerisindeki değerler **Value** şeklinde ifade edilirken, anahtarlar ise **Key** şeklinde ifade edilmektedir. Yukarıdaki örneğimizdeki **foreach** döngüsünü…

```
foreach (var eleman in dallar)
   Console.WriteLine(eleman.Key + " = " + eleman.Value);
```

… biçiminde yazarak da **Value** ve **Key** özelliklerini kullanarak anahtar ve değerleri birlikte listeletebiliriz.

KeyValuePair özelliğini kullanarak da **Dictionary** içerisinde tanımlanmış olan anahtar, değer ikilisine erişebiliriz. Bunun için **foreach** döngüsünü aşağıdaki gibi değiştirelim.

```
foreach (KeyValuePair<string, string> eleman in dallar)
   Console.WriteLine(eleman.Key + " = " + eleman.Value);
```

Bu kodumuz aynı şekilde sonuç verecektir.

Remove metodunu kullanarak **Dictionary** içerisinden anahtarı belirterek istediğiniz değeri silebilirsiniz.

```
static void Main(string[] args)
{
  Dictionary<string, string> dallar = new Dictionary<string, string>()
  {
    {"Volkan AKTAŞ", "Programlama"},
    {"Bünyamin KARAHAN", "Programlama"},
    {"İrfan MERGAN", "Donanım"},
    {"Hale KATMER", "Web Tasarım"}
  };
  dallar.Remove("İrfan MERGAN");
  foreach (var eleman in dallar)
  Console.WriteLine(eleman);
  Console.ReadLine();
}
```



Kodumuzda **Remove** metodunu kullanarak "*İrfan MERGAN*" anahtarını ve bu anahtara ait "*Donanım*" değerini sildik. **Dictionary** içerisine eleman eklenmesini **Add** metodu ile değil de, sırasıyla **{anahtar,değer}** ikilisi şeklinde yaptığımıza dikkat edelim. Bu şekilde de **Dictionary** tanımlaması yapmak mümkündür. Eleman değerlerinin girildiği sırada **Dictionary** içerisine eklenecektir. Silme işlemi sonrası **Dictionary** içerisindeki değerleri listelediğimizde "*İrfan MERGAN*" adlı anahtar ve bu anahtara ait "*Donanım*" değerinin **Dictionary** içerisinden silindiğini görebiliriz.

Dictionary içersindeki herhangi bir değerin içeriğini…

```
Dictionary Adı[Anahtar Adı] = Yeni Değer
```

… biçiminde değiştirebiliriz.

Örneğin…

```
dallar["Bünyamin KARAHAN"] = "Donanım";
```

… satırında "*Bünyamin KARAHAN*" anahtarının yeni değeri "*Donanım*" olacaktır.

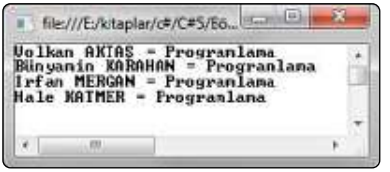
ToList metodunu kullanarak oluşturmuş olduğumuz **Dictionary**'i, **List** nesnesine dönüştürebiliriz.

```
static void Main(string[] args)
{
  Dictionary<string, string> dallar = new Dictionary<string, string>()
  {
    {"Volkan AKTAŞ", "Programlama"},
    {"Bünyamin KARAHAN", "Programlama"},
    {"İrfan MERGAN", "Donanım"},
    {"Hale KATMER", "Web Tasarım"}
  };
  foreach (var eleman in dallar.Keys.ToList())
  dallar[eleman] = "Programlama";
  foreach (KeyValuePair<string, string> eleman in dallar)
  Console.WriteLine(eleman.Key + " = " + eleman.Value);
  Console.ReadLine();
}
```

Kodumuzda…

```
foreach (var eleman in dallar.Keys.ToList())
  dallar[eleman] = "Programlama";
```

… satırlarında **dallar** adlı **Dictionary** içerisindeki anahtarları (*keys*) **ToList** metodu ile **List** nesnesine dönüştürdük.



foreach döngüsü içeriisnde sırasıyla anahtarı verilen değerleri, "*Programlama*" biçiminde değiştirdik.

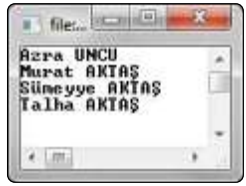
Dictionary içerisindeki anahtar ya da değerleri sıralamak için **List** nesnesine ait olan **Sort** metodunu kullanabiliriz.

```
static void Main(string[] args)
{
  Dictionary<string, int> bilgi = new Dictionary<string, int>()
  {
   {"Talha AKTAŞ", 10},
   {"Sümeyye AKTAŞ", 7},
   {"Murat AKTAŞ" , 17},
   {"Azra UNCU", 4}
  };
  List<string> lst_bilgi = new List<string>(bilgi.Keys);
  lst_bilgi.Sort();
  foreach (string anahtar in lst_bilgi)
     Console.WriteLine(anahtar);
   Console.ReadLine();
}
```

Kodumuzda…

```
List<string> lst_bilgi = new List<string>(bilgi.Keys);
```

… satırında bilgi isimli **Dictionary**'den **lst_bilgi** isimli bir **List** nesnesi türeterek, **Dictionary** içerisindeki anahtarları (**bilgi.Keys** ifadesi ile) içerisine aktardık.



Sort metodu ile listeyi sıralayarak **foreach** döngüsü ile harf sırasına göre küçükten büyüğe doğru sıralı halini konsol ekranına yazdırdık.

Dictionary içerisinde LINQ sorguları da kullanabilirsniz. Bu sayede **Dictionary** içerisindeki veriler üzerinde daha çeşitli işlemler yapabilirsiniz. Örneğin; bir önceki örneğimizde yaptığımız sıralama işlemini LINQ sorgusu ile yapalım.

```
static void Main(string[] args)
{
  Dictionary<string, int> bilgi = new Dictionary<string, int>()
  {
   {"Talha AKTAŞ", 10},
   {"Sümeyye AKTAŞ", 7},
   {"Murat AKTAŞ" , 17},
   {"Azra UNCU", 4}
  };
  var sırala = from eleman in bilgi orderby
      eleman.Key ascending select eleman.Key;
  foreach (string anahtar in sırala)
       Console.WriteLine(anahtar);
   Console.ReadLine();
}
```

Sıralama işlemini yapan LINQ sorgusu…

```
var sırala = from eleman in bilgi orderby
     eleman.Key ascending select eleman.Key;
```

… satırında yer almaktadır. **orderby** sorgusu sıralama amacıyla kullanılır. **select** sorgusunu burada anahtarları seçmek amacıyla kullanıyoruz.



ascending ise, sıralama işleminin küçükten büyüğe artan şekilde olacağını belirtmektedir. **descending** ifadesini kullanarak sıralama işleminin resimde görüldüğü gibi büyükten küçüğe doğru azalan bir şekilde olmasını sağlayabiliriz.

Örneğin anahtarlar içerisinde ilk harfi "*M*" olanları listeleyen bir LINQ sorgusu yazalım.

```
static void Main(string[] args)
{
  Dictionary<string, int> bilgi = new Dictionary<string, int>()
  {
   {"Talha AKTAŞ", 10},
   {"Sümeyye AKTAŞ", 7},
   {"Murat AKTAŞ" , 17},
   {"Mehmet GÖKMEN", 19}
  };
  var arama = bilgi.Where(eleman => eleman.Key.StartsWith("M"))
          .Select(eleman => eleman.Key);
  foreach (string anahtar in arama)
          Console.WriteLine(anahtar);
  Console.ReadLine();
}
```



where sorgusu ile **Dictionary** içerisinde arama yapıyoruz. **StartsWith** ise belirtilen karakter ile başlayan anahtarlar üzerinde arama yapmamızı sağlıyor. Belirtilen kritere uygun olan anahtarları **foreach** döngüsü içerisinde listeliyoruz.

Dictionary Initializer

C# 6'dan önceki sürümlerde Dictionary tanımlamamızı yaparken anahtar ve değerleri;

```
Dictionary<string, int> bilgi = new Dictionary<string, int>()
{
    {"Talha", 12},
    {"Sümeyye", 9},
    {"Murat" , 19},
    {"Mehmet", 21}
};
```

şeklinde yazarken, C# 6 ile birlikte;

```
Dictionary<string, int> bilgi = new Dictionary<string, int>()
{
    ["Talha"] = 12,
    ["Sümeyye"] = 9,
    ["Murat"] = 19,
    ["Mehmet"] = 21
};
```

şeklinde yazabilmekteyiz.

SortedDictionary Sinifi

`System.Collections.Generic` sınıfına ait bir sınıftır. İçerisindeki değerlerin anahtar yardımıyla saklanıp, sırılandığı bir koleksiyon türüdür. Sıralama işleminin bir miktar performans kaybına neden olmasından dolayı **`Dictionary`** sınıfına göre performansı daha yavaştır.

```
SortedDictionary<Anahtar değişken türü, Değer değişken türü >
SortedDictionaryİsmi = new Dictionary< Anahtar değişken türü,
Değer değişken türü >( );
```

… biçiminde tanımlanır.

`Dictionary` içerisinde kullanılan metodları **`SortedDictionary`** içerisinde de kullanabiliriz.

```
static void Main(string[] args)
{
  Dictionary<string, int> bilgi = new Dictionary<string, int>()
  {
   {"Talha AKTAŞ", 10},
   {"Sümeyye AKTAŞ", 7},
   {"Murat AKTAŞ" , 17},
   {"Azra UNCU", 4}
};
```



`Dictionary` kullanarak yaptığımız yukarıdaki kod satırlarımızı çalıştırdığımızda, **`Dictionary`** içerisindeki anahtar ve değerlerin girilen sırada listelendiğini görürüz.

Aynı anahtar ve değerlerin girişini **`SortedDictionary`** kullanarak yaptığımızda ise…

```
static void Main(string[] args)
{
  SortedDictionary<string, int> bilgi = new SortedDictionary<string, int>()
  {
   {"Talha AKTAŞ", 10},
   {"Sümeyye AKTAŞ", 7},
   {"Murat AKTAŞ" , 17},
   {"Azra UNCU", 4}
};
```



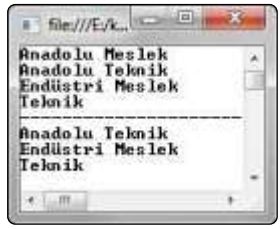
… anahtar ve değerlerin, anahtarlara göre sıralanarak listelendiğini görürüz.

SortedSet Sınıfı

System.Collections.Generics sınıfı altında yer alan bir sınıftır. Elemanları sıralı olarak yer almaktadır. Elemanları **Key** ve **Value** gibi niteliklere sahip değildir.

Add metodu ile **SortedSet** içerisine değer ekleeybilir, **Remove** metodu ile de silebiliriz.

```
static void Main(string[] args)
{
  SortedSet<string> okul = new SortedSet<string>();
  okul.Add("Teknik");
  okul.Add("Anadolu Teknik");
  okul.Add("Endüstri Meslek");
  okul.Add("Anadolu Meslek");
  foreach (string degerler in okul)
           Console.WriteLine(degerler);
  Console.WriteLine("---------------------");
  okul.Remove("Anadolu Meslek");
  foreach (string degerler in okul)
           Console.WriteLine(degerler);
  Console.ReadLine();
}
```



Kodumuzda **Add** metodu ile elemanlarımız ekletip **foreach** menüsü ile konsol ekranına listeledik. Ardından **Remove** metodunu kullanarak belirtilen değeri sildik ve **SortedSet**'in son halini yeniden listeledik.

Listeleme işleminde de görüldüğü gibi **SortedList** içerisindeki elemanlar küçükten büyüğe doğru sıralı bir şekildedir.

SortedSet içerisinde daha önceden mevcut bir değerin eklenmek istenmesi durumunda, geriye **false** değeri döner ve ekleme yapılmaz. Eklenmek istenen değer mevcut değilse, geriye **true** değeri dönerek ekleme işlemi gerçekleştirilir.

```
static void Main(string[] args)
{
  SortedSet<string> okul = new SortedSet<string>();
  okul.Add("Teknik");
  okul.Add("Anadolu Teknik");
  okul.Add("Endüstri Meslek");
  okul.Add("Anadolu Meslek");
  okul.Add("Teknik");
  foreach (string degerler in okul)
           Console.WriteLine(degerler);
  Console.ReadLine();
}
```



Uygulamamızı çalıştırdığımızda görüldüğü gibi "*Teknik*" değeri 2 defa eklenmek istenmesine rağmen eklenmemiştir.

Eklemek istediğimiz değer mevcut olduğunda geriye false değeri döneceğini belirtmiştik. Bunu test etmek için…

```
bool donen_deger=okul.Add("Teknik");
Console.WriteLine(donen_deger);
```

… satırlarını yazarak, dönen değeri kontrol ettiğimizde konsol ekranında **false** yazdığını görürüz.

RemoveWhere metodunu kullanarak silme işleminin belirteceğiniz seçeneklere göre gerçekleştirilmesini sağlayabilirsiniz.

```
static void Main(string[] args)
{
```

```
  SortedSet<string> okul = new SortedSet<string>();
  okul.Add("Teknik");
  okul.Add("Anadolu Teknik");
  okul.Add("Endüstri Meslek");
  okul.Add("Anadolu Meslek");
  foreach (string degerler in okul)
           Console.WriteLine(degerler);
  Console.WriteLine("---------------------");
  okul.RemoveWhere(deger => deger.StartsWith("A"));
  foreach (string degerler in okul)
           Console.WriteLine(degerler);
  Console.ReadLine();
}
```



Kodumuzda…

```
okul.RemoveWhere(deger => deger.StartsWith("A"));
```

…satırında **`RemoveWhere`** metodunu **`StartsWith`** seçeneği ile birlikte kullanarak **`SortedList`** içeri-sindeki elemanlardan ilk karakteri "*A*" ile başlayan-ların silinmesini sağlıyoruz.

Bir **`List`** nesnesi oluşturarak, oluşturmuş olduğunuz **`List`** nesnesinden yeni bir **`SortedList`** türetebilirsiniz.

```
static void Main(string[] args)
{
  List<string> list = new List<string>();
  list.Add("Teknik");
  list.Add("Anadolu Teknik");
  list.Add("Endüstri Meslek");
  list.Add("Anadolu Meslek");
```

```
  list.Add("Teknik");
  SortedSet<string> s_set = new SortedSet<string>(list);
  foreach (string degerler in s_set)
           Console.WriteLine(degerler);
  Console.ReadLine();
}
```



Kodumuzda öncelikle list adından yeni bir **`List`** nesnesi oluşturup içerisine değerleri ekliyoruz. "*Teknik*" adlı değerden iki adet eklendiğine dikkat edelim.

```
SortedSet<string> s_set = new SortedSet<string>(list);
```

Bu kod satırında **`list`** adlı **`List`** nesnesinden, **`s_set`** adlı yeni bir **`SortedList`** oluşturuyoruz.

Değerleri listelediğimizde "*Teknik*" elemanın yalnızca bir adet görüntülendiğine dikkat edelim.

HashSet Sınıfı

`System.Collections.Generics` sınıfı altında yer alan bir sınıftır. **`SortedSet`**'de olduğu gibi daha önceden mevcut bir değerin eklenmek istenmesi durumunda geriye **`false`** değeri döner ve ekleme yapılmaz. Eklenmek istenen değer mevcut değilse, geriye **`true`** değeri dönerek ekleme işlemi gerçekleştirilir. Farklı olarak optimize edilmiş bir **`hash`** koleksiyon olduğu için değer ekleme ve arama gibi işlemleri daha hızlı bir biçimde gerçekleştirebilirsiniz.

`SortedSet` sınıfında olduğu gibi **`Add`**, **`Remove`**, **`RemoeWhere`** metodlarını aynı şekilde kullanabilirsiniz.

```
static void Main(string[] args)
{
  HashSet<string> okul = new HashSet<string>();
```

```
  okul.Add("Teknik");
  okul.Add("Anadolu Teknik");
  okul.Add("Endüstri Meslek");
  okul.Add("Anadolu Meslek");
  foreach (string degerler in okul)
           Console.WriteLine(degerler);
  Console.WriteLine("---------------------");
  okul.Remove("Anadolu Meslek");
  foreach (string degerler in okul)
           Console.WriteLine(degerler);
  Console.ReadLine();
}
```

Yukarıdaki örneğimizde, daha önce **SortedSet** konusunda yaptığımız aynı örneği **HashSet** sınıfı ile yaptık. Görüldüğü gibi ekleme ve silme işlemi aynı şekilde gerçekleştirilmektedir.

SetEquals metodu bir **HashSet**'in başka bir koleksiyona eşit olup olmadığını belirlemek amacıyla kullanılır. Geriye **boolean** bir değer dönmektedir. Eğer dönen değer **true** ise ikisi birbirine eşittir, **false** değeri dönerse eşit değildir.

```
static void Main(string[] args)
{
  List<string> list = new List<string>();
  list.Add("C#");
  list.Add("VB.Net");
  list.Add("ASP.Net");
  list.Add("MVC");
  list.Add("C#");
  HashSet<string> h_set = new HashSet<string>();
  h_set.Add("VB.Net");
  h_set.Add("C#");
  h_set.Add("ASP.Net");
  h_set.Add("MVC");
  if (h_set.SetEquals(list))
   Console.WriteLine("Hashset, List'e eşit");
  else
   Console.WriteLine("Hashset, List'e eşit değil");
  Console.ReadLine();
}
```

Kodumuzda list adında yeni bir **List** nesnesi oluşturarak içerisine değer ekledik. Daha sonra **h_set** adında bir **HashSet** oluşturarak içerisine değer ekledik.

```
if (h_set.SetEquals(list))
```

… satırında **HashSet**'in **List** nesnesi içeriğine eşit olup olmadığını **SetEquals** metodu ile kontrol ettik.

Uygulamamızı çalıştırdığımızda "*HashSet, List'e eşit*" mesajı görüntülenecektir. **List** içerisine "*C#*" değerini iki defa eklememize rağmen, karşılaştırma yapılırken tekrar eden değer gözardı edilmiştir.

IntersectWith metodunu kullanarak **HashSet** içerisindeki değerleri bir başka koleksiyon ile karşılaştırarak ortak olan değerleri elde edebiliriz.

```
static void Main(string[] args)
{
  List<string> list = new List<string>();
  list.Add("C#");
  list.Add("VB.Net");
  list.Add("ASP.Net");
  list.Add("MVC");
  list.Add("Silverlight");
  HashSet<string> h_set = new HashSet<string>();
  h_set.Add("VB.Net");
  h_set.Add("C#");
  h_set.Add("WPF");
  h_set.IntersectWith(list);
  foreach (string degerler in h_set)
     Console.WriteLine(degerler);
  Console.ReadLine();
}
```

Kodumuzda…

```
h_set.IntersectWith(list);
```

… satırında **h_set** isimli **HashSet** ile list isimli List nesnesini karşılaştırıyoruz. Daha sonra, ortak olan değerleri foreach döngüsü içerisinde listeliyoruz.



HashSet içerisindeki değerlerden **List** içerisinde de mevcut olan değerler "*VB.Net*" ve "*C#*" değerleri olduğu için uygulamamızı çalıştırdığımızda konsol ekranında bu değerler listelenecektir.

Specialized (Özelleştirilmiş) Koleksiyonlar

Specialized koleksiyonlar, belirli işlemler ve veri türleri için özelleştirilmiş koleksiyon türleridir. **System.Collections.Specialized** namespace'i altında yer alırlar. **Specialized** koleksiyon başlığı altında **ListDictionary**, **HybridDictionary**, **OrderedDictionary** ve **StringCollection** sınıflarını inceleyeceğiz.

ListDictionary Sınıfı

System.Collections.Specialized sınıfı altında yer alan özelleştirilmiş bir sınıftır. Az sayıda eleman üzerinde yapılan işlemlerde daha performanslı çalışır. Bu nedenle eleman sayısının 10'dan az olduğu koleksiyonlarda kullanılması tavsiye edilir. İçerisindeki elemanlar **key** (*anahtar*) ve **value** (*değer*) ikilisi şeklinde saklanır. Kullanımı **HashTable** sınıfı ile aynıdır. Aynı interface ve metodlar **ListDirectory** içerisinde de kullanılabilir.

```
using System;
using System.Collections;
using System.Collections.Specialized;
using System.Text;
static void Main(string[] args)
{
  ListDictionary isimler = new ListDictionary();
  isimler.Add("Volkan AKTAŞ", 1);
  isimler.Add("Bünyamin KARAHAN", 2);
  isimler.Add("İrfan MERGAN", 3);
  isimler.Add("İsmail Hakkı GÜVEN", 4);
  foreach (DictionaryEntry eleman in isimler)
```

```
    Console.WriteLine(eleman.Key + " = " + eleman.Value);
  Console.WriteLine("---------------------");
  isimler.Remove("Volkan AKTAŞ");
  foreach (DictionaryEntry eleman in isimler)
    Console.WriteLine(eleman.Key + " = " + eleman.Value);
  Console.ReadLine();
}
```



Kodumuzda **Add** ve **Remove** metodlarını kullanarak **ListDictionary** içerisine anahtar ve değer ekleme, silme işlemi gerçekleştirdik. Önce **Add** metodu ile eklenen tüm anahtar ve değerleri **foreach** döngüsü ile listeledik. Daha sonra **Remove** metodu ile "Volkan AKTAŞ" isimli anahtarı ve buna karşılık gelen değeri silerek listenin yeni halini konsol ekranına yazdırdık. Listeleme işlemini yaparken **HashTable** sınıfı ile kullanılan **DictionaryEntry** yapısını kullandık.

ListDictionary sınıfının az sayıda eleman üzerinde yapılan işlemlerde daha performanslı çalıştığını belirtmiştik. Bunu test etmek için **ListDictionary** ile **HashTable** sınıfını karşılaştıracak bir uygulama yapalım.

```
using System;
using System.Collections;
using System.Collections.Specialized;
using System.Text;
static void Main(string[] args)
{
  DateTime baslangiczamani,bitiszamani;
  TimeSpan fark;

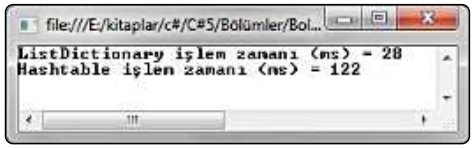
  ListDictionary isimler_list = new ListDictionary();
  isimler_list.Add("Volkan AKTAŞ", 1);
  isimler_list.Add("Bünyamin KARAHAN", 2);
  isimler_list.Add("İrfan MERGAN", 3);
```

```
  isimler_list.Add("İsmail Hakkı GÜVEN", 4);
  isimler_list.Add("Murat UNCU", 5);

  Hashtable isimler_hash = new Hashtable();
  isimler_hash.Add("Volkan AKTAŞ", 1);
  isimler_hash.Add("Bünyamin KARAHAN", 2);
  isimler_hash.Add("İrfan MERGAN", 3);
  isimler_hash.Add("İsmail Hakkı GÜVEN", 4);
  isimler_hash.Add("Murat UNCU", 5);
  baslangiczamani = DateTime.Now;
  for (int i = 0; i < 1000000; i++)
  {
   object a = isimler_list["Volkan AKTAŞ"];
  }
  bitiszamani = DateTime.Now;
  fark = bitiszamani-baslangiczamani;
  Console.WriteLine("ListDictionary işlem zamanı (ms) = " + fark.Milliseconds);

  baslangiczamani = DateTime.Now;
  for (int i = 0; i < 1000000; i++)
  {
   object a = isimler_hash["Volkan AKTAŞ"];
  }
  bitiszamani = DateTime.Now;
  fark = bitiszamani - baslangiczamani;
  Console.WriteLine("Hashtable işlem zamanı (ms) = " + fark.Milliseconds);
  Console.ReadLine();
}
```

Kodumuzda **isimler_list** adlı yeni **ListDictionary** ve **isimler_hash** adlı yeni bir **HashTable** oluşturarak içerisine anahtar ve değerleri ekledik. **TimeSpan** yapısı kullanarak başlangıç ve bitiş zamanları arasındaki farkı aldık. Her iki ekleme işleminin bitiş zamanlarını karşılaştırdığımızda **ListDictionary** ile yapılan işlemin, **HashTable** ile yapılan işleme göre daha kısa sürede sonlandığını görebiliriz. Bu sonuçtan da anlaşıldığı üzere eleman sayısının az olduğu durumlarda **LisrDictionary** kullanmak daha avantajlıdır.



HybridDictionary Sinifi

System.Collections.Specialized sınıfı altında yer alan özelleştirilmiş bir sınıftır. **HybridDictionary**, eleman sayısının belli olmadığı durumlarda kullanılan bir **ListDictionary** türüdür. **ListDictionary** sınıfının az sayıda elemana sahip koleksiyonlarda performanslı olarak çalıştığını söylemiştik. Eleman sayısının fazla olduğu durumlarda **ListDictionary** sınıfının performans sorunlarını aşmak amacıyla, **HybridDcitionary** geliştirilmiştir.

```
using System;
using System.Collections;
using System.Collections.Specialized;
using System.Text;
static void Main(string[] args)
{
  HybridDictionary hl= new HybridDictionary();
  hl.Add("C#", 1);
  hl.Add("Vb.Net", 2);
  hl.Add("Visual C++", 3);

  foreach (DictionaryEntry anahtarlar in hl)
  Console.WriteLine(anahtarlar.Key + " = " + anahtarlar.Value);
  Console.ReadLine();
}
```

Kodumuzda **hl** adından bir **HybridDictionary** tanımlayarak **Add** metodu ile içerisine anahtar ve değerleri ekledik. **HashTable** sınıfı ile kullanılan **DictionaryEntry** yapısını kullanarak anahtar ve değerleri listeledik.

HybridDictionary ile **HashTable** sınıfının performanslarını karşılaştıracak bir uygulama yapalım. Öncelikle bir sayı dizisi oluşturarak, sayısal değerler üzerinde performans karşılaştırması yapalım.

```
using System;
using System.Collections;
using System.Collections.Specialized;
using System.Text;
static void Main(string[] args)
{
   DateTime baslangiczamani, bitiszamani;
```

```
    TimeSpan fark;
    int[] dizi = new int[1000000];
    Random rs = new Random();

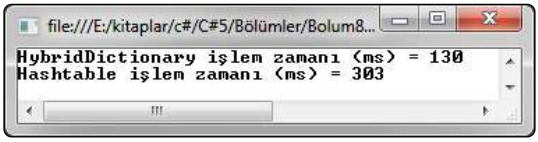
    for (int i = 0; i < 999999; i++)
    dizi[i]=rs.Next(0,100000);

    HybridDictionary hl = new HybridDictionary();
    baslangiczamani = DateTime.Now;
    for (int i = 0; i <= 999999; i++)
    hl.Add(i, dizi[i]);
    bitiszamani = DateTime.Now;
    fark = bitiszamani - baslangiczamani;
    Console.WriteLine("HybridDictionary işlem zamanı (ms) = " +
          fark.Milliseconds);

    Hashtable ht = new Hashtable();
    baslangiczamani = DateTime.Now;
    for (int i = 0; i < 999999; i++)
    ht.Add(i, dizi[i]);

    bitiszamani = DateTime.Now;
    fark = bitiszamani - baslangiczamani;
    Console.WriteLine("Hashtable işlem zamanı (ms) = " +
          fark.Milliseconds);
    Console.ReadLine();
}
```

Kodumuzda **1000000** adet rastgele sayı oluşturarak bunu bir dizi içerisine aktardık. Dizi içerisindeki elemanları da önce bir **HybridDictionary** içerisine, daha sonra da bir **HashTable** içerisine ekledik. Her iki ekleme işleminin bitiş zamanlarını incelediğimizde **HybridDictionary** ile yapılan ekleme işleminin çok daha kısa sürede sonuçlandığını görebiliriz. Bu sonuçtan da anlaşıldığı üzere sayısal değerler üzerinde yapılan işlemlerde **HybridDictionary** kullanmak **HashTable**'a göre daha avantajlıdır.



Sayısal değerler üzerinde yaptığımız performans karşılaştırmasının ardından, string bir dizi oluşturarak, string değerler üzerinde performans karşılaştırması yapalım.

```
using System;
using System.Collections;
using System.Collections.Specialized;
using System.Text;
static void Main(string[] args)
{
   string[] strChars = {"A","B","C","D","E","F","G","H","I","İ","J",
"K","L","M","N","O","Ö","P","R","S","Ş","T","U","Ü","V","Y","Z"};
   DateTime baslangiczamani, bitiszamani;
   TimeSpan fark;
   string[] dizi = new string[1000000];
   Random rnd = new Random();

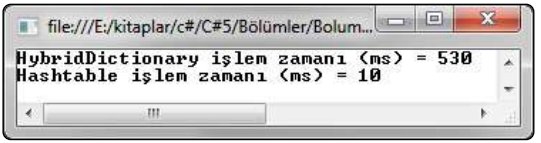
   int mb = strChars.GetUpperBound(0);
   for (int i = 0; i <999999; i++)
   dizi[i]=strChars[rnd.Next(mb)];

   HybridDictionary hl = new HybridDictionary();
   baslangiczamani = DateTime.Now;
   for (int i = 0; i <= 999999; i++)
   hl.Add(i, dizi[i]);
   bitiszamani = DateTime.Now;
   fark = bitiszamani - baslangiczamani;
   Console.WriteLine("HybridDictionary işlem zamanı (ms) = " +
            fark.Milliseconds);

   Hashtable ht = new Hashtable();
   baslangiczamani = DateTime.Now;
   for (int i = 0; i < 999999; i++)
   ht.Add(i, dizi[i]);
   bitiszamani = DateTime.Now;
   fark = bitiszamani - baslangiczamani;
   Console.WriteLine("Hashtable işlem zamanı (ms) = " +
            fark.Milliseconds);
   Console.ReadLine();
}
```

Kodumuzda alfabedeki harflerden oluşan bir karakter dizisi tanımladık. Bu karakter dizisinden `1000000` defa rastgele olarak seçilen harflerden bir string dizi oluşturarak içerisine aktardık. Dizi içerisindeki elemanları da önce bir

HybridDictionary içerisine, daha sonra da bir **HashTable** içerisine ekledik. Her iki ekleme işleminin bitiş zamanlarını incelediğimizde sayısal değerler üzerinde yaptığımız işlemin tersine olarak **HashTable** ile yapılan ekleme işleminin çok daha kısa sürede sonuçlandığını görebiliriz. Bu sonuçtan da anlaşıldığı üzere string değerler üzerinde yapılan işlemlerde **HashTable** kullanmak **HybridDictionary**'e göre daha avantajlıdır.



OrderedDictionary Sınıfı

System.Collections.Specialized sınıfı altında yer alan özelleştirilmiş bir sınıftır. İçerisindeki elemanlar **key** (*anahtar*) ve **value** (*değer*) ikilisi şeklinde saklanıp, her bir değere anahtar ya da index numarası vasıtasıyla erişebilirsiniz. İçerisine eklenen anahtar ve değerler eklenen sıraya göre saklanır.

Add metodu ile anahtar ve değer girişi yapılabileceği gibi **Insert** metodu ile de index numarasını belirterek anahtar ve değer girişi yapılabilir.

```
using System;
using System.Collections;
using System.Collections.Specialized;
using System.Text;
static void Main(string[] args)
{
  OrderedDictionary od = new OrderedDictionary();
  od.Add("T10A","Teknik");
  od.Add("A10A","Anadolu Teknik");
  od.Add("E10A","Endüstri Meslek");
  od.Insert(0,"M10A","Anadolu Meslek");

  string[] anahtarlar = new string[od.Keys.Count];
  od.Keys.CopyTo(anahtarlar, 0);
  for (int i = 0; i < od.Keys.Count; i++)
  {
   Console.WriteLine(
```

```
    "İndex = {0}, Anahtar = {1}, Değer = {2}",
    i,anahtarlar[i],od[i]);
  }
  Console.ReadLine();
}
```

Kodumuzda **od** isimli yeni bir **OrderedDictionary** oluşturarak **Add** ve **Insert** metodları ile anahtar ve değer girişi yaptık. **Add** metodu ile değer girişi yapılırken girilen sıraya göre anahtar ve değerler **OrderedDictionary** içerisinde saklanır. **Insert** metodu ile yaptığımız anahtar ve değer girişinde ise, kodumuzda belirttiğimiz 0 nolu index sırasına ekleme işlemi yapılmaktadır.

```
string[] anahtarlar = new string[od.Keys.Count];
```

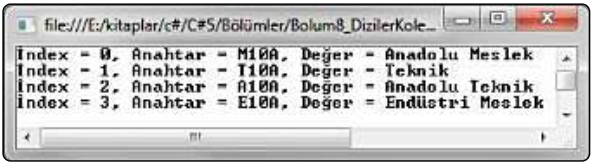
Bu kod satırında **anahtarlar** isimli **string** türde bir diziyi, **OrderedDictionary** içerisindeki anahtar sayısı kadar eleman sayısıyla tanımladık.

```
od.Keys.CopyTo(anahtarlar, 0);
```

Bu kod satırında **OrderedDictionary** içerisindeki anahtarları **CopyTo** metodunu kullanarak, anahtarlar dizisine 0 nolu index değerinden itibaren kopyaladık.

```
for (int i = 0; i < od.Keys.Count; i++)
{
   Console.WriteLine(
    "İndex = {0}, Anahtar = {1}, Değer = {2}",
    i,anahtarlar[i],od[i]);
}
```

Bu satırlarda ise, **OrderedDictionary** içerisindeki anahtar sayısı kadar bir döngü kurarak sırasıyla index numarasını, anahtarı (anahtarlar isimli diziden alınarak), değeri (**OrderedDictionary** içerisinden alınarak) listeliyoruz.

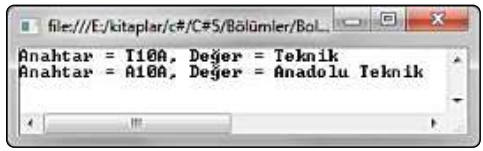


Remove metodunu kullanarak **Dictionary** içerisinden anahtarı belirterek istediğiniz değeri silebileceğiniz gibi, **RemoveAt** metodunu kullanarak da index değerini belirterek silme işlemi yapabilirsiniz.

```
using System;
using System.Collections;
using System.Collections.Specialized;
using System.Text;
static void Main(string[] args)
{
  OrderedDictionary od = new OrderedDictionary();
  od.Add("T10A", "Teknik");
  od.Add("A10A", "Anadolu Teknik");
  od.Add("E10A", "Endüstri Meslek");
  od.Insert(0, "M10A", "Anadolu Meslek");
  od.Remove("E10A");
  od.RemoveAt(0);

  IDictionaryEnumerator eleman = od.GetEnumerator();
  while (eleman.MoveNext())
  {
   Console.WriteLine("Anahtar = {0}, Değer = {1}",
   eleman.Key,eleman.Value);
  }
  Console.ReadLine();
}
```

Kodumuzda **Remove** metodunu kullanarak anahtarı belirterek, **RemoveAt** metodunu kullanarak da index değerini belirterek **OrderedDictionary** içerisinde silme işlemi gerçekleştirdik.



IDictionaryEnumerator arayüzüne ait **GetEnumerator** metodu ile **OrderedDictionary** içerisindeki elemanları aldık. **MoveNext** metodu ile **while** döngüsü içerisinde sırasıyla bu elemanları gezerek, anahtar ve değerleri konsol ekranına yazdırdık.

StringCollection Sınıfı

System.Collections.Specialized sınıfı altında yer alan özelleştirilmiş bir sınıftır. **String** koleksiyonları yönetmek için kullanışlı metodlara sahip olan bir sınıftır. Özellikle sıklıkla değişen veriler üzerinde işlem yapmak için kullanışlı bir sınıfıtr.

Add, **AddRange** ve **Insert** metodlarını kullanarak, **StringCollection** içerisine değer ekleme işlemi yapabiliriz. **Add** metodu ile eklenen değer **StringCollection** içerisinde en sona eklenecektir. **Insert** metodu ile belirtilen index numarasına değer ekleyebilirsiniz. **AddRange** metodu ise bir string dizisini, **StringCollection** içerisine eklemek amacıyla kullanılır.

```
using System;
using System.Collections.Specialized;
using System.Text;
static void Main(string[] args)
{
  StringCollection diller = new StringCollection();
  diller.Add("F#");
  diller.Add("VB.Net");
  diller.Insert(0,"C#");
  string[] scriptdiller = new string[]{"Perl", "Ruby", "Python"};
  diller.AddRange(scriptdiller);
  foreach (string isim in diller)
     Console.WriteLine(isim);
  Console.ReadLine();
}
```



Bu kodumuzda diller adında bir **StringCollection** tanımlayarak, öncelikle **Add** metodu ile **StringCollection** içerisine girilen sıraya göre değer eklemesi yaptık. Daha sonra 0 nolu index sırasına değer girişi yaptık. **scriptdiller**

adıyla string türde bir dizi oluşturarak, içerisindeki değerleri **AddRange** metodu ile **StringCollection** içerisine aktardık. Son olarak da **foreach** döngüsü içerisinde **StringCollection** içeriisndeki değerleri listeledik.

Bir **textBox** içerisindeki satırları okuyarak **StringCollection** içerisine aktaran bir uygulama yapalım.

```
private void btnAktar_Click(object sender, EventArgs e)
{
 StringCollection sc = new StringCollection();
 foreach (string satırlar in textBox1.Lines)
 {
    sc.Add(satırlar);
 }
 MessageBox.Show(sc[0]);
}
```



Kodumuzda **textBox** içerisindeki satırları **foreach** döngüsü içerisinde **Lines** özelliği ile okuyarak **Add** metodu ile **StringCollection** içerisine aktarıyoruz. **StringCollection** içerisinde ilk index sırasındaki değeri de mesaj penceresi içerisinde görüntülüyoruz.

Remove ve **RemoveAt** metodlarını kullanarak **OrderedDictionary** sınıfında olduğu gibi **StringCollection** içerisinde de değer silebilirsiniz.

.NET CORE PROJELERİNDE KULLANILAN DİZİ VE KOLEKSİYON METOTLARI

Index Yapısı

Temel olarak bir dizi veya koleksiyondaki elemanın indeks değerini saklamak amacıyla kullanılmaktadır. **Index** yapısını C# 8 sürümünden itibaren .NET Core projelerinde kullanabilirsiniz.

Örneğin;

```
int[] dizi = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

şeklinde **int** türde bir dizi tanımlayalım.

```
Index i1 = 2;
Console.WriteLine(dizi[i1])
```

Şeklinde yazacağımız kod baştan 2. indeks değeri olan **2** rakamını görüntüleyecektir.

^ operatörünü kullanarak indeks değerinin baştan değil de **sondan** itibaren alınmasını sağlayabiliriz.

```
Index i2 = ^2;
Console.WriteLine(dizi[i2])
```

Şeklinde yazacağımız kod sondan 2. indeks değeri olan **8** rakamını görüntüleyecektir.

Index türünde belirlediğimiz değişkenleri kullanarak bir **değer aralığı** belirleyebiliriz.

```
var d = dizi[i1..i2];
foreach (var eleman in d)
    Console.Write(eleman);
```

Yukarıdaki kod satırlarında Index türünde `i1` ve `i2` değişkenlerini `dizi[i1..i2]` şeklinde kullanarak baştan 2. ve sondan 2. indeks aralığındaki 2 3 4 5 6 7 değerleri yazdırılacaktır. İkinci indeks değeri olan aralığın sağındaki `i2` değerini indeks numarası değil de **sıra numarası** olarak kabul edebiliriz. Çünkü `8` değeri dizide 7. sırada olan `7` rakamına karşılık gelmektedir.

Kodumuzu herhangi bir değişken tanımlamaya gerek kalmadan

```
foreach (var eleman in dizi[2..8])
    Console.Write(eleman);
```

şeklinde düzenleyerek döngü içerisinde indeks aralığı belirtebiliriz.

`dizi[^2..^0]` şeklindeki kullanım son `2` değer olan `8` ve `9` değerlerini verecektir.

`dizi[..2]` şeklindeki kullanım ilk 2 değer olan `0` ve `1` değerlerini verecektir.

`dizi[2..]` şeklindeki kullanım 2. indeks değerinden itibaren değerler olan `23456789` değerlerini verecektir.

`dizi[..]` şeklindeki kullanım dizinin **tüm elemanlarını** verecektir.

Index yapısını stringlerin içerisindeki belli bir aralığı almak için kullanabiliriz.

Örneğin

```
string a = "KodLab yayınevi";
Console.WriteLine(a[..7]);
```

Ya da

```
Console.WriteLine("kodlab yayınevi"[..7]);
```

kod satırları "`KodLab`" sonucunu verecektir.

Range Yapısı

Bir dizi veya koleksiyonun **başlangıç** ve **bitiş** değerleri belirtilen bir aralığını temsil eder. Range yapısını C# 8 sürümünden itibaren .NET Core projelerinde kullanabilirsiniz. Yukarıda [başlangıç indeksi .. bitiş indeksi] şeklinde belirttiğimiz değer aralığını Range yapısına ait çeşitli metotlarla daha

özelleştirilmiş bir biçimde belirleyebilirsiniz. Örneğin yukarıda kullanmış olduğumuz diziyi burada tekrar kullanarak bu defa Range yapısını kullanarak elemanlara erişelim.

```
int[] dizi = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

şeklinde tanımlanmış olan diziyi **Range** yapısı ile listelemek için aşağıdaki kodları yazalım:

```
Range range = 2..8;
foreach (var eleman in dizi[range])
    Console.Write(eleman);
```

Kodumuz 2. ve 8. indeks aralığındaki **2 3 4 5 6 7** değerlerini yazdıracaktır. **Range** yapısında da **Index** yapısı konusunda belirttiğimiz gibi ikinci değer olan aralığın sağındaki 8 değerini indeks numarası değil de **sıra numarası** olarak kabul edebiliriz. Çünkü 8 değeri dizide 7. sırada olan 7 rakamına karşılık gelmektedir.

Range tanımlaması yaparken **Index** yapısını kullanarak tanımlama yapabiliriz. Buna göre yukarıdaki örneğimizi aşağıdaki gibi yazabiliriz:

```
Index i1 = 2;
Index i2 = ^2;
Range range = i1..i2;
foreach (var eleman in dizi[range])
    Console.Write(eleman);
```

Range yapısını stringlerin içerisindeki **belli bir aralığı almak için** kullanabiliriz.

Örneğin

```
Index i1 = 0;
Index i2 = 7;
Range range = i1..i2;
var a = "kodlab yayınevi"[range];
Console.WriteLine(a);
```

kod satırları "**KodLab**" sonucunu verecektir.

Range yapısı özelliklerinden **range.Start.Value** ve **range.End.Value** özellikleri

ile başlangıç ve bitiş indeks değerlerine erişebilirsiniz.

Span<T> Yapısı

Span<T> Yapısı C# 7.2 ve .NET Core 2.1 sürümünden itibaren kullanılmaya başlanan ve bitişik hafıza alanlarına **tip korumalı erişimi** sağlayan bir türdür. Tip ve bellek güvenliğini sağlayarak kodun korunmasını ve kolaylaştırır, bellek ayırma ve kopyalama işlemlerini azaltarak performansı arttırır. Bellek üzerinde yeni bir nesne oluşturmadan ya da verileri kopyalamadan bir dizinin belli bir bölümünü görünümünü elde edebilirsiniz. Bu özelliğe slice adı verilir. Bellekteki verilerin yalnız okunabilir görünümünü elde etmek için **ReadOnlySpan<T>** türü kullanılabilir.

```
int[] dizi = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
var span = dizi.AsSpan(1, 4);
foreach (var i in span)
    Console.Write(i);
```

Yukarıdaki kod satırlarımızda **`AsSpan`** metodu kullanılarak bir dizinin belirtilen aralığını alarak yeni bir **`ReadOnlySpan`** yapısı oluşturur. Kodumuzu çalıştırdığımızda **`1 2 3 4`** şeklinde çıktı verecektir.

Yukarıda tanımlamış olduğumuz **`span`** yapısının belli bir bölümünün elde etmek için kodumuzu

```
var subSpan = span[1..^1];
foreach (var i in subSpan)
    Console.Write(i);
```

Şeklinde yazabiliriz. Kodumuz **`2 3`** şeklinde **çıktı** verecektir.

```
var subSpan = span[1..^1];
```

Satırı yerine

```
var subSpan = span.Slice(1,2);
```

`Slice` özeliğini kullanarak da **`span`** yapısının belli bir bölümünü elde edebiliriz.

```
var a = Enumerable.Range(0, 100).ToArray();
```

```
int i1 = 10;
int i2 = 21;
Span<int> s = a[i1..i2];
foreach (var i in s)
    Console.Write(i);
```

Yukarıdaki kod satırlarında **System.Linq** sınıfına ait **Enumarable.Range()** metodunu kullanarak 0 ile 100 arasında bir dizi tanımladık ve **Span<int>** türünden tanımladığımız **s** değişkenine dizinin 10 ile 20 numaralı index'ler arasındaki değerleri aktardık.

HATALARI BULMAK VE KONTROL ETMEK

9

Hatalar programın çalışması esnasında oluşurlar ve programın kendi çalışmasının kesilmesine neden olurlar. Bu hatalar kullanıcı kaynaklı, yazılımcı kaynaklı ya da sistem kaynaklı olabilir. Burada yazılımcıya düşen oluşabilecek tüm hataların analizini yaparak programın çalışmasının kesilmesini engellemektir. Programın hiç hata vermemesini sağlamak da tamamen doğru bir yaklaşım değildir. Doğru olan yaklaşım; oluşabilecek hataların analizinin yapılması, gerekirse kullanıcıyı bilgilendirerek ondan alınan geri dönüşlerle gerekmiyorsa; programın kendi içerisinde yanlış sonuçlar üretmesine neden olmayacak şekilde programın akışını yönlendirerek, oluşabilecek hataları kontrol etmektir.

Oluşan hataları yakalamak için **`try catch`** yapısı kullanılır. Daha önce C++ ve Java dillerinde kullanılmakta olan bu yapı, .Net içerisinde de kullanılmaktadır.

Genel kullanımı şu şekildedir:

```
try
{
 Hatanın oluşabileceği kod satırları;
}
catch
{
 Hata oluştuğunda çalıştırılacak kod satırları;
}
finally
{
 Hata olsa da olmasa da çalıştırılacak kod satırları;
}
```

Programın hata verebilecek olan kod satırları **try catch** aralığında yazılır. **catch** ifadesinden sonra ise, hata oluşması durumunda çalıştırılacak olan kod satırları yazılmalıdır. **catch finally** aralığında ise, hata oluşup oluşmamasına bakılmaksızın çalışacak olan kod satırları yazılabilir.

En basit hali ile **try catch** kullanımına bir örnek yapalım.

```
private void button1_Click(object sender, EventArgs e)
{
 try
 {
  int sıfır = 0;
  int sonuc = 10 / sıfır;
 }
 catch
 {
  MessageBox.Show("Hata oluştu");
 }
}
```

Kodumuzda oluşan her hata durumunda "*Hata oluştu*" mesajı görüntülenmektedir. Kodumuzun çalışması kesintiye uğramasa da oluşan her hata durumunun engellenmesi pek de sağlıklı bir yöntem değildir. Ayrıca birden fazla hatanın oluşacağı durumlarda, bu şekildeki bir hata kontrolü tam anlamı ile açıklayıcı bilgi vermekten uzaktır.

Exception Nesnesi

Program içerisinde oluşan hatalar **Exception** nesnesinden türetilmektedir. **Exception** nesnesi **System** sınıfının altında yer almaktadır. Oluşan hataların niteliğini ayırt edebilmek için **Exception** nesnesi içerisinde saklanan hatanın hangi sınıfa ait olduğu değerlendirilerek işlem yapılabilir.

Bir örnek yaparak birden fazla hata durumu ortaya çıkartalım.

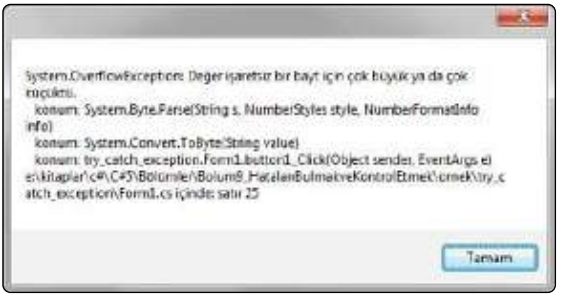
```
private void button1_Click(object sender, EventArgs e)
{
 try
 {
  byte notu;
```

```
  notu = Convert.ToByte(Interaction.InputBox("Notu Girin", "Not Girişi"));
  if (notu < 45)
  {
   MessageBox.Show("Kaldınız");
  }
  else if (notu >= 45 & notu <= 100)
  {
   MessageBox.Show("Geçtiniz");
  }
 }
 catch
 {
  MessageBox.Show("Yanlış Not Girişi Yaptınız");
 }
}
```

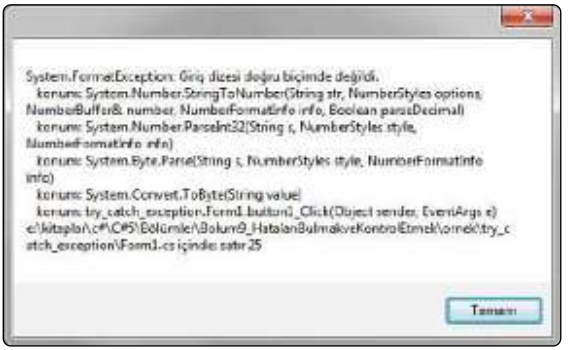
Kodumuzu çalıştırıp **notu** değişkenine **byte** değişken tipinin sınırları dışında bir değer verdiğimizde ya da **string** bir değer girdiğimizde her iki durumda da "*Yanlış Not Girişi Yaptınız*" mesajı görüntülenecektir. Bu durumu engellemek için **Exception** nesnesinden yararlanacağız.

```
private void button1_Click(object sender, EventArgs e)
{
 try
 {
  byte notu;
  notu = Convert.ToByte(Interaction.InputBox("Notu Girin", "Not Girişi"));
  if (notu < 45)
  {
   MessageBox.Show("Kaldınız");
  }
  else if (notu >= 45 & notu <= 100)
  {
   MessageBox.Show("Geçtiniz");
  }
 }
 catch (Exception ex)
 {
  MessageBox.Show(ex.ToString());
 }
}
```

Kodumuzu çalıştırıp `notu` değişkenine `byte` değişken tipinin sınırları dışında bir değer verdiğimizde, oluşan `OverflowException` (*taşma hatası*) `catch` ile yakalanarak `Exception` nesnesi ile `ex` değişkenine aktarılmakta ve `ToString` ile `string` türüne dönüştürülerek bir mesaj penceresi içerisinde görüntülenmektedir.



`String` bir değer girdiğimizde ise oluşan `FormatException` (*geçersiz format hatası*) alınarak bir mesaj penceresi içerisinde görüntülenmektedir.



`Exception` nesnesinin özellikleri kullanılarak, oluşan hata ile ilgili detaylı bilgiler alınabilir. Bu özellikler şunlardır:

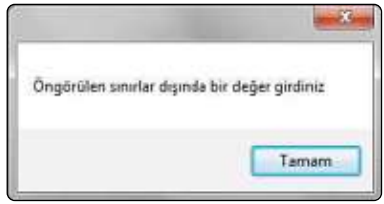
- `Message`: Oluşan hata ile ilgili açıklamaları içerir.
- `Source`: Hatanın oluştuğu nesne ya da sınıf hakkında bilgi içerir.
- `TargetSite`: Hatayı oluşturan metot hakkında bilgi içerir.
- `HelpLink`: Oluşan hata ile ilgili yardım dosyası ataması yapmak ya da atanmış yardım dosyasını görüntülemek için kullanılır.
- `StackTrace`: Hataya neden olan çağrıları içeren **stack** (*yığın*) bilgisini içerir.
- `InnerException`: Oluşan iç hataları yakalamak amacıyla kullanılır. Eğer hiçbir iç hata oluşmazsa bu özellik `null` değer içerir.

Yukarıdaki örneğimizdeki **`MessageBox.Show(ex.ToString());`** ifadesi yerine **`MessageBox.Show(ex.Source);`** yazıp çalıştırdığımızda, herhangi bir hata durumunda **`mscorlib`** mesajını görüntüleyecektir. Çünkü oluşan hatalar **`mscorlib`** sınıfına ait hatalardır. **`mscorlib`** sınıfı, **`System`** isim alanının içerisinde bulunduğu sınıftır.



Oluşan bu farklı hata türlerini kullanarak daha anlaşılır mesajlar görüntületmek mümkündür.

```
private void button1_Click(object sender, EventArgs e)
{
 try
 {
  byte notu;
  notu = Convert.ToByte(Interaction.InputBox("Notu Girin", "Not Girişi"));
  if (notu < 45)
  {
   MessageBox.Show("Kaldınız");
  }
  else if (notu >= 45 & notu <= 100)
  {
   MessageBox.Show("Geçtiniz");
  }
 }
 catch (OverflowException)
 {
  MessageBox.Show("Öngörülen sınırlar dışında bir değer girdiniz");
 }
 catch (FormatException)
 {
  MessageBox.Show("Rakam girmelisiniz");
 }
}
```



Örneğimizi çalıştırıp **byte** değişkeninin sınırları dışında bir rakam girdiğimizde **OverflowExcetion** oluşacak ve bizim belirlediğimiz hata mesajı görüntülenecektir. Görüldüğü gibi oluşan hatalara sabit ifadeler şeklinde .Net içerisinden erişmek mümkündür. Bu sabitleri kullanarak oluşan hata mesajının türü belirlenmektedir.

try catch yapısı içerisinde birden fazla **Exception** kullanmak mümkündür.

```
static void Main(string[] args)
{
 int deger, sonuc,bolen;
 try
 {
  Console.Write("Sayıyı Giriniz:");
  deger = Convert.ToInt32(Console.ReadLine());
  Console.Write("Bölen Sayıyı Giriniz:");
  bolen = Convert.ToInt32(Console.ReadLine());
  sonuc = deger / bolen;
 }
 // Sıfıra Bölme Hatası
 catch (DivideByZeroException)
 {
  Console.WriteLine("Sıfıra Bölme Hatası");
  sonuc = int.MaxValue;
 }
 // Arithmetik Hata
 catch (ArithmeticException)
 {
  Console.WriteLine("Aritmetik Hata");
  sonuc = int.MaxValue;
 }
 // Tanımlanmamış Hata
 catch
 {
  Console.WriteLine("Tanımlanmamış Hata");
  sonuc = int.MaxValue;
 }
 Console.WriteLine("Sonuç = {0}", sonuc);
 Console.ReadKey();
}
```



Örneğimizi çalıştırıp bölen sayıyı `0` olarak girdiğimizde **`DivideByZeroException`** hatası gerçekleşerek, konsol ekranında "*Sıfıra Bölme Hatası*" mesajı görüntülenecektir.



Örneğimizi tekrar çalıştırıp bu defa da **`integer`** değişken sınırını aşan bir değer girelim. **`ArithmeticException`** hatası gerçekleşerek konsol ekranında "*Aritmetik Hata*" mesajı görüntülenecektir.

NOT **`Exception`** nesnesi altında bulunan tüm hataların listesine *http://msdn.microsoft.com/en-us/library/system.systemexception.aspx* adresinden ulaşabilirsiniz.

Finally Deyimi

Hata oluşsa da oluşmasa da çalışacak olan kod satırlarını **`finally`** deyiminden sonra yazarız. Örneğin; aşağıdaki konsol uygulamasında program sayıları ekrana yazarken bir tuşa basıldığında, sayma işlemi iptal edilmektedir. İşlem iptal edildiğinde ise; **`finally`** bloğu içerisinde yazdığımız **`Console.ReadLine()`** metodu devreye girerek, programın çalışmasını bir tuşa basılana kadar bekletmektedir.

```
static void Main(string[] args)
{
 int i;
 try
 {
  for (i = 1; i <= 10000; i++)
  {
   if (Console.KeyAvailable == false)
   {
    Console.WriteLine(i);
```

```
    }
    else
    {
     break;
    }
    if (i == 10000) i = 1;
   }
  }
  finally
  {
   Console.ReadLine();
  }
 }
```

Throw Deyimi

Belirtilen hata sınıfından yeni bir **`Exception`** nesnesi oluşturarak bir hata durumu gerçekleştirmek amacı ile kullanılır. Hata oluşması durumunda görüntülenecek olan hata mesajı da belirlenebilir.

Örnek olarak bir **`OverflowException`** oluşturalım.

```
private void button1_Click(object sender, EventArgs e)
{
 byte notu;
 notu = Convert.ToByte(Interaction.InputBox("Notu Girin", "Not Girişi"));
 if (notu < 45)
 {
  MessageBox.Show("Kaldınız");
 }
 else if (notu >= 45 & notu <= 100)
 {
  MessageBox.Show("Geçtiniz");
 }
 else if (notu > 100)
 {
  try
  {
    throw new OverflowException("100'den büyük bir değer girdiniz");
  }
```

```
  catch (Exception ex)
  {
   MessageBox.Show(ex.Message);
  }
 }
}
```

Notu değişkenine 100'den büyük bir değer girdiğimizde **throw** nesnesi ile **OverflowException** türünden yeni bir hata nesnesi oluşturduk. Hata mesajı olarak da "*100'den büyük bir değer girdiniz*" metnini belirledik. Oluşturduğumuz hatayı **catch** ile yakalayarak bir mesaj penceresi içerisinde görüntüledik.



Program içerisinde kullandığımız **try catch** blokları içerisinde de **throw** nesnesi kullanabiliriz. Örnek olarak daha önce yaptığımız not giriş örneğindeki hata yakalama kodlarımızın arasına **throw** nesnesini yerleştirelim.

```
private void button1_Click(object sender, EventArgs e)
{
   try
   {
    byte notu;
    notu = Convert.ToByte(Interaction.InputBox("Notu Girin", "Not Girişi"));
    if (notu < 45)
    {
     MessageBox.Show("Kaldınız");
    }
    else if (notu >= 45 & notu <= 100)
    {
     MessageBox.Show("Geçtiniz");
    }
    else if (notu > 100)
    {
   try
   {
    throw new OverflowException("100'den büyük bir değer girdiniz");
   }
```

```
    catch (Exception ex)
    {
       MessageBox.Show(ex.Message);
    }
     }
    }
    catch (OverflowException)
    {
     MessageBox.Show("Öngörülen sınırlar dışında bir değer girdiniz");
    }
    catch (FormatException)
    {
     MessageBox.Show("Rakam girmelisiniz");
    }
}
```

Kodumuzu çalıştırdığımızda artık hemen hemen tüm hata durumlarını kontrol altına aldığımızı görürüz. Hem **`string`** değer girişi, hem değişken sınırı dışında değer girişi hem de `100` ile değişken sınırı arasında bir değer girişi durumunda oluşan hataları kontrol altına aldık.

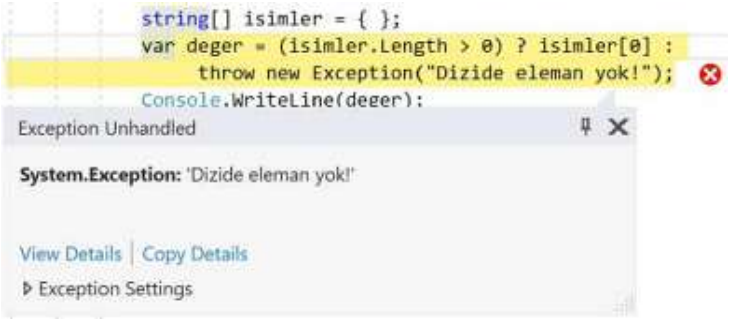
Throw Expressions

C# 7 ile birlikte kullanılmaya başlanan **Throw Expressions** özelliği ile birlikte `?` ***(Conditional)*** ve `??` ***(Null Coalescing)*** operatörlerini kullanarak kendi exception'larımızı gerçekleştirebiliyoruz.

```
string[] isimler = { "mehmet", "volkan", "bünyamin", "ömer" };
var deger = (isimler.Length > 0) ? isimler[0] :
     throw new Exception("Dizide eleman yok!");
Console.WriteLine(deger);
Console.ReadLine();
```

Örnek uygulamamızda eğer isimler dizisi içerisinde eleman varsa isimler dizisinin ilk elemanı olan "**`mehmet`**" değeri görüntülenecektir. Kodumuzda görüldüğü gibi artık throw ifadelerinde `?` ve `??` operatörlerini kullanarak daha gelişmiş exception'lar tanımlayabiliyoruz. Örneğin `?` operatörünü kullanırken `?` karakterinden sonra kullanılan değişken türü ile, : karakterinden sonra değişken türünün aynı olması gerekirken kodumuzda olduğu gibi : karakterinden sonra yeni bir exception tanımlayabiliyoruz.

Kodumuzda tanımladığımız dizinin içerisini boşaltarak örnek hata durumunu oluşturalım.



Görüldüğü gibi `throw` ile oluşturulan `exception` içerisinde belirlediğimiz hata mesajı görüntülenmektedir.

Exception.Data Özelliği

Oluşan hata durumlarında, kullanıcı tarafından tanımlanan ek bilgilerin anahtar ve değer şeklinde bir koleksiyon içerisinde saklanması ve okunması amacıyla kullanılan özelliktir. `Exception.Data` özelliği içerisinde aşağıdaki bilgiler saklanabilir.

- **stringInfo:** Oluşan hata durumu ile ilgili bilgi vermek amacıyla kullanılır.
- **IntInfo:** Oluşan hata durumu ile ilgili sayısal değer bilgisi (satır numarası, hata adedi vs.) saklamak amacıyla kullanılır.
- **DateTimeInfo:** Tarih ve zaman bilgisi içerir.
- **ExtraInfo:** Oluşan hata durumu ile ilgili ayrıntılı bilgi vermek amacıyla kullanılır.
- **MoreExtraInfo:** Oluşan hata durumu ile ilgili daha fazla bilgi vermek amacıyla kullanılır.

Örnek olarak oluşan hata durumunda hatanın oluşma zamanı bilgisini saklayarak, buna erişen bir uygulama yapalım. Daha önce yapmış olduğumuz not giriş örneğini aşağıdaki gibi değiştiriyoruz.

```
try
{
    Console.WriteLine("Notu Giriniz");
    byte notu = Convert.ToByte(Console.ReadLine());
```

```
    if (notu < 45)
        Console.WriteLine("Kaldınız");
    else if (notu >= 45 & notu <= 100)
        Console.WriteLine("Geçtiniz");
    else if (notu > 100 & notu <= 255)
        throw new OverflowException();
}
catch (Exception ex)
{
    if (ex is OverflowException)
    {
        ex.Data["stringInfo"] = "100'den büyük bir değer girdiniz";
        ex.Data["Time"] = DateTime.Now;
    }
    foreach (DictionaryEntry bilgi in ex.Data)
        Console.WriteLine(bilgi.Value.ToString());
}
```

Kodumuzda oluşan hata yakalanarak **ex** değişkeni içerisinde saklanmaktadır.

```
if (ex is OverflowException)
```

... satırında oluşan hata eğer **OverflowException** hatası ise;

```
ex.Data["stringInfo"] = "100'den büyük bir değer girdiniz";
```

... satırında oluşan hata durumu ile ilgili açıklama **stringInfo** özelliğine;

```
ex.Data["Time"] = DateTime.Now;
```

... satırında ise hatanın oluştuğu zaman bilgisi **Time** özelliğine aktarılmaktadır.

```
foreach (DictionaryEntry pair in ex.Data)
     Console.WriteLine(bilgi.Value.ToString());
```

Satırlarında ise **ex.Data** özelliği içerisinde saklanan anahtar ve değer bilgilerinden **bilgi.Value** şeklinde değerleri elde ederken, anahtarları ise **bilgi.Key** şeklinde elde edebiliriz.

Exception Filters

Yukarıdaki örnek uygulamamızda;

```
if (ex is OverflowException)
```

satırında **`if`** yapısını kullanarak oluşan hata eğer **OverflowException** hatası ise çeşitli işlemler yaptırmıştık.

C# 6 ile birlikte kullanılmaya başlanan **`exception filters`** ile birlikte **`if`** yapısını

```
catch (Exception ex) if (ex is OverflowException)
```

kod satırında olduğu gibi, **`catch`** satırında kullanarak yakalanan hataları filtreleyebilmekteyiz.

Buna göre kodumuzun yeni hali aşağıdaki gibi olacaktır:

```
try
{
    Console.WriteLine("Notu Giriniz");
    byte notu = Convert.ToByte(Console.ReadLine());
    if (notu < 45)
        Console.WriteLine("Kaldınız");
    else if (notu >= 45 & notu <= 100)
        Console.WriteLine("Geçtiniz");
    else if (notu > 100 & notu <= 255)
        throw new OverflowException();
}
catch (Exception ex) if (ex is OverflowException)
{
    ex.Data["stringInfo"] = "100'den büyük bir değer girdiniz";
    ex.Data["Time"] = DateTime.Now;

    foreach (DictionaryEntry bilgi in ex.Data)
        Console.WriteLine(bilgi.Value.ToString());
}
```

Kullanıcı Tanımlı Exception Nesnesi Oluşturmak

Application.Exception sınıfını kullanarak kendi **Exception** nesnelerimizi oluşturabiliriz.

Örnek olarak daha önce yaptığımız not giriş örneğindeki **100**'den büyük değer girişi durumunda **overflow** (*taşma*) hatası meydana getirecek bir **class** (*sınıf*) tasarlayalım. Sınıflarla ilgili ayrıntılı bilgi ayrı bir bölüm içerisinde anlatılacağından dolayı, burada mümkün oldukça basit bir sınıf tasarladık.

Yeni bir proje oluşturarak proje içerisinde **Project** menüsünden **Add Class** seçeneğini kullanarak **NotKontrol** adlı yeni bir sınıf ekledim.

NotKontrol sınıfının kodu aşağıdaki gibidir:

```
using System;
using System.Text;
using System.Windows.Forms;

namespace KullanıcıTanımlıException
{
   public class NotKontrol : ApplicationException
   {
     public NotKontrol(string HataMesajı): base(HataMesajı)
     {
     }
   }
}
```

NotKontrol:ApplicationException ifadesi, **NotKontrol** sınıfının **ApplicationException** sınıfından türetildiğini belirtmektedir. Dolayısı ile o sınıfın özellik ve metotları kullanılabilir.

Form'umuzun kodu ise aşağıdaki gibidir:

```
private void button1_Click(object sender, EventArgs e)
{
 try
 {
```

```
  byte notu;
  notu = Convert.ToByte(Interaction.InputBox("Notu Girin", "Not Girişi"));
  if (notu < 45)
  {
   MessageBox.Show("Kaldınız");
  }
  else if (notu >= 45 & notu <= 100)
  {
   MessageBox.Show("Geçtiniz");
  }
   else if (notu > 100)
   {
    throw new NotKontrol("100 den büyük değer girdiniz");
   }
  }
 catch (Exception ex)
 {
  MessageBox.Show(ex.Message);
 }
}
```

notu değişkenine **100**'den büyük bir değer girilmesi durumunda **NotKontrol** sınıfını çağırarak yeni bir **Exception** oluşturduk ve **HataMesajı** değişkenine de "*100'den büyük değer girdiniz*" mesajını gönderdik. Program çalıştırılıp **100**'den büyük bir değer girildiğinde "*100'den büyük değer girdiniz*" mesajı görüntülenecektir.

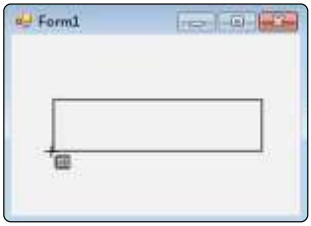
FORMLAR VE KONTROLLER

10

Şimdiye kadar yaptığımız uygulamalarda kısmen de olsa Form'ların ve kontrollerin bazı özelliklerini ve olaylarını kullandık. Bu bölümde ayrıntılı olarak inceleyeceğiz.

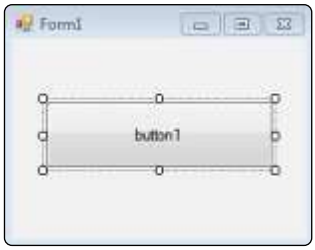
Form üzerine bir kontrolü eklemek için 2 yöntem kullanılabilir;

Birinci yöntemde fare ile kontrole çift tıklandığında form üzerine otomatik olarak eklenir.



İkinci yöntemde ise kontrol üzerine fare ile bir defa tıklanarak, form üzerine istenilen koordinata, istenilen boyutlarda eklenecek kontrolümüz çizilir.

Farenin tuşu bırakıldığında kontrol form üzerine eklenmiş olur.

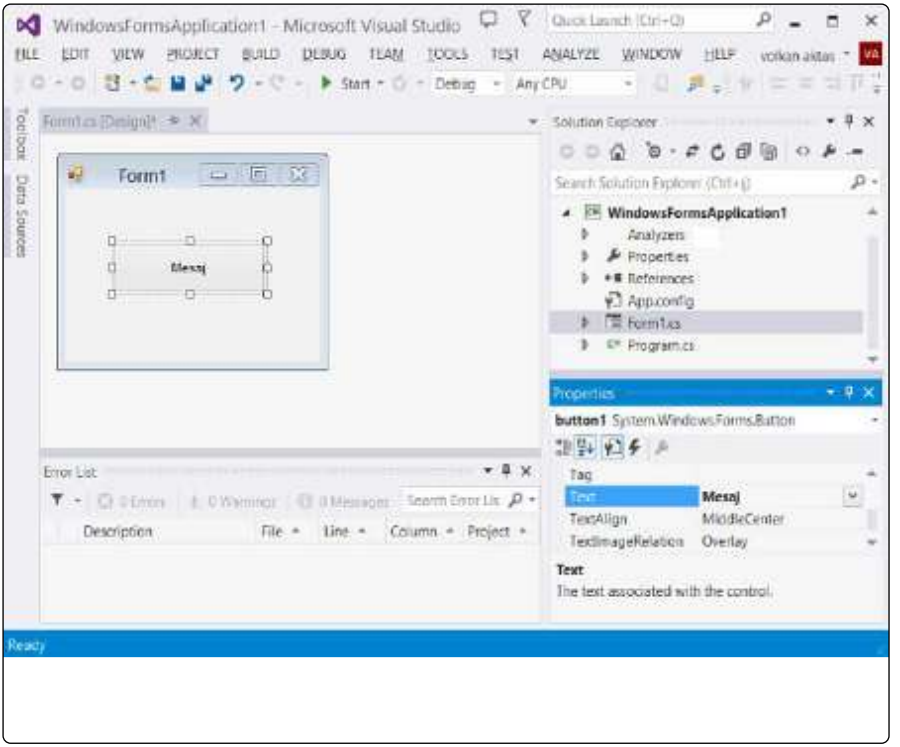


Kullanacağımız her bir kontrolün **Properties** (*Özellikler*) adı verilen kendilerine ait nitelikleri vardır. Bu özellikleri kullanarak bir kontrolün rengi, boyutu, ismi gibi niteliklerini değiştirebiliriz. Bu özelliklerinin bir kısmı bütün kontrollerde ortak olarak kullanılmaktadır.

Bu özelliklerin başlıcaları şunlardır:

- **`Name`**: Nesnenin kodlama yapısı içerisinde kullanılacak olan ismidir.
- **`Text`**: Nesnenin kullanıcı tarafında görüntülenecek olan ismidir.
- **`BackColor`**: Zemin Rengini belirler.
- **`Enabled`**: Nesnenin aktif veya pasif olacağını belirler.
- **`Font`**: Yazı tipi özelliklerini belirler.
- **`Visible`**: Nesneyi görünür ya da görünmez yapar.
- **`Location`**: Nesnenin konumunu belirler.
- **`Size`**: Nesnenin boyutunu belirler.

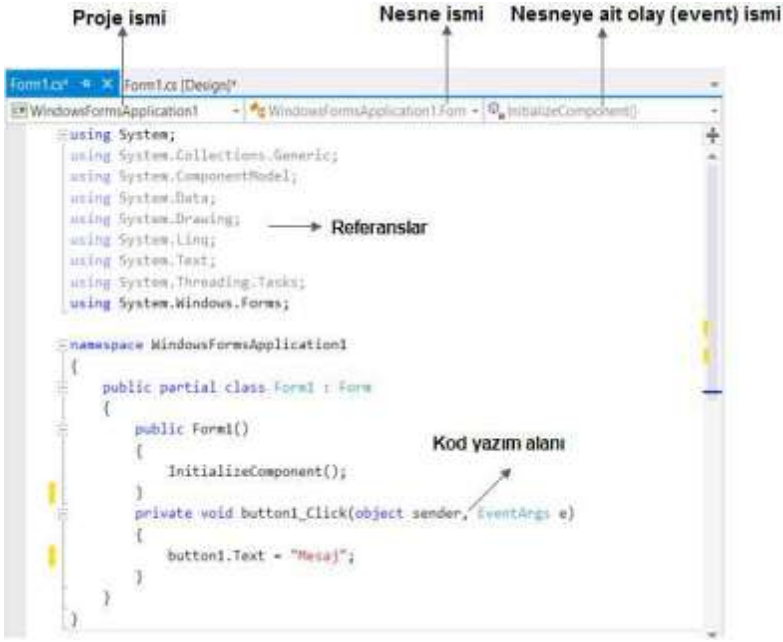
Bir kontrolün özelliğini **Properties Window** vasıtası ile herhangi bir kod yazmadan kolayca değiştirebiliriz. Örneğin; Form üzerine eklediğimiz butonumuz seçiliyken, **Properties** penceresinden **Text** özelliğini seçerek **Mesaj** olarak değiştirelim.



Aynı özellik kod yazarak da değiştirilebilir. Kod görünümüne geçiş yaparak butonun **Text** özelliğini değiştirelim.

Kod görünümüne geçiş yapmak için **Solution Explorer** penceresindeki <> butonu tıklanabilir, View menüsünden **Code** seçeneği tıklanabilir ya da **F7** kısayol tuşu kullanılabilir.

Kod yazma penceresinin üst tarafında sol tarafta nesneler, sağ tarafta ise bu nesnelere ait olaylar listelenir. Kullanıcı ilgili olayı seçtiği zaman, olayın başlangıç kodu otomatik olarak yazılmaktadır. Elbette kullanıcı isterse bunları elle de yazabilir.



Butonun **Text** özelliğini değiştirmek için…

```
Button1.Text= "Mesaj"
```

… biçiminde kodumuzu yazdık. Uygulamamızı çalıştırdığımızda, buton üzerinde görüntülenecek yazı değiştirilecektir.

Kontrollerin Yerleşimi ve Düzeni

Form üzerine birden fazla kontrol eklendiğinde, .NET bu kontrolleri daha düzenli bir şekilde yerleştirebilmek için gerekli desteği sağlar.

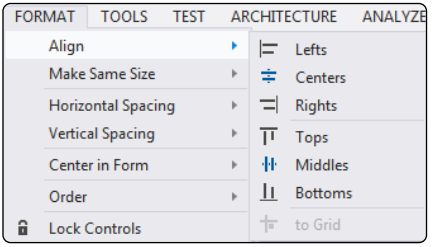


Örneğin; bir Form üzerine birden fazla buton kontrolünü ekleyerek, genişlik ve yüksekliklerini farklı şekilde belirleyip dağınık şekilde yerleştirelim.

Öncelikle butonları hizalama işlemlerini inceleyelim. Fare ile bir butonu basılı tutup bir başka butonun hizasına getirdiğimizde buton ile aynı hizaya geldiğinde klavuz çizgileri görüntülenir. Bu şekilde istediğimiz kontrole göre hizalama işlemi yapabiliriz.



Seçili olan kontrolleri birbirine göre hizalamak için ise **Format** menüsünü kullanabiliriz.



Lefts seçeneğini kullanarak seçili olan kontrolleri sola hizalayabiliriz.

Centers seçeneğini kullanarak seçili olan kontrolleri ortalayabiliriz.

Rights seçeneğini kullanarak seçili olan kontrolleri sağa hizalayabiliriz.



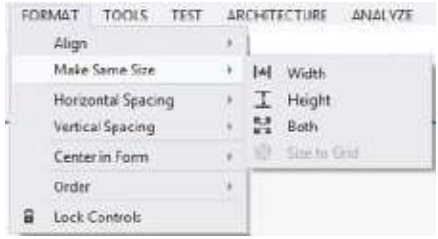
Tops seçeneğini kullanarak, seçili olan kontrolleri yatay olarak üst kenara göre hizalayabiliriz.

Middles seçeneğini kullanarak, seçili olan kontrolleri yatay olarak üst kenara göre ortalayabiliriz.

Bottoms seçeneğini kullanarak, seçili olan kontrolleri yatay olarak alt kenara göre hizalayabiliriz.



Form üzerindeki kontrollerin boyutlarını eşitlemek için **Format** menüsü altındaki **Make Same Size** başlığı altındaki seçenekleri kullanabiliriz.



Bu başlık altındaki **Width** seçeneği ile seçili olan butonların genişliklerini eşitleyebiliriz.

Height seçeneği ile seçili olan butonların yüksekliklerini eşitleyebiliriz.

Both seçeneği ile seçili olan butonların hem genişlik hem de yüksekliklerini eşitleyebiliriz.



Formlar

Form nesneleri ve **Form** üzerinde kullandığımız diğer kontroller, **`System.Windows.Forms`** sınıfından türetilirler. **Windows Form** uygulamalarımızda kontrollerimizi form üzerine yerleştirerek kullanırız. **Form** dosyaları disk üzerinde **cs** uzantısı ile kaydedilirler. Bir ya da birden fazla Form kullanmamız mümkündür.

Standart bir **Form**'un kod yapısı şu şekildedir:

```
using System;
using System.Windows.Forms;
namespace Form_Olayları
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
  }
  private void Form1_Load(object sender, EventArgs e)
  {
   this.Text = "Yeni Form";
  }
 }
}
```

`Form1`'in **`Load`** olayı ile **Form** yüklenirken, **`Form`** sınıfının **`Load`** olayı içerisinde yazdığımız kodlar çalıştırılmakta ve **`Form1`**'in başlığı "*Yeni Form*" olarak değiştirilmektedir. **Form**'un **`Text`** özelliği **Form**'un başlığını değiştirirken, **`name`** özelliği **Form**'un kod içerisinde kullanılacak ismini belirler.

Formlar Arası Geçiş

Projemize yeni bir form eklemek için **Solution Explorer**'a farenin sağ tuşu ile tıklanarak **Add / Windows Form** ya da **Project** menüsünden **Add / Windows Form** seçeneği ile yeni bir form eklenebilir.

Formlar arası geçişlerde **`Show`**, **`Hide`**, **`Visible`** ve **`ShowDialog`** metodları kullanılabilir.

Örnek olarak projemizde 2 adet Form olsun. Her ikisine de birer adet **Button** ekleyerek **Form**'lar arası geçiş sağlayalım.

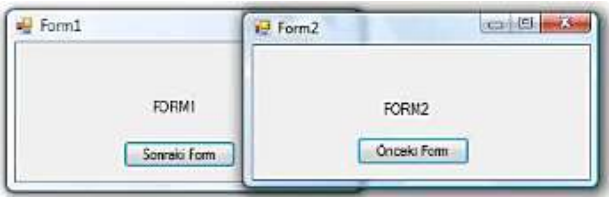
Öncelikle **Show** metodunu kullanalım.

Form1'den **Form2**'ye geçiş için…

```
Form2 frm2 = new Form2();
frm2.Show();
```

… şeklinde geçiş yapabiliriz.

Yaptığımız örnekte **Form2**'yi görüntülediğimizde **Form1** ekranda kalarak, **Form2** görüntülenecektir.



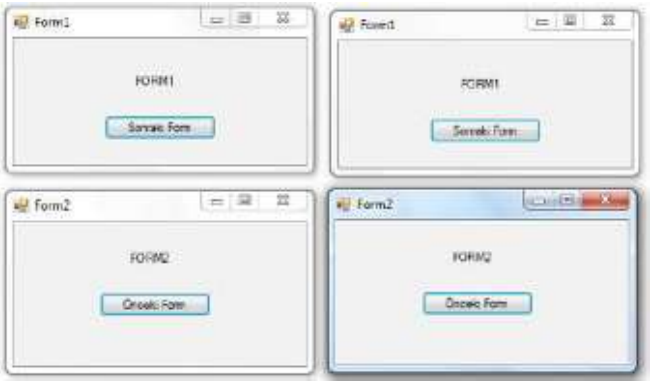
Ancak **Form**'ları görüntülerken dikkat edilirse, aynı Form birden fazla açılabilmektedir. Kodumuzu aşağıdaki gibi düzenleyip çalıştıralım.

Form1'in kodu:

```
private void btnSonrakiForm_Click(object sender, EventArgs e)
{
 Form2 frm2 = new Form2();
 frm2.Show();
}
```

Form2'nin kodu:

```
private void btnÖncekiForm_Click(object sender, EventArgs e)
{
 Form1 frm1 = new Form1();
 frm1.Show();
}
```



Görüldüğü gibi butonlara her tıkladığımızda, yeni bir Form açılmaktadır.

`Hide` metodunu kullanarak **`Form2`**'yi görüntülemeden önce **`Form1`**'i gizleyebiliriz. Ancak bu şekilde **`Form1`** hafızada kalmaya devam edip, sadece gizlenecektir.

```
private void btnSonrakiForm_Click(object sender, EventArgs e)
{
 this.Hide();
 Form2 frm2 = new Form2();
 frm2.Show();
}
```

`Show` ve **`Hide`** metodları yerine, **`Visible`** metodunu kullanarak da **Form**'u gösterip gizleyebiliriz.

`Form1`'in kodu:

```
private void btnSonrakiForm_Click(object sender, EventArgs e)
{
 this.Visible = false;
 Form2 frm2 = new Form2();
 frm2.Visible = true;
}
```

Form2'nin kodu:

```
private void btnÖncekiForm_Click(object sender, EventArgs e)
{
 this.Visible = false;
 Form1 frm1 = new Form1();
 frm1.Visible = true;
}
```

Close metodunu kullanarak **Form**'un tamamen kapatılmasını ve hafızadan silinmesini sağlayabiliriz. Ancak bu işlemi ana formda yapamayız. Ana **Form**'daki form kapatma olayını, C#.NET programı sonlandırma olayı gibi algılamaktadır. Dolayısı ile ana **Form**'da formu **Hide** metodu ile gizleyecek, diğer formlarda ise **Close** metodu ile kapatacağız.

Örneğimizi buna göre düzenleyecek olursak;

Form1'in kodu:

```
private void btnSonrakiForm_Click(object sender, EventArgs e)
{
 this.Hide();
 Form2 frm2 = new Form2();
 frm2.Show();
}
```

Form2'nin kodu:

```
private void btnÖncekiForm_Click(object sender, EventArgs e)
{
 this.Close();
 Form1 frm1 = new Form1();
 frm1.Show();
}
```

Form'u görüntülemede kullanılan bir diğer metod ise, **ShowDialog** metodudur. **Show** metodunda Form açık iken, diğer **Form**'lar aktifleştirilebilir ve geçiş yapılabilir. **ShowDialog** metodu ile görüntülenen **Form**'lar kapatılmadığı sürece diğer formlara geçiş yapılamaz ve aktifleştirilemez.

Yani kodumuzu…

```
private void btnSonrakiForm_Click(object sender, EventArgs e)
{
 Form2 frm2 = new Form2();
 frm2. ShowDialog();
}
```

… şeklinde yazdığımızda eğer Form1 açık ise Form2 kapatılmadığı sürece Form1'e geçiş yapamayız.

`Show` metodu geriye bir değer döndürmezken, **`ShowDialog`** metodu ise, geriye **`DialogResult`** türünden bir değer döndürür. Bu şekilde **Form**'un nasıl kapatıldığı hakkında bilgi sahibi olabiliriz.

`ShowDialog` ile ilgili bir şifre giriş **Form**'u örneği yapalım. Yeni bir **Windows Form** projesi açıp içerisine **`Form1`** dışında yeni bir **Form** ekleyerek üzerine **Tamam** ve **İptal** butonlarını ekliyoruz. Ayrıca bilgi görüntüleme amaçlı olarak kullanacağımız 1 adet de **Label** nesnesi ekleyelim. Eklediğimiz **Form**'un ismini **`frmDialog`** olarak verelim.

Dialog Form'umuzun kodu şu şekilde olmalı:

```
private void btn_tamam_Click(object sender, EventArgs e)
{
 this.DialogResult = System.Windows.Forms.DialogResult.OK;
 this.Close();
}
private void btn_iptal_Click(object sender, EventArgs e)
{
 this.DialogResult = System.Windows.Forms.DialogResult.Cancel;
 this.Close();
}
```

Ayrıca **`Form1`** içerisinden **Dialog Form**'u içerisindeki **Label** nesnesine erişim için **frmDialog.Designer.cs** isimli Form'un tasarım kodlarını içeren dosya içerisinde…

```
private System.Windows.Forms.Label label1;
```

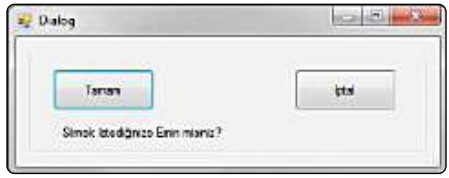
… satırını…

```
public System.Windows.Forms.Label label1;
```

… şeklinde düzenleyerek **label1** nesnesinin erişim tipini, **private** yerine **public** olarak değiştiriyoruz.

Form1'in kodu ise şu şekildedir:

```
private void button1_Click(object sender, EventArgs e)
{
 frmDialog diyalog_formu = new frmDialog();
 diyalog_formu.label1.Text = "Silmek İstediğinize Emin misiniz?";
 diyalog_formu.ShowDialog();
 if (diyalog_formu.DialogResult == System.Windows.Forms.DialogResult.OK)
 {
  MessageBox.Show("Dosyalar Silindi");
 }
}
```



Dialog Form'unda basılan butonların bilgisi formun **DialogResult** bilgisine aktarılmakta ve **Form1** içerisinde **Dialog Form**'unun **DialogResult** bilgisi kontrol edilerek, eğer dönen değer **OK** butonu ise, ekrana "*Dosyalar Silindi*" şeklinde bir mesaj çıkmasını sağlıyoruz.

ShowDialog metodunu kullandıktan sonra, **Dispose** metodunu kullanmak yerinde olacaktır. **Dispose** metodu; nesnelerin kullanımının ardından hafıza kaynaklarını temizlemek amacı ile kullanılır. **Show** metodu kullanıldıktan sonra Form kapatıldığında, otomatik olarak **Dispose** işlemi gerçekleştirildiğinden dolayı kullanmaya gerek yoktur. Ancak **ShowDialog** metodu metodu ile görüntülenen **Form Dispose** işlemi gerçekleştirilene kadar gizlenmiş olarak çalışmaya devam eder. Bu nedenle **Dispose** metodunu kullanmak gerekmektedir.

Örneğimizde **Dialog Form**'u için yapılacak **`Dispose`** işlemini, **`Form1`** içerisinde yapmak daha uygun olacaktır. Çünkü aksi takdirde hafıza içeriği temizleneceğinden dolayı, **Dialog Form**'unun geriye döndüreceği değer içeriği de temizleneceğinden, geriye boş değer dönecektir.

```
private void button1_Click(object sender, EventArgs e)
{
 frmDialog diyalog_formu = new frmDialog();
 diyalog_formu.label1.Text = "Silmek İstediğinize Emin misiniz?";
 diyalog_formu.ShowDialog();
 if (diyalog_formu.DialogResult == System.Windows.Forms.DialogResult.OK)
 {
  MessageBox.Show("Dosyalar Silindi");
 }
 diyalog_formu.Dispose();
}
```

Projemizde yüklü olan birden fazla formdan hangisinin ilk sırada yer alacağını belirlemek için **Program.cs** dosyası açılarak içerisindeki…

```
Application.Run(new Form1());
```

… satırında bulunan **`Form1`** yerine istenilen **Form** ismi yazılarak, başlangıç formu değiştirilebilir.

Görüntülenen **Form**'lardan hangisinin en önde, hangisinin en arkada olacağını belirlememiz mümkündür.

`BringToFront` metodu ile görüntülenen formu, diğer formlara göre en önde görüntületebiliriz.

`SendToBack` metodu ile görüntülenen formu, diğer formlara göre en arkada görüntületebiliriz.

```
private void btnOn_Click(object sender, EventArgs e)
{
 Form2 frm2 = new Form2();
 frm2.Show();
 frm2.BringToFront();
}
```

```
private void btnArka_Click(object sender, EventArgs e)
{
 Form2 frm2 = new Form2();
 frm2.Show();
 frm2.SendToBack();
}
```

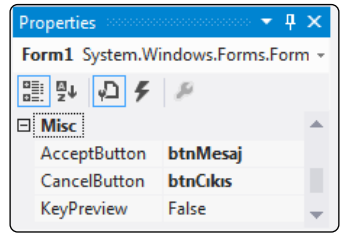
btnOn butonuna tıkladığımızda, **Form2** görüntülenerek en önde yer alacaktır. **btnArka** butonuna tıkladığımızda ise **Form2** görüntülenerek en arkada yer alacaktır.

Form Nesnesinin Özellikleri

- **Icon:** Form'un simgesini belirler. ico uzantılı bir simge dosyası seçilmelidir.
- **BackColor:** Formun zemin rengini değiştirmek için kullanılır. **Custom**, **Web** ve **System** adıyla 3 adet sekmesi vardır. **Custom** sekmesinde mevcut renk tablosundan istenilen renk fare ile tıklanarak seçilebilir. **Web** ve **System** sekmelerinde ise, mevcut renk sabiti isimlerinden herhangi birisi seçilebilir.
- **AcceptButton:** **Form** üzerinde **Enter** tuşuna basıldığında çalıştırılacak olan butonu belirler.
- **CancelButton:** **Form** üzerinde **ESC** tuşuna basıldığında çalıştırılacak olan butonu belirler.

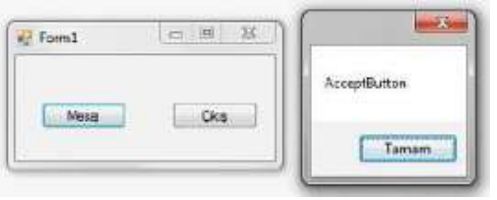
Örnek olarak bir **Form** üzerine iki adet **Button** ekleyelim. Butonlardan birisi ekranda mesaj penceresini görüntülesin (**btnMesaj**), diğeri ise programdan çıkışı sağlasın (**btnCıkıs**).

Form üzerinde **Enter** tuşuna basıldığında, **btnMesaj** butonunun çalışması için Form'un **AcceptButton** özelliğine **btnMesaj**, **ESC** tuşuna basıldığında **btnCıkıs** butonunun çalışması için **Form**'un **CancelButton** özelliğine **btnCıkıs** yazıyoruz.



Kodumuz aşağıdaki şekilde olacak:

```
private void btnCıkıs_Click(object sender, EventArgs e)
{
 Application.Exit();
}
private void btnMesaj_Click(object sender, EventArgs e)
{
 MessageBox.Show("AcceptButton");
}
```



Form Boyutu ile İlgili Özellikler

Form'un ekrana hangi boyutlarda geleceğini **`WindowState`** özelliği ile belirleyebiliriz. Bu özellik 3 değer alır. **`Normal`**, **`Minimized`**, **`Maximized`**.

Normal seçeneği, **Form**'u hangi boyutlarda ayarladı isek formun o boyutlarda ekrana gelmesini sağlar.

Minimized seçeneği, **Form**'u simge durumuna küçültülmüş olarak ekrana gelmesini sağlar.

Maximized seçeneği, **Form**'un tam ekran boyutunda ekranı kaplamış olarak ekrana gelmesini sağlar.

FormBorderStyle özelliği ile **Form**'un biçimi ile birlikte, boyutlarının değiştirilip değiştirilemeyeceğini de belirleyebiliriz. Bu özellik şu değerleri alır:

- **`None`**: **Form**, sağ üst köşesindeki tüm kontrolleri ve formun kenar çizgilerini siler.
- **`Fixed Single`**: **Form**'un boyutunun değiştirilmesini engeller. Bu seçenek aktif iken **`MaximizeBox`** özelliğini de **`false`** yaparak tam ekran butonunu pasif yapıp, formun ekranı kaplamasını da engelleyerek **Form**'un boyutlarının değiştirilmesini tamamen engelleyebiliriz. İsteğe bağlı olarak

`MinimizeBox=false` ile simge duruma küçültme düğmesini, **`ControlBox=false`** ile de kapatma düğmesini pasif yapabiliriz.

- **`Fixed3D`**: **Form**'a 3 boyutlu görünüm vererek aynı zamanda boyutlarının değiştirilmesini engeller.
- **`FixedDialog`**: **Form**'un boyutları değiştirilemez. Aynı zamanda **Form**'un simgesi de görünmez. Form simgesi, **Icon** özelliği ile değiştirilebilmektedir.
- **`Sizable`**: Varsayılan görüntüleme ayarıdır. **Form**'un boyutlarının değiştirilebilmesini, simgesinin görüntülenebilmesini sağlar.
- **`FixedToolWindow`**: **Form**'un boyutları değiştirilemez, simgesi görünmez, ekranı kaplama ve simge durumuna küçültme butonları görünmez. Yalnızca kapatma butonu aktiftir.
- **`SizableToolWindow`**: **`FixedToolWindow`** ile hemen hemen aynıdır. Farklı olarak **Form**'un boyutları değiştirilebilir.

Size başlığı altındaki **Width** ve **Height** özellikleri ile **Form**'un boyutları belirlenir. **Width** özelliği ile genişlik belirlenirken, **Height** özelliği ile yükseklik belirlenir.

Form'un boyutlarının hiçbir şekilde değiştirilmesini istemiyorsak, öncelikle **Form**'un **`FormBorderStyle`** özelliğini **`FixedSingle`** yaparak **Form**'un boyutunun fare ile değiştirilmesini engelleriz. Daha sonra **`MaximizeBox`** özelliğini **`false`** yaparak da tam ekran boyutuna getirilmesini engelleriz.

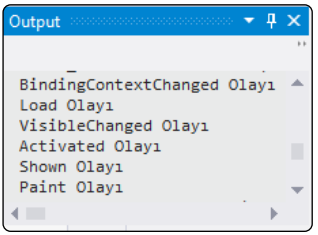
Form Olayları

Açılış esnasında gerçekleşen olayların sırasını test etmek için bir kod yazalım. **Output** ekranına **Debug** metodu ile yazdırma işlemi yapacağımız için **`System.Diagnostics`** sınıfını projemize dahil etmeliyiz.

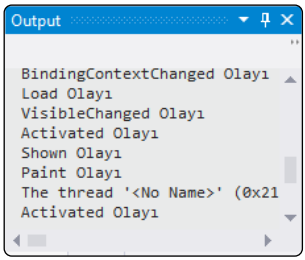
```
using System;
using System.Windows.Forms;
using System.Diagnostics;
namespace Form_Acilis
{
 public partial class Form1 : Form
 {
   public Form1()
   {
```

```
        InitializeComponent();
    }
    private void Form1_Activated(object sender, EventArgs e)
    {
        Debug.Print("Activated Olayı");
    }
    private void Form1_BindingContextChanged(object sender, EventArgs e)
    {
        Debug.Print("BindingContextChanged Olayı");
    }
    private void Form1_Load(object sender, EventArgs e)
    {
        Debug.Print("Load Olayı");
    }
    private void Form1_Paint(object sender, PaintEventArgs e)
    {
        Debug.Print("Paint Olayı");
    }
    private void Form1_Shown(object sender, EventArgs e)
    {
        Debug.Print("Shown Olayı");
    }
    private void Form1_VisibleChanged(object sender, EventArgs e)
    {
        Debug.Print("VisibleChanged Olayı");
    }
  }
}
```

Her olay gerçekleştiğinde **Output Window** içerisinde olayın adı yazılacak, dolayısı ile olayların hangi sırada gerçekleştiğini görebileceğiz.



Kodumuzu çalıştırıp **Output Window** içerisine baktığımızda olaylarımızın şu sırada gerçekleştiğini görüyoruz.



Form fare ile tıklanarak seçildiğinde ise, **`Activated`** olayı gerçekleşmektedir.

Form'un açılışı esnasında gerçekleşen bu olayları inceleyelim…

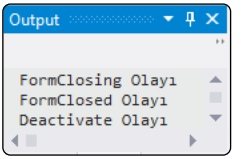
- **`BindingContextChanged`**: Bu olayla birlikte form görünür hale gelmeye başlar.
- **`Load`**: **Form**'un yüklenmesi olayıdır.
- **`Activated`**: **Form** aktifleştiğinde gerçekleşen olaydır.
- **`VisibleChanged`**: **Form** görüntülendiğinde gerçekleşen olaydır.
- **`Shown`**: **Form**'un görüntülenmesi bittiği zaman gerçekleşen olaydır.
- **`Paint`**: **Form** ve **Form** üzerindeki kontroller ekrana çizilip yerleştirildiği zaman gerçekleşen olaydır.

Kapanış esnasında gerçekleşen olayların sırasını test etmek için bir kod yazalım.

```
using System;
using System.Diagnostics;
using System.Windows.Forms;
namespace Form_Kapanis
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
  }
  private void Form1_FormClosing(object sender, FormClosingEventArgs e)
  {
   Debug.Print("FormClosing Olayı");
  }
  private void Form1_FormClosed(object sender, FormClosedEventArgs e)
  {
   Debug.Print("FormClosed Olayı");
```

```
    }
    private void Form1_Deactivate(object sender, EventArgs e)
    {
     Debug.Print("Deactivate Olayı");
    }
  }
}
```

Kodumuzu çalıştırıp **Output Window** içerisine baktığımızda, olaylarımızın şu sırada gerçekleştiğini görüyoruz.



Form'un kapanışı esnasında gerçekleşen bu olayları inceleyelim.

- **`FormClosing`**: Form kapatılırken gerçekleşen olaydır.
- **`FormClosed`**: **Form** kapatıldığında gerçekleşen olaydır.
- **`Deactivate`**: **Form** odağını kaybettiğinde ve artık aktif form olmadığında gerçekleşen olaydır.

Cancel niteliğini kullanarak **Form**'un kapatılmasını engelleyebilirsiniz. Kodumuzu **Form**'un kapanış (**`FormClosing`**) olayına yazalım.

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
 MessageBox.Show("Uygulamadan Çıkamazsınız!");
 e.Cancel = true;
}
```

Kodumuz **`Form_Closing`** olayına yazıldığı için **Form**'un kapatılması esnasında çalışmaktadır. Kod içerisinde geçen **`e.Cancel=true`** ifadesi ile çıkış işlemi iptal edilmektedir.

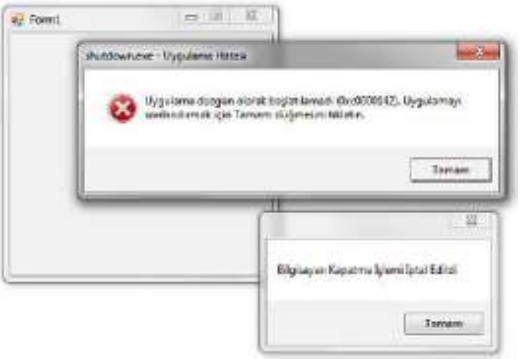
`Form_Closing` olayının bize sağladığı en büyük avantajlardan biri ise, **`CloseReason`** metodu ile kullanıcının kapatma işlemini hangi şekilde gerçekleştirdiğine dair bilgi alınabilmesidir. **`CloseReason`** metodu şu değerleri alabilir:

- **`None`:** Nedeni bilinmeyen kapatma işlemi.
- **`ApplicationExitCall`: `Application`** sınıfının çıkış metodu çağrıldığında gerçekleşen olaydır.
- **`FormOwnerClosing`: Owner Form** kapatıldığında gerçekleşen olaydır.
- **`MdiFormClosing`: MDI Ana Form**'u kapatılmaya başlanıldığında gerçekleşen olaydır.
- **`TaskManagerClosing`: Form**, görev yöneticisi tarafından sonlandırmak istenildiğinde gerçekleşen olaydır.
- **`UserClosing`:** Kullanıcı tarafından form kapatılmaya başlanıldığında örneğin **Alt+F4** tuşlarına basılarak, formun üst köşesindeki (**x**) işaretine basarak ya da formun kontrol menüsünden **Kapat** seçeneği seçildiğinde gerçekleşen olaydır.
- **`WindowsShutDown`:** İşletim sisteminin bilgisayar kapanmadan önce bütün uygulamaları kapatmaya başladığında gerçekleşen olaydır.

```
using System;
using System.Windows.Forms;
using System.Diagnostics;
namespace Form_CloseReason
{
  public partial class Form1 : Form
  {
   public Form1()
   {
    InitializeComponent();
   }
   private void Form1_FormClosing(object sender, FormClosingEventArgs e)
   {
    if (e.CloseReason == CloseReason.WindowsShutDown)
    {
     Process.Start("shutdown","-a");
     MessageBox.Show("Bilgisayarı Kapatma İşlemi İptal Edildi");
    }
  }
 }
}
```

Örneğimizi denemek için **exe** dosyamızı çalıştırıp, ardından bilgisayarı kapat-

ma işlemi gerçekleştirdim. Ancak bilgisayarı kapatma işlemi kodumuz tarafından engellendi.



Process.Start metodunu **-a** parametresi ile birlikte kullanarak bilgisayarın yeniden başlatılma işleminin yapılmasını sağladık.

KeyPress Olayı:

Form üzerinde bir tuşa basıldığında gerçekleşen olaydır.

Örnek olarak **Form** üzerinde basılan tuş sayısını görüntüleyen bir uygulama yapalım.

```
byte i;
private void Form1_KeyPress(object sender, KeyPressEventArgs e)
{
 i ++;
 if (e.KeyChar == 27) MessageBox.Show(i - 1 + " tuşa basıldı");
}
```

e.KeyChar ile basılan tuşun karakter kodu kontrol edilmektedir. Eğer basılan tuşun karakter kodu **27** ise, yani **ESC** tuşu ise, o ana kadar basılmış tuşların sayısı bir mesaj penceresi içerisinde gösterilmekte, **ESC** tuşuna basılmamış ise basılan tuşların sayısının saklandığı i değişkeninin değeri bir arttırılmaktadır.

Ancak **Keypress** olayında **Alt**, **AltGr**, **Ctrl** gibi kontrol tuşlarına basıldığını algılayamayız.

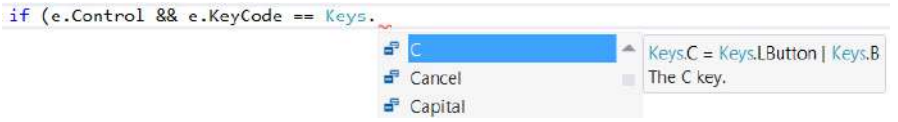
KeyDown Olayı:

Form üzerinde yön tuşları ve kontrol tuşları da dâhil olmak üzere, herhangi bir tuşa basıldığında gerçekleşen olaydır.

`e.KeyCode` metodu ile basılan tuşun kodu belirtilir. Burada istenilirse tuşun karakter kodu yazılır ya da aşağıda görüldüğü gibi **`Keys.TuşAdı`** şeklinde tuşun adı belirtilebilir.

Eğer kontrol tuşları kullanımı kontrol edilecekse **`e.Shift`**, **`e.Ctrl`** ya da **`e.Alt`** şeklinde kullanılabilir.

`Keys` ifadesinden sonra noktaya basıldığında kullanılabilecek tuşlar listelenir.



Kodumuz aşağıdaki gibi olacak:

```
private void Form1_KeyDown(object sender, KeyEventArgs e)
{
  if (e.Control && e.KeyCode == Keys.C)
  {
 DialogResult cikis = MessageBox.Show
   ("Ctrl+C tuşlarına basıldı. Çıkmak istediğinize emin misiniz",
   "Çıkış", MessageBoxButtons.OKCancel);
 if (cikis == DialogResult.OK ) Application.Exit();
  }
}
```

Örneğimizde **Ctrl+C** tuşlarına basıldığında bir mesaj penceresi yardımı ile **Form**'dan çıkış işlemi yapılmaktadır. **`Application.Exit`** metodu uygulamadan çıkış yapılmasını sağlar.

Keyup Olayı:

Basılı olan tuşun bırakılması sırasında gerçekleşen olaydır. **`Keydown`** olayında olduğu gibi kontrol tuşlarının da kontrolü yapılabilir.

Owned Formlar

C# içerisinde bir formu **`owner`** (*sahip*) olarak belirleyip ondan **`owned`** (*sahiplenilmiş*) **Form**'lar oluşturmak mümkündür. Bir **Owner Form** simge durumuna küçültüldüğünde owned form da simge durumuna küçülür, kapatıldığında o da kapatılır.

Owned Form'lar, **`AddOwnedForm`** metoduyla bir **Form**'a eklenir ve **`RemoveOwnedForm`** metoduyla çıkarılır.

```
Form ownedForm1 = new Form();
private void btnOwnedFormEkle_Click(object sender, EventArgs e)
{
 this.AddOwnedForm(ownedForm1);
 ownedForm1.Show();
}
private void btnOwnedFormSil_Click(object sender, EventArgs e)
{
 this.RemoveOwnedForm(ownedForm1);
}
```

Örneğimizde **`AddOwnedForm`** metodu ile **`Form1`**'e bağlı olan **`OwnedForm1`** adı ile bir **Owned Form** oluşturduk. Eğer **`Form1`**'i simge durumuna küçültürsek, **Owned Form** da simge durumuna küçülecektir. **`OwnedFormSil`** butonunu tıkladıktan sonra **`Form1`**'i simge durumuna küçülttüğümüzde ise, **Owned Form**'un simge durumuna küçülmediğini görürüz. Çünkü **`RemoveOwnedForm`** metodu ile **Owned Form** olma özelliğini sildik.



Diğer Kontroller

A Label Kontrolü

İçeriği sabit kalıp değiştirilmeyecek bilgilerin görüntülenmesinde kullanılır. **`Text`** özelliği ile görüntülenecek olan metin belirlenir.

Örnek olarak **Label** nesnesini **Form** üzerinde kodla oluşturup, bazı özelliklerini değiştirelim:

```
using System;
using System.Windows.Forms;
using System.Drawing;
namespace Label_Kontrolu
{
  public partial class Form1 : Form
  {
   public Form1()
   {
    InitializeComponent();
   }
   private void Form1_Load(object sender, EventArgs e)
   {
  Label Label1 =new Label() ;
  this.Controls.Add(Label1);
  Label1.Text = "Güzel gören güzel düşünür. \r\n Güzel düşünen
hayatından lezzet alır.";
  Label1.Font = new Font("Comic Sans MS", 15, FontStyle.Italic);
  Label1.ForeColor = Color.DarkBlue;
  Label1.AutoSize = true;
   }
  }
}
```

Kodumuzda **`Label1`** adında yeni bir **Label** nesnesi oluşturarak…

```
this.Controls.Add(Label1);
```

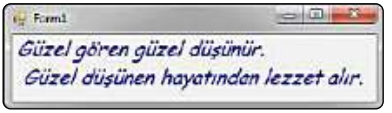
… satırında **Form** üzerine **`Add`** metodunu kullanarak ekledik. **`this`** **Form**'u temsil ederken, **`this.Controls`** **Form** üzerindeki kontrolleri temsil eder.

Yazıtipleri üzerinde işlem yapabilmek için **System.Drawing** sınıfını projemize dahil ettik.

Font özelliği ile eklediğimiz **Label** nesnesinin yazı tipi özelliklerini belirledik.

ForeColor özelliği ile yazı rengini belirledik.

AutoSize özelliğini **true** yaparak, **Label** nesnesinin boyutunun, içerisindeki yazının boyutuna göre otomatik olarak belirlenmesini sağladık.



LinkLabel Kontrolü

Tıklandığında herhangi bir web adresini ya da mail adresini açan **Label** nesnesidir. **LinkLabel** nesnesinin **Links** özelliğine **Add** metodu ile web adresi girilerek hedef site ya da mail adresi belirtilir.

Şu şekilde kullanılır:

```
linkLabel.Links.Add(Adresin başlangıç değeri, Adresin bitiş değeri,
"site adresi ya da mail adresi")
```

Örnek olarak programlarımız için hakkında **Form**'u tasarlayalım.

```
private void linkLabel1_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
  System.Diagnostics.Process.Start(e.Link.LinkData.ToString());
  linkLabel1.LinkVisited = true;
}
private void Form1_Load(object sender, EventArgs e)
{
  linkLabel1.Links.Add(0, 19, "www.volkanaktas.com");
  linkLabel2.Links.Add(0, 24, "volkanaktas@yahoo.com");
}
private void linkLabel2_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
  System.Diagnostics.Process.Start
   ("mailto:volkanaktas@yahoo.com?subject=Soru Bankasi");
  linkLabel2.LinkVisited = true;
}
```



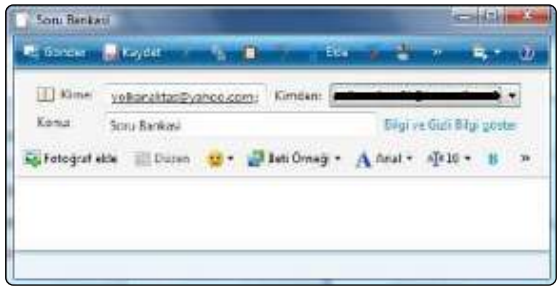
Örneğimizde mail adresine tıklandığında…

```
System.Diagnostics.Process.Start("mailto:volkanaktas@yahoo.
com?subject=Soru Bankası");
```

… komut satırı ile varsayılan mail adresi açılmaktadır.

- **`Mailto`**: Mail'in gönderileceği adresi
- **`Subject`**: Mail konusunu temsil eder.
- **`Body`**: Mail içerisine eklenecek yazıyı belirler.

Ayrıca **`attachment=eklenecek dosya adı`** şeklinde de mailimize dosya ekleyebiliriz.



`linkLabel1.LinkVisited = true` satırı ile linke tıklandıktan sonra, linkin farklı bir renk ile belirtilmesini sağlıyoruz.

TextBox Kontrolü

İçeriği değiştirilebilecek bilgilerin görüntülenmesi amacı ile kullanılır. Bilgi girişi amacıyla da kullanılabilir. **`Text`** özelliği ile görüntülenecek ya da girişi yapılacak metin belirlenir. En fazla **255** karakterlik bilgi görüntülenebilir.

TextBox Kontrolünün Özellikleri

Multiline seçeneği, bir satırdan daha uzun olan metinlerin görüntülenmesinde kullanılır. **Multiline** özelliğini **true** yaparak, daha fazla satır görüntülenmesini sağlayabiliriz.

ScrollBars seçeneği ile **TextBox** nesnemiz multiline modunda iken kaydırma çubukları eklenebilir. **ScrollBars** seçeneği şu değerleri alır:

- **None**: Kaydırma çubukları görüntülenmez. Varsayılan seçenektir.
- **Vertical**: Dikey kaydırma çubuğu eklenir.
- **Horizontal**: Yatay kaydırma çubuğu eklenir.
- **Both**: Hem yatay hem de dikey kaydırma çubuğu eklenir.

PasswordChar seçeneği genellikle şifre giriş işlemlerinde **TextBox** içerisinde girilen şifrenin gizlenip başkaları tarafından görülmemesi amacı ile kullanılır. **PasswordChar** olarak belirlenen karakter örneğin * karakteri girildiğinde **TextBox** içerisindeki metin, * karakterleri şeklinde görüntülenecektir.

ReadOnly özelliği **false** konumunda iken **TextBox** içerisine yazı girişi yapılabilir. **true** konumunda ise yazı girişi yapılamaz, **TextBox** içerisindeki bilgiler yalnız okunabilir.

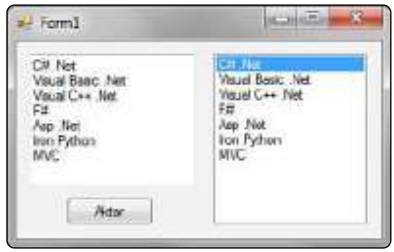
Font özelliği yazı tipini değiştirmek amacı ile kullanılır.

Forecolor özelliği yazı rengini değiştirir.

- **Lines**: **TextBox** içerisine girilen satırların **String[] Array** türünden saklanması amacıyla kullanılır. **TextBox** içerisine girilen satır sayısına **Lines** özelliğinin **Length** metodu ile ulaşılabilir.

Örnek olarak **TextBox** içerisine girilen satırları bir **ListBox** içerisine aktaran bir uygulama yapalım.

```
private void btnAktar_Click(object sender, EventArgs e)
{
  for (int i = 0; i<textBox1.Lines.Length; i++)
  {
   listBox1.Items.Add(textBox1.Lines[i]);
  }
}
```



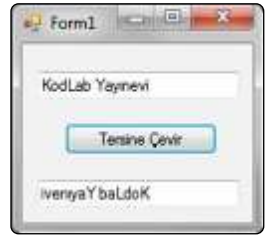
Kodumuzda **`Lines.Length`** özelliğini kullanarak **TextBox** içerisindeki satırların sayısı kadar tekrar eden bir **`for`** döngüsü kuruyoruz. Döngü içerisinde, **ListBox**'ın **`Items.Add`** özelliğini kullanarak **TextBox**'ın satırlarını (**`Lines`**) sırasıyla **ListBox** içerisine ekliyoruz.

`Text`: TextBox içerisindeki yazıyı temsil eder.

Örnek olarak **`textBox1`** içerisindeki yazıyı tersine çeviren bir uygulama yapalım.

```
private void btnAktar_Click(object sender, EventArgs e)
{
  char[] str = textBox1.Text.ToCharArray();
  Array.Reverse(str);
  string strReversed = new string(str);
  textBox2.Text = strReversed;
}
```

Kodumuzda **`ToCharArray`** metodunu kullanarak **TextBox** içerisinde girilmiş olan metni karakter dizisine çeviriyor ve **`str`** isimli karakter diziye aktarıyoruz. **`Array.Reverse`** metodunu kullanarak karakter dizisini tersine çeviriyoruz ve **`strReversed`** isimli string değişkene aktarıyoruz. Son olarak da **`strReversed`** değişkeninin içerisinde tersine çevirili halde bulunan içeriği **`textBox2`** içerisine aktarıyoruz.



TextBox Kontrolünün Metodları

- **`Cut`**: Seçili olan metni keserek taşınmak üzere hafızaya alır.
- **`Copy`**: Seçili olan metni seçerek kopyalanmak üzere hafızaya alır.
- **`Paste`**: Hafızada olan metni yapıştırır.
- **`Undo`**: Son yapılan işlemi geri alır.
- **`Clear`**: **TextBox** içerisindeki yazıyı temizleyerek siler.
- **`Dispose`**: **TextBox** nesnesini siler.

Örnek olarak bir **TextBox** içerisinde temel editör işlemleri gerçekleştiren bir uygulama yapalım. Projemize yazı rengini değiştirmek için 1 adet **ColorDialog** nesnesi, yazı tipini değiştirmek için de 1 adet **FontDialog** nesnesi ekliyoruz.

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace textbox_metodlar
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
  }
  private void btnKes_Click(object sender, EventArgs e)
  {
   if (textBox1.SelectionLength > 0) {
    textBox1.Cut(); }
   else {
    MessageBox.Show("Seçili Alan Yok");  }
  }
  private void btnKopyala_Click(object sender, EventArgs e)
  {
    if (textBox1.SelectionLength > 0) {
    textBox1.Copy(); }
    else {
    MessageBox.Show("Seçili Alan Yok");  }
  }
  private void btnYapistir_Click(object sender, EventArgs e)
```

```
  {
   textBox1.Paste();
  }
  private void btnGeriAl_Click(object sender, EventArgs e)
  {
   textBox1.Undo();
  }
  private void btnYaziTipi_Click(object sender, EventArgs e)
  {
   if (FontDialog1.ShowDialog() == DialogResult.OK)
     textBox1.Font = FontDialog1.Font;
  }
  private void btnYaziRengi_Click(object sender, EventArgs e)
  {
   if (ColorDialog1.ShowDialog() == DialogResult.OK)
     textBox1.ForeColor = ColorDialog1.Color;
  }
  private void btnKalin_Click(object sender, EventArgs e)
  {
   textBox1.Font = new Font(Font, FontStyle.Bold );
  }
  private void btnEgik_Click(object sender, EventArgs e)
  {
   textBox1.Font = new Font(Font, FontStyle.Italic );
  }
  private void btnAltiCizili_Click(object sender, EventArgs e)
  {
   textBox1.Font = new Font(Font, FontStyle.Underline);
  }
 }
}
```

textBox1 nesnesinin **Font** özelliğine yeni bir **Font** nesnesi oluşturarak, belirtilen yazı sitili özelliklerini aktardık. Biz **Bold**, **Italic** ve **Underline** değerlerini kullandık. Buna ek olarak üstü çizili yazı sitili için **Strike**, normal yazı biçimi için **Regular** değerlerini kullanabiliriz.



- **`AppendText`: TextBox** içerisinde mevcut olan yazı silinmeden, **TextBox** içerisinde yazı eklemek amacı ile kullanılır.

Kullanımı şu şekildedir:

```
textBox.AppendText("Eklenecek Metin")
```

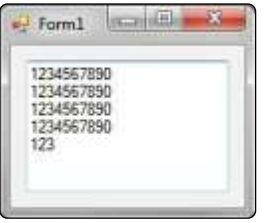
Bu metod yerine…

```
textBox.Text=textBox.Text + "Eklenecek Metin"
```

… yazılışını da kullanabiliriz.

Örneğin **TextBox** içerisinde yazı yazılırken her 10 karakter yazıldığında otomatik olarak bir alt satıra inecek bir uygulama yapalım.

```
int karakteradedi = 0;
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
 if (karakteradedi>=10)
 {
   if (karakteradedi % (10) == 0)
    textBox1.AppendText(Environment.NewLine);
 }
 karakteradedi = karakteradedi + 1;
}
```



Kodumuzda **TextBox**'ın **`KeyPress`** olayı içerisinde her bir tuşa basıldıkça, tuş sayısını bir değişken içerisinde saklıyoruz. Eğer basılan tuş sayısı **10**'un katlarına ulaşmışsa, **`AppendText`** metodu ile **TextBox** içerisine **`Environment.NewLine`** özelliğini kullanarak yeni bir

satır ekliyoruz. Kodumuzun daha henüz bir karakter girişi yapılmamışken boş satır eklememesi için de…

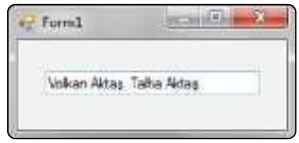
```
if (karakteradedi>=10)
```

… kontrolünü yapıyoruz.

TextBox Kontrolünün Olayları

- **`TextChanged`: TextBox** içerisine değer eklenirken ya da içerisindeki değerler üzerinde değişiklik yapıldığında gerçekleşen olaydır.

```
using System;
using System.Globalization;
using System.Threading;
using System.Windows.Forms;namespace textbox_textchanged
{
  public partial class Form1 : Form
  {
   public Form1()
   {
    InitializeComponent();
   }
   private void textBox1_TextChanged(object sender, EventArgs e)
   {
  CultureInfo CI = Thread.CurrentThread.CurrentCulture;
  TextInfo TI = CI.TextInfo;
  textBox1.Text = TI.ToTitleCase(textBox1.Text);
  textBox1.Select(textBox1.Text.Length, 1);
   }
  }
}
```



Örneğimizde **TextBox** içerisine metin girişi yapılırken öncelikle geçerli kültür ayarlarını **TextBox** içerisine uyguluyoruz. Daha sonra **`ToTitleCase`** özelliği ile cümle ve kelimelerin ilk harflerinin büyük harfle yazılmasını sağlıyoruz.

Button Kontrolü

Bir işlemi, olayı gerçekleştirmek amacı ile kullanılan kontroldür.

Button Kontrolünün Özellikleri

- **`Image`: Button** nesnesi üzerinde görüntülenecek olan resmi belirler.
- **`TabStop`: Button** nesnesinin **Tab** tuşu ile aktifleştirilebilmesini sağlar.
- **`TabIndex`: Tab** tuşuna basıldığında nesnelerin hangi sıra ile aktifleşeceğini belirler. Bu özelliğe nesnenin sıra numarası yazılır. **`TabIndex`** değerleri **0**'dan başlamak üzere yazılır. **`TabIndex`** özelliğini kullanabilmek için **`TabStop`** özelliğini true yapmak gerekmektedir.

Button Kontrolünün Olayları

- **`Click`:** Bir defa tıklama olayıdır.
- **`DoubleClick`:** Çift tıklama olayıdır.

```
private void button1_Click(object sender, EventArgs e)
{
 if (button1.Text == "Başla")
  button1.Text = "Bitir";
 else
  button1.Text = "Başla";
}
```

Örneğimiz için öncelikle tasarım görünümünde **`button1`**'in **`Text`** özelliğini "*Başla*" olarak belirliyoruz. Uygulamamızı çalıştırdığımızda **Button** nesnesi üzerindeki metin "*Başla*" ise "*Bitir*" olarak, "*Bitir*" ise "*Başla*" olarak değiştirilmektedir.

- **`MouseMove`:** Farenin imleci **Button** nesnesi üzerine geldiğinde gerçekleşen olaydır.
- **`MouseLeave`:** Farenin imleci **Button** nesnesi üzerinden ayrıldığında gerçekleşen olaydır.

```
System.Drawing.Color renk;
private void button1_MouseLeave(object sender, EventArgs e)
{
 button1.BackColor = renk;
}
private void button1_MouseMove(object sender, MouseEventArgs e)
{
 button1.BackColor = System.Drawing.Color.DarkGray;
}
private void Form1_Load(object sender, EventArgs e)
{
 renk = button1.BackColor;
}
```

Kodumuzu çalıştırdığımızda fare ile **Button** nesnesinin üzerine geldiğimizde renginin koyu gri renk aldığını, farenin imleci **Button** üzerinden ayrıldığında ise eski rengine döndüğünü görürüz.



Form yüklenirken renk değişkenine `button1`'in arkaplan rengini aktarıyoruz. Farenin imleci üzerine geldiğinde `button1`'in zemin rengini koyu gri olarak değiştiriyoruz. Farenin imleci `button1` üzerinden ayrıldığında ise; renk değişkeninin içerisindeki değeri yeniden `button1` içerisine aktararak, `button1`'in eski renk değerlerine dönmesini sağlıyoruz.

ComboBox Kontrolü

İçerisindeki değerleri açılabilen liste şeklinde görüntüleyen nesnedir. **ComboBox** içerisindeki elemanlara index numaraları yardımı ile ulaşılabilir. İlk eleman 0 index numarası ile belirtilir.

ComboBox Kontrolünün Özellikleri

- `Items`: **ComboBox** içerisindeki değerleri temsil eder.
- `Text`: **ComboBox** içerisinde görüntülenen metindir.
- `SelectedIndex`: **ComboBox** içerisinde seçili olan elemanın index numarasını verir.
- `Sorted`: `true` değerini aldığında **ComboBox** içerisindeki elemanların sıralı şekilde listelenmesini sağlar.

ComboBox Kontrolünün Metodları

Items özelliği aşağıdaki metodları içerir.

- **`Add`**: **ComboBox** içerisine belirtilen değeri listenin sonuna eklemek amacı ile kullanılır.
- **`Remove`**: **ComboBox** içerisindeki belirtilen değeri silmek amacı ile kullanılır.
- **`RemoveAt`**: **ComboBox** içerisindeki belirtilen index numaralı değeri silmek amacı ile kullanılır.
- **`Count`**: **ComboBox** içerisindeki toplam değer sayısını verir.
- **`Insert`**: Belirtilen değeri **ComboBox** içerisinde istenilen bir konuma eklemek amacı ile kullanılır.
- **`Clear`**: **ComboBox** içerisindeki değerlerin hepsini siler.
- **`Contains`**: Eğer belirtilen değer mevcutsa **`true`** değerini, mevcut değilse **`false`** değerini döndürür.
- **`CopyTo`**: Belirtilen elemanı bir dizi içerisine kopyalayarak aktarmak amacı ile kullanılır.

```
private void btnEkle_Click(object sender, EventArgs e)
{
 byte i;
 for (i = 0;i<=100;i++)
 {
  if (i % 2 == 0)
   comboBox1.Items.Add(i);
 }
 label1.Text = comboBox1.Items.Count + " adet eleman var";
 comboBox1.SelectedIndex = 0;
}
private void btnYerlestir_Click(object sender, EventArgs e)
{
 comboBox1.Items.Insert(0, -1);
 comboBox1.SelectedIndex = 0;
 label1.Text = comboBox1.Items.Count + " adet eleman var";
}
private void btnTemizle_Click(object sender, EventArgs e)
{
 comboBox1.Items.Clear();
```

```
 label1.Text = comboBox1.Items.Count + " adet eleman var";
}
private void btnSil_Click(object sender, EventArgs e)
{
 if (comboBox1.SelectedIndex >= 0)
 {
  comboBox1.Items.RemoveAt(comboBox1.SelectedIndex);
 }
 else
  comboBox1.Items.Remove(comboBox1.Items[0]);
 comboBox1.SelectedIndex = 0;
 label1.Text = comboBox1.Items.Count + " adet eleman var";
}
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
 label2.Text = "Seçili eleman= " +
   comboBox1.Items[comboBox1.SelectedIndex];
}
```

ComboBox içerisine…

```
if (i % 2 == 0)
```



… komut satırı ile 0 ile 100 arasındaki sayılardan çift olanları seçerek…

```
comboBox1.Items.Add(i);
```

… komut satırı ile **ComboBox** içerisine sıra ile elemanlarımızı ekliyoruz.

```
comboBox1.Items.Insert(0, -1);
```

… satırında **`Insert`** metodu ile **`0`** nolu index sırasına **`-1`** adlı elemanımız ekliyoruz.

`label2` adlı **Label** içerisinde seçili olan elemanı **`SelectedIndex`** özelliğini kullanarak görüntüledik. **`comboBox1.Items`** şeklinde **ComboBox** içerisindeki eleman listesine ulaşarak, index numarası olarak **`combobox1.SelectedIndex`** ifadesi ile seçili olan elemanın index numarasını verdik.

ComboBox içerisinde eğer seçili eleman varsa seçili elemanı, seçili eleman yoksa ilk sıradaki elemanı sildik.

```
comboBox1.Items.Remove(comboBox1.Items[0]);
```

… satırı ile ilk sıradaki elemanı silme işlemi yaptık.

```
comboBox1.Items.RemoveAt(comboBox1.SelectedIndex);
```

… satırı ile ise seçili elemanın index numarası hangisi ise o elemanı sildik.

ListBox Kontrolü

İçerisindeki değerleri liste şeklinde görüntüleyen nesnedir. **ListBox** içerisindeki elemanlara index numaraları yardımı ile ulaşılabilir. İlk eleman 0 index numarası ile belirtilir.

ListBox nesnesinin **ComboBox** nesnesi ile hemen hemen tüm özellik, metot ve olayları aynı olduğu için burada farklı olanlara değinelim.

ListBox içerisinde sıralama işlemi yapan bir örnek kod yazalım.

```
int i, sayi;
private void btnEkle_Click(object sender, EventArgs e)
{
 for(i = 1;i<=10;i++)
 {
  sayi =  Convert.ToInt32(Interaction.InputBox
      (i + ". sayıyı giriniz."));
  listBox1.Items.Add(sayi);
 }
}
private void btnSirala_Click(object sender, EventArgs e)
{
 int l, m;
 for (l = 0; l <= 9; l++)
 {
  for (m = l + 1; m <= 9; m++)
  {
    if (Int32.Parse(listBox1.Items[l].ToString()) >
```

```
   Int32.Parse(listBox1.Items[m].ToString()))
    {
    int aradeger = Int32.Parse(listBox1.Items[l].ToString());
    listBox1.Items[l] = listBox1.Items[m];
    listBox1.Items[m] = aradeger;
     }
  }
 }
}
```

Kodumuzda öncelikle iç içe 2 adet **for** döngü kurulmakta. Bu döngülerden ikincisi birincisinden bir fazla değer almakta. Bunun nedeni önceki değerle sonraki değeri karşılaştırmak için. Burada önceki değer sonraki değerden büyükse, yer değiştirme işlemi yapılmaktadır. Yer değiştirme işlemi sırasında **aradeger** isimli bir değişken vasıtası ile yer değiştirme işlemini yapmaktayız. Değerler yer değiştirilmeden önce **aradeger** değişkenine aktarılmaktadır.

Programımızı çalıştırıp **Ekle** düğmesine bastığımızda, yandaki şekilde görüldüğü gibi bir giriş kutusu yardımı ile 10 adet sayı eklenmektedir





Sırala düğmesine bastığımızda ise sayılar sıralanarak yine aynı **ListBox** içerisine aktarılmaktadır.

Başka bir örnek ile bir **ListBox** içerisinde arama yaptıralım:

```
int i;
Boolean bulundu= false;
private void btnAktar_Click(object sender, EventArgs e)
{
 for (i = 0; i <= 100; i++)
     listBox1.Items.Add(i);
}
```

```
private void btnArama_Click(object sender, EventArgs e)
{
 for(i = 1;i<=listBox1.Items.Count - 1;i++)
 {
  if (txtAranan.Text == listBox1.Items[i].ToString())
  {
   bulundu = true;
   listBox1.SelectedIndex = Int32.Parse(listBox1.Items[i].ToString());
   break;
  }
  else bulundu = false;
 }
 if (bulundu == true)
  MessageBox.Show("Bulundu");
 else
  MessageBox.Show("Bulunamadı");
}
```

Kodumuzda öncelikle **btnAktar** isimli **Button** içerisinde 0 ile 100 arasındaki sayıları **ListBox** içerisine **Items.Add** metodunu kullanarak **for** döngüsü içerisinde aktardık.

btnArama isimli **Button** içerisinde ise **Items.Count** özelliği ile **ListBox** içerisindeki eleman sayısı kadar bir for döngüsü kuruyoruz.

```
if (txtAranan.Text == listBox1.Items[i].ToString())
```

… satırında **ListBox** içerisindeki değerler sırasıyla TextBox içerisine girilen değer ile karşılaştırılıyor. Değerler eşit ise, bulundu değişkeninin değeri **true** olmaktadır.

```
listBox1.SelectedIndex = Int32.Parse(listBox1.Items[i].ToString( ));
```

Komut satırı ile **ListBox** içerisinde bulunan değerin index numarasını vererek, o elemanı **ListBox** içerisinde seçili duruma getiriyoruz.

Eğer karşılaştırılan değerler birbirine eşit değil ise, **bulundu** değişkeninin değeri **false** olmaktadır.



Son olarak karşılaştırma işleminin bitiminde **bulundu** değişkeninin değerine göre ekrana mesaj penceresi, içerisinde değerin bulunup bulunmadığı hakkında bilgi verilmektedir.

Drag And Drop (Fare İle Sürükleme ve Bırakma) Olayları

Bir nesnenin içerisindeki içeriği alıp başka bir nesnenin içerisine taşımaya **drag and drop** ya da **sürükle ve bırak** denilir. Bir **ListBox** nesnesi içerisinde sürükle ve bırak olayını gerçekleştirebilmek için öncelikle **AllowDrop** özelliğinin **true** olarak ayarlanması gerekmektedir. Aksi takdirde sürükleme ve bırakma işlemi gerçekleştirilemez.

Örneğimiz için **Form** üzerine iki adet **ListBox** ekleyelim ve **AllowDrop** özelliklerini **true** olarak ayarlayalım.

```
private void Form1_Load(object sender, EventArgs e)
{
 byte i;
 for(i = 1;i<=10;i++)
  listBox1.Items.Add(i);
}
private void listBox1_MouseDown(object sender, MouseEventArgs e)
{
 if (listBox1.Items.Count == 0) return;
 string deger = listBox1.Items[listBox1.
IndexFromPoint(e.X,e.Y)].ToString();
 if (DoDragDrop(deger, DragDropEffects.All) == DragDropEffects.All)
    listBox1.Items.RemoveAt(listBox1.IndexFromPoint(e.X, e.Y));
}
private void listBox2_DragDrop(object sender, DragEventArgs e)
{
```

```
if (e.Data.GetDataPresent(DataFormats.StringFormat))
listBox2.Items.Add(e.Data.GetData(DataFormats.StringFormat));
}
private void listBox2_DragOver(object sender, DragEventArgs e)
{
 e.Effect = DragDropEffects.All;
}
```

Taşıma işlemini **listBox1**'den **listBox2**'ye doğru yapacağız. Dolayısı ile **ListBox** içerisindeki bir elemanı seçeceğimiz için **listBox1**'in **mousedown** olayını kullanıyoruz.

```
string deger = listBox1.Items[listBox1.IndexFromPoint(e.X,e.Y)].ToString();
```

… satırı ile **ListBox** nesnesinin **IndexFromPoint** metodunu kullanarak belirtilen **X** ve **Y** koordinatlarında **ListBox** nesnesinin hangi index numaralı değeri olduğunun kontrolü yapılarak **ListBox**'ın ilgili elemanı **deger** adlı değişkene aktarılır.

DoDragDrop metodu taşıma işleminin başlamasını sağlar.

DragDropEffects, taşınacak olan verinin nasıl taşınacağını belirler. Şu değerleri alır:

- **All**: Veri kopyalanır, sürükleme kaynağından silinir ve hedefe bırakılır.
- **Copy**: Sürükleme kaynağındaki veri kopyalanarak hedefe bırakılır.
- **Link**: Sürükleme kaynağındaki veri, hedef nesneye bağlanır.
- **Move**: Sürükleme kaynağındaki veri taşınarak hedefe bırakılır.
- **None**: Hedef nesne sürüklenip bırakılan verileri kabul etmez.
- **Scroll**: Hedef nesnede kaydırma yapıldığını bildirir.

```
if (DoDragDrop(deger, DragDropEffects.All) == DragDropEffects.All)
```

… satırı ile **DoDragDrop** metodu vasıtası ile eğer sürükleme işlemi sonucunda taşıma işlemi başlamışsa…

```
listBox1.Items.RemoveAt(listBox1.IndexFromPoint(e.X, e.Y));
```

ListBox nesnesinin **X** ve **Y** koordinatlarında **ListBox** nesnesinin hangi index numaralı değeri olduğunun kontrolü yapılarak **ListBox**'ın ilgili elemanı silinir.

listBox1'den değerlerimizi seçip taşımaya başladık ve **listBox1**'den taşınan değeri sildik.

Taşınan nesne **listBox2**'nin üzerine geldiğinde ne gibi bir işlem yapılacağını, **listBox2**'nin **DragOver** olayı ile belirliyoruz.

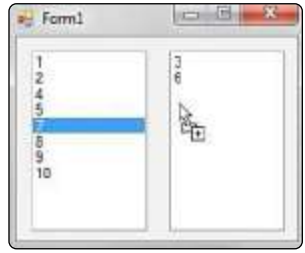
Son aşamada ise taşınan değeri **listBox2** içerisine bırakacağız. Bunun için **listBox2**'nin **DragDrop** olayını kullanıyoruz.

```
if (e.Data.GetDataPresent(DataFormats.StringFormat))
```

… satırı ile taşınan verinin formatının string bir değer olup olmadığının kontrolünü yaparak…

```
listBox2.Items.Add(e.Data.GetData(DataFormats.StringFormat));
```

… satırı ile **listBox2** içerisine taşınan string formattaki veriyi aktarıyoruz.

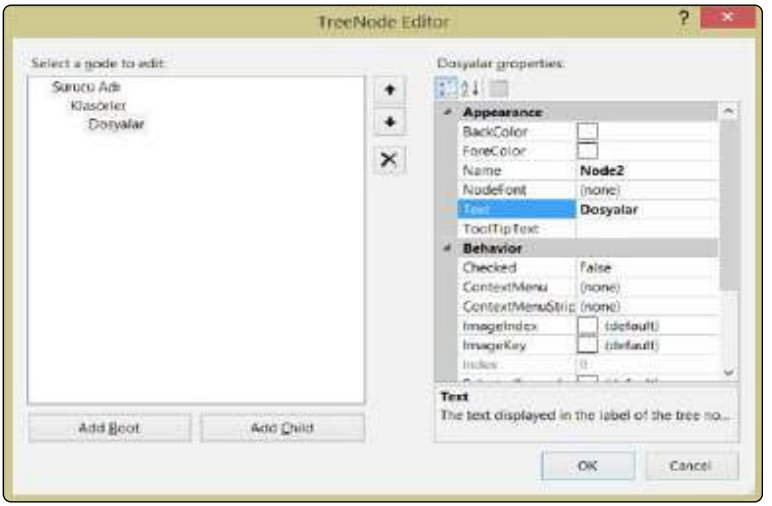


TreeView Kontrolü

İçerisindeki değerleri ağaç biçiminde hiyerarşik olarak görüntüleyen nesnedir. **TreeView** içerisindeki elemanlar **node** (*düğüm*) olarak isimlendirilir. Düğümler **parent node** (*ana düğüm*) ve **child node** (*çocuk düğüm*) olarak iki gruba ayrılırlar. Bir ana düğüm birden çok çocuk düğüme sahip olabilir. **TreeView** nesnesi içerisinde en az bir adet parent node bulunmalıdır. **TreeView** içerisindeki düğümlere index numaraları yardımı ile ulaşılabilir. İlk düğüm 0 index numarası ile belirtilir. **TreeView** içerisindeki düğümler tanımlanırken, **TreeNode** nesnesi olarak tanımlanır.

TreeView Kontrolünün Özellikleri

- **`CheckBoxes`: `True`** değerini aldığında her bir düğümün yanında **CheckBox** görüntülenmesini sağlar.
- **`HideSelection`: `True`** değerini aldığında **TreeView** nesnesi aktifliğini kaybettiğinde seçili olan elemanın seçimi kaldırılır. **`False`** değerini aldığında ise **TreeView** nesnesi aktifliğini kaybetse bile eleman seçili halde kalmaya devam eder.
- **`HotTracking`: `True`** değerini aldığında farenin imleci, **TreeView** içerisindeki elemanlar üzerinden geçtiğinde rengi koyulaşarak altı çizili hale gelir.
- **`LabelEdit`: `True`** değerini aldığında düğümler üzerinde bulunan yazıların değiştirilebilmesini sağlar.
- **`Nodes`:** Parent node ve ve child node elemanlarının eklenip düzenlenebildiği **TreeNode Editör** penceresini açar.



- **`SelectedNode`:** Seçili olan bir düğümü görüntülemek ya da bir düğümü seçmek amacı ile kullanılır.
- **`SelectedImageIndex`:** Index değerini kullanarak **TreeView** içerisinde seçili olan bir imajı görüntülemek ya da seçmek amacı ile kullanılır.
- **`SelectedImageKey`: `Key`** değerini kullanarak **TreeView** içerisinde seçili olan bir imajı görüntülemek ya da seçmek amacı ile kullanılır.

- **`ShowLines`:** Düğümleri birbirine bağlayan çizgilerin görüntülenmesini ya da görüntülenmemesini sağlar.
- **`ShowNodeToolTips`:** Düğümler hakkında açıklayıcı bilgi pencerelerinin görüntülenmesini ya da görüntülenmemesini sağlar.
- **`ShowPlusMinus`:** Alt düğümlerden önce + ya da - işaretinin görüntülenmesini ya da görüntülenmemesini sağlar. Bu işaret ile alt dallar görüntülenir ya da kapatılabilir. Düğüm kapalı iken + , açık iken – işareti görüntülenir.
- **`ShowRootLines`:** Ana düğümlerden önce + ya da - işaretinin görüntülenmesini ya da görüntülenmemesini sağlar.
- **`StateImageList`:** Düğümlerin her birinin durumunu görüntülemek için kullanılan **`ImageList`** nesnesinin adıdır.

TreeView Kontrolünün Metodları

- **`ExpandAll`: TreeView** içerisinde bulunan tüm düğümlerin açık vaziyette listelenmesini sağlar.
- **`CollapseAll`: TreeView** içerisinde bulunan tüm düğümlerin kapalı vaziyette listelenmesini sağlar.
- **`GetNodeCount`: TreeView** içerisinde bulunan düğümlerin sayısını listelemek için kullanılır.

  `GetNodeCount(true ya da false)` şeklinde kullanılır. Eğer **`true`** değerini alırsa, alt düğümlerin sayısını bulmayıp sadece ana düğümlerin sayısını verir. **`false`** değerini aldığında ise, ana düğümlerle beraber tüm alt düğümlerin sayısını da verir.

- **`TopNode`: TreeView** nesnesi içerisinde listelenen düğümlerin ana düğümünü verir.

  Örneğin…

```
MessageBox.Show(treeView1.TopNode.ToString());
```

  Şeklindeki kullanım ile **TreeView** nesnesi içerisinde listelenen düğümlerin ana düğümünü bir mesaj penceresi içerisinde görüntüler.

- **`VisibleCount`: TreeView** nesnesi içerisinde listelenen düğümlerden, o an görüntülenmekte olan düğüm sayısını verir.

TreeView Kontrolünün Olayları

- **`AfterCheck`:** Düğümlerden biri üzerinde bulunan **CheckBox** işaretlendikten sonra ya da işareti kaldırıldıktan sonra gerçekleşen olaydır.
- **`AfterCollapse`: TreeView** içerisinde açık halde bulunan düğümler kapatıldıktan sonra gerçekleşen olaydır.
- **`AfterExpand`: TreeView** içerisinde kapalı halde bulunan düğümler açıldıktan sonra gerçekleşen olaydır.
- **`AfterLabelEdit`: TreeView** içerisinde bulunan düğümler üzerinde bulunan yazılar düzenlendikten sonra gerçekleşen olaydır.
- **`AfterSelect`: TreeView** içerisinde düğümler seçildikten sonra gerçekleşen olaydır.
- **`BeforeCheck`:** Düğümlerden biri üzerinde bulunan **CheckBox** işaretlenmeden önce ya da işareti kaldırılmadan önce gerçekleşen olaydır.
- **`BeforeCollapse`: TreeView** içerisinde açık halde bulunan düğümler kapatılmadan önce gerçekleşen olaydır.
- **`BeforeExpand`: TreeView** içerisinde kapalı halde bulunan düğümler açılmadan önce gerçekleşen olaydır.
- **`BeforeLabelEdit`: TreeView** içerisinde bulunan düğümler üzerinde bulunan yazılar düzenlenmeden önce gerçekleşen olaydır.
- **`BeforeSelect`: TreeView** içerisinde düğümler seçilmeden önce gerçekleşen olaydır.
- **`NodeMouseClick`:** Düğümlerden biri tıklandığında gerçekleşen olaydır.
- **`NodeMouseDoubleClick`:** Düğümlerden biri çift tıklandığında gerçekleşen olaydır.
- **`NodeMouseHover`:** Düğümlerden birinin üzerinden fare ile geçildiğinde gerçekleşen olaydır.
- **`Sort`: TreeView** içerisindeki düğümleri küçükten büyüğe doğru sıralamak için kullanılır. Farklı biçimde sıralamak amacıyla **TreeViewNodeSorter** adlı metodu kullanabiliriz.

Belirtilen bir yol içerisindeki klasörleri **TreeView** içerisinde listeleyen örnek bir uygulama yapalım.

```
using System;
using System.IO;
using System.Windows.Forms;
namespace treeview_kontrolu
{
 public partial class Form1 : Form
 {
   public Form1()
   {
  InitializeComponent();
   }
   private void btnListele_Click(object sender, EventArgs e)
   {
  string[] klasorler  = Directory.GetDirectories(textBox1.Text);
  treeView1.Nodes[0].Text = textBox1.Text;
  TreeNode anadugum = treeView1.Nodes[0];
  if (klasorler.Length != 0)
  {
    foreach(string aktifklasor in klasorler)
    {
     TreeNode dugum = new TreeNode(aktifklasor);
     anadugum.Nodes.Add(dugum);
    }
  }
  treeView1.ExpandAll();
   }
   private void Form1_Load(object sender, EventArgs e)
   {
  treeView1.Nodes.Add("Klasör Listesi");
   }
 }
}
```



Uygulamamız ile **TextBox** içerisine girdiğimiz yol bilgisi alınarak, bu yol içerisindeki klasörler **TreeView** nesnesi içerisinde listelenmektedir.

Kodumuzda **Form**'un `Load` olayı içerisinde

```
treeView1.Nodes.Add("Klasör Listesi");
```

… satırı ile **TreeView** içerisine "*Klasör Listesi*" adında bir adet **parent node** ekliyoruz. Diğer node'ları bu parent node altına ekleyeceğiz.

Klasörler üzerinde işlem yapmak için `System.IO` sınıfını projemize dahil ettik.

```
string[] klasorler  = Directory.GetDirectories(textBox1.Text);
```

… metodu ile `textBox1` içerisinde yazılan yol bilgisini alıp, bu yol içerisinde bulunan klasörlerin listesini elde ediyoruz.

```
treeView1.Nodes[0].Text = textBox1.Text;
```

… satırı ile eklemiş olduğumuz **parent node** ismini **TextBox** içerisine yazan yol bilgisi ile değiştiriyoruz.

```
TreeNode anadugum = treeView1.Nodes[0];
```

`anadugum` adında bir **TreeNode** nesnesini tanımlayarak, buna **Form**'un `Load` olayı esnasında oluşturmuş olduğumuz **parent node** nesnesini aktarıyoruz.

```
foreach(string aktifklasor in klasorler)
```

… satırı ile `klasorler` isimli değişkenimiz içerisinde bulunan `klasor` listesi

elemanları üzerinde işlem yapmak için bir döngü kuruyoruz.

```
TreeNode dugum = new TreeNode(aktifklasor);
```

… satırı ile döngü içerisinde **dugum** adında yeni bir **TreeNode** nesnesi oluşturarak, **aktifklasor** değişkeni içerisinde saklanmakta olan klasör adı bilgisini aktarıyoruz.

```
anadugum.Nodes.Add(dugum);
```

Yine döngü içerisinde, oluşturmuş olduğumuz yeni düğümü **anadugum** adlı parent node altına ekliyoruz.

```
treeView1.ExpandAll();
```

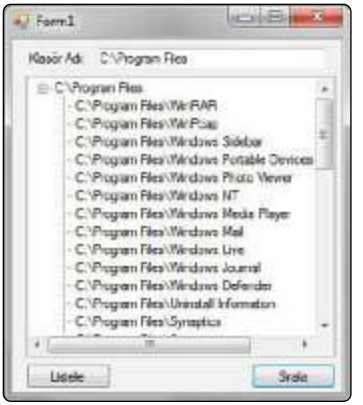
… satırı ile de ana düğüm ve tüm alt düğümlerin açık vaziyette listelenmesini sağlıyoruz.

TreeView içerisindeki sıralama şeklini küçükten büyüğe doğru değil de, tam tersi biçimde büyükten küçüğe doğru sıralayacak biçimde yukarıdaki örneğimizi düzenleyelim. **Form** üzerine **btnSirala** adıyla yeni bir **Button** nesnesi daha ekliyoruz.

```
using System.IO;
using System.Windows.Forms;
namespace treeview_kontrolu
{
 public partial class Form1 : Form
 {
   public Form1()
   {
  InitializeComponent();
   }
    private void btnSirala_Click(object sender, EventArgs e)
   {
    treeView1.TreeViewNodeSorter = new NodeSorter();
   }
   private void btnListele_Click(object sender, EventArgs e)
   {
  string[] klasorler = Directory.GetDirectories(textBox1.Text);
```

```
  treeView1.Nodes[0].Text = textBox1.Text;
  TreeNode anadugum = treeView1.Nodes[0];
  if (klasorler.Length != 0)
  {
    foreach(string aktifklasor in klasorler)
    {
     TreeNode dugum = new TreeNode(aktifklasor);
     anadugum.Nodes.Add(dugum);
    }
  }
  treeView1.ExpandAll();
   }
   private void Form1_Load(object sender, EventArgs e)
   {
  treeView1.Nodes.Add("Klasör Listesi");
   }
 }
 public class twSorter : IComparer
 {
   public int Compare(object x, object y)
   {
  TreeNode tx = x as TreeNode;
  TreeNode ty = y as TreeNode;
  return string.Compare(ty.Text, tx.Text);
   }
 }
}
```

Kodumuzda **twSorter** isimli **IComparer** interface'den türettiğimiz bir karşılaştırma sınıfı yazıyoruz. Bu sınıfın içerisinde **Compare** metodunu kullanıyoruz. **Compare** metodu, iki adet parametre almaktadır. Bu parametrelerle **TreeView** içerisindeki düğümler karşılaştırılmaktadır. Eğer geriye dönen değer 0 ise, her iki düğüm birbirine eşit, 0'dan küçükse ilk parametredeki düğüm daha üst sıraya taşınır, 0'dan büyükse ikinci parametredeki düğüm daha üst sıraya taşınır.



Kodumuzu bu hali ile çalıştırdığımızda sıralama işleminin yönünün tersine çevrilerek büyükten küçüğe doğru sıralama yapıldığını görürüz.

Eğer sıralama işleminin yönünü küçükten büyüğe olacak şekilde değiştirmek istersek, **Compare** metodunu…

```
string.Compare(tx.Text, ty.Text);
```

… şeklinde **x** ve **y** parametrelerini yer değiştirerek kullanmalıyız.

☑ CheckBox Kontrolü

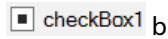
İki ya da üç duruma sahip olabilen, onay ve seçim işlemleri için kullanılan bir kontroldür.

CheckBox Kontrolünün Özellikleri

- **Checked:** **True** değerini aldığında **CheckBox** nesnesi işaretlidir, **false** değerini aldığında ise işaretli değildir.
- **CheckState:** 3 adet değer alır.

  Unchecked: İşaretsiz.

  Checked: İşaretli.

  Indeterminate: Belirsiz. Bu konumda iken 

biçimini alarak ne işaretli ne de işaretsiz bir konum alır.
- **ThreeState:** **CheckBox**'ın **Checked**, **Unchecked** şeklinde iki değer değil de, **Checked**, **UnChecked** ve **Indeterminate** şeklinde üç değer almasını sağlar.
- **FlatStyle:** **CheckBox** nesnesinin görünüm şeklini değiştirir. Aldığı değerler ve bunun sonucunda oluşan görünüm şekilleri şunlardır.

Flat: 

Popup: 

Standard: 

System: 

CheckBox Kontrolünün Olayları

CheckedChanged: **CheckBox** nesnesinin durumu değiştirildiğinde gerçekleşen olaydır.

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace checkbox_font_style
{
  public partial class Form1 : Form
  {
 public Form1()
 {
   InitializeComponent();
 }
 private void chkKalin_CheckedChanged(object sender, EventArgs e)
 {
   textBox1.Font = new Font(textBox1.Font.Name,
    textBox1.Font.Size,textBox1.Font.Style ^ FontStyle.Bold);
 }
 private void chkEgik_CheckedChanged(object sender, EventArgs e)
 {
   textBox1.Font = new Font(textBox1.Font.Name,
    textBox1.Font.Size,textBox1.Font.Style ^ FontStyle.Italic);
 }
 private void chkAltiCizili_CheckedChanged(object sender, EventArgs e)
 {
   textBox1.Font = new Font(textBox1.Font.Name,
    textBox1.Font.Size,textBox1.Font.Style ^ FontStyle.Underline);
 }
}
}
```



Örneğimizde **CheckBox**'lar işaretliyken yazı stilinin uygulanmasını, işaretli değilken yazı stilinin uygulanmamasını sağladık. **^** (*XOR-Özel veya*) mantıksal operatörü bir tarafın **`true`** diğer tarafın **`false`** olduğu durumlarda çalışır. Burada da eğer daha önceden o stil özelliği uygulanmışsa iptal edilmesini, uygulanmamışsa uygulanmasını sağlıyoruz.

CheckedListBox Kontrolü

İçerisindeki değerleri işaretlenebilir liste şeklinde görüntüleyen nesnedir. **CheckedListBox** içerisindeki işaretli ya da işaretsiz elemanlara index numaraları yardımı ile ulaşılabilir. İlk eleman 0 index numarası ile belirtilir.

CheckedListBox Kontrolünün Özellikleri

`Items`: **CheckedListBox** içerisindeki tüm elemanları temsil eder.

- `CheckedItems`: **CheckedListBox** içerisindeki işaretlenmiş tüm elemanları temsil eder.

CheckedListBox Kontrolünün Metodları

`Items` özelliği aşağıdaki metodları içerir.

- `Add`: Belirtilen değeri listenin sonuna eklemek amacı ile kullanılır.
- `Remove`: Belirtilen değeri silmek amacı ile kullanılır.
- `RemoveAt`: Belirtilen index numaralı değeri silmek amacı ile kullanılır.
- `Count`: Eleman sayısını verir.
- `Insert`: Belirtilen değeri listenin istenilen bir yerine eklemek amacı ile kullanılır.
- `Clear`: **CheckedListbox** içerisindeki değerlerin hepsini siler.
- `Contains`: Eğer belirtilen değer mevcutsa `true` sonucunu, mevcut değilse `false` değerini döndürür.
- `IndexOf`: Değeri belirtilen elemanın index numarasını geriye döndürür.
- `CopyTo`: Belirtilen elemanı bir dizi içerisine kopyalayarak aktarmak amacı ile kullanılır.

CheckedItems özelliği ise; **Count**, **IndexOf**, **Contains** ve **CopyTo** metodlarına sahiptir.

SetItemChecked Metodu:

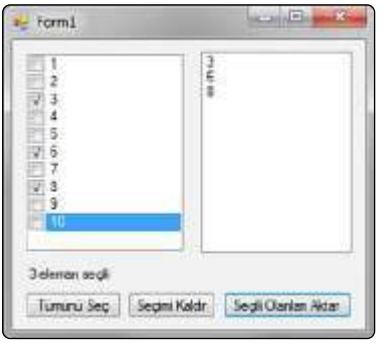
CheckedListBox içerisindeki bir elemanı işaretli duruma getirmek ya da seçili ise, işaretini kaldırmak için kullanılır. Kullanımı şu şekildedir:

```
SetItemChecked(index değeri, true ya da false)
```

İkinci parametreye `true` değeri verildiğinde belirtilen index numaralı eleman işaretlenir, `false` değeri verildiğinde ise işareti kaldırılır.

```
byte i;
private void btnTumunuSec_Click(object sender, EventArgs e)
{
 for(i = 0; i<= checkedListBox1.Items.Count - 1; i++)
 {
  checkedListBox1.SetItemChecked(i, true);
 }
 label1.Text = checkedListBox1.CheckedItems.Count + " eleman seçili";
}
private void Form1_Load(object sender, EventArgs e)
{
 for (i = 1; i <= 10; i++)
  checkedListBox1.Items.Add(i);
}
private void btnAktar_Click(object sender, EventArgs e)
{
 for (i = 0; i <= checkedListBox1.CheckedItems.Count - 1; i++)
  listBox1.Items.Add(checkedListBox1.CheckedItems[i]);
}
private void btnSecimiKaldir_Click(object sender, EventArgs e)
{
 for(i = 0;i <= checkedListBox1.Items.Count - 1;i++)
  checkedListBox1.SetItemChecked(i, false);
 label1.Text = checkedListBox1.CheckedItems.Count + " eleman seçili";
}
private void checkedListBox1_ItemCheck(object sender, ItemCheckEventArgs e)
{
 label1.Text = checkedListBox1.CheckedItems.Count - 1 + " eleman seçili";
}
```

Kodumuzda öncelikle **`Form`**'un **`Load`** olayı içerisinde **`for`** döngüsü içerisinde **`1`** ile **`10`** arasındaki sayıları **`Items.Add`** metodu ile **CheckedListBox** içerisine aktarıyoruz.



btnTumunuSec butonu içerisinde **Items.Count** özelliği ile **CheckedListBox** içerisindeki elemanların sayısı kadar bir döngü kurarak, **SetItemChecked** metoduna **true** değerini verip sırasıyla tüm elemanları seçili duruma getiriyoruz.

btnSecimiKaldir butonu içerisinde ise, aynı şekilde bir döngü kurarak **SetItemChecked** metoduna **false** değerini vererek, sırasıyla tüm elemanların işaretlerini kaldırıyoruz.

CheckedListBox'ın **ItemCheck** olayı içerisinde **CheckedItems.Count** özelliği ile seçili olan elemanların sayısı **Label** nesnesi içerisinde görüntülenmektedir.

btnAktar isimli **Button** içerisinde ise **CheckedItems.Count** özelliği ile **CheckedListBox** içerisinde seçili elemanların sayısı kadar bir döngü kurarak, seçili olan elemanlar (**CheckedItems**), **ListBox** nesnesi içerisine aktarılmaktadır.

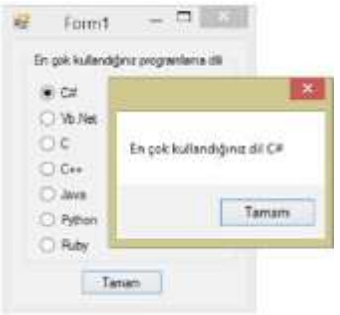
RadioButton Nesnesi

Birden fazla seçenekten yalnızca birini seçme durumlarında kullanılan nesnedir. **Form** üzerine eklenen birden fazla **RadioButton** içerisinden aynı anda yalnızca biri seçili olabilir.

RadioButton Kontrolünün Özellikleri

Checked Özelliği: RadioButton nesnesinin seçili olup olmamasını belirler.

```
private void btnTamam_Click(object sender, EventArgs e)
{
 if (radioButton1.Checked )
  MessageBox.Show("En çok kullandığınız dil C#");
 else if (radioButton2.Checked)
  MessageBox.Show("En çok kullandığınız dil Vb.Net");
 else if (radioButton3.Checked)
  MessageBox.Show("En çok kullandığınız dil C");
 else if (radioButton4.Checked)
  MessageBox.Show("En çok kullandığınız dil C++");
 else if (radioButton5.Checked)
  MessageBox.Show("En çok kullandığınız dil Python");
 else if (radioButton6.Checked)
  MessageBox.Show("En çok kullandığınız dil Ruby");
 else if (radioButton7.Checked)
  MessageBox.Show("En çok kullandığınız dil Delphi");
}
```



Örneğimizde **RadioButton** nesnesinin işaretli olup olmadığı `Checked` özelliği ile kontrol edilmektedir. Eğer geriye dönen değer `true` ise işaretli, `false` ise işaretli değildir. Eğer işaretli ise, kullanılan dile ait değer bir mesaj penceresi içerisinde görüntülenmektedir.

Bu uzun kodu nesnel bir yaklaşımla kısaltabiliriz. **RadioButton** nesnelerimizi **GroupBox** içerisinde bulundurmamız nedeni ile `groupBox.Controls` şeklinde **GroupBox** elemanlarına erişebiliriz.

```
private void btnTamam_Click(object sender, EventArgs e)
{
 foreach(Control radio in groupBox1.Controls)
 {
  RadioButton rdb = (RadioButton)(radio);
  if (rdb.Checked) MessageBox.Show
    ("En çok kullandığınız dil " + rdb.Text);
 }
}
```

Kodumuzda **GroupBox** içerisindeki elemanlara bir döngü içerisinde erişiyoruz. Eriştiğimiz elemanları…

```
RadioButton rdb = (RadioButton)(radio);
```

… satırında **RadioButton** nesnesine dönüştürüyoruz. Bu işlemin ardından **RadioButton** nesnelerinin `Checked` özelliğini kontrol ederek dönen değer `true` ise, işaretli olduğunu belirleyerek bir mesaj penceresi içerisinde işaretli olan **RadioButton** nesnesinin metninin görüntülenmesini sağlıyoruz.

GroupBox Kontrolü

Nesneleri gruplandırmak amacı ile kullanılır. **GroupBox** nesnesi içerisinde yerleştirilen elemanlar **GroupBox** nesnesi ile birlikte taşınabilir.

Örneğin; **CheckedListBox** konusunda yaptığımız örnekteki nesneleri **GroupBox** nesnesi ile gruplandırdığımızda form tasarımı şekildeki gibi olacaktır.



Kodla **GroupBox** nesnesi oluşturarak içerisinde 1 adet **Label**, 1 adet de **TextBox** nesnesi ekleyelim.

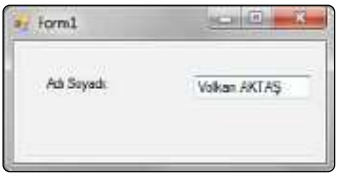
```
using System;
using System.Windows.Forms;
using System.Drawing;
namespace groupBox_Kontrolu
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
  }
  private void Form1_Load(object sender, EventArgs e)
  {
   GroupBox groupBox1 = new GroupBox();
   Label label1 = new Label();
   label1.Location = new Point(20, 30);
   label1.Text = "Adı Soyadı: ";

   TextBox text1 = new TextBox();
   text1.Text = "Volkan AKTAŞ";
   text1.Location = new Point(150, 30);
   Controls.Add(groupBox1);
   groupBox1.FlatStyle = FlatStyle.Flat;
   groupBox1.AutoSize = true;
```

```
    groupBox1.Left = groupBox1.Padding.Left +
          groupBox1.Margin.Left;

    groupBox1.Controls.Add(label1);
    groupBox1.Controls.Add(text1);
   }
  }
}
```

Oluşturduğumuz **GroupBox** kontrolü ve içerisindeki kontrollerin görünümü yandaki gibidir.



Oluşturduğumuz **Label** ve **TextBox** nesnelerini...

```
groupBox1.Controls.Add(label1);
groupBox1.Controls.Add(text1);
```

… komut satırları ile **GroupBox** içerisine ekledik.

GroupBox'ın içerisindeki nesnelere göre otomatik boyut alması için **`Autosize`** özelliğine **`true`** değerini verdik.

GroupBox'ın yatay konumunu (**`Left`**) belirlemek için…

```
groupBox1.Left = groupBox1.Padding.Left + groupBox1.Margin.Left;
```

… komut satırı ile yatay aralık değeri (**`Padding.Left`**) ile yatay kenar boşluklarını (**`Margin.Left`**) topladık.

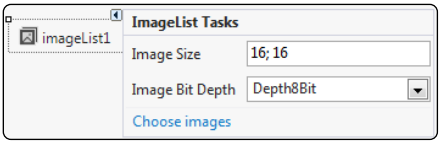
ImageList Kontrolü

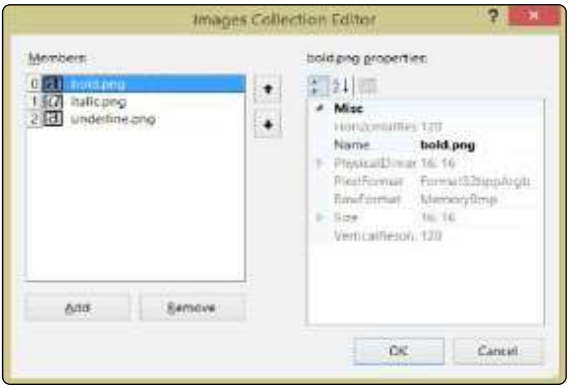
Resim dosyalarının bir arada saklanıp herhangi bir nesne içerisinden resim dosyalarına erişilmesini sağlamak amacı ile kullanılır. Form üzerine eklenmeyip **Visual Studio IDE** penceresinin alt tarafındaki panele eklenir.

ImageList Kontrolünün Özellikleri

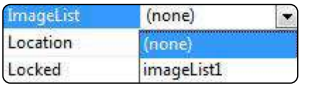
- **`ImageSize`: ImageList** içerisine eklenecek resimlerin genişlik ve yükseklik değerlerini belirler.

- `ColorDepth`: **ImageList** nesnesi içerisindeki resimlerin renk derinliğini belirler.
- `Images`: **Images Collection Editor** penceresini görüntüler. Resim ekleme, silme, düzenleme işlemleri yapılabilir. Aynı pencereyi görüntülemek için **ImageList** nesnesinin sağ üst köşesinde bulunan kulakçığa tıklayarak gelen menüden **Choose Images** seçeneği kullanılabilir.

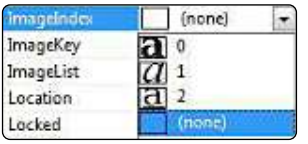




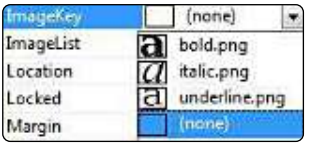
Bir **ImageList** oluşturulduktan sonra herhangi bir nesneye, **ImageList** nesnesini bağlamak için **ImageList** özelliğinden **ImageList** nesnesinin adını seçmek yeterlidir.



Daha sonra bağlantı kurduğumuz nesnenin **ImageIndex** özelliği ile resim dosyalarının index numaralarını seçebiliriz.



Aynı işlemi **ImageKey** özelliği ile de resim dosyalarının isimlerini seçerek gerçekleştirebiliriz.



Bu işlemi kodla şu şekilde gerçekleştiririz:

```
button1.ImageIndex = 0;
button2.ImageIndex = 1;
button3.ImageIndex = 2;
```

… veya…

```
button1.Image = imageList1.Images[0];
button2.Image = imageList1.Images[1];
button3.Image = imageList1.Images[2];
```



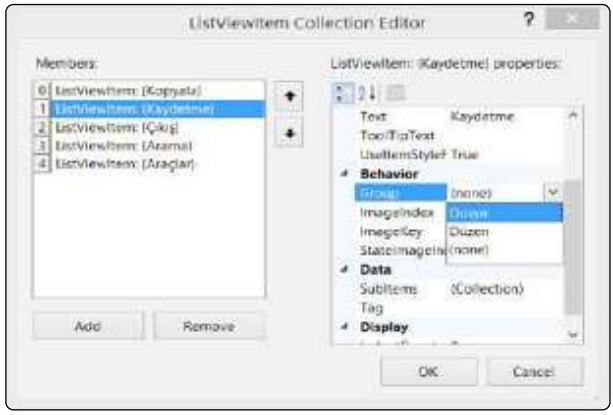
ListView Kontrolü

İçerisindeki değerlerin simgeler ile birlikte saklandığı bir kontroldür. **Form** üzerine eklenmeyip, **Visual Studio IDE** penceresinin alt tarafındaki panele eklenir.

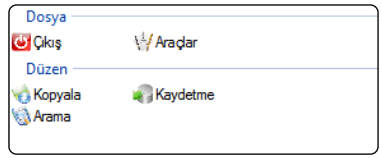
ListView Kontrolünün Özellikleri

- **`Items`: ListView** içerisindeki elemanları temsil eder. **`Items`** özelliğine tıklandığında karşımıza gelen **ListViewItem Collection Editor** yardımıyla **ListView** içerisine eleman ekleyip, silebilir ve mevcut elemanlar üzerinde düzenlemeler yapabiliriz.
- **`Columns`: ListView** içerisindeki sütunları temsil eder. **`Columns`** özelliğine tıklandığında karşımıza gelen **ColumnHeader Collection Editor** yardımıyla **ListView** içerisine sütun ekleyip, silebilir ve mevcut sütunlar üzerinde düzenlemeler yapabiliriz.
- **`Groups`: ListView** içerisindeki grupları temsil eder. Gruplar ortak özelliğe sahip elemanları bir arada listelemek amacıyla kullanılır. **`Groups`** özelliğine tıklandığında karşımıza gelen **ListViewGroup Collection Editor** yardımıyla **ListView** içerisine grup ekleyip, silebilir ve mevcut gruplar üzerinde düzenlemeler yapabiliriz. Grupları görüntülemek için **`ShowGroups`** özelliğinin

değeri `true` olarak belirlenmelidir. Grup eklendikten sonra `ListViewItem` **CollectionEditor** yardımıyla **Item** eklemesi yaptığımız pencerede, **Group** özelliği ile eklenmiş olan gruplardan birini seçerek, belirtilen **Item**'i istenilen grup altına ekleyebiliriz.

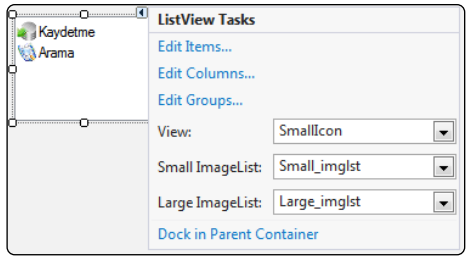


Örneğin; resimde görüldüğü gibi `Groups` özelliği ile **Dosya** ve **Düzen** adlı iki grup ekledik ve **ListViewItem CollectionEditor** yardımıyla `Item` eklemesi yaparken, **Item**'lerin her birini bir grup altına aldık.



NOT

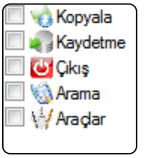
Properties Window içerisinde `Items`, `Columns` ve `Groups` özelliklerine tıklayarak düzenleme pencerelerini açabileceğimiz gibi **ListView**'ın sağ üst köşesindeki kulakçığa farenin sağ tuşu ile tıklayarak da aynı pencereleri görüntüleyebiliriz.



- `Sorting`: **ListView** içerisindeki değerlerin sıralı olarak listelenmesi amacıyla kullanılır. Aldığı değerler şunlardır:
  - `None`: Listeleme işleminin sırasız bir biçimde yapılmasını sağlar.

- **`Ascending`:** Listeleme işleminin küçükten büyüğe doğru artan bir şekilde yapılmasını sağlar.
- **`Descending`:** Listeleme işleminin büyükten küçüğe doğru azalan bir şekilde yapılmasını sağlar.

- **`CheckBoxes`: ListView** içerisinde görüntülenen elemanlarla birlikte **CheckBox** görüntülenmesini sağlar.



- **`LargeImageList`: ListView** içerisindeki simgelerin büyük resimler şeklinde görüntülenmesinde kullanılacak olan simgelerin saklanacağı **`imageList`** ismini belirtir.
- **`SmallImageList`: ListView** içerisindeki simgelerin küçük resimler şeklinde görüntülenmesinde kullanılacak olan simgelerin saklanacağı **ImageList** ismini belirtir.
- **`StateImageList`:** Seçim ya da onay durumları gibi işlemler için kullanılır. **`CheckBoxes`** özelliği ile birlikte kullanılır. **StateImageList** içerisindeki **`0`** ve **`1`** index numaralı simgeler **CheckBox** işlemleri için kullanılır. **`0`** nolu index değeri seçili olmayan **CheckBox**'ı, **`1`** nolu index değeri ise seçili olan **CheckBox**'ı temsil eder.
- **`View`: ListView** içerisindeki simgelerin görünüm şeklini belirler. Aldığı değerler şunlardır:
  - **`LargeIcon`:** Eğer **LargeImageList** ismi seçilmişse, seçilen **ImageList** içerisindeki simgelerin görüntülenmesini sağlar.
  - **`Details`:** Eğer **ListView** içerisinde sütun eklenmişse, **ListView** içerisindeki simgelerin sütunlar halinde görüntülenmesini sağlar.



Menü	Durum
Kopyala	Aktif
Kaydetme	Aktif
Çıkış	Aktif
Arama	Pasif
Araçlar	Pasif

ListViewItem CollectionEditor yardımıyla item eklemesi yaparken **`SubItems`** özelliğini kullanarak sütunlar içerisine değer eklemesi yapabiliriz.



- **`SmallIcon`:** Eğer **`SmallImageList`** ismi seçilmişse, seçilen **ImageList** içerisindeki simgelerin görüntülenmesini sağlar.



- **`List`:** Her bir elemanın açıklama ve simgesiyle beraber birer satırda olacak şekilde listelenmesini sağlar.
- **`Tile`:** Her bir elemanın büyük boyuttaki simge, açıklama ve **`subitem`** değerleriyle birlikte listelenmesini sağlar. Ancak bu özellik **`CheckBoxes`** özelliği ile birlikte kullanılamaz. Dolayısı ile bu özelliği kullanmak için **`CheckBoxes`** özelliği **`false`** olarak belirlenmelidir.



Örnek olarak belirtilen bir klasör içerisindeki klasör ve dosyaları **ListView** içerisinde listeleyen bir uygulama yapalım. Klasör ismini **Form** üzerindeki **TextBox**'dan alacağız. **ListView** içerisinde listelenen klasörler için bir simge, dosyalar için de bir simge belirleyelim. Resimleri, *projemizin bin/debug* klasörü altında **resimler** adında bir klasör oluşturarak içerisine kopyalayalım. Bu resimleri kodumuz içerisinde oluşturacağımız **ImageList** içerisine ekleyeceğiz. **Form**'uzun üzerine 1 adet **ListView**, 1 adet **Button**, 1 adet **Label**, 1 adet de **TextBox** ekliyoruz.

```
using System;
using System.Drawing;
using System.IO;
using System.Windows.Forms;
namespace listView_KlasorListeleme
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
  }
```

```
  private void button1_Click(object sender, EventArgs e)
  {
    ImageList imglist = new ImageList();
    imglist.Images.Add(Bitmap.FromFile("./resimler/Dosya.ico"));
    imglist.Images.Add(Bitmap.FromFile("./resimler/Klasor.ico"));
    listView1.SmallImageList = imglist;

    listView1.Columns.Add("İsim");
    listView1.Columns.Add("Boyut");
    listView1.Columns.Add("Tür");

    ListViewItem item;
    String[] klasorler = Directory.GetDirectories(textBox1.Text);
    int boyut = klasorler.Length;
    if (boyut != 0)
    {
   foreach (string aktifklasor in klasorler)
   {
     item = new ListViewItem(Path.GetFileName(aktifklasor), 1);
     item.SubItems.Add("");
     item.SubItems.Add("Klasör");
     listView1.Items.Add(item);
   }

   String[] dosyalar = Directory.GetFiles(textBox1.Text);
   boyut = dosyalar.Length;
   foreach (string aktifdosya in dosyalar)
   {
     item = new ListViewItem(Path.GetFileName(aktifdosya), 0);
     FileInfo dosya_bilgisi = new FileInfo(aktifdosya);

     item.SubItems.Add(Convert.ToString(dosya_bilgisi.Length));
     item.SubItems.Add("Dosya");
     listView1.Items.Add(item);
   }
  }
    listView1.View = View.Details;
  }
 }
}
```

Kodumuzun satırlarını açıklayalım…

```
ImageList imglist = new ImageList();
```

imglist isimli yeni bir **ImageList** nesnesi oluşturuyoruz.

```
imglist.Images.Add(Bitmap.FromFile("./resimler/Dosya.ico"));
```

ImageList içerisine, proje klasörü içerisindeki (*bin/debug*) resimler klasörü altında bulunan simge dosyasını ekliyoruz.

```
listView1.SmallImageList = imglist;
```

ListView içerisindeki simgelerin küçük resim boyutunda görüntülenmesinde kullanılacak olan **ImageList** ismini belirliyoruz.

```
listView1.Columns.Add("İsim");
```

ListView içerisine sütunları ekliyoruz. Dosya ve klasörlerin üç çeşit bilgisini (isim, boyut, tür) sütun olarak **ListView** içerisine ekleyerek bu bilgileri ilgili sütunlarda görüntüleyeceğiz.

```
ListViewItem item;
```

ListView içerisine ekleyeceğimiz elemanları belirtecek **item** adlı değişkenimizi tanımlıyoruz.

```
String[] klasorler = Directory.GetDirectories(textBox1.Text);
```

TextBox içerisine girilmiş olan klasör yolu altında bulunan klasörlerin listesini, **Directory.GetDirectories** metodu ile **klasorler** adlı dizi değişkeni içerisine aktarıyoruz.

```
int boyut = klasorler.Length;
```

klasorler değişkeninin boyutunu, **boyut** isimli değişkene **Length** özelliğini kullanarak aktarıyoruz.

```
if (boyut != 0)
```

boyut değişkeninin içeriğinin **0**'dan farklı olup olmadığını kontrol etmekle, **klasorler** değişkeni içerisine aktarılan klasör olup olmadığını kontrol ediyoruz.

```
foreach (string aktifklasor in klasorler)
```

klasorler değişkeni içerisinde bulunan klasör listesini okumak için **foreach** döngüsü kuruyoruz.

```
item = new ListViewItem(Path.GetFileName(aktifklasor), 1);
```

item değişkenine, okunan klasörün **Path.GetFileName** metodu ile ismini alarak, **ImageList** içerisindeki **1** index numarasına sahip resim ile birlikte aktarıyoruz.

```
item.SubItems.Add("");
```

ListView içerisinde klasör özelliklerini listelemede kullanılacak boş bir **SubItem** ekliyoruz. Dosya özelliklerini listelerken ve dosya boyutunu görüntülerken, klasör özelliklerinde dosya boyutunu boş bırakacağımız için boş bir **SubItem** ekledik.

```
item.SubItems.Add("Klasör");
```

ListView içerisinde listelenen elemanın "*Klasör*" olduğunu belirtmek için "*Klasör*" metnini **SubItem** olarak ekliyoruz.

```
listView1.Items.Add(item);
```

item nesnesini belirlediğimiz özellikleri ve **SubItem**'ler ile beraber **ListView** içerisine ekliyoruz.

```
String[] dosyalar = Directory.GetFiles(textBox1.Text);
```

textBox içerisine girilmiş olan klasör yolu altında bulunan dosyaların listesini, **Directory.GetFiles** metodu ile **dosyalar** adlı dizi değişkeni içerisine aktarıyoruz.

```
boyut = dosyalar.Length;
```

Dosyaların boyutunu, **boyut** isimli değişkene **Length** özelliğini kullanarak aktarıyoruz.

```
foreach (string aktifdosya in dosyalar)
```

dosyalar değişkeni içerisinde bulunan dosya listesini okumak için **foreach** döngüsü kuruyoruz.

```
item = new ListViewItem(Path.GetFileName(aktifdosya), 0);
```

item değişkenine, okunan dosyanın **Path.GetFileName** metodu ile ismini alarak, **ImageList** içerisindeki **0** index numarasına sahip resim ile birlikte aktarıyoruz.

```
item.SubItems.Add(Convert.ToString(dosya_bilgisi.Length));
```

ListView içerisinde **Length** özelliği ile bulunan dosya boyutunu listelemede kullanılacak bir **SubItem** ekliyoruz.

```
item.SubItems.Add("Dosya");
```

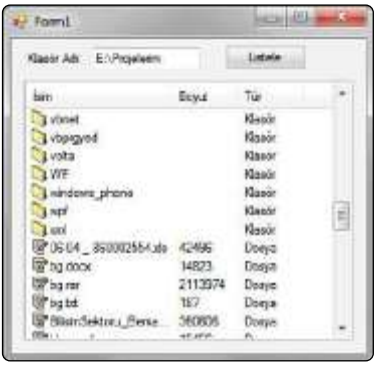
ListView içerisinde listelenen elemanın dosya olduğunu belirtmek için "*Dosya*" metnini **SubItem** olarak ekliyoruz.

```
listView1.Items.Add(item);
```

item nesnesini belirlediğimiz özellikleri ve **SubItem**'ler ile beraber **ListView** içerisine ekliyoruz.

```
listView1.View = View.Details;
```

ListView içerisindeki dosya ve klasörleri belirttiğimiz özellikleri ile beraber sütunlar halinde listeliyoruz.



Panel Kontrolü

GroupBox nesnesinde olduğu gibi nesneleri gruplamak için kullanılır.

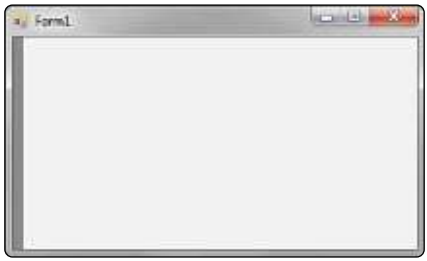
Örnek olarak otomatik olarak gizlenebilen bir **Panel** tasarlayalım. **Form** üzerine 1 adet **Panel**, içerisine de 5 adet **Button** ekleyelim.

Form içerisinde sola dayalı olarak bir **Panel** hazırlayıp içerisinde butonlar ekleyelim.

Kodumuz aşağıdaki gibi olacak:

```
int konum;
private void panel1_MouseLeave(object sender, EventArgs e)
{
 panel1.Left = konum - 90;
}
private void panel1_MouseMove(object sender, MouseEventArgs e)
{
 panel1.Left = konum;
}
private void Form1_Load(object sender, EventArgs e)
{
 konum = panel1.Left;
 panel1.Left = panel1.Left - 90;
}
```

Farenin imleci **Panel** üzerine gelmeden önce, panelimizin küçük bir kısmı görünür durumdadır.





Farenin imlecini **Panel** üzerine getirdiğimizde ise, panel tam olarak görüntülenecektir.

Form'un **`Load`** olayı ile form ilk yüklendiğinde

```
konum = panel1.Left;
```

… satırı ile **Panel**'in yatay konumunu konum adlı değişkene aktarıyoruz.

```
panel1.Left = panel1.Left - 90;
```

Panel'in yatay konumunu 90 pixel azaltarak gizlenmesini sağlıyoruz.

Panel'in **`MouseMove`** olayı ile panel üzerine farenin imleci geldiğinde…

```
panel1.Left = konum;
```

… **Panel**'in yatay konumuna, konum değişkeninde saklanan değeri aktarıyoruz.

Panel'in **`MouseLeave`** olayı ile panel üzerinden farenin imleci ayrıldığında…

```
panel1.Left = konum - 90;
```

… **Panel**'in yatay konumunu 90 pixel azaltarak yeniden gizlenmesini sağlıyoruz.

`Dock` özelliğini kullanarak panelin, **Form** üzerinde istenilen bir kenara otomatik olarak yerleştirilmesini sağlayabiliriz.

```
Panel İsmi.Dock=DockStyle.Özellik İsmi
```

… biçiminde kullanılır.

DockStyle özelliği `Bottom`, `Fill`, `Left`, `None`, `Right`, `Top` değerlerini alır.

- `Bottom`: **Panel**'i **Form**'un alt tarafına yerleştirmek için kullanılır.
- `Fill`: **Panel**'i **Form**'un boyutuna genişletmek için kullanılır.
- `Left`: **Panel'**i **Form**'un sol tarafına yerleştirmek için kullanılır.
- `Right`: **Panel**'i **Form**'un alt tarafına yerleştirmek için kullanılır.
- `Top`: **Panel**'i **Form**'un üst tarafına yerleştirmek için kullanılır.
- `None`: **Panel**'i **Form**'un üzerindeki yerleşimini iptal etmek için kullanılır.



Örneğin; yukarıdaki örneğimizdeki **Panel**'i formun sağ kenarına hizalamak için…

```
panel1.Dock = DockStyle.Right;
```

… şeklinde kodumuzu yazarız.

`Dock` işlemi sırasında **Panel** boyutunun, **Panel** içeriğine göre otomatik olarak ayarlanması gerektiği durumlarda `AutoSize` özelliğini `true` yapabilirsiniz.

FlowLayoutPanel Kontrolü

İçerisindeki nesneleri gruplandırma amaçlı olarak kullanılan bir çeşit **Panel** kontrolüdür. Farklı olarak `FlowLayoutPanel` içerisindeki nesneler, panel'in yerleşimine göre otomatik olarak yerleştirilebilir.

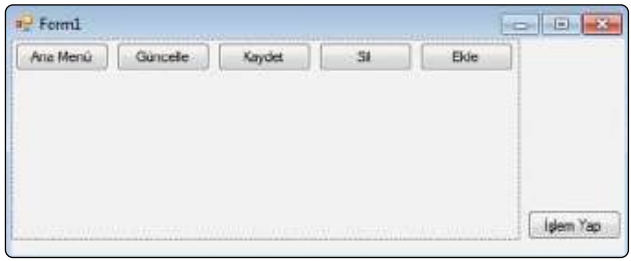
FlowLayoutPanel Kontrolünün Özellikleri

`FlowDirection`: `FlowLayoutPanel` içerisindeki nesnelerin yerleşim düzenini belirler. Varsayılan olarak `LeftToRight` değerini alır.

Aşağıdaki değerleri alır:

- **`BottomUp`**: **`FlowLayoutPanel`** içerisindeki nesneler dikey olarak **Form**'un alt tarafından üst tarafına yerleşecek biçimde yerleştirilir.
- **`LeftToRight`**: **`FlowLayoutPanel`** içerisindeki nesneler yatay olarak **Form**'un solundan sağına doğru yerleşecek biçimde yerleştirilir.
- **`RightToLeft`**: **`FlowLayoutPanel`** içerisindeki nesneler yatay olarak **Form**'un sağından soluna doğru yerleşecek biçimde yerleştirilir.
- **`TopDown`**: **`FlowLayoutPanel`** içerisindeki nesneler dikey olarak **Form**'un üst tarafından alt tarafına yerleşecek biçimde yerleştirilir.

Örnek olarak **Form** üzerine **`FlowLayout`** **Panel**'i içerisindeki butonlarla beraber aşağıdaki gibi yerleştirelim.



Yerleşim şeklini yukarıdan aşağıya doğru olacak biçimde değiştirmek için kodumuzu aşağıdaki gibi yazıyoruz:

```
private void btnislem_Click(object sender, EventArgs e)
{
 flowLayoutPanel1.FlowDirection = FlowDirection.TopDown;
}
```

`FlowLayoutPanel` yerleşimini yukarıdan aşağıya doğru olacak biçimde değiştirmek için **`FlowDirection`** özelliğini **`TopDown`** olarak belirliyoruz.



- **`WrapContents`:** **`FlowLayoutPanel`** içerisindeki nesnelerin **`FlowLayoutPanel`** boyutuna uygun olarak yerleşim biçimlerinin düzenlenmesini sağlar. Varsayılan olarak **`True`** değerini alır.



Yerleşim biçimini otomatik olarak değiştirmek için öncelikle **Properties** penceresinden **`WrapContents`** özelliğini **`False`** olarak belirleyerek, **Form** üzerine **`FlowLayout`** panelimizi, içerisindeki butonlarla beraber, yandaki şekilde olduğu gibi yerleştiriyoruz. Görüldüğü gibi **Panel** genişliğini butonlara göre daha dar yaparak, butonların bir kısmının görünmemesini sağladık.

Kodumuzu aşağıdaki gibi yazıyoruz:

```
private void btnislem_Click(object sender, EventArgs e)
{
 flowLayoutPanel1.WrapContents = true;
}
```



Kodumuzda, **`FlowLayoutPanel`** içerisindeki nesnelerin yerleşiminin **Panel** boyutuna göre otomatik olarak belirlenmesi için **`WrapContents`** özelliğinin değerini **`True`** olarak belirliyoruz. Görüldüğü **Panel** içerisine sığmayan butonlar da görüntülenerek otomatik olarak yerleştirildi.

- **`AutoScroll`:** **`FlowLayoutPanel`** içerisine sığmayan nesneleri görüntülemek için kaydırma çubuklarının görüntülenmesini sağlar. Varsayılan olarak **`False`** değerini alır.

Örnek olarak **`FlowLayoutPanel`** içerisindeki butonların bir kısmı **Panel** sınırları dışında kalacak biçimde düzenleme yaparak **Form** üzerine ekleyelim. **`WrapContents`** özelliğini **Properties** penceresinden **`False`** olarak belirleyelim.

Kodumuzu aşağıdaki gibi yazıyoruz:

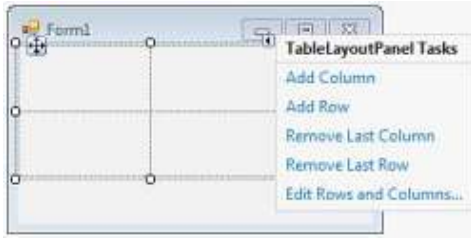
```
private void btnislem_Click(object sender, EventArgs e)
{
 flowLayoutPanel1.AutoScroll = true;
}
```

Kodumuzda kaydırma çubuklarını görüntülemek için `AutoScroll` özelliğine `true` değerini atadık. Uygulamamızı çalıştırdığımızda görüldüğü gibi **Panel** sınırları dışında kalan butonları görüntülemek için panel içerisine kaydırma çubuğu eklendi.



TableLayoutPanel Kontrolü

İçerisindeki nesnelerin tablo içerisine eklenen satır ve sütunlar şeklinde eklenebildiği bir çeşit panel kontrolüdür. **TableLayoutPanel** üzerinde yer alan kulakçığa tıklandığında aktifleşen **TableLayoutPanel Tasks** menüsü altındaki…

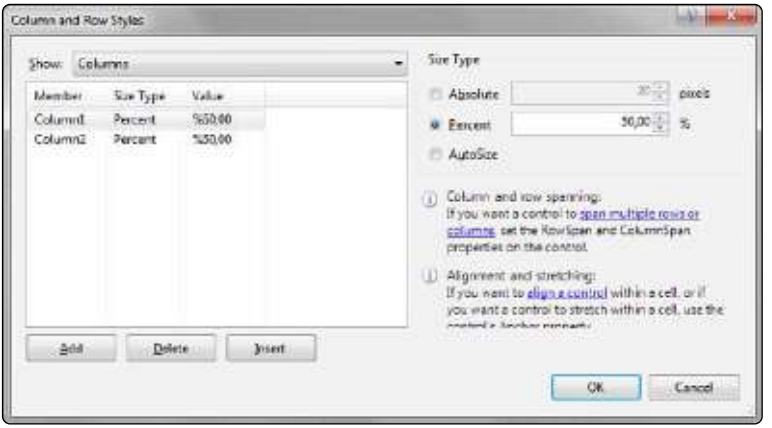


- **Add Column** seçeneği ile sütun ekleme,
- **Add Row** seçeneği ile satır ekleme,
- **Remove Last Column** seçeneği ile tablonun son sütununu silme,
- **Remove Last Row** seçeneği ile tablonun son satırını silme,
- **Edit Rows and Columns** seçeneği ile ise tabloda eklenmiş olan satır ve sütunlar üzerinde düzenleme işlemleri yapılabilir.

TableLayoutPanel içerisine ekleyeceğiniz nesnelerin her biri ayrı ayrı birer hücreye eklenecektir.



Tablodaki satır ve sütunları düzenleme penceresine **Edit Rows and Columns** seçeneğini kullanarak ulaşabileceğiniz gibi **Properties Window** içerisinde **Rows** ya da **Colums** özellikleri yardımıyla da ulaşabilirsiniz. Bu penceredeki **Add** seçeneği ile yeni bir satır ya da sütun ekleyebilir, **Delete** seçeneği ile silebilirsiniz. **Insert** seçeneği ise **Add** seçeneğinden farklı olarak en son satır ya da sütun olarak değil, istediğiniz konuma ekleme yapmanızı sağlar.



Size Type bölümünde satır ve sütunların ölçüleri belirlenir. Aldığı değerler şunlardır:

- **`Absolute`:** Her bir satır ya da sütunun boyutu birbirinden bağımsız olarak pixel cinsinden belirlenir.
- **`Percent`:** Her bir satır ya da sütun boyutu yüzde oranı olarak belirlenerek satır ya da sütunlar arasında genişlik yüzde olarak paylaştırılır. (Örneğin; tabloda üç satır varsa %25, %50, %25 olarak satırlar ölçeklendirilebilir.)



- **`AutoSize`:** Satır ya da sütunların genişlikleri içeriğe göre otomatik olarak belirlenir.

TableLayoutPanel içerisindeki nesneler **Controls** koleksiyonu içerisinde saklanır. **Controls** koleksiyonunun **Add** metodunu kullanarak panel içerisine ekleme yapabilirsiniz.

`Controls.Add` metodu…

```
Controls.Add(Kontrol Adı, sütun numarası, satır numarası)
```

… şeklinde kullanılır.

Örnek olarak bir **TableLayoutPanel** nesnesini kodla oluşturarak içerisine 2 adet **Label**, 2 adet de **TextBox** nesnesi ekleyelim. Kodumuzu **Form**'un **`Paint`** olayı içerisine yazarak kodumuzun, **Form** üzerindeki kontroller ekrana yerleştirilmeden önce çalışmasını sağlıyoruz.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
 TableLayoutPanel tlp = new TableLayoutPanel();
 tlp.Location = new System.Drawing.Point(10,10);
 tlp.Name = "TableLayoutPanel1";
 tlp.Size = new System.Drawing.Size(200, 100);
 tlp.BackColor = Color.DarkGray;
 tlp.ColumnCount = 2;
 tlp.RowCount = 2;
 tlp.ColumnStyles.Add(new ColumnStyle(SizeType.Absolute, 90F));
 tlp.ColumnStyles.Add(new ColumnStyle(SizeType.Absolute, 90F));
 tlp.RowStyles.Add(new RowStyle(SizeType.Percent, 50F));
 tlp.RowStyles.Add(new RowStyle(SizeType.Percent, 50F));

 Label label1 = new Label();
 label1.Text = "Kullanıcı Adı:";
 Label label2 = new Label();
 label2.Text = "Şifre:";
 TextBox textBox1 = new TextBox();
 TextBox textBox2 = new TextBox();

 tlp.Controls.Add(label1, 0, 0);
 tlp.Controls.Add(textBox1, 1, 0);
 tlp.Controls.Add(label2, 0, 1);
 tlp.Controls.Add(textBox2, 1, 1);
 this.Controls.Add(tlp);
}
```

Kodumuzun satırlarını açıklayalım…

```
TableLayoutPanel tlp = new TableLayoutPanel();
```

tlp isimli yeni bir **TableLayoutPanel** nesnesi oluşturuyoruz.

```
tlp.Location = new System.Drawing.Point(10,10);
```

Location özelliği ile **TableLayoutPanel** nesnesinin **Form** üzerindeki **x** ve **y** koordinatlarını belirliyoruz.

```
tlp.Size = new System.Drawing.Size(200, 100);
```

Size özelliği ile **TableLayoutPanel** nesnesinin genişlik ve yüksekliğini belirliyoruz.

```
tlp.BackColor = Color.DarkGray;
```

BackColor özelliği ile **TableLayoutPanel** nesnesinin zemin rengini koyu gri (**DarkGray**) olarak belirliyoruz.

```
tlp.ColumnCount = 2;
```

ColumnCount özelliği ile **TableLayoutPanel** nesnesinin sütun sayısını **2** olarak belirliyoruz.

```
tlp.RowCount = 2;
```

RowCount özelliği ile **TableLayoutPanel** nesnesinin satır sayısını **2** olarak belirliyoruz.

```
tlp.ColumnStyles.Add(new ColumnStyle(SizeType.Absolute, 90F));
```

ColumnStyles.Add metodu ile **TableLayoutPanel** nesnesinin sütun özelliğine yeni bir stil tanımlaması yaparak ekleme yapıyoruz. **SizeType.Absolute**, **90F** şeklinde tanımladığımız özellik ile sütun genişliğini **90** olarak belirliyoruz.

```
tlp.RowStyles.Add(new RowStyle(SizeType.Percent, 50F));
```

RowStyles.Add metodu ile **TableLayoutPanel** nesnesinin satır özelliğine

yeni bir stil tanımlaması yaparak ekleme yapıyoruz. **`SizeType.Percent, 50F`** şeklinde tanımladığımız özellik ile satır genişliğini **%50** olarak belirliyoruz.

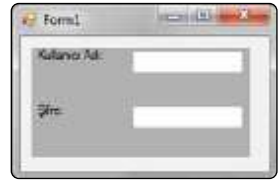
```
tlp.BackColor = Color.DarkGray;
```

`BackColor` özelliği ile **TableLayoutPanel** nesnesinin zemin rengini koyu gri (**`DarkGray`**) olarak belirliyoruz.

Daha sonra **Label** ve **TextBox** nesnelerini tanımlıyor ve…

```
tlp.Controls.Add(label1, 0, 0);
```

… satırında olduğu gibi **`Controls.Add`** metodu ile **TableLayoutPanel** üzerine ekliyoruz.



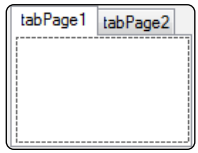
TabControl Nesnesi

Kontrolleri, sekmelerden oluşan paneller içerisinde görüntülemek amacıyla kullanılır.

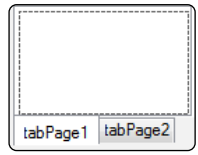
TabControl Kontrolünün Özellikleri

- **`Alignment`**: **TabControl** nesnesi içerisindeki sekmelerin hizalanma şeklini belirler. Şu değerleri alır:

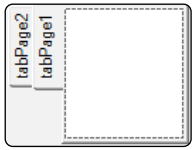
Top:



Bottom:



Left:



Right:

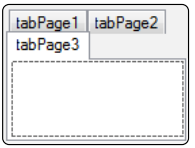


- **`DrawMode`**: **TabControl** nesnesindeki sekmeler üzerinde çizim olayları gerçekleştirilebilmesini sağlar. Ancak bunun için **`DrawMode`** özelliği **`OwnerDrawFixed`** olarak değiştirilmelidir.
- **`HotTrack`**: Farenin imleci sekmeler üzerine geldiğinde sekme biçimlerinin değişmesini sağlar.

- **`ImageList`: TabControl** nesnesindeki sekmelerde görüntülenecek olan resimlerin saklandığı **ImageList** nesnesini belirtir.
- **`ItemSize`:** Sekme boyutlarını belirler.
- **`Multiline`: TabControl** nesnesindeki sekmelerin birden fazla satırda görüntülenmesini sağlar.

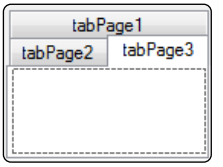
Yandaki görünümde **`MultiLine`** özelliği **`false`** yapılmıştır.

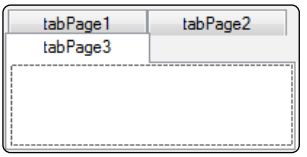




Soldaki görünümde ise **`MultiLine`** özelliği **`true`** yapılmıştır:

- **`TabCount`:** Sekme sayısını belirler.
- **`SizeMode`:** Sekmelerin boyutlandırma biçimini belirler. Şu değerleri alır:
  - **`Normal`:** Sekme boyutları farklı olarak belirlenebilir.
  - **`FillToRight`:** Sekmeler kenar boşlukları bırakılmayacak biçimde yerleştirilir.
  - **`Fixed`:** Sekme genişlikleri eşit olacak şekilde belirlenir.





TabControl Kontrolünün Metodları

- **`SelectedIndex`:** Index değeri verilen sekmeyi seçmek amacı ile kullanılır.

```
tabControl1.SelectedIndex = 1;
```

`SelectedTab`: Adı belirtilen sekmeyi seçmek amacı ile kullanılır.

```
tabControl1.SelectedTab = TabPage2;
```

- **`TabPages`:** Bir **TabControl** nesnesi içerisindeki sekme listesini döndürür.

TabControl Nesnesi İçerisine Sekme Eklemek ve Silmek



TabControl nesnesine farenin sağ tuşu ile tıklanarak gelen menüden **Add Tab** ve **Remove Tab** eçenekleri ile sekme eklenip silme işlemi gerçekleştirilebilir.

Ya da **TabControl** nesnesinin sağ üst köşesinde bulunan kulakçığa tıklanarak gelen menüden **Add Tab** ve **Remove Tab** seçenekleri sekmeler eklenebilir ya da silinebilir.



Bir değer yöntem ise **TabControl** nesnesi seçili iken, **Properties Window** içerisinde **TabPages** özelliğinden **Collection** seçeneğine tıklayarak **PageCollectionEditor** penceresini kullanmaktır.



Kod yolu ile yeni bir sekme eklemek için `TabPages.Add` metodu kullanılabilir.

```
tabControl1.TabPages.Add("tabPage1");
```

Kod yolu ile bir sekmeyi silmek için **`TabPages.Remove`** ve **`TabPages.RemoveAt`** metodları kullanılabilir. **`TabPages.Remove`** metodu ile sekmenin adı verilerek silme işlemi yapılırken **`TabPages.RemoveAt`** metodu ile ise sekmenin index numarası verilerek silme işlemi yapılır.

```
tabControl1.TabPages.Remove(tabPage2);
tabControl1.TabPages.RemoveAt(2);
```

Örnek olarak belirtilen bir klasör içerisindeki resimleri listeleyen bir uygulama yapalım. Klasör ismini **Form** üzerindeki **TextBox**'dan alacağız. Belirtilen klasör yolu altında bulunan her bir klasör için bir sekme ve her bir sekme altında bir **ListView** oluşturulacak. **ListView** içerisinde listelenenecek resimler önce bir **ImageList**'e oradan **ListView**'e aktarılacak. **Form**'uzun üzerine 1 adet **TabControl**, 1 adet **buton**, 1 adet **Label**, 1 adet de **TextBox** ekliyoruz.

```
using System;
using System.Drawing;
using System.IO;
using System.Windows.Forms;
namespace tabControl_ResimListeleme
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
  }
  private void btnListele_Click(object sender, EventArgs e)
  {
    int i = 0;
    string[] klasorler = Directory.GetDirectories(textBox1.Text);
    if (klasorler.Length != 0)
    {
   foreach (string aktifklasor in klasorler)
   {
     i++;
     TabPage tp = new TabPage("tp" + i);
     tabControl1.TabPages.Add(tp);

     ListView lstv = new ListView();
```

```
     lstv.Size = new System.Drawing.Size(520, 330);
     lstv.View = View.LargeIcon;

     ImageList imgl = new ImageList();
     imgl.ImageSize = new Size(128, 128);

     tabControl1.TabPages[i].Text = Path.GetFileName(aktifklasor);

     string[] dosyalar = Directory.GetFiles(aktifklasor, "*.jpg",
SearchOption.AllDirectories);
     imgl.Images.Clear();
     foreach (string aktifdosya in dosyalar)
     {
      imgl.Images.Add(Image.FromFile(aktifdosya));
     }
     for (int j = 0; j < imgl.Images.Count; j++)
     {
      ListViewItem item = new ListViewItem();
      item.ImageIndex = j;
      lstv.Items.Add(item);
     }
     lstv.LargeImageList = imgl;
     tabControl1.TabPages[i].Controls.Add(lstv);
   }
    }
    tabControl1.TabPages.RemoveAt(0);
  }
 }
}
```

Kodumuzun satırlarını açıklayalım…

```
String[] klasorler = Directory.GetDirectories(textBox1.Text);
```

TextBox içerisine girilmiş olan klasör yolu altında bulunan klasörlerin listesini, **`Directory.GetDirectories`** metodu ile **`klasorler`** adlı dizi değişkeni içerisine aktarıyoruz.

```
if (klasorler.Length != 0)
```

`Length` özelliği ile **`klasorler`** değişkeninin boyutunu bularak, boyutun 0'dan

farklı olup olmadığını kontrol etmekle belirtilen **klasorler** değişkeni içerisine, klasör aktarılıp aktarılmadığını kontrol ediyoruz.

```
foreach (string aktifklasor in klasorler)
```

klasorler değişkeni içerisinde bulunan klasör listesini okumak için **foreach** döngüsü kuruyoruz.

```
TabPage tp = new TabPage("tp" + i);
```

tp isimli yeni bir sekme (**tabPage**) tanımlıyoruz.

```
tabControl1.TabPages.Add(tp);
```

Tanımladığımız sekmeyi **TabPages.Add** metodu ile **TabControl** içerisine ekliyoruz.

```
ListView lstv = new ListView();
lstv.Size = new System.Drawing.Size(520, 330);
```

lstv isimli yeni bir **ListView** tanımlayarak, genişlik ve yüksekliğini belirliyoruz.

```
lstv.View = View.LargeIcon;
```

ListView içerisinde resimleri görüntüleyeceğimiz için **ListView** içerisindeki elemanların büyük simgeler halinde görüntülenmesi amacıyla, **View** özelliğini **LargeIcon** olarak belirliyoruz.

```
ImageList imgl = new ImageList();
imgl.ImageSize = new Size(128, 128);
```

… **imgl** isimli yeni bir **ImageList** tanımlayarak, genişlik ve yüksekliğini belirliyoruz.

```
tabControl1.TabPages[i].Text = Path.GetFileName(aktifklasor);
```

TabControl içerisindeki sekmelerin başlıklarını döngü içerisinde okunan klasörlerin ismi olarak belirliyoruz.

```
string[] dosyalar = Directory.GetFiles(aktifklasor,
"*.jpg", SearchOption.AllDirectories);
```

TextBox içerisine girilmiş olan klasör yolu altında bulunan dosyaların listesini, **`Directory.GetFiles`** metodu ile **`dosyalar`** adlı dizi değişkeni içerisine aktarıyoruz. Aktarılacak dosyaların türünü "*jpg*" oalrak belirliyoruz. Arama işleminin alt klasörlerde de yapılması için **`SearchOption`** özelliğini **`AllDirectories`** olarak belirliyoruz. Alt klasörlerde arama yapılmaması için **`SearchOption`** özelliğini **`TopDirectoryOnly`** olarak belirlememiz gerekmektedir.

```
imgl.Images.Clear();
```

ImageList içerisinde daha önceden mevcut olan resim varsa temizliyoruz. Böylece her bir klasör içerisinde mevcut olan resimler üzerinde yapılan işlemler bittikten sonra, yeni bir klasöre geçmeden önce **ImageList** içerisini temizliyoruz.

```
foreach (string aktifdosya in dosyalar)
```

`dosyalar` değişkeni içerisinde bulunan dosya listesini okumak için **`foreach`** döngüsü kuruyoruz.

```
imgl.Images.Add(Image.FromFile(aktifdosya));
```

ImageList içerisine, döngü içerisinde okunan dosyanın ismini belirterek aktarıyoruz.

```
for (int j = 0; j < imgl.Images.Count; j++)
```

ImageList içerisindeki resimleri okumak için bir döngü kuruyoruz. Döngümüz **ImageList** içerisindeki resimlerin sayısı kadar (**`Images.Count`**) tekrarlanacak.

```
ListViewItem item = new ListViewItem();
```

Her bir resim dosyasını görüntülemede kullanılacak yeni bir **ListViewItem** nesnesi tanımlıyoruz.

```
item.ImageIndex = j;
```

Tanımladığımız **ListViewItem** nesnesinin index numarasını döngü değişkenine eşitleyerek, otomatik olarak artmasını sağlıyoruz. Bu şekilde **ImageList** içerisindeki resim dosyaları index numarasına göre sıra ile okunacaktır.

```
lstv.Items.Add(item);
```

... **item** nesnesini belirlediğimiz resim dosyası ile beraber **ListView** içerisine ekliyoruz.

```
lstv.LargeImageList = imgl;
```

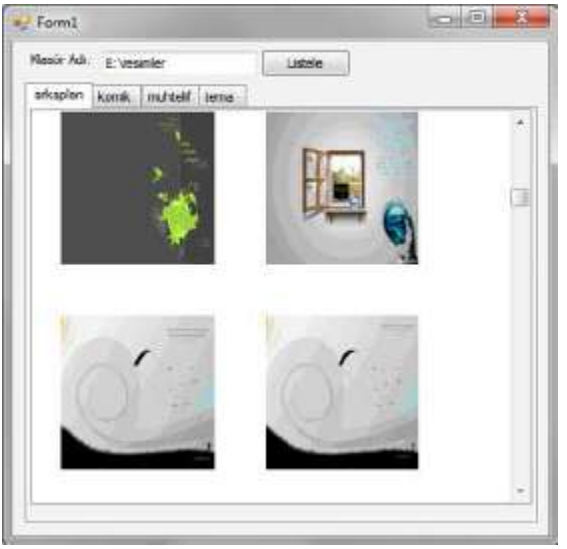
ListView içerisindeki resimlerin büyük simge görünümünde görüntülenmesi için kullanılacak **ImageList** ismini belirtiyoruz.

```
tabControl1.TabPages[i].Controls.Add(lstv);
```

TabControl sekmesi içerisinde **ListView** nesnesini ekliyoruz.

```
tabControl1.TabPages.RemoveAt(0);
```

Formumuzun üzerine **TabControls** nesnesini ilk olarak eklediğimizde boş olarak eklememek için boş bir sekme eklemiştik. Bu satırda ilk index değerine sahip bu boş sekmeyi siliyoruz.



Timer Kontrolü

Zamanlama işlemleri için kullanılır. .Net içerisinde 3 farklı sınıfı kullanarak zamanlama işlemi gerçekleştirilebilir.

```
System.Windows.Forms.Timer
System.Timers.Timer
System.Threading.Timer
```

Timer kontrolü **`System.Windows.Forms.Timer`** sınıfına ait bir kontroldür.

Timer Kontrolünün Özellikleri

- **`Interval`**: Zaman aralığını belirtir. Milisaniye cinsinden değer alır. Örneğin; 1000 değerini aldığında, bu 1 saniye demektir. Alabileceği maksimum değer **`Int32.MaxValue`** değeridir. Bu da 2.147.483.647 rakamına karşılık gelmektedir.
- **`Enabled`**: **`True`** değerini aldığında **`Timer`** nesnesi aktif olurken, **`False`** değerini aldığında pasif olur.

Timer Kontrolünün Metodları

- **`Enabled`**: **`True`** değerini aldığında Timer nesnesini aktif yapar. **`False`** değerini aldığında ise pasif yapar
- **`Start`**: Timer nesnesini başlatır.
- **`Stop`**: Timer nesnesini durdurur.

NOT

Timer nesnesi **`enabled=false`** seçeneği ile pasif yapıldıktan sonra, **`enabled=true`** seçeneği ile yeniden aktif yapıldığında sayma işleminde ilk başa dönülür. Ancak **`stop`** ile durdurulup, **`start`** ile tekrar başlatılırsa sayma işlemine kalınan yerden devam edilir.

Timer Kontrolünün Olayları

- **`Tick`**: **`Interval`** özelliğinde belirlenen zaman aralığına göre zamanlananan olaylar **`Tick`** olayı içerisinde gerçekleşir.

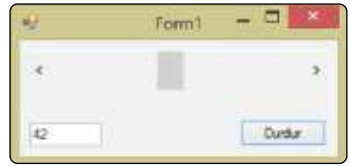
Form üzerindeki bir yatay kaydırma çubuğunu sağdan sola, soldan sağa sürekli olarak gidip gelecek şekilde hareket ettiren bir örnek uygulama yapalım.

Properties penceresinden **ScrollBar** nesnesinin **`value`** değerini **`1`**, **`minumum`** değerini **`1`**, **`maksimum`** değerini **`100`** olarak, **Timer** nesnesinin **`interval`** özelliğini **`100`**, **`enabled`** özelliğini **`False`** olarak ayarlayalım.

```
Boolean son=false;
private void Form1_Load(object sender, EventArgs e)
{
 timer1.Interval = hScrollBar1.Value;
}
private void hScrollBar1_Scroll(object sender, ScrollEventArgs e)
{
 timer1.Interval = hScrollBar1.Value;
 if (hScrollBar1.Value == hScrollBar1.Maximum) son = true;
}
private void timer1_Tick(object sender, EventArgs e)
{
 if (son == false)
 {
  textBox1.Text = hScrollBar1.Value.ToString();
  if (hScrollBar1.Value == 100)
  {
   son = true;
   return;
  }
  hScrollBar1.Value = hScrollBar1.Value + 1;
 }
 else
 {
  textBox1.Text = hScrollBar1.Value.ToString();
  if (hScrollBar1.Value == 1)
  {
   son = false;
   return;
  }
  hScrollBar1.Value = hScrollBar1.Value - 1;
 }
}
private void button1_Click(object sender, EventArgs e)
{
 if (button1.Text == "Başla")
 {
  button1.Text = "Durdur";
  timer1.Enabled = true;
```

```
 }
 else
 {
  button1.Text = "Başla";
  timer1.Enabled = false;
 }
 }
}
```

Kaydırma çubuğumuz soldan sağa doğru ilerleyecek, en sağa geldiğinde ise bu defa sağdan sola doğru ilerleyecektir. Bu esnada kaydırma çubuğunun o anki değeri **TextBox** nesnesi içerisinde görüntülenecektir.



`HScrollBar1_Scroll` olayı içerisinde…

```
timer1.Interval = hScrollBar1.Value;
```

… satırında **`timer1`**'in **`interval`** değerini kaydırma çubuğunun değerine bağlayarak, **`interval`** arttıkça ya da azaldıkça kaydırma çubuğunun da ilerlermesini sağlıyoruz.

```
if (hScrollBar1.Value == hScrollBar1.Maximum) son = true;
```

… satırında kaydırma çubuğunun değeri kaydırma çubuğunun maksimum değerine ulaştığında, kaydırma çubuğunun son noktasına geldiğini anlayarak son değişkeninin değerine **`true`** değerini aktarıyoruz.

```
Timer1_Tick olayı içerisinde…
if (son == false)
```

… satırı ile kaydırma çubuğunun maksimum değere ulaşmadığının kontrolünü yapar.

```
if (hScrollBar1.Value == 100)
```

Eğer kaydırma çubuğunun değeri **`100`** oldu ise (100 değeri bizim kaydırma çubuğu için belirlediğimiz Maximum değeridir)…

```
son = true;
return;
```

… son değişkeninin değerini **`true`** yaparak alt programdan çıkış yapıyoruz.

Eğer kaydırma çubuğunun değeri henüz `100` değerine ulaşmamışsa, kaydırma çubuğunun değerini `1` arttırıyoruz.

```
hScrollBar1.Value = hScrollBar1.Value + 1;
```

Son değişkenin değeri **`true`** olduğunda kaydırma çubuğunu geriye doğru döndürmek için ise…

```
hScrollBar1.Value = hScrollBar1.Value - 1;
```

… satırını kullanıyoruz.

```
if (button1.Text == "Başla")
```

Eğer butona tıklandığında, başlığı "*Başla*" ise…

```
button1.Text = "Durdur";
timer1.Enabled = true;
```

… butonun başlığını "*Durdur*" olarak değiştirerek, **Timer**'ı aktifleştiriyoruz. Böyle **ScrollBar** çalışmaya başlıyor.

Eğer butona tıklandığında, başlığı "*Durdur*" ise…

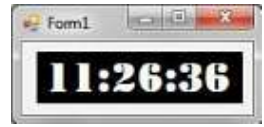
```
button1.Text = "Başla";
timer1.Enabled = false;
```

… butonun başlığını "*Başla*" olarak değiştirerek, **Timer**'ı pasif yapıyoruz. Böyle **ScrollBar** duruyor.

Basit bir saat uygulaması yapalım.

Form üzerine 1 adet **label** nesnesi ile 1 adet de **Timer** nesnesi ekliyoruz. Timer nesnesinin **`interval`** özelliğini `1000` ve **`Enabled`** özelliğini **`True`** olarak belirliyoruz.

```
private void Timer1_Tick(object sender, EventArgs e)
{
 Label1.Text = string.Format("{0:hh:mm:ss}", DateTime.Now);
}
```



`DateTime.Now` ile o anki zamanı görüntületip **Label** nesnesine aktarıyoruz. Bu işlem saniyede bir yapıldığı için, o anki zaman her saniye güncellenerek kullanıcıya gösteriliyor. Saat, dakika, saniye biçiminde görüntülenmesi için format özelliğini **`{0:hh:mm:ss}`** olarak belirliyoruz.

Bir kronometre örneği için **Form** üzerine 2 adet **Button**, 2 adet **Label**, 1 adet **TextBox**, 1 adet de **Timer** nesnesi ekleyelim. Timer nesnesinin **`interval`** değeri **`1000`**, **`Enabled`** özelliği **`False`** olacak.

```
int saat,dakika,saniye;
DateTime zaman,kalınanzaman;
Boolean duraklama = false;
private void btnBasla_Click(object sender, EventArgs e)
{
 if (btnBasla.Text == "Başla")
 {
  btnBasla.Text = "Dur";
  if (duraklama == false)
  {
   zaman = Convert.ToDateTime(textBox1.Text);
   saat = zaman.Hour;
   dakika = zaman.Minute;
   saniye = zaman.Second;
  }
  else
  {
   zaman = kalınanzaman;
   saat = zaman.Hour;
   dakika = zaman.Minute;
   saniye = zaman.Second;
  }
  timer1.Start();
  }
```

```
        else
        {
         zaman = Convert.ToDateTime(label1.Text);
         btnBasla.Text = "Başla";
         timer1.Stop();
         duraklama = true;
         kalınanzaman = Convert.ToDateTime(label1.Text);
        }
       }
       private void timer1_Tick(object sender, EventArgs e)
       {
        label1.Text = ((Convert.ToString(saat) + ":") +
          (Convert.ToString(dakika) + ":") + Convert.ToString(saniye));
        if ((saniye == 0) && (dakika > 0))
        {
         dakika = dakika - 1;
         saniye = 59;
        }
        else if ((saniye == 0) && (dakika == 0) && (saat > 0))
        {
         saniye = 59;
         dakika = 59;
         saat = saat - 1;
        }
        if ((dakika == 0) && (saat > 0))
        {
         dakika = 59;
         saat = saat - 1;
        }
        if ((saat == 0) && (dakika == 0) && (saniye == 0))
        {
         timer1.Enabled = true;
         MessageBox.Show("Süre Bitti");
        }
        saniye = saniye - 1;
       }
       private void btnBitir_Click(object sender, EventArgs e)
       {
        timer1.Enabled = false;
        label1.Text = textBox1.Text;
       }
```

Kodumuzda…

```
zaman = Convert.ToDateTime(textBox1.Text);
```

…satırında **`Convert.ToDateTime`** metodu ile **TextBox** içerisine girilen değerleri **Tarih-Zaman** türüne dönüştürüyoruz.

```
saat = zaman.Hour;
dakika = zaman.Minute;
saniye = zaman.Second;
```

… satırları ile de **TextBox** içerisine girilmiş zaman değerini saat, dakika, saniye şeklinde parçalara ayırıyoruz.

```
if ((saniye == 0) && (dakika > 0))
```

… satırında **`saniye=0`** işlemi ile **`dakika>0`** işlemi **`&&`** (*AndAlso*) işlemine tabi tutulur. **`&&`** operatöründe eğer ilk değişken **`true`** ise **`And`** işlemi gerçekleştirilir. Ancak ilk değer **`false`** ise ikinci değişkenin değerine bakılmadan **`false`** sonucu döndürülür.

```
true &&  true = true
true && false = false
false && x = false
```

Bizim örneğimizde **`saniye=0`** ve **`dakika>0`** iken, ikisi de **`true`** değerini aldığı için **`true`** değeri üretilecek ve kod bloğu içerisindeki işlemler gerçekleştirilecektir.



Birbiri ile bağlantılı çalışan 2 adet **Timer**'ı kullanabileceğimiz örnek bir uygulama yapalım. **Form** üzerine 2 adet **Timer** ekleyerek **Timer**'ın **`Enabled`** özelliğini **`True`**, **`timer2`**'nin **`Enabled`** özelliğini **`False`** ve her iki **Timer**'ın da **`interval`** özelliğini **`1000`** olarak ayarlayalım.

```
byte i;
private void timer1_Tick(object sender, EventArgs e)
{
 i++;
 if (i == 5)
 {
  MessageBox.Show("Timer1 saymayı bitirdi. Şimdi sıra Timer2 de.");
  timer1.Enabled = false;
  timer2.Enabled = true;
  i = 0;
 }
}
private void timer2_Tick(object sender, EventArgs e)
{
 i++;
 if (i == 3)
 {
  MessageBox.Show("Timer2 de saymayı bitirdi.");
  timer2.Enabled = false;
 }
}
```

İlk **Timer** saymaya başlayacak 5 saniye sonra **`timer1`** pasif yapılarak, **`timer2`** aktif yapılacak ve "*Timer1 saymayı bitirdi. Şimdi sıra Timer2 de.*" şeklinde mesaj görüntülenecek. 3 saniye sonra da "*Timer2 de saymayı bitirdi.*" Şeklinde bir mesaj daha görüntülenecek ve **`timer2`** pasif yapılacaktır.

`System.Timers` sınıfını kullanarak konsol uygulamaları içerisinde zamanlayıcı işlemleri yapabiliriz. Örnek olarak konsol ekranında, o anki saati görüntüleyecek bir uygulama yapalım.

```
using System;
using System.Text;
using System.Timers;
namespace timer_console
{
  class Program
  {
   static void Main(string[] args)
   {
```

```
   Timer tmr = new Timer();
   tmr.Elapsed += tmr_Elapsed;
   tmr.Interval = 1000;
   tmr.Start();
   Console.ReadLine();
  }
  static void tmr_Elapsed(object sender, ElapsedEventArgs e)
  {
   Console.Clear();
   Console.Write(string.Format("{0:hh:mm:ss}", DateTime.Now));
  }
 }
}
```

Kodumuzda **`Elapsed` event handler** ile belirtilen metod içerisindeki kod satırları, **Timer** **`interval`**'ı içerisinde çalıştırılacaktır. **`tmr_Elapsed`** metodunu kodumuza elle yazarak eklemiyoruz. Bunun için **`tmr.Elapsed+=`** ifadesinden sonra **Tab** tuşuna basarak, kodumuza bu metodun otomatik olarak eklenmesini sağlıyoruz.

Aynı işlemi **`System.Threading`** sınıfı ile de yapabiliriz.

```
using System;
using System.Text;
using System.Threading;
namespace timer_Console_Threading
{
  class Program
  {
   static void Main(string[] args)
   {
    TimerCallback tmrCb = new TimerCallback(Saat);
    Timer tmr = new Timer(tmrCb, null, 0, 1000);
    Console.ReadLine();
    tmr.Dispose();
   }
   static void Saat(object sender, ElapsedEventArgs e)
   {
    Console.Clear();
    Console.Write(string.Format("{0:hh:mm:ss}", DateTime.Now));
   }
 }
}
```

Kodumuzda…

```
TimerCallback tmrCb = new TimerCallback(Saat);
```

… satırında **`TimerCallback`** delegate kullanarak **`Saat`** isimli metodu çalıştırdık.

```
Timer tmr = new Timer(tmrCb, null, 0, 1000);
```

… satırında yeni bir **Timer** nesnesi oluşturarak, **`TimerCallback`** delagate ile belirtilen **`Saat`** metodu için bir **thread** başlatıyoruz. **`null`** olarak belirttiğimiz parametre **`TimerCallBack`**'e geçecek parametreyi belirtir. Herhangi bir parametre geçilmeyecekse, **`null`** olarak bırakılmalıdır. **`0`** olarak belirttiğimizi parametre **Timer** çalıştırılmadan önceki bekleme zamanını belirtir. Bekleme yapılması istenmiyorsa, **`0`** olarak belirtilmelidir. **`1000`** olarak belirttiğimiz parametre ise **Timer**'ın zaman aralığını belirtir (1000 milisaniye = 1saniye olarak belirttik).



Yukarıda geçen 3 farklı Timer sınıfı dışında, **`System.Diagnostics`** namespace'i altında yer alan **`StopWatch`** sınıfı kullanarak da zaman ölçümü yapabilirsiniz. **`StopWatch`** sınıfı, özellikle uygulamaların işlem zamanlarını ölçmede sıklıkla kullanılmaktadır.

`Start` metodu ile **`StopWatch`** başlatılırken, **`Stop`** metodu ile de durdurulur. Yapılan işlemin ardından **`StartWatch`** nesnesini sıfırlayarak yeniden başlatmak için **`StartNew`** metodu kullanılır. Zaman ölçümünü durdurarak **`StopWatch`** nesnesini sıfırlamak için ise **`Reset`** metodu kullanılır.

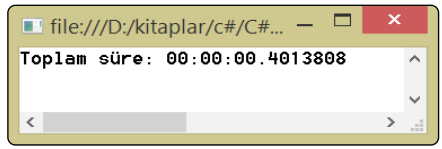
`Elapsed` özelliği, ölçülen zamanı belirtir. Ölçülen zamanı milisaniye cinsinden görüntülemek için **`ElapsedMiliseconds`** özelliği kullanılır.

`Frequency` özelliği ile saniye başına **tick** sayısını (*frekans değerini*) elde edebilirsiniz. Ölçülen toplam tick sayısını elde etmek için ise, **`ElapsedTicks`** özelliği kullanılır.

Örnek olarak bir döngü içerisinde bir diziye değer aktarımı yaparak, toplam işlem zamanını ölçelim.

```
using System;
using System.Text;
using System.Diagnostics;
namespace StopWatch_Olcum
{
 class Program
 {
  static void Main(string[] args)
  {
   int[] toplam = new int[10000000];
   Random rnd = new Random();
   int rnd_sayi = 0;
   Stopwatch sw = new Stopwatch();
   sw.Start();
   for (int i = 0; i < 10000000; i++)
   {
     rnd_sayi=rnd.Next(1,100);
     if (rnd_sayi < 50)
      toplam[i] = rnd_sayi;
   }
   sw.Stop();
   Console.WriteLine("Toplam süre: {0}",sw.Elapsed);
   Console.ReadLine();
  }
 }
}
```

Kodumuzda **sw** isimli yeni bir **StopWatch** tanımladık. **Start** metodu ile ölçüm işlemini başlattık. 10.000.000 adetlik bir dizi tanımladık. **for** döngüsü içerisinde **1** ile **100** arasında rastgele sayılar oluşturarak, **50**'den küçük olanlarını dizi içerisine aktardık. Aktarma işleminin bitmesinin ardından **Stop** metodu ile ölçüm işlemini sonlandırdık. Ölçülen süreyi **Elapsed** özelliği ile konsol ekranında görüntüledik.



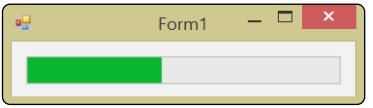
ProgressBar Kontrolü

Gerçekleştirilen bir işlem sırasında işlemin ilerleme durumunu göstermek amacıyla kullanılır.

ProgressBar Kontrolünün Özellikleri

- **`Maximum`: ProgressBar** nesnesinin alabileceği en büyük değeri belirtir.
- **`Minimum`: ProgressBar** nesnesinin alabileceği en küçük değeri belirtir.
- **`Step`: ProgressBar** nesnesinin artış miktarını belirtir.

```
private void timer1_Tick(object sender, EventArgs e)
{
 if (progressBar1.Value == progressBar1.Maximum)
  progressBar1.Value = 1;
 else
  progressBar1.Value = progressBar1.Value + 1;
}
```



Form üzerine 1 adet **Timer** ekleyerek **`Enabled=True`**, **`Interval=100`** olarak belirledik. Bir adet de **ProgressBar** nesnesi ekleyerek **`Minimum=1`**, **`Maksimum=100`**, **`Step=10`** olarak belirledik.

ProgressBar nesnesinin değeri maksimum değerine ulaştığında **ProgressBar** nesnesinin değeri `1` yapılıp tekrar başa dönmesi sağlanmakta, aksi durumda ise değeri `1` arttırılmaktadır.

PictureBox Kontrolü

Resim dosyalarını görüntülemek için kullanılan nesnedir.

PictureBox Kontrolünün Özellikleri

- **`Image`: PictureBox** nesnesi içerisinde görüntülenecek olan resim dosyasını belirtir.
- **`ImageLocation`:** Resim dosyasının web adresini belirterek, internet ortamındaki bir resim dosyasının görüntülenmesini sağlar.
- **`SizeMode`:** Resmin boyutlandırılması ile ilgili seçenekleri içerir. Şu değerleri alır:

- **`Normal`:** Eğer resmin boyutu **PictureBox** nesnesinin boyutundan büyükse, **PictureBox** nesnesi içine resmin ne kadarı sığıyorsa, o kısmı görüntülenir.
- **`StretchImage`:** Resmin boyutu **PictureBox** nesnesinin boyutunu alır.
- **`AutoSize`: PictureBox** penceresi resmin boyutunu alır.
- **`CenterImage`:** Resim **PictureBox** nesnesinin içerisinde ortalanır.
- **`Zoom`:** Resmin genişlik ve yüksekliğine göre uygun oranlarda büyütülerek ya da küçültülerek **PictureBox** içerisinde görüntülenir.

PictureBox Kontrolünün Metodları

`Load` Metodu: PictureBox nesnesi içerisine resim dosyası aktarırken, **`Image`** özelliğini kullanabileceğimiz gibi **`Load`** metodunu da kullanabiliriz.

```
pictureBox1.Load("C:\ornekresim.bmp");
```

Şeklinde **PictureBox** nesnesi içerisinde bir resim dosyası görüntüleyebiliriz.

Aynı işlemi **`Image`** özelliği ile

```
pictureBox1.Image = new Bitmap("C:\ornekresim.bmp");
```

Şeklinde belirtilen resim dosyası isminden yeni bir **Bitmap** nesnesi oluşturarak, **PictureBox** nesnesi içerisinde bir resim dosyası görüntüleyebiliriz.

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace picturebox_kontrolu
{
 public partial class Form1 : Form
 {
   public Form1()
   {
    InitializeComponent();
   }
   private void Form1_Load(object sender, EventArgs e)
   {
    openFileDialog1.FileName = "dosya seç";
    openFileDialog1.Filter = "jpeg dosyaları (*.jpeg) |*.jpg|bmp
dosyaları (*.bmp)|*.bmp|gif dosyaları (*.gif)|*.gif";
```

```
    openFileDialog1.Title = "Lütfen Bir Resim Dosyası Seçin";
    openFileDialog1.InitialDirectory = "C:\\Users\\volkan\\Pictures";
   }
   private void btnSec_Click(object sender, EventArgs e)
   {
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
     pictureBox1.Image = new Bitmap(openFileDialog1.OpenFile());
   }
   private void btnGoster_Click(object sender, EventArgs e)
   {
    pictureBox1.Load(textBox1.Text);
   }
  }
}
```

Örneğimizde kullanıcı isterse,

```
pictureBox1.Image = new Bitmap(openFileDialog1.OpenFile());
```

… satırı ile **`Image`** özelliğini kullanarak, **OpenFileDialog** nesnesi yardımıyla seçilen resim dosyasını **PictureBox** nesnesi içerisine ekleyebilir. İsterse de **`Load`** özelliğini kullanarak…

```
pictureBox1.Load(textBox1.Text);
```

… satırı ile **TextBox** içerisine yazılan resim dosyasını **PictureBox** nesnesi içerisine ekleyebilir.

Örneğimizde **PictureBox**'ın **`SizeMode`** özelliği, **Properties Window** penceresinden **`Zoom`** olarak seçilmiştir.

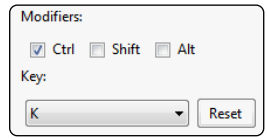


MenuStrip Kontrolü

Menü tasarımı yapmak amacı ile kullanılan nesnedir. **Form** üzerine eklenmeyip **Visual Studio IDE** penceresinin alt tarafındaki panele eklenir.

MenuStrip Kontrolünün Özellikleri

- **`Enabled`:** Varsayılan değer olarak **`True`** seçilidir. **`False`** olarak değiştirildiğinde ilgili menü öğesi pasif yapılarak kullanılamaz.
- **`Checked`:** Varsayılan değer olarak **`False`** seçilidir. **`True`** olarak değiştirildiğinde ilgili menü öğesinin yanına **check** işareti eklenir.
- **`CheckOnClick`:** Varsayılan değer olarak **`False`** seçilidir. **`True`** seçili iken menü öğesindeki **check** işaretinin fare tıklaması ile işaretlenmesini ya da işaretinin kaldırılmasını sağlar.
- **`ShowShortcutKeys`:** Varsayılan değer olarak **`True`** seçilidir. Bu konumda iken menü elemanları ile birlikte kısayol tuşlarının kullanılabilmesini sağlar. **`False`** olarak değiştirildiğinde ilgili menü öğesi ile birlikte kısayol tuşu kullanılamaz.
- **`ShortcutKeys`:** Seçili olan menü eleman ile birlikte kullanılacak kısayol tuşunu belirler. Seçilecek olan kısayol tuşunda harf, rakam, fonksiyon tuşları kullanılabileceği gibi kontrol tuşları ile birlikte de tuş kombinasyonları yapılabilir.



- **`ShortCutKeyDisplayString`:** Kısayol tuşu ile açıklayıcı bilgi görüntülenmek istenirse, bu özellik yardımıyla belirtilir.
- **`DisplayStyle`:** Menü elemanlarının gösterim biçimini belirtir. Aldığı değerler şunlardır:

  `None`: Menü elemanının boş olarak görüntülenmesini sağlar.

  `Text`: Menü elemanında yalnızca yazı görüntülenir.

  `Image`: Menü elemanında yalnızca resim görüntülenir.

  `ImageAndText`: Menü elemanında hem yazı hem de resim görüntülenir.

- **`TextAlign`:** Menü elemanları içerisindeki yazının hizalama biçimini belirtir
- **`TextDirection`:** Menü elemanları içerisindeki yazının yönünü belirtir. Aldığı değerler şunlardır:

  `Horizontal`: Menü elemanları içerisindeki yazıyı yatay yönde görüntüler.

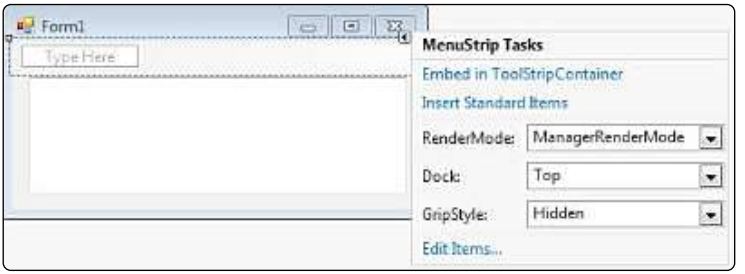
Vertical90: Menü elemanları içerisindeki yazıyı dikey yönde 90 derecelik açıyla görüntüler.

Vertical270: Menü elemanları içerisindeki yazıyı dikey yönde 270 derecelik açıyla görüntüler.

- **ToolTipText:** Menü öğesi ilgili görüntülencek açıklamalar bu özellik yardımıyla belirtilir.
- **Image:** Menü öğesi ile birlikte görüntülenecek olan resim dosyası seçilir. Menü öğesinde resim seçili iken işareti görüntülenemez.
- **Items:** Ana menü altındaki alt menü öğeleri listesine erişimi sağlar.

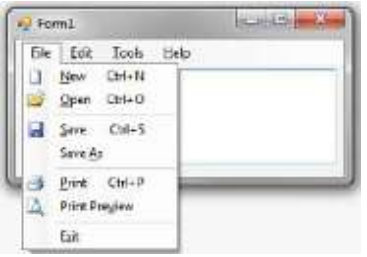
MenuStripTasks

MenuStrip nesnesi seçili iken sağ üst köşede bulunan kulakçığa tıklayarak, **MenuStripTasks** menüsünü görüntüleriz.



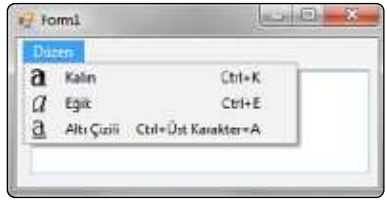
Bu menüde bulunan seçenekleri inceleyelim…

- **Embed in ToolStripContainer:** Menüyü **ToolStripContainer** içerisine yerleştirir (**ToolStripContainer** nesnesi daha sonra ayrıntılı olarak anlatılacaktır).
- **Insert Standart Items:** Alt menüleri ile birlikte hazır standart bir menü ekler.



- **`RenderMode`:** Menü çubuğunun görünümü ile ilgili ayarlar yapılır. Şu değerleri alır:

  `System`: Menü çubuğunun varsayılan Windows görünüm ayarları ile görüntülenmesini sağlar.

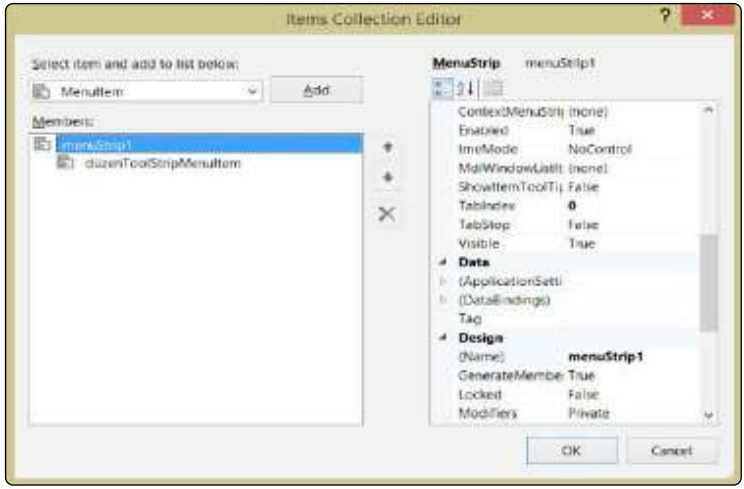


  `Professional`: Menü çubuğunun Office uygulamaları menü görünümü şeklinde görüntülenmesini sağlar.

  `ManagerRenderMode`: Özel menü stilleri tanımlamamıza olanak sağlar.



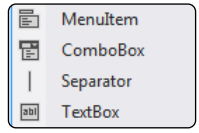
- **`Dock`:** Menünün form üzerindeki yerleşim yerini belirler.
- **`EditItems`:** Bu seçenek tıklanarak ana menü nesneleri üzerinde her türlü ekleme, silme, sıralama ve düzenleme işlemleri gerçekleştirilebilir. Ancak burada yalnızca ana menü nesneleri eklenebilir ve üzerlerinde işlemler gerçekleştirilebilir.



Alt menü seçenekleri üzerinde işlem yapmak için ise alt menülerin bağlı olduğu ana menü (örneğimizde **Düzen** menüsü) farenin sağ tuşu ile tıklanarak **EditDropDownItems** seçeneği tıklanır.







Menü öğeleri olarak **Insert** menüsü ile ya da **Items Collection Editor** penceresinde **Add** butonunun yan tarafındaki açılır liste ile **ComboBox**, **TextBox** ya da menü elemanları arasına ayırıcı çizgi (*separator*) ekleyebileceğimiz gibi **ConvertTo** seçeneği ile de daha önceden oluşturduğumuz menü elemanlarını da dönüştürebiliriz.

Örnek bir menü tasarımı yapalım.

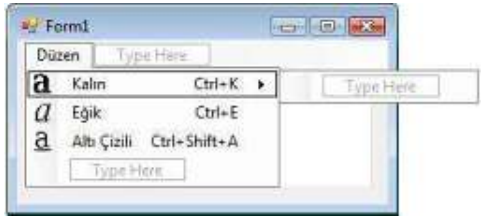
Öncelikle **Form** üzerine bir adet **TextBox** nesnesi ekleyerek `Multiline` özelliğini `true` yaparak **TextBox** nesnesini çok satırlı hale getirelim.

MenuStrip nesnesini ekleyelim. Eklenen **MenuStrip** nesnesi **Form** üzerine değilde **IDE**'nin alt tarafındaki panele eklenir. Bu panel içerisinde **MenuStrip** nesnesine tıklayarak menümüzü oluşturabilir ve özelliklerini değiştirebiliriz.

Örnek olarak **Düzen** adlı bir ana menü, bu ana menü altında da **Kalın**, *Eğik* ve Altı Çizili şeklinde alt menüleri oluşturduk.



Alt menü öğelerinin yanına simgeler ve kısayol tuşlarını ekleyelim. Simgeleri **Image** özelliği ile kısayol tuşlarını ise **ShortcutKeys** özelliği ile belirleriz.




```
private void KalınToolStripMenuItem_Click(object sender, EventArgs e)
{
  textBox1.Font = new Font(textBox1.Font.Name,
 textBox1.Font.Size, textBox1.Font.Style ^ FontStyle.Bold);
}
private void EğikToolStripMenuItem_Click(object sender, EventArgs e)
{
  textBox1.Font = new Font(textBox1.Font.Name,
   textBox1.Font.Size, textBox1.Font.Style ^ FontStyle.Italic);
}
private void AltıÇiziliToolStripMenuItem_Click(object sender,
EventArgs e)
{
  textBox1.Font = new Font(textBox1.Font.Name,
   textBox1.Font.Size, textBox1.Font.Style ^ FontStyle.Underline);
}
```

Kalın menüsü tıklandığında **TextBox** içerisindeki yazı kalın yapılır. Bir defa daha tıklanırsa, kalınlık özelliği iptal edilir. Diğer menü öğeleri de aynı şekildedir. Aynı zamanda kısayol tuşları da kullanılabilir. **^** (*XOR- Özel veya*) mantıksal operatörü

bir tarafın `true`, diğer tarafın `false` olduğu durumlarda çalışır. Burada da eğer daha önceden o stil özelliği uygulanmışsa iptal edilmesini, uygulanmamışsa uygulanmasını sağlıyoruz.

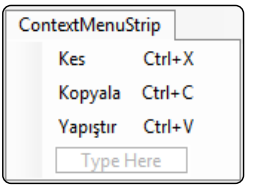


ContextMenuStrip Kontrolü

Herhangi bir nesneye sağ tuş menüsü eklemek amacı ile kullanılan menü nesnesidir. **Form** üzerine eklenmeyip **Visual Studio IDE** penceresinin alt tarafındaki panele eklenir.

Birçok özelliği **MenuStrip** nesnesi ile aynı olduğu için burada ayrıca değinmeyeceğiz. **MenuStrip** nesnesi konusunda yaptığımız örneği geliştirerek, **TextBox** nesnesi içerisinde farenin sağ tuşu ile tıklandığında **Kes**, **Kopyala**, **Yapıştır** seçeneklerinden oluşan sağ menü çıkmasını sağlayalım.

Form'umuzun üzerine ek olarak **ContextMenuStrip** nesnesini ekliyoruz. **ContextMenuStrip** tasarımını yandaki gibi yapıyoruz:



TextBox'ın **ContextMenuStrip** özelliğinden `ContextMenuStrip1` değerini seçerek, **ContextMenuStrip** nesnemizi **TextBox** nesnesine bağlıyoruz. Uygulamamızın **ContextMenuStrip** kodları eklenmiş son hali aşağıdaki gibi olacak:

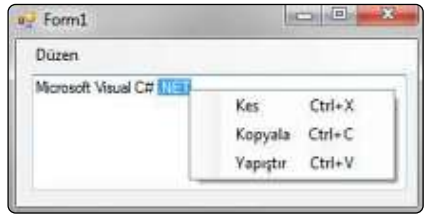
```
private void KalınToolStripMenuItem_Click(object sender, EventArgs e)
{
  textBox1.Font = new Font(textBox1.Font.Name, textBox1.Font.Size,
textBox1.Font.Style ^ FontStyle.Bold);
}
private void EğikToolStripMenuItem_Click(object sender, EventArgs e)
{
  textBox1.Font = new Font(textBox1.Font.Name,
 textBox1.Font.Size, textBox1.Font.Style ^ FontStyle.Italic);
}
private void AltıÇiziliToolStripMenuItem_Click(object sender, EventArgs e)
```

```
{
  textBox1.Font = new Font(textBox1.Font.Name,
   textBox1.Font.Size, textBox1.Font.Style ^ FontStyle.Underline);
}
private void KesToolStripMenuItem_Click(object sender, EventArgs e)
{
  if (textBox1.SelectionLength > 0)
   textBox1.Cut();
  else
   MessageBox.Show("Seçili Alan Yok");
}
private void KopyalaToolStripMenuItem_Click(object sender, EventArgs e)
{
  if (textBox1.SelectionLength > 0)
   textBox1.Copy();
  else
   MessageBox.Show("Seçili Alan Yok");
}
private void YapıştırToolStripMenuItem_Click(object sender, EventArgs e)
{
  textBox1.Paste();
}
```

Kodumuzda…

```
if (textBox1.SelectionLength > 0)
```

… satırında **TextBox** içerisinde seçili karakter uzunluğunun 0'dan büyük olmasını kontrol ederek, kodlarımızın yalnızca **TextBox** içerisinde seçili kısım varsa çalıştırılmasını sağlıyoruz.



Kodumuzu çalıştıralım. **TextBox** içerisinde farenin sağ tuşu ile tıkladığımızda, sağ menü görüntülenir.

NotifyIcon Kontrolü

Form'u **Taskbar** içerisine simge durumuna küçültmek amacı ile kullanılan menü nesnesidir. **Form** üzerine eklenmeyip **Visual Studio IDE** penceresinin alt tarafındaki panele eklenir.

NotifyIcon Kontrolünün Özellikleri

- **`ContextMenuStrip`:** Program **Taskbar** içerisine simge durumuna küçültüldüğünde, görüntülenecek olan menü nesnesinin ismi atanır.
- **`BalloonTipText`:** Uygulama simge durumuna küçültüğünde görüntülenecek olan **balon** metnini belirler.
- **`BalloonTipTitle`:** Farenin imleci program simgesinin üzerine getirildiğinde görüntülenecek olan **metni** belirler.
- **`BalloonTipIcon`:** Uygulama simge durumuna küçültüğünde görüntülenecek olan balon metninin **simgesini** belirler. Şu değerleri alır.

  `None`: Herhangi bir simge görüntülenmez.

  `Info`: İkaz simgesi görüntüler.

  `Error`: X simgesi görüntüler.

  `Warning`: Ünlem işareti görüntüler.

NotifyIcon Kontrolünün Metodları

- **`ShowBalloonTip`:** Belirtilen milisaniye cinsinden değer kadar süre, balon yardımını görüntüler

Örneğin; `ShowBalloonTip(1000)` kodu, balon yardımını 1 saniye boyunca görüntüler.

Örnek olarak **Form** kapatıldığında simge durumuna küçülten bir uygulama yapalım. **Form** üzerine 1 adet **NotifyIcon** kontrolu ve **Form** simge durumuna küçültüldüğünde görüntülenecek menüler için 1 adet de **ContextMenuStrip** kontrolü ekliyoruz.

ContextMenuStrip içerisine Önceki **`Boyut`** ve **`Çıkış`** adı ile 2 adet menü elemanı ekliyoruz.

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
  if (e.CloseReason == CloseReason.UserClosing)
  {
   e.Cancel = true;
   this.Hide();
   notifyIcon1.Visible = true;
   notifyIcon1.ShowBalloonTip(1000);
  }
}
private void Form1_Load(object sender, EventArgs e)
{
  notifyIcon1.ContextMenuStrip = contextMenuStrip1;
  notifyIcon1.BalloonTipText = "Uygulama Hala Çalışıyor...";
  notifyIcon1.BalloonTipTitle = "Örnek Uygulama";
  notifyIcon1.BalloonTipIcon = ToolTipIcon.Info;
}
private void Form1_Resize(object sender, EventArgs e)
{
  if (this.WindowState == FormWindowState.Minimized)
  {
   this.Hide();
   notifyIcon1.Visible = true;
   notifyIcon1.ShowBalloonTip(1000);
  }
}
private void notifyIcon1_DoubleClick(object sender, EventArgs e)
{
  this.Show();
  notifyIcon1.Visible = false;
}
private void ÖncekiBoyutToolStripMenuItem_Click(object sender, EventArgs e)
{
  this.Show();
  this.WindowState = FormWindowState.Normal;
  notifyIcon1.Visible = false;
}
private void ÇıkışToolStripMenuItem_Click(object sender, EventArgs e)
{
  Application.Exit();
}
```

Form'u simge durumunda görüntülemek için öncelikle **Form**'un **`ShowInTaskBar`** seçeneği **`True`** olarak ayarlanmalıdır. Daha sonra **Form** simge durumuna küçültüldüğünde görüntülenecek olan simgeyi Icon özelliği ile belirliyoruz.

```
if (e.CloseReason == CloseReason.UserClosing)
```

Komut satırı ile **Form**, kullanıcı tarafından kapatıldığında **`e.Cancel = true`** komutu ile çıkış işleminin iptal edilmesini ve daha sonra…

```
this.Hide();
notifyIcon1.Visible = true;
notifyIcon1.ShowBalloonTip(1000);
```

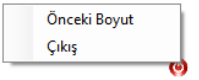
… **Form**'un gizlenmesini, task bar içerisinde simgenin görüntülenmesini ve balon yardımının bir saniye süre ile görüntülenmesini sağlıyoruz.



Oluşturduğumuz **ContextMenu** öğelerinden birisi…

```
this.WindowState = FormWindowState.Normal;
```

… komutu ile uygulamanın önceki boyutuna dönmesini, diğeri ise programın kapatılmasını sağlıyor.



`notifyIcon1_DoubleClick` olayı ile simge üzerine çift tıklanıldığında, **Form**'un görüntülenerek simgenin gizlenmesini sağlıyoruz.

ToolStrip Kontrolü

Araç çubuğu oluşturmak amacı ile kullanılır. **Form** üzerine eklenmeyip **Visual Studio IDE** penceresinin alt tarafındaki panele eklenir.

ToolStrip Kontrolünün Özellikleri

- **`DisplayStyle`:** Araç çubuğu üzerindeki elemanların üzerinde görüntülenecek yazı ve resimler ile ilgili ayarlar yapılır. Şu değerleri alır:

  `None`: Yazı ve resim görüntülenmez.

  `Text`: Yalnızca yazı görüntülenir.

  `Image`: Yalnızca resim görüntülenir.

  `ImageAndText`: Hem yazı hem de resim görüntülenir.

- **`ImageScaling`:** Araç çubuğu üzerindeki elemanlar üzerinde görüntülenecek olan resmin boyutlandırılma ayarını yapar. **`None`** değerini aldığında resim orijinal boyutunda görüntülenir. **`SizeToFit`** değerini aldığında ise, araç çubuğunun boyutuna göre resim otomatik olarak boyutlandırılır.
- **`TextDirection`:** Araç çubuğu üzerinde görüntülenecek olan yazının açısını belirler. **`DisplayStyle`** özelliği **`Text`** ya da **`ImageAndText`** değerini almış olmalıdır. **`TextDirection`** özelliği şu değerleri alabilir:

  `Horizontal`: Yazı yatay olarak görüntülenir.

  `Vertical90`: Yazı 90 derece açıyla dikey olarak görüntülenir.

  `Vertical270`: Yazı 270 derece açıyla dikey olarak görüntülenir.



- **`TextImageRelation`:** Resim ve yazının bir arada yerleşim şeklini belirler. **`DisplayStyle`** özelliği **`ImageAndText`** değerini almış olmalıdır. **`TextImageRelation`** özelliği şu değerleri alabilir:

  `Overlay`: Yazı resmin ortasına yerleşir.

  `ImageAboveText`: Resim yazının üstüne yerleşir.

  `TextAboveImage`: Yazı resmin üzerine yerleşir.

  `ImageBeforeText`: Resim yazıdan önce yerleştirilir.

  `TextBeforeImage`: Yazı resimden önce yerleştirilir.

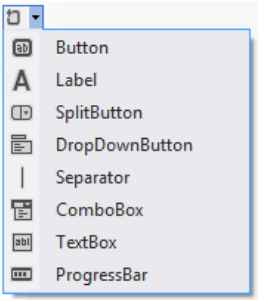


- **`ToolTipText`:** Araç çubuğu elemanı üzerinde fare ile beklenildiğinde görüntülenecek olan açıklama yazısını belirler.

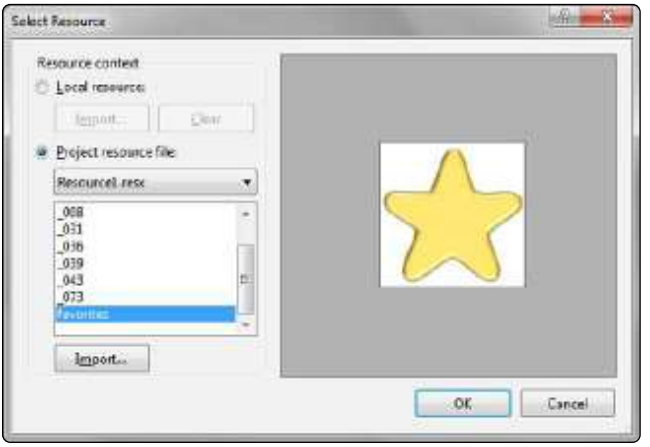


Örnek bir web **Browser** uygulaması yaparak örneğimiz içerisinde araç çubuğu tasarımı yapalım. Form üzerine 1 adet **MenuStrip**, 1 adet **ContextMenuStrip**, 1 adet **ToolStrip** nesnesi, 1 adet de **WebBrowser** nesnesi ekleyelim.

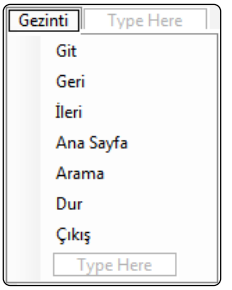
ToolStrip nesnesi üzerindeki aşağı ok işaretine tıklayarak, araç çubuğu üzerine butonlarımızı ekleyebiliriz. Aşağıda doğru açılan liste kutusu şeklinde buton eklemek için **DropDownButton** nesnesi, araç çubuğu elemanları arasına ayırıcı çizgi koymak için ise, **Seperator** nesnesi eklemeliyiz.



Araç çubuğu üzerine eklediğimiz butonların simgeleri için yeni bir **resource** dosyası oluşturarak resimleri buradan ekleyelim.



MenuStrip nesnesinin elemanları ise şu şekilde olacak:



Uygulamamızın kod satırları aşağıdaki gibi olacak:

```
byte i;
private void GeriToolStripMenuItem_Click(object sender, EventArgs e)
{
 webBrowser1.GoBack();
}
private void İleriToolStripMenuItem_Click(object sender, EventArgs e)
{
 webBrowser1.GoForward();
}
private void AnaSayfaToolStripMenuItem_Click(object sender, EventArgs e)
{
 webBrowser1.GoHome();
}
private void AramaToolStripMenuItem_Click(object sender, EventArgs e)
{
 webBrowser1.GoSearch();
}
private void DurToolStripMenuItem_Click(object sender, EventArgs e)
{
 webBrowser1.Stop();
}
private void ÇıkışToolStripMenuItem_Click(object sender, EventArgs e)
{
 Application.Exit();
}
private void GitToolStripMenuItem_Click(object sender, EventArgs e)
{
 i++;
 if (!(txtadres.Text.StartsWith("http://")))
  txtadres.Text = txtadres.Text.Insert(0, "http://");
 webBrowser1.Navigate(txtadres.Text);
 Cursor.Current = Cursors.Default;

 ToolStripMenuItem yenimenu = new
  ToolStripMenuItem(String.Format(txtadres.Text, i));
  tbSayfalar.DropDownItems.Add(yenimenu);
}
private void tbSayfalar_DropDownItemClicked(object sender,
ToolStripItemClickedEventArgs e)
{
 webBrowser1.Navigate(e.ClickedItem.Text);
}
```

WebBrowser nesnesi ile;

Bir önceki sayfaya gitmek için **WebBrowser** nesnesinin **`GoBack`** metodu kullanılır.

Bir sonraki sayfaya gitmek için **WebBrowser** nesnesinin **`GoForward`** metodu kullanılır.

Ana sayfaya gitmek için **WebBrowser** nesnesinin **`GoHome`** metodu kullanılır.

Arama sayfasına gitmek için **WebBrowser** nesnesinin **`GoSearch`** metodu kullanılır.

Bir sayfanın görüntülenmesi için **WebBrowser** nesnesinin **`Navigate`** metodu kullanılır. **`WebBrowserNesnesi.Navigate("Web site adresi")`** biçiminde kullanılır.

Bir sayfanın görüntülenmesi işlemini durdurmak için ise **WebBrowser** nesnesinin **`Stop()`** metodu kullanılır.

Uygulamamızda…

```
ToolStripMenuItem yenimenu = new
    ToolStripMenuItem(String.Format(txtadres.Text, i));
```

… satırı ile index numarası **`i`** olan, adı ise **`txtadres`** isimli text kutusuna yazılan web adresi olan bir menü nesnesi oluşturuyoruz.

```
tbSayfalar.DropDownItems.Add(yenimenu);
```

Satırı ile de oluşturmuş olduğumuz **`yenimenu`** nesnesini **`tbSayfalar`** adlı **DropDown** menümüze alt menü olarak ekliyoruz. Her yeni sayfa görüntülenişinde **Git** butonuna basılması ile birlikte görüntülenen web sitesinin adı tbSayfalar menüsüne alt menü olarak eklenecektir.

Oluşturduğumuz alt menülere tıklanınca **WebBrowser** içinde o adresin görüntülenmesi için ise, **`tbSayfalar_DropDownItemClicked`** olayını kullanıyoruz. Bu olay içerisinde…

```
webBrowser1.Navigate(e.ClickedItem.Text);
```

… satırı ile tıklanan menü nesnesinin text özelliğinde bulunan web sitesi adresinin **WebBrowser** nesnesi içerisinde görüntülenmesini sağlıyoruz.

Eğer web adresinin başına "*http://*" ifadesinin konulması unutulursa…

```
if (!(txtadres.Text.StartsWith("http://")))
    txtadres.Text = txtadres.Text.Insert(0, "http://");
```

… satırı ile adresin baş kısmına "*http://*" ifadesini ekliyoruz. **`StartsWith`** metodu bir string'in, belirtilen bir ifade ile başlayıp başlamadığını bulmak amacı ile kullanılır. Geriye **`true`** ya da **`false`** olarak boolean değer döndürür.

Kodumuzda da görüldüğü gibi kodlarımızı **Gezinti** menüsü altındaki menüler için yazdık. Ayrıca araç çubuğu içerisindeki simgelere tıklandığında, aynı işlevlerin çalıştırılması için kodlarımızı yeniden yazmamız gerekmektedir. Ancak bu kodları yeniden yazmak yerine **event handler** kullanarak araç çubuğundaki simgelere tıklandığında ilgili menü kodlarının çalıştırılmasını sağlayabiliriz.

Menü elemanlarına tıklandıkça araç çubuğu üzerindeki aynı işleve sahip butonları da çalıştırmak için **Form1.Designer.cs** dosyasını açarak **`private void InitializeComponent()`** satırını bulup **+** işaretine tıklıyoruz. Açılan kod satırları içerisinde event handler metodu ile istediğimiz olayları birbirine bağlayabiliriz. Örneğin; **`GeriToolStripMenuItem`** isimli menü öğesine tıklandığında, **`tbgeri`** isimli **Tooltip** buton nesnesinin de çalıştırılmasını sağlamak için...

```
this.tbgeri.Click += new System.EventHandler(this.
GeriToolStripMenuItem_Click);
```

… biçiminde olayları birbirine bağlarız.

Tüm olayları birbirine bağlamak için **Form1.Designer.cs** dosyasında **`private void InitializeComponent()`** bölümünün en altına

```
this.tbgeri.Click += new
System.EventHandler(this.GeriToolStripMenuItem_Click);
this.tbileri.Click += new
System.EventHandler(this.İleriToolStripMenuItem_Click);
this.tbAnaSayfa.Click += new
System.EventHandler(this.AnaSayfaToolStripMenuItem_Click);
```

```
this.tbArama.Click += new
System.EventHandler(this.AramaToolStripMenuItem_Click);
this.tbDur.Click += new
System.EventHandler(this.DurToolStripMenuItem_Click);
this.tbCikis.Click += new
System.EventHandler(this.ÇıkışToolStripMenuItem_Click);
this.tbgit.Click += new
System.EventHandler(this.GitToolStripMenuItem_Click);
this.btngit.Click += new
   System.EventHandler(this.GitToolStripMenuItem_Click);
```

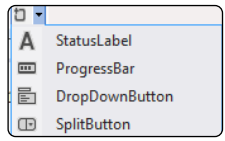
… satırlarını ekliyoruz.

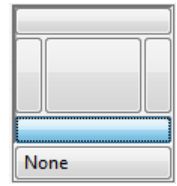


StatusStrip Kontrolü

Form nesnesinin alt tarafında bir durum çubuğu görüntülenerek kullanıcıya bilgi verilmesini sağlar. **Form** üzerine eklenmeyip **Visual Studio IDE** penceresinin alt tarafındaki panele eklenir.

StatusStrip nesnesini `Dock` özelliği vasıtası ile formun herhangi bir bölümüne konumlandırabilirsiniz.





StatusStrip nesnesi üzerine **StatusLabel**, **ProgressBar**, **DropDownButton**, **SplitButton** eklenebilir.

ToolStrip konusunda yapmış olduğumuz **WebBrowser** örneğini **StatusStrip** ekleyerek geliştirelim. Ayrıca 1 adet de **Timer** nesnesi ekleyeceğiz. **Timer** nesnesinin `Interval` özelliği `1000`, `Enabled` özelliği `False` olacak.

StatusStrip içerisine ise ziyaret edilen sayfaların isimlerinin görüntülenmesi için 1 adet **StatusStripLabel**, sayfanın görüntülenme hızını göstermesi için 1 adet **ToolStripProgressBar**, ziyaret edilen sayfaların listesinin görüntülenmesi için de 1 adet **ToolStripDropDownButton** nesnesi ekliyoruz. **ToolStripDropDownButton** nesnesinin `DisplayStyle` özelliğini `Text` olarak ayarlayarak, yalnızca yazı görüntülenmesini sağlıyoruz.

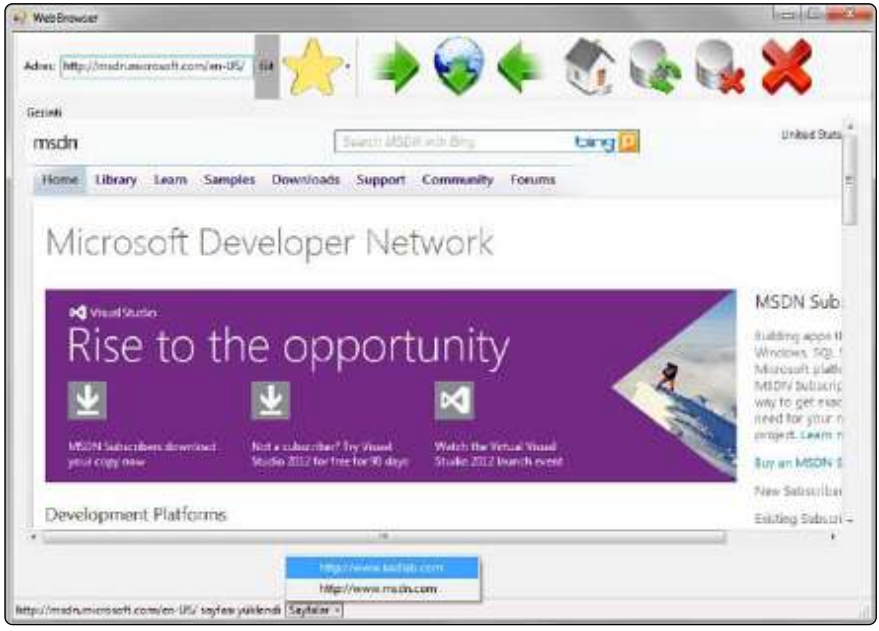
WebBrowser kodumuzun son hali, yaptığımız eklemelerle aşağıdaki hali alacak:

```
byte i;
private void GeriToolStripMenuItem_Click(object sender, EventArgs e)
{
 webBrowser1.GoBack();
}
private void İleriToolStripMenuItem_Click(object sender, EventArgs e)
{
 webBrowser1.GoForward();
}
private void AnaSayfaToolStripMenuItem_Click(object sender, EventArgs e)
{
 webBrowser1.GoHome();
}
private void AramaToolStripMenuItem_Click(object sender, EventArgs e)
{
 webBrowser1.GoSearch();
}
private void DurToolStripMenuItem_Click(object sender, EventArgs e)
{
```

```
 webBrowser1.Stop();
}
private void ÇıkışToolStripMenuItem_Click(object sender, EventArgs e)
{
 Application.Exit();
}
private void GitToolStripMenuItem_Click(object sender, EventArgs e)
{
 i++;
 if (!(txtadres.Text.StartsWith("http://")))
  txtadres.Text = txtadres.Text.Insert(0, "http://");
 webBrowser1.Navigate(txtadres.Text);
 Cursor.Current = Cursors.Default;

 ToolStripMenuItem yenimenu = new
  ToolStripMenuItem(String.Format(txtadres.Text, i));
 tbSayfalar.DropDownItems.Add(yenimenu);
 tsSayfalar.DropDownItems.Add(yenimenu);
}
private void tbSayfalar_DropDownItemClicked(object sender,
ToolStripItemClickedEventArgs e)
{
 webBrowser1.Navigate(e.ClickedItem.Text);
}
private void tsSayfalar_DropDownItemClicked(object sender,
ToolStripItemClickedEventArgs e)
{
 webBrowser1.Navigate(e.ClickedItem.Text);
}
private void webBrowser1_DocumentCompleted(object sender,
WebBrowserDocumentCompletedEventArgs e)
{
 txtadres.Text = e.Url.ToString();
 timer1.Stop();
 toolStripStatusLabel1.Text = txtadres.Text + " sayfası yüklendi";
 toolStripProgressBar1.Value = 0;
 toolStripProgressBar1.Visible = false;
}
private void webBrowser1_Navigating(object sender,
WebBrowserNavigatingEventArgs e)
{
```

```
 toolStripLabel1.Visible = true;
 toolStripProgressBar1.Visible = true ;
 toolStripStatusLabel1.Text = txtadres.Text +
       " adresi yüklenmeye başladı";
 toolStripProgressBar1.Value = 0;
 timer1.Start();
}
private void timer1_Tick(object sender, EventArgs e)
{
 toolStripStatusLabel1.Text = txtadres.Text + " Sayfa Yükleniyor ";
 if (toolStripProgressBar1.Value == 100)
  toolStripProgressBar1.Value = 0;
 else
  toolStripProgressBar1.Value = toolStripProgressBar1.Value + 10;
}
private void Form1_Load(object sender, EventArgs e)
{
 toolStripProgressBar1.Visible = false;
}
```



Bu örneğimizde daha önce yapmış olduğumuz **WebBrowser** örneğine ek olarak, **WebBrowser** içerisinde bir siteye yönlenilmeye başlandığında gerçekleşen **Navigating** olayını kullandık. Bu olay içerisinde timer nesnesini başlatıyoruz.

Timer nesnesinin **Tick** olayı içerisinde de **ProgressBar**'ı ilerletiyor ve ilgili sayfanın görüntülendiğini **StatusBar** içerisinde belirtiyoruz.

Ayrıca **WebBrowser** içerisinde sayfanın görüntülenmesi tamamlandığında gerçekleşen `DocumentCompleted` olayı ile de o an görüntülenmesi tamamlanmış olan sayfayı

```
txtadres.Text = e.Url.ToString();
```

… satırı ile `txtadres` isimli adres satırımıza yazıyoruz. Bunu yazmamızın nedeni; bir site içerisinde alt sayfalara girildiğinde bunu adres kutusuna yazdırmak amacıyladır.

Son olarak **Timer** nesnesini durduruyoruz.

Ayrıca daha önce görüntülenen sayfaların listesini araç çubuğunda gösterdiğimiz gibi burada ek olarak durum çubuğunda da görüntülüyoruz.

ToolTip Kontrolü

Farenin imleci bir kontrolün üzerine geldiğinde o kontrolle ilgili olarak bir açıklama görüntülenmesini sağlar. **Form** üzerine eklenmeyip **Visual Studio IDE** penceresinin alt tarafındaki panele eklenir.

Tooltip Kontrolünün Özellikleri

- **`Active`: `False`** iken kontrol üzerinde herhangi bir açıklama görüntülenmez. **`True`** iken açıklama görüntülenir.
- **`InitialDelay`:** Açıklamanın, kontrol üzerine odaklanıldıktan ne kadar süre sonra görüntüleneceğini milisaniye cinsinden belirler. Varsayılan değeri `500`'dür.
- **`AutoPopDelay`:** Açıklamanın kontrol üzerinde ne kadar süre görüntüleneceğini milisaniye cinsinden belirler. Varsayılan değeri `5000`'dir.
- **`ReshowDelay`:** Farenin imleci bir kontrol üzerine getirilip açıklama görüntülenip,

farenin imleci kontrol üzerinden çekildiğinde açıklamanın ne kadar süre ile görüntülenmeye devam edeceğini milisaniye cinsinden belirler. Varsayılan değeri `100`'dür.

- **`AutomaticDelay`**: **`AutoPopDelay`**, **`InitalDelay`** ve **`ReshowDelay`** değerlerinin **`AutomaticDelay`** değerine bağlı olarak otomatik olarak değiştirilmesini sağlar.

  `AutoPopDelay`: **`AutomaticDelay`** değerinin 10 katı değer alır.

  `InitialDelay`: **`AutomaticDelay`** değerine eşit değer alır.

  `ReshowDelay`: **`AutomaticDelay`** değerinin `1/5`'i oranında değer alır.

- **`ShowAlways`**: **`True`** değerini aldığında kontrol üzerinde açıklamanın halen bulunup bulunmadığı kontrol edilir. Eğer açıklama görüntülenmeye devam ediyorsa, açıklama görüntülenmeye devam eder. **`False`** değerini aldığında ise, açıklamanın görüntülenmesi bitirilip yeniden görüntülenir.
- **`StripAmbersands`**: **`True`** değerini aldığında açıklama metni içerisinde **`&`** işareti varsa, **`&`** işaretinden sonra gelen harfin altını çizer, **`false`** değerini aldığında ise altını çizmez. Menü elemanları ya da **Label** gibi bir harfin kısayol tuşu olarak belirlendiği metinleri açıklama satırı olarak kullanmak istediğimizde faydalı olan bir özelliktir.
- **`ToolTipIcon`**: Açıklama üzerinde yer alacak simgeyi belirler. Şu değerleri alır:

  `None`: Herhangi bir simge görüntülenmez.

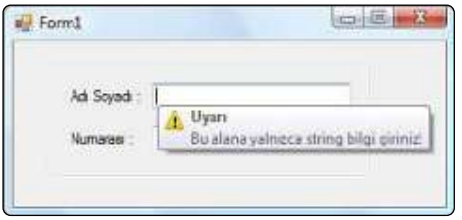
  `Info`: İkaz simgesi görüntüler.

  `Error`: X simgesi görüntüler.

  `Warning`: Ünlem işareti görüntüler.

- **`ToolTipTitle`**: Açıklama penceresinin başlığında görüntülenecek metni belirler.
- **`UseAnimation`**: **`True`** değerini aldığında açıklama penceresi görüntülenirken ya da kaybolurken animasyon efekti verilerek görüntülenmesini ve kaybolmasını sağlar. **`False`** değerini aldığında ise herhangi bir efekt uygulanmaz.
- **`UseFading`**: Açıklama penceresinin görünürken saydam görünümden normal görünüme doğru geçiş yaparak, kaybolurken de normal görünümden saydam görünüme geçiş yaparak kapanmasını sağlar.
- **`IsBallon`**: **`True`** değerini aldığında açıklama penceresi balon şeklinde görüntülenir.

IsBallon özelliği **false** değerini aldığında:



IsBallon özelliği **true** değerini aldığında:



ToolTip Kontrolünün Metodları

- **SetToolTip:** Belirtilen bir kontrol için görüntülenecek olan açıklama mesajını belirler. Kullanımı şu şekildedir:

```
SetToolTip(kontrol adı, "açıklama mesajı");
```

- **GetToolTip:** Belirtilen bir kontrol için ayarlanmış olan açıklama mesajını geriye döndürür. Kullanımı şu şekildedir:

```
GetToolTip(kontrol adı);
```

Örneğin…

```
MessageBox.Show(toolTip1.GetToolTip(txtAdSoyad));
```

…. satırı ile **txtAdSoyad** adlı kontrol için ayarlanmış olan hata mesajı, bir mesaj penceresi içerisinde görüntülenir.

- **RemoveAll:** Kontrollere atanmış olan tüm **ToolTip** nesnesi atamalarını siler.
- **Show:** Açıklama penceresinin istenilen herhangi bir kontrolün üzerinde ya da istenilen bir noktada görüntülemek için kullanılır.

```
toolTip1.Show("Bu alana yalnızca string bilgi giriniz!", txtAdSoyad) ;
```

txtadSoyad kontrolü üzerinde açıklama penceresini görüntüler.

```
toolTip1.Show("Bu alana yalnızca string bilgi giriniz!", txtAdSoyad, 5000);
```

txtadSoyad kontrolü üzerinde açıklama penceresini 5 saniye görüntüler.

```
toolTip1.Show("Bu alana yalnızca string bilgi giriniz!",
txtAdSoyad, txtAdSoyad.Width,txtAdSoyad.Height - 20, 5000);
```

txtAdSoyad kontrolünün üzerinde **x** koordinatı **txtAdSoyad** kontrolünün genişliği, **y** koordinatı ise **txtAdSoyad** kontrolünün yüksekliğinin 20 piksel aşağısı olan noktada açıklama penceresini 5 saniye görüntüler.



Yukarıda saydığımız özellikleri **Properties** penceresinden değil de kod ile yazalım.+

```
private void Form1_Load(object sender, EventArgs e)
{
 toolTip1.AutoPopDelay = 5000;
 toolTip1.InitialDelay = 1000;
 toolTip1.ReshowDelay = 500;
 toolTip1.ToolTipIcon = ToolTipIcon.Warning;
 toolTip1.ToolTipTitle = "Uyarı";
 toolTip1.IsBalloon = true;
 toolTip1.SetToolTip(txtAdSoyad,
    "Bu alana yalnızca string bilgi giriniz!");
 toolTip1.SetToolTip(txtNumara,
    "Bu alana yalnızca rakam giriniz!");
}
```



Kodumuzda…

```
toolTip1.AutoPopDelay = 5000;
```

… satırında **AutoPopDelay** özelliği ile açıklamanın kontrol üzerinde 5 saniye görüntüleneceğini belirliyoruz.

```
toolTip1.InitialDelay = 1000;
```

… satırında **InitialDelay** özelliği ile açıklamanın, kontrol üzerine odaklanıldıktan 1 saniye sonra görüntüleneceğini belirliyoruz.

```
toolTip1.ReshowDelay = 500;
```

… satırında **ReshowDelay** özelliği ile açıklama görüntülendikten sonra, farenin imleci kontrol üzerinden çekildiğinde açıklamanın 500ms (*yarım saniye*) daha görüntülenmeye devam edeceğini belirliyoruz.

```
toolTip1.ToolTipIcon = ToolTipIcon.Warning;
```

… satırında **ToolTipIcon** özelliği ile açıklama üzerinde ünlem işareti şeklinde simge görüntülenmesini sağlıyoruz.

```
toolTip1.ToolTipTitle = "Uyarı";
```

… satırında **ToolTipTitle** özelliği ile açıklama penceresinin başlığını "*Uyarı*" olarak belirliyoruz.

```
toolTip1.IsBalloon = true;
```

… satırında **IsBalloon** özelliği ile açıklama penceresinin balon şeklinde görüntüleneceğini belirliyoruz.

```
toolTip1.SetToolTip(txtAdSoyad,
    "Bu alana yalnızca string bilgi giriniz!");
toolTip1.SetToolTip(txtNumara,
    "Bu alana yalnızca rakam giriniz!");
```

… satırlarında **SetToolTip** metodu ile belirtilen kontrole belirtilen açıklama metni ile **ToolTip** kontrolünü bağlıyoruz.

ErrorProvider Kontrolü

Kullanıcı tarafından TextBox, ListBox vs. gibi nesnelere veri girişi yapılırken oluşacak bir hata durumunda, kullanıcıyı görsel olarak uyararak yönlendirmek amacı ile kullanılır. **Form** üzerine eklenmeyip **Visual Studio IDE** penceresinin alt tarafındaki panele eklenir.

ErrorProvider Kontrolünün Özellikleri

- **`BlinkRate`:** Hata durumunda görüntülenecek olan mesajın simgesinin ne sıklıkla görüntüleneceğini milisaniye cinsinden belirler. Varsayılan değeri **250**'dir.
- **`BlinkStyle`:** Hata simgesinin görüntülenme biçimini belirler. Şu değerleri alır:

  `BlinkIfDifferentError`: Hata simgesi birkaç defa yanıp söner. Bir sonraki oluşacak hataya kadar bir daha yanıp sönmez.

  `AlwaysBlink`: Hata simgesi sürekli yanıp söner.

  `NeverBlink`: Hata simgesi yanıp sönmez.

- **`Icon`:** Hata simgesini belirler. Varsayılan hata simgesi şeklindedir. İstenilirse kullanıcı bilgisayarında mevcut olan simgelerden birini seçebilir.

ErrorProvider Kontrolünün Metodları

- **`SetError`:** Belirtilen bir kontrol için görüntülenecek olan hata mesajını belirler. Kullanımı şu şekildedir:

```
SetError(kontrol adı, "hata mesajı");
```

- **`GetError`:** Belirtilen bir kontrol için ayarlanmış olan hata mesajını geriye döndürür. Kullanımı şu şekildedir:

```
GetError(kontrol adı);
```

Örneğin…

```
MessageBox.Show(ErrorProvider1.GetError(txtAdSoyad));
```

… satırı ile **`txtAdSoyad`** adlı kontrol için ayarlanmış olan hata mesajı bir mesaj penceresi içerisinde görüntülenir. Örnek olarak **TextBox** içerisine metin girişi ve rakam girişi sırasında oluşan hata durumlarında bizi uyaracak bir uygulama yapalım.

```
Boolean rakamvar;
int rakam;
private void txtAdSoyad_Validating(object sender, System.
ComponentModel.CancelEventArgs e)
{
  errorProvider1.Clear();
  rakamvar = int.TryParse(txtAdSoyad.Text , out rakam);
  if (txtAdSoyad.Text == "")
  {
   errorProvider1.SetError(txtAdSoyad, "Ad Soyad boş bırakılamaz");
  }
  else
  {
   if (rakamvar == true)
     errorProvider1.SetError(txtAdSoyad,
      "Ad Soyad alanına rakam girilemez");
  }
}
private void txtNumara_Validating(object sender, System.
ComponentModel.CancelEventArgs e)
{
  errorProvider1.Clear();
  rakamvar = int.TryParse(txtNumara.Text, out rakam);
  if (txtNumara.Text == "")
  {
   errorProvider1.SetError(txtNumara, "Numara boş bırakılamaz");
  }
  else
  {
   if (rakamvar == false)
    errorProvider1.SetError(txtNumara, "Numara alanına harf girilemez");
  }
}
private void Form1_Load(object sender, EventArgs e)
{
  rakamvar = false;
}
```

txtAdSoyad nesnesinin **Validating** olayında öncelikle text kutusuna değer

girişi yapılıp yapılmadığı kontrol edilmekte, eğer değer girişi yapılmamışsa **errorProvider** nesnesine "*Ad Soyad boş bırakılamaz*" mesajı hata mesajı olarak gönderilmektedir. Eğer değer girişi yapılmışsa, text kutusuna girilen değerin içerisindeki karakterler sıra ile incelenerek karakterin rakam olup olmadığı **TryParse** metodu ile kontrol edilmekte; eğer rakam varsa, **rakamvar** adlı mantıksal değişken **true** değerini almaktadır. Bu değişken **true** değerini aldığında ise **errorProvider** nesnesine hata mesajı olarak "*Ad Soyad alanına rakam girilemez*" mesajı gönderilmektedir.



Benzer bir kontrol işlemi **txtNumara** adlı text kutusu için de yapılmakta ve boş değer kontrolü ve harf girişi kontrolü yapılarak bu durumlarda **ErrorProvider** nesnesine gerekli parametreler gönderilmektedir.



HelpProvider Kontrolü

Herhangi bir uygulama ya da kontrolle ilgili yardım görüntülemek için kullanılır. **Form** üzerine eklenmeyip **Visual Studio IDE** penceresinin alt tarafındaki panele eklenir.

HelpProvider Kontrolünün Özellikleri

- **HelpNamespace:** Hazırlanmış olan yardım dosyasının ya da görüntülenmesi istenilen web adresinin belirtildiği özelliktir.

```
helpProvider1.HelpNamespace = Application.StartupPath + "\\yardim.chm";
```

… şeklinde kullanıldığında, **F1** tuşuna basıldığında uygulamanın çalışma klasöründe bulunan **yardim.chm** adlı yardım dosyasını görüntüler.

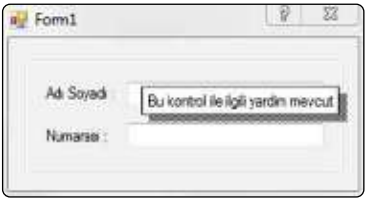
```
helpProvider1.HelpNamespace = "http://www.volkanaktas.com";
```

… şeklinde kullanıldığında ise, **F1** tuşuna basıldığında basıldığında belirtilen internet sayfası görüntülenir.

HelpProvider Kontrolünün Metodları

- **`SetHelpString`:** Belirtilen kontrol aktifleştirildiğinde görüntülenecek olan açıklama mesajını belirler. Görüntülenebilmesi için **`HelpNameSpace`** özelliğinin boş bırakılması gerekmektedir. **`SetHelpString`** özelliği ile belirlenmiş olan açıklama mesajını görüntüleyebilmek için **Form** penceresinde yardım simgesinin görüntülenmesi gerekmektedir. Yardım simgesini görüntüleyebilmek için ise tam ekran yapma ve simge durumuna küçültme simgeleri pasif olmalıdır.

```
private void Form1_Load(object sender, EventArgs e)
{
  this.HelpButton = true;
  this.MaximizeBox = false;
  this.MinimizeBox = false;
  helpProvider1.SetHelpString(txtAdSoyad,
  "Bu kontrol ile ilgili yardim mevcut");
}
```

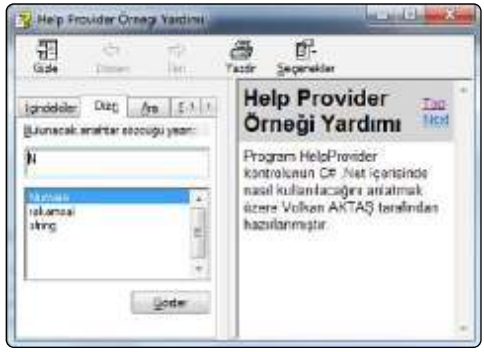


- **`GetHelpString`:** Belirtilen bir kontrol için **`SetHelpString`** metodu ile atanmış olan açıklama metnini görüntüler.
- **`SetShowHelp`: `true`** ya da **`false`** değerini alır. Belirtilen bir kontrol için yardım görüntülenip görüntülenmeyeceğini belirler.
- **`SetHelpKeyword`:** Yardım dosyasının belirtilen keyword'a göre açılmasını sağlar. Keyword'lar, yardım dosyası içerisinde arama esnasında ulaşımı kolaylaştırmak için tanımlanan anahtar kelimelerdir.
- **`GetHelpKeyword`:** Belirtilen bir konrol için **`SetHelpKeyword`** metodu ile atanmış olan anahtar kelimeyi görüntüler.

- **`SetHelpNavigator`:** Açılan yardım dosyasının açılış şeklini belirler. Şu değerleri alır:

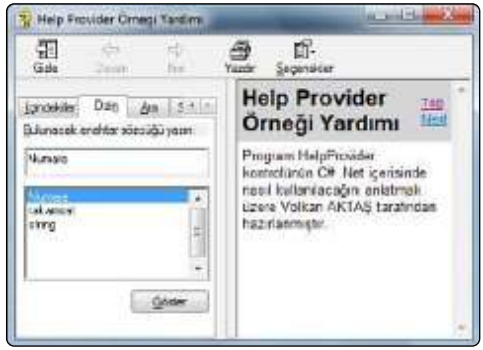
  `AssociateIndex`: Yardım dosyası açılırken **Dizin** sekmesinin aktif olarak açılmasını ve dizin bölümündeki arama kutusuna **`SetHelpKeyword`** metodunda yazılan keyword'un baş harfinin yazılmasını sağlar.

```
private void Form1_Load(object sender, EventArgs e)
{
   helpProvider1.HelpNamespace = Application.StartupPath + "\\yardim.chm";
   helpProvider1.SetHelpKeyword(txtAdSoyad, "Numara");
   helpProvider1.SetHelpNavigator(txtAdSoyad, HelpNavigator.AssociateIndex);
   helpProvider1.SetShowHelp(txtAdSoyad, true);
}
```



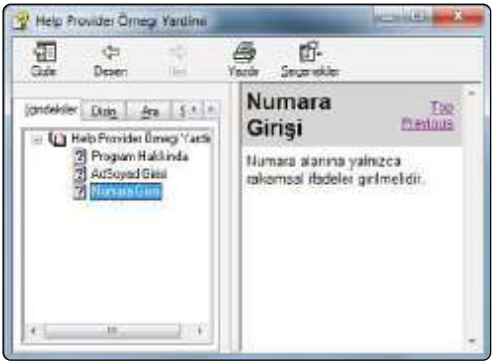
`Find`: Yardım dosyası açılırken **Ara** sekmesinin aktif olarak açılmasını sağlar.

`Index`: Yardım dosyası açılırken **Dizin** sekmesinin aktif olarak açılmasını sağlar.

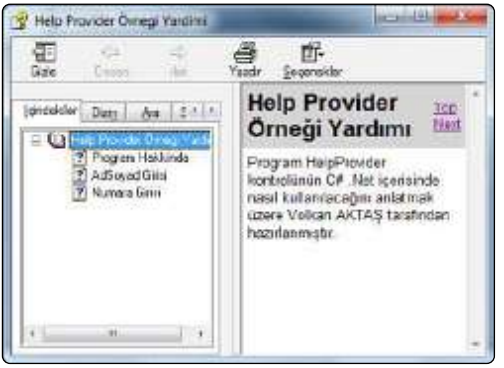


`KeywordIndex`: Yardım dosyası açılırken **`SetHelpKeyword`** metodu ile belirtilmiş keyword'un bulunduğu ilgili yardım bölümünün görüntülenmesini sağlar.

```
private void Form1_Load(object sender, EventArgs e)
{
   helpProvider1.HelpNamespace = Application.StartupPath + "\\yardim.chm";
   helpProvider1.SetHelpKeyword(txtAdSoyad, "Numara");
   helpProvider1.SetHelpNavigator(txtAdSoyad, HelpNavigator. KeywordIndex);
   helpProvider1.SetShowHelp(txtAdSoyad, true);
}
```

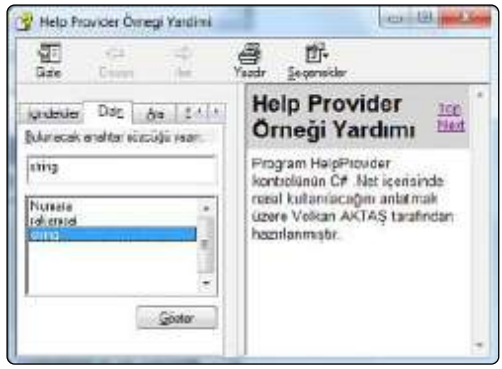


`TableOfContents`: Yardım dosyası açılırken İçindekiler sekmesinin aktif olarak açılmasını sağlar.



`Topic`: Yardım dosyası içerisine eklenmiş bir HTML dosyası varsa görüntülenmesini sağlar.

TopicId: Yardım dosyası açılırken **Dizin** sekmesinin aktif olarak açılmasını sağlayarak **SetHelpKeyword** metodu ile belirtilmiş keyword'un **Dizin** bölümünde seçili olmasını sağlar.



- **GetHelpNavigator:** Belirtilen bir kontrol için **SetHelpNavigator** metodu ile atanmış olan yardım dosyası açılış şekli değerini verir.
- **ResetShowHelp:** Bir kontrol için atanmış olan yardım dosyası atamasını siler.

DateTimePicker Kontrolü

Açılabilen pencere şeklinde tarih ve zamanı görüntülemek için kullanılır **DateTimePicker** kontrolü eklendiğinde varsayılan olarak o anki tarih ve zamanı gösterir.

DateTimePicker Kontrolünün Özellikleri

- **Format:** Görüntülenecek olan tarih veya zamanın gösterim biçimini belirler. Aşağıdaki değerleri alır:

  Long: Tarih ve zamanı birlikte görüntüler. 26 Temmuz 2014 Cumartesi

  Short: Yalnızca tarihi görüntüler. 26. 7.2014

  Time: Yalnızca zamanı görüntüler. 13:32:01

  Custom: Bu özellik seçildiğinde, **CustomFormat** özelliği ile kullanıcı tarafından özel bir tarih ya da zaman formatı belirlenebilir.

- **CustomFormat:** Kullanıcı tarafından özel bir tarih formatı belirtmek amacı ile kullanılır. Bunun için kullanılacak karakterler ve anlamları aşağıdaki gibidir:

  d: Gün bilgisini verir. 1 ile 31 arası değer alır.

dd: İki basamaklı olarak gün bilgisini verir. 1 ile 31 arası değer alır.

ddd: Gün bilgisini kısaltarak görüntüler. Örneğin; **Pazartesi** gününü **Pzt** olarak görüntüler.

dddd: Gün bilgisini tam hali ile görüntüler. Örneğin; **Pazartesi** gibi...

m: Ay bilgisini verir. 1 ile 12 arası değer alır.

mm: İki basamaklı olarak ay bilgisini verir. 1 ile 12 arası değer alır.

mmm: Ay bilgisini kısaltarak görüntüler. Örneğin; **Ağustos** ayını **Ağu** olarak görüntüler.

mmmm: Ay bilgisini tam hali ile görüntüler. Örneğin; **Ağustos** gibi...

y: Yıl bilgisinin son basamağını verir. **2009** yılı için 9 rakamı görüntülenir.

yy: Yıl bilgisinin son iki basamağını verir. 2009 yılı için 09 rakamı görüntülenir.

yyy: Yıl bilgisini tam hali ile görüntüler. Yıl bilgisi 2009 şekinde görüntülenir.

yyyy: Yıl bilgisini tam hali ile görüntüler. Yıl bilgisi 2009 şekinde görüntülenir.

h: Saat bilgisini tek basamaklı olarak görüntüler. 1 ile 12 arası değer alır.

hh: Saat bilgisini iki basamaklı olarak görüntüler. 1 ile 12 arası değer alır.

H: Saat bilgisini tek basamaklı olarak görüntüler. 1 ile 24 arası değer alır.

HH: Saat bilgisini iki basamaklı olarak görüntüler. 01 ile 24 arası değer alır.

m: Dakika bilgisini tek basamaklı olarak görüntüler. 0 ile 59 arası değer alır.

mm: Dakika bilgisini iki basamaklı olarak görüntüler. 00 ile 59 arası değer alır.

s: Saniye bilgisini tek basamaklı olarak görüntüler. 0 ile 59 arası değer alır.

ss: Saniye bilgisini iki basamaklı olarak görüntüler. 00 ile 59 arası değer alır.

Bu karakterleri kullanarak örnek formatlar tanımlayalım…

26/07/2014 tarihi ve **13:32:01** zamanı için işlemler yapalım.

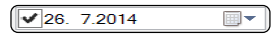
hh:mm:ss `:01:32:01` değerini görüntüler.

HH:mm:ss `:13:32:01` değerini görüntüler.

dd MM yyyy `:26 07 2014` değerini görüntüler.

dd MMMM yyyy `:26 Temmuz 2014` değerini görüntüler.

HH:mm:ss - dd MMMM yyyy `:26 Temmuz 2014 - 13:32:01` değerini görüntüler.

- **ShowCheckBox: True** değerini aldığında **DateTimePicker** kontrolünün sol tarafında **CcheckBox** bulunmasını sağlar. 
- **Checked: True** değerini aldığında **DateTimePicker** kontrolünün sol tarafında bulunan **CheckBox**'ın işaretli olmasını sağlar. **False** değerini aldığında ise işaret kaldırılır.
- **ShowUpDown: True** değerini aldığında **DateTimePicker** kontrolünün sağ tarafında tarih ya da zaman değerleri arasında ileri geri hareket edilmesini sağlayan yukarı aşağı ok işaretinin bulunmasını sağlar. 
- **Value: DateTimePicker** kontrolünün aldığı tarih ve zaman değerini verir. Görüntülenen tarih ve zaman formatı değiştirilebilir. Şu format değerlerini alabilir:

  ToLongDateString: 26 Temmuz 2014 Cumartesi örneğinde olduğu gibi tarihi görüntülerken, hem ay hem de gün ismi ile beraber görüntüler.

  ToShortDateString: 26.07.2014 örneğinde olduğu gibi tarihi rakamlar şeklinde görüntüler.

  ToLongTimeString: 13:32:01 örneğinde olduğu gibi zamanı **saat: dakika:saniye** şeklinde görüntüler.

  ToShortTimeString: 13:32 örneğinde olduğu gibi zamanı saat: dakika şeklinde görüntüleyip, saniyeyi görüntülemez.

- **MinDate: DateTimePicker** kontrolünün alabileceği minimum tarih değerini belirler. Varsayılan değeri **01.01.1753** tarihidir.
- **MaxDate: DateTimePicker** kontrolünün alabileceği maksimum tarih değerini belirler. Varsayılan değeri **31.12.9998** tarihidir.

Tarih zaman formatını **CustomFormat** özelliği ile değiştiren örnek bir uygulama yapalım.

```
private void Form1_Load(object sender, EventArgs e)
{
 dateTimePicker1.Format = DateTimePickerFormat.Custom;
 dateTimePicker1.CustomFormat = "hh:mm:ss - dd MMMM yyyy";
}
```

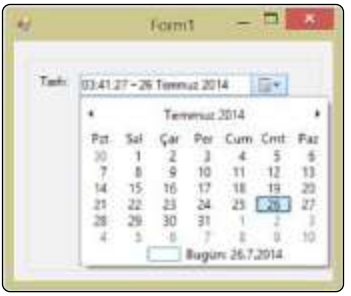
Kodumuzda…

```
dateTimePicker1.Format = DateTimePickerFormat.Custom;
```

… satırında **`CustomFormat`** seçeneği ile özel bir tarih formatı seçeceğimizi belirttik.

```
dateTimePicker1.CustomFormat = "hh:mm:ss - dd MMMM yyyy";
```

… satırında ise **`CustomFormat`** biçimi olarak, önce saat sonra tarihin görüntüleneceği format biçimini belirledik.

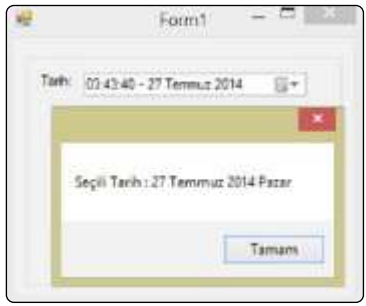


DateTimePicker Kontrolünün Olayları

`ValueChanged`: DateTimePicker kontrolünün görüntülediği tarih değiştirildiğinde gerçekleşen olaydır.

```
private void Form1_Load(object sender, EventArgs e)
{
 dateTimePicker1.Format = DateTimePickerFormat.Custom;
 dateTimePicker1.CustomFormat = "hh:mm:ss - dd MMMM yyyy";
}
private void dateTimePicker1_ValueChanged(object sender, EventArgs e)
{
 MessageBox.Show("Seçili Tarih : " +
dateTimePicker1.Value.ToLongDateString());
}
```

Kodumuzda **DateTimePicker** kontrolü içerisinde herhangi bir tarih değiştirildiğinde, mesaj penceresi içerisinde seçili tarih görüntülenmektedir.

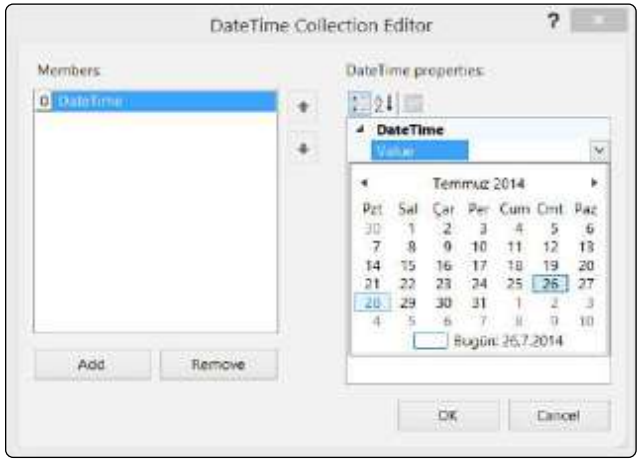


MonthCalendar Kontrolü

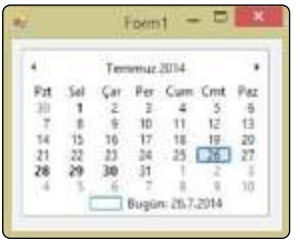
Bir tarihi grafiksel olarak göstererek bu tarihler arasında gezinmeyi sağlayan bir kontroldür.

MonthCalendar Kontrolünün Özellikleri

- **`BoldedDates`:** Koyu renkli olarak görüntülenecek olan günlerin listesini belirler. **DateTimeCollection Editor** yardımı ile günler eklenebilir.



- **`AnnuallyBoldedDates`: `BoldedDates`** özelliğinde olduğu gibi koyu renkli olarak görüntülenecek olan günlerin listesini belirler. **`BoldedDates`**'den farklı olarak belirlenmiş tarihler, her yılda koyu olarak görüntülenecektir.



Örneğin; takvimimizde **`BoldedDates`** özelliğine değer olarak **`28, 29, 30 Temmuz 2014`** tarihlerini ekleyelim. **`AnnuallyBoldedDates`** özelliğine ise, değer olarak **`1 Temmuz 2014`** tarihlerini ekleyelim.



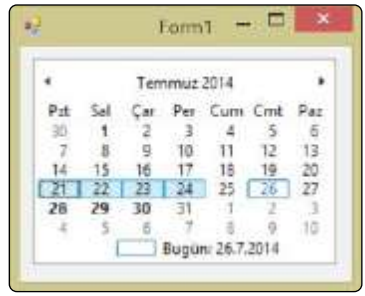
Tarihi değiştirip **`Temmuz 2015`** yaptığımızda **`AnnuallyBoldDates`** özelliği ile eklediğimiz **`1Temmuz`** gününün koyu renkli olarak gösterildiğini, ancak **`BoldedDates`** özelliği ile eklediğimiz **`28, 29, 30 Temmuz`** tarihlerinin koyu renkli olarak gösterilmediğini görüyoruz.

- **`CalendarDimensions`**: **MonthCalendar** nesnesinin **`x`** ve **`y`** koordinatlarında kaç adet ayı görüntüleyeceğini belirler. Örneğin; bu değeri **`2;2`** yaptığımızda aşağıdaki gibi bir görünüm elde ederiz:

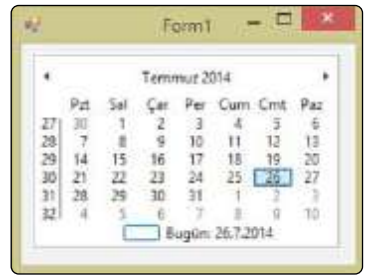


- **`FirstDayOfWeek`**: Haftanın ilk günü olarak hangi günün görüntüleneceğini belirler.
- **`MaxDate`**: MonthCalendar nesnesinin görüntüleyebileceği maksimum tarihi belirler. Varsayılan değeri **`31.12.9998`** tarihidir.
- **`MinDate`**: MonthCalendar nesnesinin görüntüleyebileceği minimum tarihi belirler. Varsayılan değeri **`01.01.1753`** tarihidir.

- **`MaxSelectionCount`:** Aynı anda seçilebilecek en fazla gün sayısını belirler. Varsayılan değeri `7`'dir.
- **`MonthlyBoldedDates`:** Her ayın belli bir gün ya da günlerini koyu renkli olarak görüntülemek amacı ile kullanılır.
- **`SelectionRange`:** Seçili olarak görüntülenecek tarih aralığını belirler. Örneğin; bu özelliğe `21.07.2014; 24.07.2014` değerini girdiğimizde 21-24 Ekim 2014 günleri seçili olarak görüntülenir. `Start` metodu ile seçimin başlangıç noktası, `end` metodu ile de bitiş noktası belirlenebilir ya da belirlenmişse görüntülenebilir.



- **`ScrollChange`:** Ok tuşları ile aylar arasında geçiş yapılırken her defasında kaç ayın atlanacağını belirler. Varsayılan değeri `0`'dır. Örneğin; bu özelliğe `2` değerini verdiğimizde, **Ağustos** ayı seçili iken ileri doğru hareket etmek için ok tuşuna bastığımızda **Ekim** ayına hareket edilecektir.
- **`ShowToday`:** "*Bugün: Tarih*" şeklinde o günkü tarih yazısının görüntülenip görüntülenmeyeceğini belirler.
- **`ShowTodayCircle`:** Bugünün tarihi yazısının yan tarafındaki çerçevenin görüntülenip görüntülenmeyeceğini belirler.
- **`ShowWeekNumber`:** **MonthCalendar** nesnesi üzerinde aktif tarihin yılın kaçıncı haftası olduğuna dair bir cetvel görüntülenip görüntülenmeyeceğini belirler. `True` değeri verilip bu cetvel görüntülendiğinde, görünüm yandaki gibi olacaktır.



- **`TodayDate`:** Bugünün tarihi olarak otomatik olarak belirlenen tarihten farklı bir tarih girilmek isteniyorsa, bu özellik ile belirlenebilir.

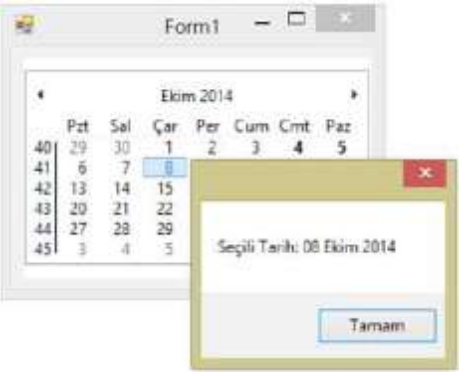
MonthCalendar Kontrolünün Metodları

- **AddBoldedDate:** Koyu renkli olarak görüntülenecek olan günlerin belirlenmesi amacı ile kullanılır.
- **AddAnnuallyBoldedDate:** **AddBoldedDate** özelliğinde olduğu gibi koyu renkli olarak görüntülenecek olan günlerin belirlenmesi amacı ile kullanılır. **AddBoldedDate** ile eklenmiş günlerden farklı olarak belirlenmiş tarihler her yılda koyu olarak görüntülenecektir.
- **UpdateBoldedDates:** Koyu renkli olarak görüntülenmek üzere eklenmiş olan günlerin görüntülenmesi amacı ile kullanılır.

MonthCalendar Kontrolünün Olayları

- **DateSelected:** **MonthCalendar** kontrolü üzerinde bir tarih seçilip, görüntülenen tarih değiştirildiğinde gerçekleşen olaydır.

```
private void Form1_Load(object sender, EventArgs e)
{
 monthCalendar1.ShowToday = false;
 monthCalendar1.ShowTodayCircle = false;
 monthCalendar1.AddBoldedDate(DateTime.Parse("2014/10/4"));
 monthCalendar1.AddBoldedDate(DateTime.Parse("2014/10/5"));
 monthCalendar1.UpdateBoldedDates();
}
private void monthCalendar1_DateSelected(object sender,
DateRangeEventArgs e)
{
 MessageBox.Show("Seçili Tarih: " +
    monthCalendar1.SelectionRange.Start.ToString("dd MMMM yyyy"));
}
```



Örneğimizde `AddBoldedDate` metodu ile manuel olarak tarihleri ekleyerek, koyu renkli olarak gösterilmesini sağladık. Tarihleri ekledikten sonra değişikliğin aktif olması için `UpdateBoldedDates()` metodunu kullandık. Tarih seçidiğinde mesaj penceresinin görüntülenmesi için `DateSelected` olayı içerisinde `SelectionRange.Start.ToString()` metodu ile seçili tarihi görüntüledik.

- `DateChanged`: **MonthCalendar** kontrolünün üzerinde görüntülenen tarih değiştirildiğinde gerçekleşen olaydır.

DriveListBox Kontrolü

ToolBox üzerinde standart olarak yer almayan bu kontrol, `Microsoft.VisualBasic.Compatibility.VB6` sınıfı altında yer almaktadır. **ToolBox** üzerine farenin sağ tuşu ile tıklayarak **Choose Items** seçeneği seçilerek gelen pencereden **.NET Framework Components** sekmesi altından seçilerek **ToolBox** içerisine eklenir. Bilgisayarda mevcut bulunan sürücü isimlerini listelemek amacı ile kullanılır.

DriveListBox Kontrolünün Özellikleri

- `Drive`: Bilgisayarda mevcut bulunan sürücü isimlerini listeler.

DirListBox Kontrolü

ToolBox üzerinde standart olarak yer almayan bu kontrol, `Microsoft.VisualBasic.Compatibility.VB6` sınıfı altında yer almaktadır. Seçili olan sürücü içerisindeki klasör ya da bir klasör içerisindeki alt klasörleri listelemek amacı ile kullanılır.

DirListBox Kontrolünün Özellikleri

- `Path`: İçerisinde bulunulan klasör bilgisini verir.

FileListBox Kontrolü

ToolBox üzerinde standart olarak yer almayan bu kontrol, `Microsoft.VisualBasic.Compatibility.VB6` sınıfı altında yer almaktadır. Seçili olan klasör içerisindeki dosyaları listelemek amacı ile kullanılır.

FileListBox Kontrolünün Özellikleri

- **Path**: İçerisinde bulunulan klasör bilgisini verir.
- **FileName**: Seçili olan dosya ismi bilgisini verir.
- **Pattern**: **FileListBox** içerisinde görüntülenecek olan dosya türlerini belirler. Dosya türleri yazılırken aralarına ; işareti konulur.

  "*.bmp;*.jpg;*.jpeg;*.png;*.gif"

- **Archive**: Arşiv dosyalarını listelemek amacı ile kullanılır.
- **Hidden**: Gizli dosyalarını listelemek amacı ile kullanılır.
- **System**: Sistem dosyalarını listelemek amacı ile kullanılır.
- **ReadOnly**: Yalnız okunabilir dosyaları listelemek amacı ile kullanılır.

Örnek olarak dosyaları ve klasörleri listeleyen ve bir dosya ismine çift tıklandığında çalıştıran bir uygulama yapalım.

```
private void driveListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
  dirListBox1.Path = driveListBox1.Drive;
}
private void dirListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
  fileListBox1.Path = dirListBox1.Path;
}
private void fileListBox1_DoubleClick(object sender, EventArgs e)
{
  int n=fileListBox1.Path.IndexOf("\\");

  if (fileListBox1.Path == fileListBox1.Path.Substring(n,1))
 System.Diagnostics.Process.Start
    (fileListBox1.Path + fileListBox1.FileName);
  else
 System.Diagnostics.Process.Start
    (fileListBox1.Path + "\\" + fileListBox1.FileName);
}
private void Form1_Load(object sender, EventArgs e)
{
  cmbDosyaTuru.SelectedIndex = 0;
  cmbNitelik.SelectedIndex = 0;
```

```
  fileListBox1.Pattern = cmbDosyaTuru.Text;
  fileListBox1.Archive = true;
}
private void cmbNitelik_SelectedIndexChanged(object sender, EventArgs e)
{
  if (cmbNitelik.Text == "Arşiv Dosyaları")
       fileListBox1.Archive = true;
  if (cmbNitelik.Text == "Gizli Dosyalar")
       fileListBox1.Hidden = true;
  if (cmbNitelik.Text == "Sistem Dosyaları")
       fileListBox1.System = true;
  if (cmbNitelik.Text == "Yalnız Okunabilir Dosyalar")
       fileListBox1.ReadOnly = true;
  fileListBox1.Refresh();
}
private void cmbDosyaTuru_SelectedIndexChanged(object sender, EventArgs e)
{
  fileListBox1.Pattern = cmbDosyaTuru.Text;
}
```

Örneğimizde…

```
int n=fileListBox1.Path.IndexOf("\\");
```

…. satırında **`IndexOf`** metodu ile içerisinde bulunduğumuz klasör bilgisinde, \ işaretine kadar olan kısmı alıyoruz.

```
if (fileListBox1.Path == fileListBox1.Path.Substring(n,1))
```

… satırında **FileListBox** nesnesinin **`Path`** bilgisinin, **`Path`** içerisindeki \ işaretine kadar olan kısmına eşit olup olmadığını sorguluyoruz. Örneğin **`Path`** bilgisi `C:\` olduğunda, bu bilgi \ işaretine kadar olan kısma eşit olacaktır. Alt klasörlerde ise bu bilgi eşit olmayacaktır.

Kök dizinde iken dosya bilgisini..

```
fileListBox1.Path + fileListBox1.FileName
```

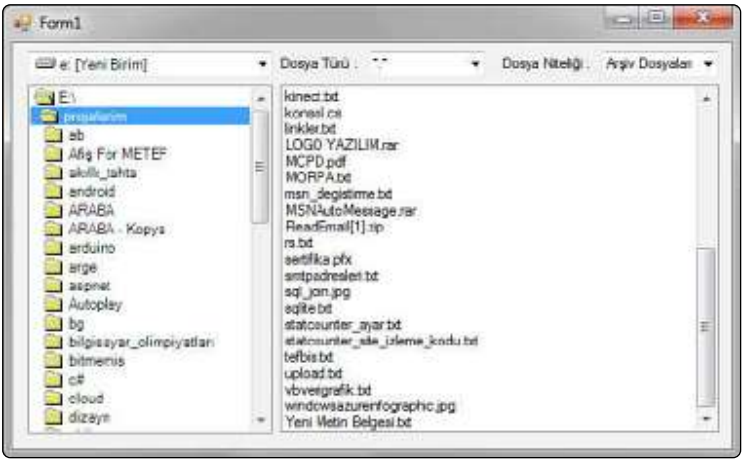
… şeklinde elde ederken, alt klasörde iken…

```
fileListBox1.Path + "\\" + fileListBox1.FileName
```

… şeklinde elde etmekteyiz.

```
System.Diagnostics.Process.Start
```

Metodu ile de **FileListBox** içerisinde seçili olan dosya çift tıklanıldığında çalıştırıyoruz.

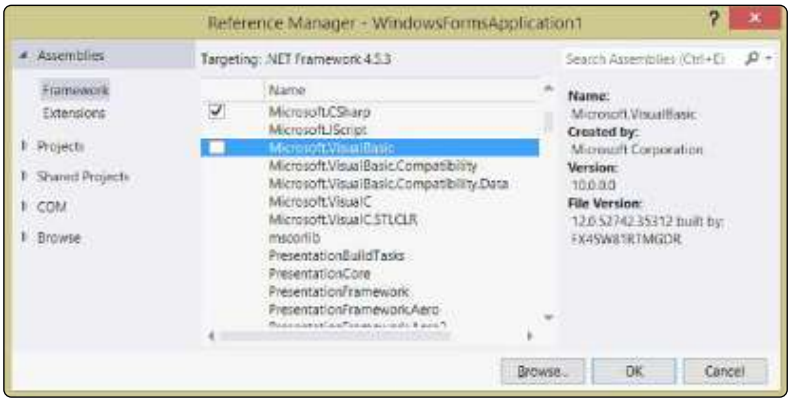


DİYALOG PENCERELERİ

11

INPUTBOX (GİRİŞ PENCERESİ)

C# içerisinde **InputBox** nesnesi mevcut değildir. Bunu kullanabilmek için `Microsoft.VisualBasic` sınıfını kullanmamız gerekmektedir. **Project / Add Reference** seçeneği ile **Assemblies / Framework** sekmesi altından `Microsoft.VisualBasic` sınıfını projemize referans olarak eklemeliyiz.



Ayrıca `using` anahtar sözcüğünü kullanarak `Microsoft.VisualBasic` sınıfını projemize eklemeliyiz.

InputBox, veri girişi amacı ile kullanılan giriş pencereleridir.

Bir değişkene atanarak kullanılırlar. Kullanımı şu şekildedir:

```
Inputbox("Görüntülenecek olan mesaj", "Pencere başlığı", "Varsayılan
mesaj", x pozisyonu, y pozisyonu)
```

InputBox içerisinde görüntülenecek olan mesajın uzunluğu en fazla **1024** karakter olabilir.

Örneğin…

```
using System;
using Microsoft.VisualBasic;
using System.Windows.Forms;
namespace InputBox_NotGirisi
{
 public partial class Form1 : Form
 {
   public Form1()
   {
    InitializeComponent();
   }
   private void button1_Click(object sender, EventArgs e)
   {
  byte notu;
  notu = Convert.ToByte(Interaction.InputBox("Notu Girin", Not
Girişi", "100", 10, 10));
   }
 }
}
```

… şeklinde kullanıldığında ekranın sol üst köşesi olan 100,100 koordinatlarında pencere çıktısı verir.



InputBox ile iki adet notu girişi yapıp ortalamasını bulan bir örnek uygulama yapalım.

```
private void btnislemyap_Click(object sender, EventArgs e)
{
  byte not1 = Convert.ToByte(Interaction.InputBox("1. Notu Girin",
"Not Girişi", "100", 10, 10));
  byte not2 = Convert.ToByte(Interaction.InputBox("2. Notu Girin",
"Not Girişi", "100", 10, 10));
  MessageBox.Show("Ortalama= " + (not1 + not2) / 2);
}
```

Kodumuzda **InputBox** içerisine girilen değerler **`string`** türde olduğu için, **`Convert.ToByte`** metodu ile byte türüne dönüştürdük.

Not girişine dair yaptığımız örneği geliştirip, **InputBox** üzerinde bulunan **Cancel** butonuna basılıp basılmadığını tespit edelim.

```
private void btnislemyap_Click(object sender, EventArgs e)
{
  string not1 = Interaction.InputBox("Notu Girin", "Not Girişi",
    "100", 10, 10);
  if (not1.Length == 0)
  {
   System.Environment.Exit(0);
  }
   else
  {
   string not2 = Interaction.InputBox("Notu Girin", "Not Girişi",
"100", 10, 10);
   MessageBox.Show("Ortalama= " + (Convert.ToByte(not1) +
Convert.ToByte(not2)) / 2);
  }
}
```

Örneğimizde daha önce byte **`not1`** ve byte **`not2`** şeklinde olan tanımlamamızı string **`not1`** ve string **`not2`** şeklinde string olarak tanımladık. Çünkü **InputBox** içerisinde **Cancel** butonuna basılıp basılmadığını, string değişkenlerin uzunluğunu bulan **`Length`** metodu ile belirledik. Eğer **`Length`** metodu **`0`**

değerini alıyorsa, **InputBox** içerisinde **Cancel** butonuna basılmıştır ya da **InputBox** içerisinde değer girişi yapılmamıştır. Bu durumda `Environment.Exit(0)` komutu ile programın çalışmasını sona erdiriyoruz.

Örneğimizde `not1` ve `not2` değişkenleri artık string tipte olduğu için toplama işlemi öncesinde `Convert.ToByte` metodu ile bu değişkenlerin aldıkları string türdeki değerleri byte değişken türüne çeviriyoruz.

MessageBox (Mesaj Penceresi)

Program içerisinde üretilen sonuçların ekranda gösterilmesi amacı ile kullanılan pencerelerdir. Kullanım biçimi aşağıdaki gibidir;

```
MessageBox.Show( Mesaj, Başlık, Buton Tipi, Simge);
```

- **`Mesaj`:** mesaj penceresinde görüntülenecek olan mesajdır. `string` veri türünden değer alır. Mesajın uzunluğu en fazla **1024** karakter olabilir.
- **`Tip`:** Mesaj penceresi üzerinde görüntülenecek olan butonları belirtir. Aşağıdaki işlemler yapılabilir.

  Mesaj penceresi üzerinde görüntülenecek buton tipleri belirlenebilir. Şu değerleri alır:

  `MessageBoxButtons.OKCancel`: Mesaj penceresinde **Tamam**, **İptal** düğmeleri görüntülenir.

  `MessageBoxButtons.OK`: Mesaj penceresinde **Tamam** düğmesi görüntülenir.

  `MessageBoxButtons.AbortRetryIgnore`: Mesaj penceresinde **Durdur**, **Yeniden Dene**, **Yoksay** düğmeleri görüntülenir.

  `MessageBoxButtons.RetryCancel`: Mesaj penceresinde **Yeniden Dene**, **İptal** düğmeleri görüntülenir.

  `MessageBoxButtons.YesNo`: Mesaj penceresinde **Evet**, **Hayır** düğmeleri görüntülenir.

  `MessageBoxButtons.YesNoCancel`: Mesaj penceresinde **Evet**, **Hayır**, **İptal** düğmeleri görüntülenir.

- **`Simge`:** Mesaj penceresinde görüntülenecek **simgeleri** belirlemek mümkündür.

Kullanabileceğimiz simgeler aşağıdaki gibidir:

`MessageBoxIcon.Error`

`MessageBoxIcon.Hand`

`MessageBoxIcon.Stop`

Dur işaretini görüntüler.



`MessageBoxIcon.Exclamation`

`MessageBoxIcon.Warning`

Ünlem işaretini görüntüler.



`MessageBoxIcon.Information`

`MessageBoxIcon.Asterisk`

Bilgi işaretini görüntüler.



`MessageBoxIcon.Question`

Soru işaretini görüntüler.



Mesaj penceresi üzerinde buton ve simgelerin birlikte kullanımı ile ilgili bir kaç örnek yapalım.

İlk örneğimizde üzerinde **uyarı işareti** ve **tamam**, **iptal** düğmeleri olan bir mesaj penceresi görüntületelim.

```
private void button1_Click(object sender, EventArgs e)
{
 MessageBox.Show("Mesaj", "Başlık", MessageBoxButtons.OKCancel,
  MessageBoxIcon.Information);
}
```

Kodumuz aşağıdaki gibi sonuç verecektir.



Birden fazla satırdan oluşacak şekilde mesaj içeriklerini yazdırmak için **`\n`** karakteri ya da **`Environment.NewLine`** ifadesi kullanılabilir.

Örneğin…

```
MessageBox.Show ("Birinci Satır \n" + "İkinci Satır", "Başlık",
   MessageBoxButtons.OKCancel,MessageBoxIcon.Information);
```

… kod satırı ile…

```
MessageBox.Show ("Birinci Satır" + Environment.NewLine + "İkinci Satır",
"Başlık",MessageBoxButtons.OKCancel,MessageBoxIcon.Information);
```

Kod satırının her ikisi de aşağıdaki sonucu verecektir.

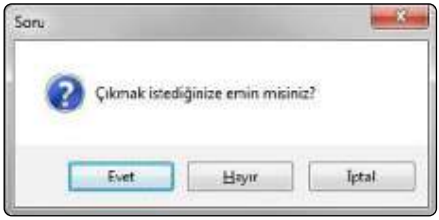


Sıradaki örneğimizde ise üzerinde **soru işareti** ve **evet**, **hayır**, **iptal** düğmeleri olan bir mesaj penceresi görüntületelim.

Kodumuz aşağıdaki gibi olacak:

```
private void button1_Click(object sender, EventArgs e)
{
 MessageBox.Show("Çıkmak istediğinize emin misiniz?", "Soru",
  MessageBoxButtons. YesNoCancel,MessageBoxIcon. Question);
}
```

Kodumuz aşağıdaki sonucu verecektir.



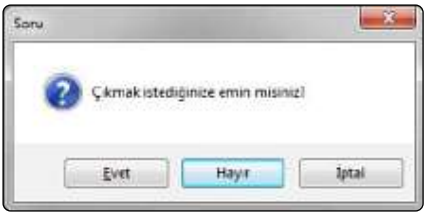
Mesaj penceresi üzerindeki butonlardan herhangi birini, varsayılan buton olarak seçip aktifleştirmek mümkündür. Bunun için aşağıdaki değer ya da sabitler kullanılabilir.

Değer	Görevi
MessageBoxDefaultButton.DefaultButton1	İlk buton aktif
MessageBoxDefaultButton.DefaultButton2	İkinci buton aktif
MessageBoxDefaultButton.DefaultButton3	Üçüncü buton aktif

Üzerinde **evet** ve **hayır** butonları ile soru işareti olan ve aktif butonu hayır olan bir pencere için kod yazalım.

```
private void button1_Click(object sender, EventArgs e)
{
 MessageBox.Show("Çıkmak istediğinize emin misiniz?", "Soru",
  MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question,
  MessageBoxDefaultButton.Button2);
}
```

Kodumuzu çalıştırdığımızda pencere görüntüsünün aşağıdaki gibi olduğunu görürüz. **Hayır** butonu seçili biçimdedir.



Mesaj penceresi üzerindeki yazıları sağa yanaşık olarak yazmak istersek, **`MessageBoxOptions.RightAlign`** değerini kullanırız. Örneğin; …

```
private void button1_Click(object sender, EventArgs e)
{
 MessageBox.Show("Çıkmak istediğinize emin misiniz?", "Soru",
  MessageBoxButtons. YesNo,MessageBoxIcon. Question,0,
  MessageBoxOptions.RightAlign);
}
```

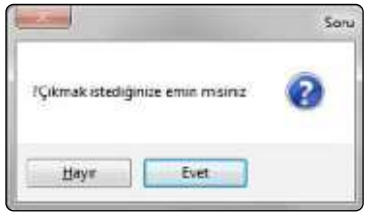
… şeklinde yazacağımız kod yazıları sağa dayalı mesaj penceresi üretecektir. Kodumuzda kullanmadığımız **`MessageBoxDefaultButton`** parametresinin yerine 0 yazıyoruz.



Arapça gibi sağdan başlayan yazıları mesaj pencerelerini kullanmak için **`MessageBoxOptions.RtlReading`** değeri kullanılabilir. Örneğin…

```
private void button1_Click(object sender, EventArgs e)
{
 MessageBox.Show("Çıkmak istediğinize emin misiniz?", "Soru",
  MessageBoxButtons. YesNo,MessageBoxIcon. Question,0,
  MessageBoxOptions.RtlReading);
}
```

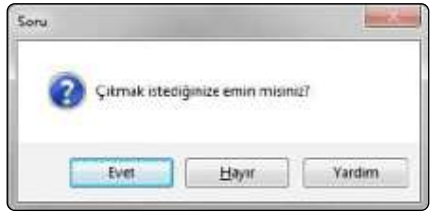
… şeklinde yazacağımız kod şekildeki sonucu üretecektir.



Mesaj pencerelerinde yardım düğmesini görüntülemek için mesaj penceresinin son parametresi olarak **`true`** değerini yazmalıyız. Örneğin…

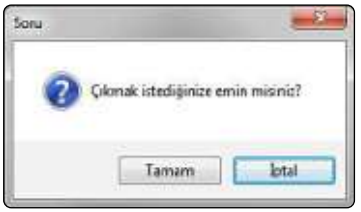
```
private void button1_Click(object sender, EventArgs e)
{
 MessageBox.Show("Çıkmak istediğinize emin misiniz?", "Soru",
  MessageBoxButtons.YesNo, MessageBoxIcon.Question,0,0,true);
}
```

… şeklinde yazdığımız kodu çalıştırdığımızda üzerinde yardım butonu da bulunan bir mesaj penceresi görüntülenecektir. Kodumuzda kullanmadığımız **`MessageBoxDefaultButton`** ve **`MessageBoxOptions`** parametrelerinin yerine `0` yazıyoruz.



Mesaj penceresi üzerindeki butonlardan hangisine basılıp basılmadığını anlayabilmek için **`DialogResult`** nesnesini kullanabiliriz. **`DialogResult`** bize basılan tuşu döndürecektir.

```
private void btnCikis_Click(object sender, EventArgs e)
{
 DialogResult  basilan_tus;
 basilan_tus=MessageBox.Show("Çıkmak istediğinize emin misiniz?",
    "Soru",MessageBoxButtons.OKCancel,MessageBoxIcon.Question,
     MessageBoxDefaultButton.Button2);
 if (basilan_tus==DialogResult.OK) Application.Exit();
}
```



Eğer basılan tuş **Tamam** butonu ise, `Application.Exit` metodu ile uygulamamızdan çıkılmaktadır. **İptal** butonuna basılırsa, uygulamamız çalışmaya devam etmektedir.

File Dialog Pencereleri



Diyalog pencereleri **toolBox** içerisinde **Dialogs** başlığı altında bulunurlar. `ColorDialog`, `FolderBrowserDialog`, `FontDialog`, `OpenFileDialog` ve `SaveFileDialog` adı ile 5 adet Dialog kontrolü vardır.

Diyalog pencerelerini görüntülemek için `ShowDialog` metodu kullanılırken, diyalog penceresi üzerinde basılan tuş `DialogResult` özelliği ile geriye döndürülmektedir.

Color Dialog

Renk seçme amaçlı olarak kullanılan diyalog penceresidir. `Color` özelliği ile `colorDialog` penceresi içerisinde seçilen renk değerini elde edebiliriz.

Örneğin; bir **Form**'un rengini **colorDialog** nesnesi yardımı ile değiştiren örnek bir uygulama yapalım.

Form üzerine bir adet buton ve bir adet **colorDialog** nesnesi ekliyoruz. Kodumuz aşağıdaki gibi olacak:

```
private void btnRenkSec_Click(object sender, EventArgs e)
{
 DialogResult basilan_tus;
 basilan_tus = colorDialog1.ShowDialog();
 if (basilan_tus == DialogResult.OK)
     this.BackColor = colorDialog1.Color;
}
```

Kodumuzda **colorDialog** penceresi görüntülenerek, `basilan_tus` değişkenine **colorDialog** penceresi üzerinde basılan tuş aktarıldı. Eğer basılan tuş **Tamam**

butonu ise, **Form**'un zemin rengine **colorDialog** penceresinde seçili olan renk, `Color` özelliği kullanılarak aktarıldı.





Uygulamamızı çalıştırıp, **colorDialog** penceresinde rengimizi seçip **Tamam** düğmesini tıkladığımızda, **Form**'un renginin seçtiğimiz renkle değiştirildiğini görürüz.

Folder Browser Dialog

Klasör seçmek amacı ile kullanılır. `SelectedPath` özelliği ile `folderBrowserDialog` penceresi içerisinde seçilen klasör yolunu elde edebiliriz.

Örnek olarak **folderBrowserDialog** nesnesi yardımı ile seçilecek bir klasördeki Word belgelerini listeleyecek olan bir örnek uygulama yapalım.

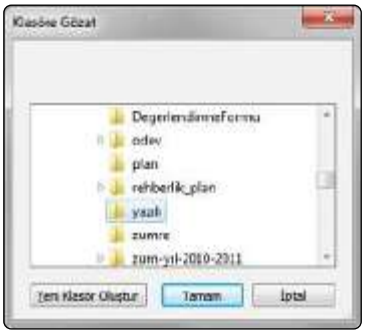
Form üzerine 1 adet **Button**, 1 adet **ListBox** ve 1 adet de **folderBrowserDialog** nesnesi ekleyelim.

Kodumuz aşağıdaki gibi olacak.

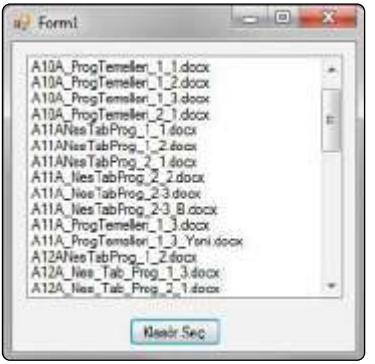
```
using System;
using System.IO;
using System.Windows.Forms;
namespace FolderBrowser_Dialog
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
  }
```

```
private void btnKlasorSec_Click(object sender, EventArgs e)
{
 DialogResult basilan_tus;
 basilan_tus = folderBrowserDialog1.ShowDialog();
 if (basilan_tus == DialogResult.OK)
 {
  DirectoryInfo dosya = new
   DirectoryInfo(folderBrowserDialog1.SelectedPath);
  FileInfo[] belgeler = dosya.GetFiles("*.doc");
  foreach( FileInfo b in belgeler)
    listBox1.Items.Add (b.Name);
   }
  }
 }
}
```

Kodumuzda **`DirectoryInfo`** sınıfını kullanarak **`folderBrowserDialog`** nesnesi içerisinde seçili olan klasör ile ilgili bilgileri elde ediyoruz. **`GetFiles`** metodunu kullanarak seçili olan klasörde, belirtilen kritere uyan dosyaları **`FileInfo`** türünden olan belgeler değişkenine aktarıyoruz. Daha sonra belgeler değişkeni içerisinde saklanan dosyaları **`foreach`** döngüsü içerisinde okuyarak, **ListBox** içerisine **`Add`** metodunu kullanarak ekliyoruz.



Kodumuzu çalıştırıp **Klasör Seç** butonuna tıklayalım ve karşımıza gelen **folderBrowserDialog** penceresi içerisinde bir klasör seçelim.



Daha sonra tamam butonuna tıkladığımızda Word belgelerinin isimlerinin **ListBox** nesnesi içerisine aktarıldığını görürüz.

Font Dialog

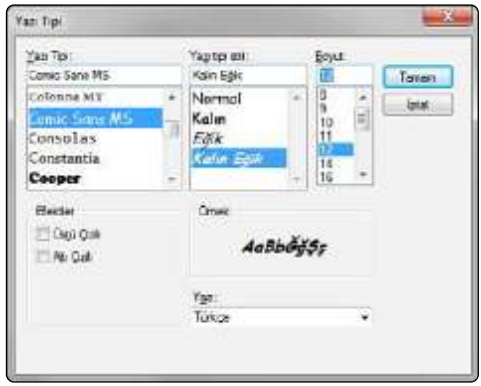
Yazı tipi ile ilgili özellikleri seçmek ve değiştirmek amaçlı kullanılır. Bir text kutusu içerisindeki yazının fontunu değiştiren bir örnek yapalım. **Form** üzerine 1 adet **Button**, 1 adet de **TextBox** ekleyeceğiz.

Kodumuz aşağıdaki gibi olacak:

```
private void btnYazıTipi_Click(object sender, EventArgs e)
{
 DialogResult basilan_tus;
 basilan_tus = fontDialog1.ShowDialog();
 if (basilan_tus == DialogResult.OK)
   textBox1.Font = fontDialog1.Font;
}
```

Kodumuzda **fontDialog** penceresi görüntülenerek, `basilan_tus` değişkenine **fontDialog** penceresi üzerinde basılan tuş aktarıldı. Eğer basılan tuş **Tamam** butonu ise, **TextBox** içerisindeki yazının `font` özelliğine, **fontDialog** penceresinde seçili olan font özellikleri aktarıldı. Kodumuzu çalıştıralım. **TextBox** içerisine yazdığımız yazının ilk hali aşağıdaki gibidir.





Yazı Tipi Özellikleri butonuna tıklayıp yazı tipi özelliklerini seçelim.



Seçtiğimiz özelliklerin **TextBox** içerisindeki yazıya uygulandığını görürüz.

Open File Dialog

Bilgisayarda daha önceden kayıtlı olan dosyaları açmaç amaçlı kullanılır. Kullanılan başlıca metotları şunlardır:

- **`FileName`:** Diyalog penceresindeki **dosya adı** bölümünde varsayılan olarak görüntülenecek metni belirler.
- **`Filter`:** Diyalog penceresinde görüntülenecek olan **dosya türlerini** belirlemek için kullanılır.

```
Dosya türü adı (*.dosya türü) |*. Dosya türü| Dosya türü adı(*.Dosya türü)|*. Dosya türü
```

… şeklinde kullanılır.

- **`Title`:** Diyalog penceresinin başlığında görüntülenecek olan yazıyı belirler.
- **`InitialDirectory`:** Diyalog penceresinde görüntülenecek olan varsayılan klasörü belirler.

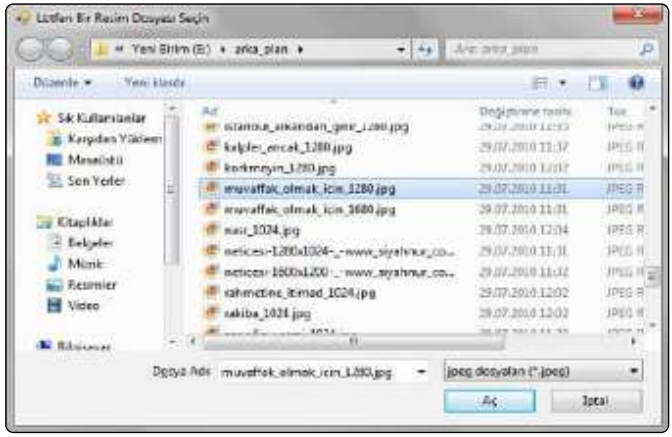
Örnek olarak **OpenFileDialog** nesnesi ile seçilen bir resmi, bir **PictureBox** nesnesi içerisinde görüntüleyen bir uygulama yazalım.

Form üzerine 1 adet **Button**, 1 adet **PictureBox** nesnesi ve 1 adet de **OpenFileDialog** nesnesi ekleyelim. **PictureBox** nesnesinin **`SizeMode`** özelliğini **`Zoom`** olarak belirleyerek resmin **PictureBox** içerisine sığacak şekilde ortalanarak yerleştirilmesini sağlıyoruz. Kodumuz aşağıdaki gibi olacak:

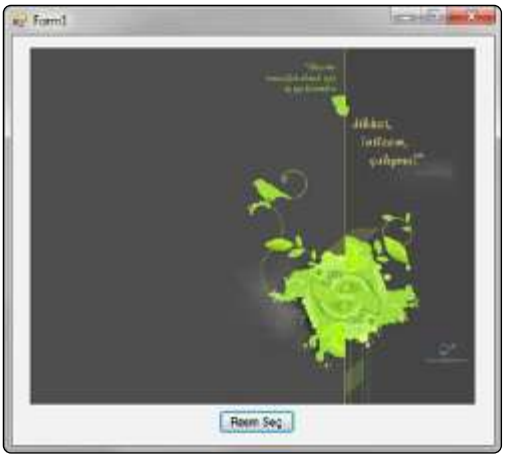
```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace OpenFileDialog
{
  public partial class Form1 : Form
  {
```

```
  public Form1()
 {
   InitializeComponent();
 }
 private void btnResimSec_Click(object sender, EventArgs e)
 {
   if (openFileDialog1.ShowDialog() == DialogResult.OK)
     pictureBox1.Image = new Bitmap(openFileDialog1.OpenFile());
 }
 private void Form1_Load(object sender, EventArgs e)
 {
   openFileDialog1.FileName = "dosya seç";
   openFileDialog1.Filter = "jpeg dosyaları (*.jpeg)
|*.jpg|bmp dosyaları (*.bmp)|*.bmp|gif dosyaları(*.gif)|*.gif";
   openFileDialog1.Title = "Lütfen Bir Resim Dosyası Seçin";
   openFileDialog1.InitialDirectory = "E:\\arka_plan";
  }
 }
}
```

Kodumuzda **Form**'un `Load` olayı içerisinde **OpenFileDialog** penceresinin `FileName`, `Filter`, `Title` ve `InitialDirectory` özelliklerinin, henüz form yüklenirken belirlenmesini sağlıyoruz. Butona tıklandığında da `ShowDialog` metodu ile diyalog penceresini görüntüleyerek, eğer basılan tuş **Tamam** butonu ise, **PictureBox** nesnesi içerisindeki resmin **OpenFileDialog** penceresi içerisinde seçili olan dosya olarak belirlenmesini sağlıyoruz. Kodumuzu çalıştırıp butonu tıkladığımızda, `openFileDialog` penceresi ekrana gelecektir.



Resmi seçip **Aç** düğmesine bastığımızda, resim **PictureBox** içerisinde görüntülenecektir.

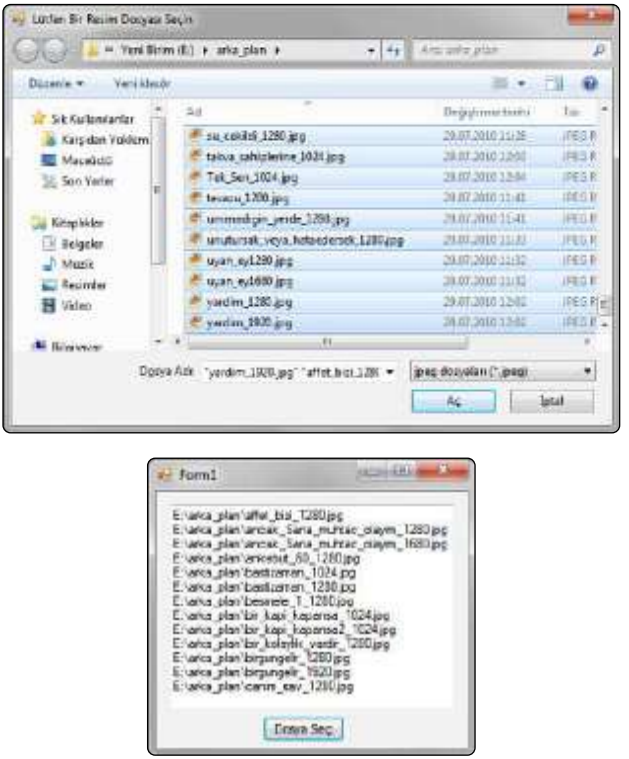


Eğer birden fazla dosya seçmemiz gerekiyorsa, **`FileName`** özelliği yerine **`FileNames`** özelliğini kullanmamız gerekmektedir. **`FileNames`**, dosya isimlerini dizi biçiminde geriye döndürür. Örnek olarak birden fazla dosya ismini seçerek bir **TextBox** içerisine ekletelim.

```
private void btnDosyaSec_Click(object sender, EventArgs e)
{
 byte i;
 openFileDialog1.Multiselect = true;
 openFileDialog1.FileName = "dosya seç";
 openFileDialog1.Filter = "jpeg dosyaları (*.jpeg) |*.jpg|bmp
dosyaları (*.bmp)|*.bmp|gif dosyaları(*.gif)|*.gif";
 openFileDialog1.Title = "Lütfen Bir Resim Dosyası Seçin";
 openFileDialog1.InitialDirectory = "E:\\arka_plan";
 openFileDialog1.ShowDialog();
 for (i=0;i<openFileDialog1.FileNames.Length-1;i++)
 {
  textBox1.Text = textBox1.Text +
   openFileDialog1.FileNames[i].ToString() +
   Environment.NewLine;
 }
}
```

Kodumuzda **OpenFileDialog** nesnesinin `MultiSelect` özelliğine `true` değerini aktararak, birden fazla dosya isminin seçilmesini sağlıyoruz. `FileNames.Length` özelliği ile **OpenFileDialog** penceresi içerisinde seçili olan dosya sayısını elde ederek, `for` döngüsü içerisinde **TextBox** içerisine sırasıyla `FileNames` koleksiyonu içerisindeki elemanları aktarıyoruz. Her bir dosya isminin **textBox** içerisine eklenmesinin ardından `Environment.NewLine` özelliği ile yeni bir satır ekliyoruz.

Uygulamamızı çalıştırıp, **OpenFileDialog** penceresini görüntüleyerek birden fazla dosya ismi seçelim.



OpenFileDialog penceresi içerisinde **Tamam** butonuna tıklamamızın ardından, seçili olan dosyaların **TextBox** içerisine aktarıldığını görürüz.

Save File Dialog

Herhangi bir dosyayı kaydetmek amaçlı kullanılır. Kullanılan başlıca metotları şunlardır:

- **`FileName`:** Diyalog penceresindeki dosya adı bölümünde varsayılan olarak görüntülenecek metni belirler.
- **`Filter`:** Diyalog penceresinde görüntülenecek olan dosya türlerini belirlemek için kullanılır.

```
Dosya türü adı (*.dosya türü) |*. Dosya türü| Dosya türü adı
(*.Dosya türü) |*. Dosya türü
```

… şeklinde kullanılır.

- **`Title`:** Diyalog penceresinin başlığında görüntülenecek olan yazıyı belirler.
- **`InitialDirectory`:** Diyalog penceresinde görüntülenecek olan varsayılan klasörü belirler.
- **`DefaultExt`:** Diyalog penceresinde görüntülenecek olan varsayılan dosya uzantısını belirler.

Örnek olarak **`openFileDialog`** nesnesi ile seçilen bir resmi bir **PictureBox** nesnesi içerisinde görüntüleyen ve **`SaveFileDialog`** nesnesi ile bunu farklı bir isimle kaydeden bir uygulama yazalım.

Form üzerine 2 adet **Button**, 1 adet **PictureBox** nesnesi, 1 adet **OpenFileDialog** nesnesi ve 1 adet de **SaveFileDialog** nesnesi ekleyelim. Kodumuz aşağıdaki gibi olacak:

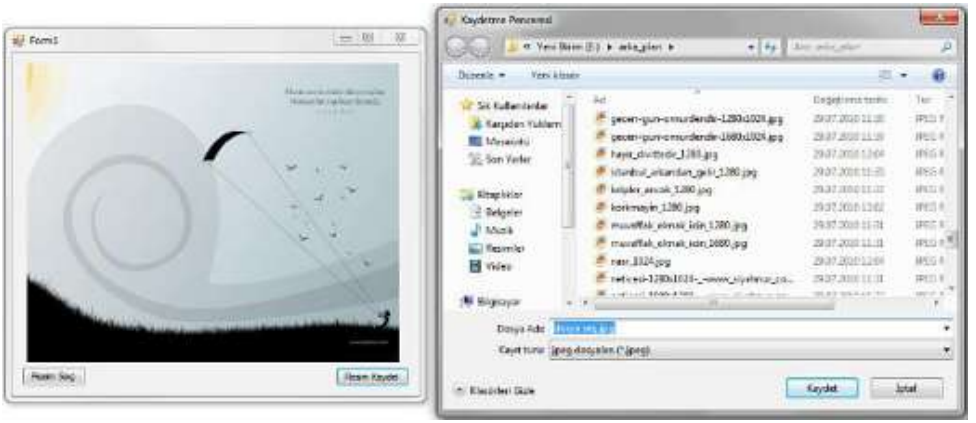
```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Windows.Forms;
namespace SaveFile_Dialog
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
```

```
}
private void Form1_Load(object sender, EventArgs e)
{
  openFileDialog1.FileName = "dosya seç";
  openFileDialog1.Filter =
   "jpeg dosyaları (*.jpeg) |*.jpg|bmp dosyaları (*.bmp)|*.bmp|gif
dosyaları(*.gif)|*.gif";
  openFileDialog1.Title = "Lütfen Bir Resim Dosyası Seçin";
  openFileDialog1.InitialDirectory = " E:\\arka_plan";

  saveFileDialog1.FileName = "dosya seç";
  saveFileDialog1.Filter = "jpeg dosyaları (*.jpeg) |*.jpg|bmp
dosyaları (*.bmp)|*.bmp|gif dosyaları (*.gif)|*.gif|İkon dosyaları
(*.ico)|*.ico|png dosyaları (*.png)|*.png|tiff dosyaları(*.tiff)|*.tiff";
  saveFileDialog1.Title = "Kaydetme Penceresi";
  saveFileDialog1.InitialDirectory = " E:\\arka_plan";
  saveFileDialog1.DefaultExt = "jpeg";
}
private void btnResimSec_Click(object sender, EventArgs e)
{
  if (openFileDialog1.ShowDialog() == DialogResult.OK)
      pictureBox1.Image = new Bitmap(openFileDialog1.OpenFile());
}
private void btnKaydet_Click(object sender, EventArgs e)
{
  if (saveFileDialog1.ShowDialog() == DialogResult.OK )
  {
   switch (saveFileDialog1.FileName.Substring
          (saveFileDialog1.FileName.Length - 3, 3))
   {
    case "bmp":
     pictureBox1.Image.Save(saveFileDialog1.FileName, ImageFormat.Bmp);
     break;
    case "gif":
     pictureBox1.Image.Save (saveFileDialog1.FileName, ImageFormat.Gif);
     break;
    case "jpeg":
     pictureBox1.Image.Save (saveFileDialog1.FileName, ImageFormat.Jpeg);
   break;
    case "png":
     pictureBox1.Image.Save(saveFileDialog1.FileName, ImageFormat.Png);
```

```
    break;
     case "ico":
       pictureBox1.Image.Save(saveFileDialog1.FileName, ImageFormat.Icon);
    break;
     case "tiff":
       pictureBox1.Image.Save(saveFileDialog1.FileName, ImageFormat.Tiff);
    break;
    }
   }
  }
 }
}
```

Kodumuzda **SaveFileDialog** nesnesini görüntüleyerek, **SaveFileDialog** nesnesi üzerinde **Tamam** butonuna tıklandığında `SubString` metodunu kullanarak, seçili olan dosyanın uzantısını tespit ediyoruz. Dosya uzantısına göre `switch case` ile seçim yaparak `Save` metodunu kullanıp, seçili olan dosyayı `ImageFormat` özelliği ile belirtilen dosya formatında kaydediyoruz.



STRING İŞLEMLERİ

12

String'ler, karakterlerin bir araya gelmesi ile oluşturulan kelimeler ve cümleler topluluğudur. C# dilinde string'ler üzerinde işlem yapmayı kolaylaştıran pek çok metot ve özellik bulunmaktadır. String'ler şu şekilde tanımlanır:

```
String Değişken adı = İlk Değer;
```

Örneğin;

```
string adsoyad ;
```

… şeklinde ilk değer ataması yapılmadan tanımlanabileceği gibi…

`string adsoyad = "Volkan AKTAŞ"` şeklinde ilk değer ataması yapılarak da kullanılabilir.

String değişkenler; hafızada 10 Byte (0 ile 2 milyar karakter arası) yer kaplarlar.

String'ler üzerinde işlem gerçekleştirilirken, **`String`** sınıfını ya da **`StringBuilder`** sınıfını kullanabiliriz.

String Sinifi

ToString Metodu

Belirtilen bir değişken türünü string türüne dönüştürmek amacı ile kullanılır.

Örneğin; aşağıdaki kodu çalıştırmaya kalktığımızda "*cannot convert from 'int' to 'string'*" şeklinde hata verecektir:

```
private void btnislem_Click(object sender, EventArgs e)
{
 int sonuc;
 string mesaj;
 sonuc = Convert.ToInt32(txtBirinciSayı.Text) +
         Convert.ToInt32(txtİkinciSayı.Text);
 MessageBox.Show (sonuc);
}
```

Çünkü **sonuc** değişkeninin içeriği **integer** olduğundan dolayı, mesaj penceresi ile görüntülemek istediğimizde **string** değere dönüştürmediğimizden dolayı hata verecektir. Onun yerine…

```
MessageBox.Show (sonuc.ToString());
```

… yazdığımızda, programımız doğru şekilde çalışacaktır.



Copy Metodu

Bir string'in içeriğini başka bir string'e kopyalamak amacı ile kullanılır.

Kullanımı şu şekildedir:

```
Hedef String Adı = String.Copy(Kaynak String Adı)
```

Aşağıdaki örnekte **a** değişkenin içeriği **b** değişkenine kopyalanarak, "*Bilgisayar*" değerini alacaktır.

```
static void Main(string[] args)
{
 string a;
 string b;
```

```
 a = "Bilgisayar";
 b = string.Copy(a);
 Console.WriteLine(b);
 Console.ReadKey();
}
```

Concat Metodu

Birden fazla string'i birleştirerek tek bir string haline getirmek amacı ile kullanılır. Eklenen her **`string`** diğer **`string`**'in sonuna eklenecektir.

Kullanımı şu şekildedir:

```
String.Concat(String1 , String2, … , String n);
```

```
static void Main(string[] args)
{
 string a = "Bilgisayar ";
 string b = "Bölümü";
 Console.WriteLine(string.Concat(a, b));
 Console.ReadKey();
}
```

`a` değişkeninin içeriği ile **`b`** değişkeninin içeriği birleştirilerek ekrana "*Bilgisayar Bölümü*" yazdırılacaktır.

Aynı işlemi **`string`** değişkenlerden birinin ismini yazmak yerine, içeriğini yazarak da gerçekleştirebiliriz.

```
static void Main(string[] args)
{
 string a = "Bilgisayar ";
 Console.WriteLine(string.Concat(a, "Bölümü"));
 Console.ReadKey();
}
```

Kodumuzda string türdeki a değişkeni ile "*Bölümü*" string ifadesini birleştirdik.

Empty Metodu

Bir string'in içeriği boşaltmak amacı ile kullanılır.

Kullanımı şu şekildedir:

```
String Değişken adı= String.Empty;
```

Concat metodu ile yaptığımız örnek üzerinde **Empty** metodunu kullanarak değişiklik yapalım.

```
static void Main(string[] args)
{
 string a = "Bilgisayar ";
 string b = "Bölümü";
 b = String.Empty;
 Console.WriteLine(string.Concat(a, b));
 Console.ReadKey();
}
```

b değişkeninin içeriğini **Empty** metodu ile boşalttığımız için ekrana "*Bilgisayar*" yazdırılacaktır.

Compare Metodu

İki string'i karşılaştırmak amacı ile kullanılır.

Kullanımı şu şekildedir:

```
String.Compare(String1, String2, Karşılaştırma Seçenekleri )
```

Karşılaştırma seçenekleri aşağıdaki değerleri alabilir.

- **CurrentCulture:** String'ler karşılaştırılırken geçerli kültür ayarları kullanılır. Karşılaştırma esnasında büyük küçük harf ayrımı göz önünde bulundurulur.
- **CurrentCultureIgnoreCase:** String'ler karşılaştırılırken geçerli kültür ayarları kullanılır. Karşılaştırma esnasında büyük küçük harf ayrımı göz önünde bulundurulmaz.
- **InvariantCulture:** Geçerli kültür ayarlarından farklı bir kültür ayarı kullanmak amacı ile kullanılır. Karşılaştırma esnasında büyük küçük harf ayrımı göz önünde bulundurulur.

- **`InvariantCultureIgnoreCase`:** Geçerli kültür ayarlarından farklı bir kültür ayarı kullanmak amacı ile kullanılır. Karşılaştırma esnasında büyük küçük harf ayrımı göz önünde bulundurulmaz.
- **`Ordinal`:** Karşılaştırılan string'ler içerisinde mevcut olan rakamlar da göz önüne alınarak karşılaştırma işlemi yapılır. Karşılaştırma esnasında büyük küçük harf ayrımı göz önünde bulundurulur.
- **`OrdinalIgnoreCase`:** Karşılaştırılan stringler içerisinde mevcut olan rakamlar da göz önüne alınarak karşılaştırma işlemi yapılır. Karşılaştırma esnasında büyük küçük harf ayrımı göz önünde bulundurulmaz.

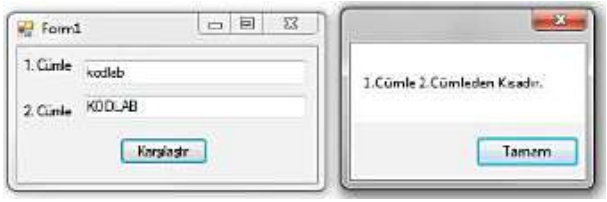
Karşılaştırma sonucunda geriye aşağıdaki değerler döner:

Dönen Değer	Sonuç
0'dan küçük	String1, String2'den kısadır.
0	String1, String2'ye eşittir.
0'dan büyük	String1, String2'den uzundur.

`String.Compare` metodunu **`StringComparison.CurrentCulture`** parametresi ile kullanalım.

```
private void btnCompare_Click(object sender, EventArgs e)
{
  string yazi1 = textBox1.Text;
  string yazi2 = textBox2.Text;
  if (string.Compare(yazi1, yazi2,
   StringComparison.CurrentCulture) == 0)
  {
   MessageBox.Show("1.Cümle 2.Cümleye Eşittir.");
  }
  else if (string.Compare(yazi1, yazi2,
   StringComparison.CurrentCulture) < 0)
  {
   MessageBox.Show("1.Cümle 2.Cümleden Kısadır.");
  }
  else
   MessageBox.Show("1.Cümle 2.Cümleden Uzundur.");
}
```

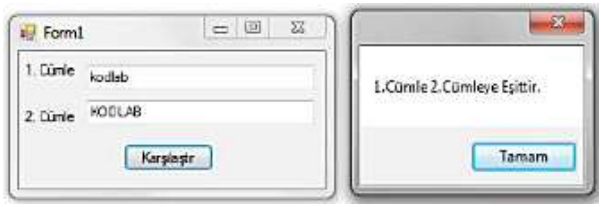
Uygulamamızı çalıştırıp aşağıdaki örnekte görüldüğü gibi **textBox1** içerisine yazdığımız yazıyı **textBox2** içerisine büyük harfle yazarak, karşılaştırma işlemi yaptığımızda küçük harfle yazılmış cümle için daha kısadır şeklinde mesaj görüntülenecektir.



Kodumuzu değiştirerek **String.Compare** metodunu **StringComparison.CurrentCultureIgnoreCase** parametresi ile kullanalım.

```
private void btnCompare_Click(object sender, EventArgs e)
{
  string yazi1 = textBox1.Text;
  string yazi2 = textBox2.Text;
  if (string.Compare(yazi1, yazi2,
  StringComparison.CurrentCultureIgnoreCase) == 0)
  {
  MessageBox.Show("1.Cümle 2.Cümleye Eşittir.");
  }
  else if (string.Compare(yazi1, yazi2,
  StringComparison.CurrentCultureIgnoreCase) < 0)
  {
  MessageBox.Show("1.Cümle 2.Cümleden Kısadır.");
  }
  else
  MessageBox.Show("1.Cümle 2.Cümleden Uzundur.");
}
```

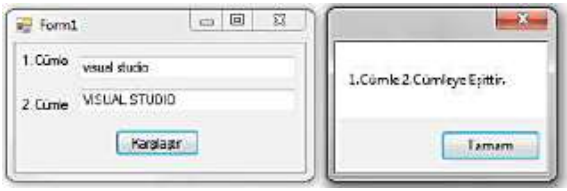
Kodumuzu çalıştırdığımızda bu defa büyük küçük harf ayrımı yapılmadan karşılaştırma işlemi yapıldığından dolayı, cümlelerin eşit olduğu şeklinde sonuç görüntülenecektir.



String.Compare metodunu **StringComparison.InvariantCultureIgnoreCase** parametresi ile kullanalım.

```
using System;
using System.Windows.Forms;
using System.Threading;
using System.Globalization;
namespace Compare_InvariantCultureIgnoreCase
{
  public partial class Form1 : Form
  {
   public Form1()
   {
    InitializeComponent();
   }
  private void btnCompare_Click(object sender, EventArgs e)
  {
   Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
   string yazi1 = textBox1.Text;
   string yazi2 = textBox2.Text;
   if (string.Compare(yazi1, yazi2,
     StringComparison.InvariantCultureIgnoreCase) == 0)
   {
    MessageBox.Show("1.Cümle 2.Cümleye Eşittir.");
   }
   else if (string.Compare(yazi1, yazi2,
     StringComparison.InvariantCultureIgnoreCase) < 0)
   {
    MessageBox.Show("1.Cümle 2.Cümleden Kısadır.");
   }
   else
   {
    MessageBox.Show("1.Cümle 2.Cümleden Uzundur.");
   }
  }
 }
}
```

Örneğimizde geçerli kültür ayarlarını "*Amerikan İngilizcesi*" şekline çevirerek İngilizce karakterlerdeki "*i*" ve "*I*" gibi karakterlerin ayrımı yapılmadan karşılaştırma işleminin gerçekleştirilmesini sağladık.



`String.Compare` metodunu **`StringComparison.Ordinal`** parametresi ile kullanalım.

```
private void btnCompare_Click(object sender, EventArgs e)
{
  string yazi1 = textBox1.Text;
  string yazi2 = textBox2.Text;
  if (string.Compare(yazi1, yazi2,
    StringComparison.Ordinal) == 0)
  {
   MessageBox.Show("1.Cümle 2.Cümleye Eşittir.");
  }
  else if (string.Compare(yazi1, yazi2,
    StringComparison.Ordinal) < 0)
  {
   MessageBox.Show("1.Cümle 2.Cümleden Kısadır.");
  }
  else
  {
   MessageBox.Show("1.Cümle 2.Cümleden Uzundur.");
  }
}
```

Karşılaştırma işleminde string'ler içerisinde bulunan rakamlar karşılaştırılarak, bu rakamların büyüklük ve küçüklük durumuna göre string'lerin uzunluk ve kısalığı değerlendirilir.





Split Metodu

Bir metni ya da metin dizisini belirtilen ayırıcı karaktere göre parçalara bölmek amacı ile kullanılır. Ayırıcı karakter olarak herhangi bir harf, özel karakter (nokta, virgül gibi) kullanılabileceği gibi boşluk, tab gibi ayırıcı karakterler de kullanılabilir.

Kullanımı şu şekildedir:

```
String Değişken Adı.Split(Karakter)
```

Örnek olarak bir text kutusuna girilen yazının kelime sayısını bulan programı, **`Split`** metodu ile yazalım.

Kodumuz aşağıdaki gibi olacak:

```
private void textBox1_KeyDown(object sender, KeyEventArgs e)
{
  if (e.KeyCode == Keys.Enter ) MessageBox.Show
  ("Cümlenin Kelime Sayısı= " + textBox1.Text.Split(' ').
  Length.ToString());
}
```

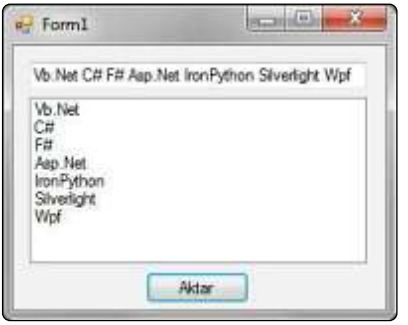


Örneğimizde kodumuzu **`textBox1`** içerisinde **Enter** tuşuna basıldığında, **`Split`** metodunu kullanarak, cümlemizi boşluk karakterine göre parçalara ayırıp, **`Length`** metodu ile ayrılan parçaların sayısını buluyoruz.

Diğer bir örnekle de bir metin kutusu içerisindeki yazıyı kelimelere ayırıp, bir **ListBox** içerisine ekleyen bir örnek yapalım.

```
private void btnAktar_Click(object sender, EventArgs e)
{
 byte i;
 string[] yazi;
 yazi = textBox1.Text.Split(' ');
 for (i = 0; i <= yazi.Length - 1; i++)
  listBox1.Items.Add(yazi[i].ToString());
}
```

Text kutusuna girilen yazıyı aradaki boşluk karakterine göre parçalara ayırıp, bunu **`yazi`** dizisine, **`yazi`** dizisinin elemanlarını da **`for`** döngüsü ile **ListBox** nesnesi içerisine aktarıyoruz.



`Split` metodunu kullanırken birden fazla karakteri de aratıp buna göre işlem yapabiliriz. Text kutusuna girilen yazıyı parçalara bölüp **ListBox**'a aktaran örneğimizi hem boşluk hem de virgül karakterlerini arayacak şekilde genişletelim.

```
private void btnAktar_Click(object sender, EventArgs e)
{
 byte i;
 string[] yazi;
 yazi = textBox1.Text.Split(' ',',');
 for (i = 0; i <= yazi.Length - 1; i++)
  ListBox1.Items.Add(yazi[i].ToString());
}
```

Örneğimizi çalıştırdığımızda görüldüğü gibi hem boşluk hem de virgül karakterlerini arayıp, kelimeleri bu karakterlere göre parçalara böldük ve **ListBox**'a aktardık.



Arayacağımız karakterleri karakter dizisi şeklinde tanımlayarak da işlem yapabiliriz.

```
private void btnAktar_Click(object sender, EventArgs e)
{
 byte i;
 string[] yazi;
 char[] karakterler = { ' ', ','};
 yazi = textBox1.Text.Split(karakterler);
 for (i = 0; i <= yazi.Length - 1; i++)
  ListBox1.Items.Add(yazi[i].ToString());
}
```

Arayacağımız karakterleri, karakterler adında **`char`** tipinde bir dizi tanımlayarak buraya aktardık ve **`Split`** metodu ile karakterler dizisinin elemanlarının aranarak, bu dizinin elemanlarına göre metnin parçalara bölünmesini sağladık.

Yukarıdaki örneğimizi yeniden çalıştırıp değer girelim. Ancak bu defa text kutusu içerisinde değerleri girerken virgülden sonra bir karakter boşluk bırakalım.

Metni parçalara bölerken virgül ve boşluk karakterlerine göre parçalara bölünmesini belirttiğimiz için virgülden sonra bir defa, boşluktan sonra da bir defa olmak üzere **`F#`** ve **`IronPython`** değerlerinden sonra 2 defa bölme işlemi yapıldı. Ancak bu bölme işlemleri sonucunda geriye boş değer döndürüldü. Boş değerlerin döndürülmesini engellemek için **`Split`** metodunun **`RemoveEmptyEntries`** parametresini kullanabiliriz.



Kodumuzu buna göre değiştirelim.

```
private void btnislem_Click(object sender, EventArgs e)
{
 byte i;
 string[] yazi;
 char[] karakterler = { ' ', ','};
 yazi = textBox1.Text.Split(karakterler,
        StringSplitOptions.RemoveEmptyEntries );
 for (i = 0; i <= yazi.Length - 1; i++)
   ListBox1.Items.Add(yazi[i].ToString());
}
```

Kodumuzu çalıştırdığımızda boş değerlerin listelenmediğini görürüz.



Join Metodu

Bir metin ya da metin dizisi içerisindeki değerlerin hepsini bir string değişkene, ayırıcı karakter kullanılarak aktaran metottur.

Kullanımı şu şekildedir:

```
String Değişken Adı.Join(ayırıcı karakter, dizi değişkeni adı)
```

Örnek olarak **`Split`** metodu ile yaptığımız örneğin tersini yapalım. Bir **ListBox** içerisindeki değerleri **`Join`** metodu ile "," karakterini kullanarak bir **TextBox** içerisine aktaralım.

```
private void btnAktar_Click(object sender, EventArgs e)
{
 string[] diller = new string[ListBox1.Items.Count];
 byte i;

 for (i = 0; i <= ListBox1.Items.Count - 1; i++)
   diller[i]=ListBox1.Items[i].ToString();
 textBox1.Text = string.Join(",", diller);
}
```



Kodumuzu çalıştırdığımızda görüldüğü gibi **ListBox** içerisindeki değerler **`diller`** adında bir dizi oluşturularak, bu dizi içerisine aktarıldı ve daha sonra bu dizinin elemanları **TextBox** içerisine **`Join`** metodu yardımıyla for döngüsü içerisinde "," karakteri kullanılarak aktarıldı.

IndexOf, IndexOfAny, LastIndex ve LastIndexOfAny Metodları

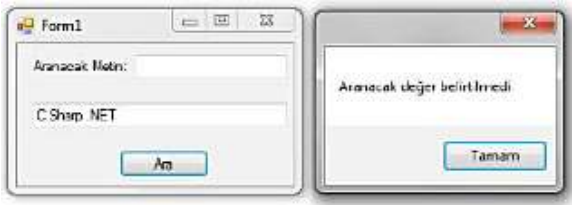
- **`IndexOf`**: Belirtilen bir karakter ya da string'in başka bir string içerisinde olup olmadığını, eğer mevcutsa hangi konumda olduğu bilgisini verir.
- **`IndexOfAny`**: Belirtilen bir dizi içerisindeki karakterlerin başka bir string içerisinde olup olmadığını, eğer mevcutsa hangi konumda olduğu bilgisini verir.
- **`LastIndexOf`**: **`IndexOf`** metodu gibi belirtilen bir karakter ya da string'in başka bir string içerisinde olup olmadığını, eğer mevcutsa hangi konumda olduğu bilgisini verir. Ancak arama işlemini sondan itibaren yapar.
- **`LastIndexOfAny`**: **`IndexOfAny`** metodu gibi belirtilen bir dizi içerisindeki karakterlerin başka bir string içerisinde olup olmadığını, eğer mevcutsa hangi konumda olduğu bilgisini verir. Ancak arama işlemini sondan itibaren yapar.

Her dört metot da geriye 3 değer döndürür.

- Eğer dönen değer `0` ise aranacak karakter ya da string belirtilmemiştir.
- Eğer dönen değer `-1` ise aranan karakter ya da string bulunamamıştır.
- Aranan karakter ya da **`string`** bulunursa, geriye bulunduğu konum döndürülür.

`IndexOf` metodunu uygulayarak örnek uygulama yapalım.

```
private void btnAra_Click(object sender, EventArgs e)
{
  if ((textBox1.Text.IndexOf(txtAranacakMetin.Text) == 0))
   MessageBox.Show("Aranacak değer belirtilmedi");
  else if (textBox1.Text.IndexOf(txtAranacakMetin.Text)==-1)
   MessageBox.Show("Aranan değer mevcut değil");
  else
   MessageBox.Show("Aranan değer mevcut. Konumu= "
    +(textBox1.Text.IndexOf(txtAranacakMetin.Text)));
}
```



Aranacak metni belirtmeyip işlem yaptığımızda dönen değer `0` olacağından dolayı, "*Aranacak değer belirtilmedi*" mesajı görüntülenecektir.

Aranan metin bulunamadığında dönen değer `-1` olacağından dolayı, "*Aranan değer mevcut değil*" mesajı görüntülenecektir.



Aranan metin bulunduğunda "*Aranan değer mevcut*" mesajı ile birlikte bulunan metnin başlangıç konumu görüntülenecektir.



`IndexOfAny` metodunu uygulayarak örnek uygulama yapalım.

```
private void btnislemyap_Click(object sender, EventArgs e)
{
  string deger = textBox1.Text  ;
  int index = deger.IndexOfAny(new char[] { ':', '\\' , '.'});
  if (index == -1)
   MessageBox.Show("Web site adresi geçerli değil");
  else
   MessageBox.Show("Web site adresi geçerli");
}
```



Örneğimizde **TextBox** içerisine girilen web sitesi adresinin geçerli bir adres olup olmadığı, bir dizi içerisindeki karakterler **`IndexOfAny`** metodu ile text kutusu içerisinde aranarak kontrol ediliyor. Eğer karakterler mevcut değilse, "*Web site adresi geçerli değil*" şeklinde uyarı veriliyor.

Substring Metodu

Bir metnin içerisinden, belirtilen bir kısmını almak amacı ile kullanılır.

Kullanımı şu şekildedir:

```
String Değişken Adı.SubString(Başlangıç pozisyonu, uzunluk)
```

Örnek olarak bir text kutusuna girilen site adresinin isim kısmını alacak bir uygulama yazalım:

Kodumuz aşağıdaki gibi olacak;

```
private void btnSubString_Click(object sender, EventArgs e)
{
  int pozisyon;
  string cumle;
  cumle = textBox1.Text;
  pozisyon = (textBox1.Text.IndexOf(".") + 1);
  MessageBox.Show (cumle.Substring(pozisyon,
      cumle.Length  - pozisyon));
}
```

Kodumuzda öncelikle **`IndexOf`** metodu ile **TextBox** içerisindeki ilk nokta işaretinin konumunu bularak pozisyon değişkenine aktarıyoruz. Daha sonra **`Substring`** metodu ile nokta işaretinin pozisyonundan itibaren cümle sonuna kadar olan kısmı alarak mesaj penceresi içerisinde görüntülüyoruz.

Kodumuzu çalıştırıp örnek bir adres girelim.

Görüldüğü gibi site adresinin ilk noktadan sonrasını keserek bir mesaj penceresi içerisinde görüntüledi.



Replace Metodu

Bir metnin içerisinde, belirtilen bir ya da birden fazla karakteri, başka bir karakter ya da karakterler ile değiştirmek amacıyla kullanılır.

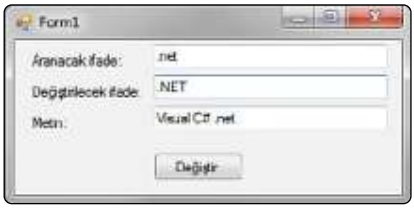
Kullanımı şu şekildedir:

```
String Değişken Adı.Replace(Eski Karakter, Yeni Karakter)
```

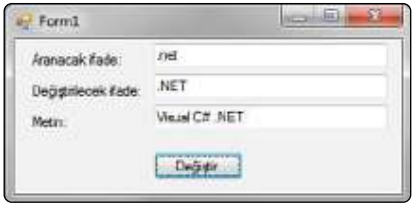
Örnek olarak bir text kutusu içerisinde değiştirme işlemi yapacak bir uygulama yazalım.

```
private void btnDegistir_Click (object sender, EventArgs e)
{
  textBox1.Text.Replace(txtAranacak.Text, txtDegistirilecek.Text);
}
```

Örneğimizi çalıştırıp **textBox1** içerisine girilen yazı içerisindeki bir bölümü aratarak, aranan ifadeyi büyük harfli hali ile değiştirdik.



Replace öncesi.



Replace işleminin sonrası.

ToUpper ve ToLower Metodları

ToUpper metodu bir string'i büyük harfe dönüştürürken, **ToLower** metodu ise küçük harfe dönüştürür.

Kullanımları şu şekildedir:

```
String Değişken Adı.ToUpper
String Değişken Adı.ToLower
```

Örneğin; **textBox1** içerisindeki bir ifadeyi büyük harfe çevirmek için…

```
textBox1.Text = textBox1.Text.ToUpper();
```

…kod satırlarını kullanırken, küçük harfe çevirmek için ise….

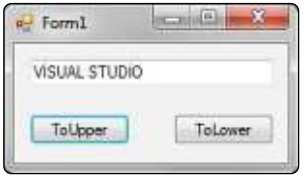
```
textBox1.Text = textBox1.Text.ToLower();
```

… kod satırı kullanılır.

NOT **`ToUpper`** ve **`ToLower`** metotlarını kullanarak yapılan dönüştürme işlemlerinde, İngilizce karakterlerdeki "`i`" ve "`I`" gibi karakterler, dönüştürme işlemi sonrasında birebir dönüştürüldüğünden dolayı, İngilizce gibi dillerle yazılan kelimelerin dönüşümünde hatalı sonuç olacaktır.



Örneğin **TextBox** içerisine "*visual studio*" metnini girip **`ToUpper`** metodu ile büyük harfe dönüştürelim.



Görüldüğü gibi "*VİSUAL STUDİO*" biçiminde dönüştürüldü. Her ne kadar büyük harfe dönüştürme işlemi doğru biçimde olsa da İngilizce karakterler istediğimiz biçimde dönüştürülmedi.

İngilizce karakterlerin doğru bir şekilde büyük harfe çevrilmesi için, **`ToUpperInvariant`** ve **`ToLowerInvariant`** metotlarını kullanabilir. Örneğimizdeki büyük harfe dönüştürme işlemini…

```
textBox1.Text = textBox1.Text.ToUpperInvariant();
```



… biçiminde **`ToUpperInvariant`** metodunu kullanarak yaptığımızda, büyük harfe dönüştürme işleminin "*VISUAL STUDIO*" biçiminde doğru olarak yapıldığını görürüz.

PadLeft ve PadRight Metodları

Bir string içerisine belirtilen karakteri belirtilen adette eklemek, eğer herhangi bir karakter belirtilmezse de boşluk karakteri eklemek amacıyla kullanılır. **`PadLeft`** metodu string'in sol tarafına karakter eklerken, **`PadRight`** ise string'in sağ tarafına karakter ekler.

Kullanımları şu şekildedir:

```
String Değişken Adı.PadLeft(Karakter Adedi)
String Değişken Adı.PadRight(Karakter Adedi)
```

Bu şekilde kullanıldığında, boşluk karakterini belirtilen adet kadar ekler.

```
String Değişken Adı.PadLeft(Karakter Adedi, "Karakter")
String Değişken Adı.PadRight(Karakter Adedi, "Karakter")
```

Bu şekilde kullanıldığında ise belirtilen karakteri belirtilen adette ekler.

Örneğin…

```
Console.WriteLine("Deneme".PadLeft(20));
```

… satırında "*Deneme*" cümlesi sol tarafına boşluklar eklenerek, 20 karakter sayısına genişletilmektedir.

```
Console.WriteLine("Deneme".PadLeft(20, '-'));
```

… satırında ise "*Deneme*" cümlesi sol tarafına "-" eklenerek, 20 karakter boyutuna genişletilmektedir.

```
static void Main(string[] args)
{
 Console.Write("Adınız".PadRight(15, '.'));
 Console.WriteLine("Volkan");
 Console.Write("Soyadınız".PadRight(15, '.'));
 Console.WriteLine("Aktaş");
 Console.ReadLine();
}
```

Kod satırlarını çalıştırdığımızda ise ekran çıktısı aşağıdaki gibi olmaktadır:



Kodumuzda "*Adınız*" ve "*Soyadınız*" ifadelerinden sonra " . " karakterini ekleyerek, her bir ifadeyi 15 karakter sayısına genişlettik.

StringBuilder Sınıfı

Özellikle birden fazla string üzerinde işlem yapmak amacıyla kullanılan ve bu işlemleri **String** sınıfına göre daha performanslı bir biçimde gerçekleştiren bir sınıftır. **StringBuilder** sınıfını kullanabilmek için projemize **System.Text** namespace'ini dahil etmemiz gerekir.

```
StringBuilder Değişken İsmi = new StringBuilder(String Değişken İsmi)
```

… şeklinde tanımlanırlar.

```
using System;
using System.Text;
using System.Windows.Forms;
namespace String_Builder
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
  }
  private void btnOlustur_Click(object sender, EventArgs e)
  {
   string yazi = "C Sharp .NET";
   StringBuilder sb = new StringBuilder(yazi);
   MessageBox.Show(sb.ToString());
  }
 }
}
```

Örneğimizde **sb** adlı yeni bir **StringBuilder** nesnesi oluşturarak içeriğine **yazi** adlı **string** değişkenin içeriğini aktarıyoruz ve **sb** değişkeninin içeriğini bir mesaj penceresi içerisinde görüntülüyoruz.

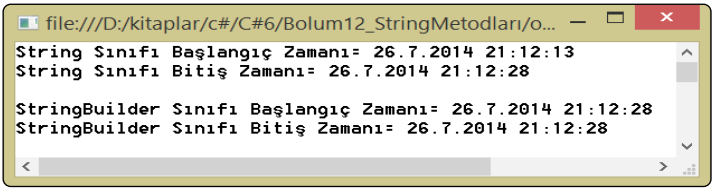
StringBuilder sınıfı ile **String** sınıfının performansını karşılaştıran örnek bir uygulama yapalım.

```
using System;
using System.Linq;
using System.Text;
namespace StringBuilder_ve_StringSınıfı_Karsılastırma
{
 class Program
 {
  static void Main(string[] args)
  {
   int i;
   string yazi="";
   DateTime baslangiczamani;
   DateTime bitiszamani;
   baslangiczamani = DateTime.Now;
   for (i = 0; i <= 50000; i++)
    yazi = yazi + i.ToString();
   bitiszamani = DateTime.Now;
   Console.WriteLine("String Sınıfı Başlangıç Zamanı= " +
                     baslangiczamani.ToString());
   Console.WriteLine("String Sınıfı Bitiş Zamanı= " +
                     bitiszamani.ToString());
   Console.WriteLine();

   StringBuilder sb = new StringBuilder();
   baslangiczamani = DateTime.Now;
   for (i = 0; i <= 50000; i++)
        sb.Append(i.ToString());
   bitiszamani = DateTime.Now;
   Console.WriteLine("StringBuilder Sınıfı Başlangıç Zamanı= " +
                     baslangiczamani.ToString());
   Console.WriteLine("StringBuilder Sınıfı Bitiş Zamanı= " +
                     bitiszamani.ToString());
   Console.ReadKey();
 }
}
}
```

Kodumuzda başlangıç ve bitiş zamanlarını **`DateTime.Now`** özelliği ile elde ediyoruz. **`String`** ve **`StringBuilder`** sınıfları için for döngüsü içerisinde 100000 adet **`string`** değeri ekleme işlemi gerçekleştiriyoruz.

`String` sınıfı ile yaptığımız işlem 15 saniye sürerken, `StringBuilder` sınıfı ile yaptığımız işlem 1 saniye sürmektedir.



`String` değişkenlerin içeriğine bir kez atama yapıldıktan sonra içerikleri bir daha değiştirilemez. Dolayısı ile `String` sınıfı ile yaptığımız işlemlerde mevcut `string` üzerine yeni bir `string` değer eklemek istediğimizde; mevcut `string` üzerine ekleme yapılamayacağından dolayı yeni bir `string` oluşturularak, mevcut `string` üzerine ekleniyor. Mevcut `string` ise `Garbage Collector` ile temizleniyor.

`StringBuilder` sınıfı ile yaptığımız işlemlerde ise, yapılan her ekleme işleminde yeni bir `string` oluşturulmayacağından dolayı performans kaybı olmamaktadır.

StringBuilder Sınıfı Özellikleri

`StringBuilder` sınıfı özelliklerinden, en çok kullanılan bazı özellikleri inceleyelim.

Length Özelliği

`StringBuilder` içerisinde mevcut olan `string` değerlerin karakter sayısını verir.

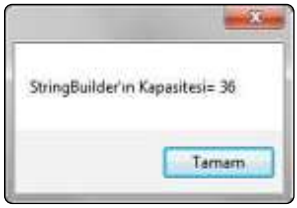
Capacity Özelliği

`StringBuilder` içerisinde saklanabilecek `string` değer kapasitesini verir.

EnsureCapacity Özelliği

`StringBuilder` içerisinde saklanabilecek `string` değer kapasitesini arttırmak amacı ile kullanılır.

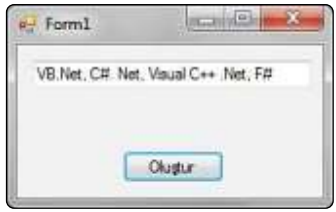
```
using System;
using System.Text;
using System.Windows.Forms;
namespace StringBuilder_EnsureCapacity
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
  }
  private void btnOlustur_Click(object sender, EventArgs e)
  {
   string yazi = "VB.Net, C#. Net, Visual C++ .Net, F#";
   StringBuilder sb = new StringBuilder();
   sb.Append(yazi.ToString());
   MessageBox.Show("StringBuilder'ın Kapasitesi= " + sb.Capacity);
   sb.EnsureCapacity(50);
   MessageBox.Show("StringBuilder'ın Yeni Kapasitesi= " + sb.Capacity);
   textBox1.Text = sb.ToString();
  }
 }
}
```



StringBuilder'ın ilk kapasitesi



StringBuilder'ın son kapasitesi



StringBuilder içeriğinin textBox'a aktarılması

Kodumuzu çalıştırdığımızda öncelikle, **Capacity** özelliği ile oluşturmuş olduğumuz **StringBuilder**'ın varsayılan kapasite değerini, daha sonra ise, **EnsureCapacity** özelliği ile yapılan kapasite arttırma işleminin ardından **StringBuilder**'ın yeni kapasitesini görüntülüyoruz.

MaxCapacity Özelliği

StringBuilder içerisinde saklanabilecek maksimum veri kapasitesini belirtir. Bu değer varsayılan olarak 2147483647 değeridir.

StringBuilder Sınıfı Metodları

StringBuilder sınıfı, **string**'ler üzerinde işlem yapmak üzere birçok metoda sahiptir. Bunların başlıcalarını inceleyelim.

Append ve AppendLine Metodları

Append metodu ile **StringBuilder** içerisine bir **string** eklenir.

AppendLine metodu ile de **StringBuilder** içerisine bir **string** eklenir. Ancak ekleme işlemi yapılırken satır sonu karakteri ile birlikte eklenmektedir. Dolayısı ile **string** eklendikten sonra bir alt satıra geçilir.

```
private void btnOlustur_Click(object sender, EventArgs e)
{
 StringBuilder sb = new StringBuilder();

 sb.Append("Programlama");
 sb.Append("Dilleri");
 sb.AppendLine();
 sb.AppendLine("----------------------");
 sb.AppendLine();
 sb.AppendLine("VB.Net");
 sb.AppendLine("C#.Net");
 sb.AppendLine("Visual C++.Net");
 sb.AppendLine("F#");
 sb.AppendLine("Iron Python");

 textBox1.Text = sb.ToString();
 }
```



AppendLine metodunu bir değer belirtmeden kullandığımızda **StringBuilder** içerisine bir **string** değer eklenmeden bir alt satıra geçilir.

APPENDFORMAT METODU

StringBuilder içerisine **string** eklenirken çeşitli formatlara göre biçimlendirilerek aktarılmasını sağlar.

Kullanımı şu şekildedir:

```
StringBuilder.AppendFormat {index[,length][:formatString]}
```

- **Index: String** değişkenlerin sırasını belirtir.
- **FormatString:** Biçimlendirme işlemi sırasında kullanılacak karakterleri belirtir. Kullanılabilecek format biçimleri, bir sonraki bölümde ayrıntılı olarak işleneceğinden dolayı burada yazılmamıştır.
- **Length:** Biçimlendirme işleminin uygulanacağı karakter uzunluğunu belirtir.

```
private void btnislem_Click(object sender, EventArgs e)
{
  StringBuilder sb = new StringBuilder();
  sb.AppendFormat("{0:hh:mm:ss}", DateTime.Now);
  sb.AppendLine();
  sb.AppendFormat("{0:HH:mm:ss}", DateTime.Now);
  sb.AppendLine();
  sb.AppendFormat("{0:dd MM yyyy}", DateTime.Now);
  sb.AppendLine();
  sb.AppendFormat("{0:dd MMMM yyyy}", DateTime.Now);
  sb.AppendLine();
  sb.AppendFormat("{0:HH:mm:ss dd MMMM yyyy}", DateTime.Now);
  textBox1.AppendText(sb.ToString());
}
```

Kodumuzda, tarih ve zaman değerlerini biçimlendirmede kullandığımız format karakterleri ve anlamları aşağıdadır:

- **dd:** İki basamaklı olarak gün bilgisini verir. 1 ile 31 arası değer alır.
- **dddd:** Gün bilgisini tam hali ile görüntüler. Örneğin; Pazartesi gibi.
- **MM:** İki basamaklı olarak ay bilgisini verir. 1 ile 12 arası değer alır.
- **MMMM:** Ay bilgisini tam hali ile görüntüler. Örneğin; Eylül gibi.
- **yy:** Yıl bilgisinin son iki basamağını verir. 2009 yılı için 09 rakamı görüntülenir.
- **yyyy:** Yıl bilgisini tam hali ile görüntüler. Yıl bilgisi 2009 şeklinde görüntülenir.
- **hh:** Saat bilgisini iki basamaklı olarak görüntüler. 1 ile 12 arası değer alır.
- **HH:** Saat bilgisini iki basamaklı olarak görüntüler. 01 ile 24 arası değer alır.
- **mm:** Dakika bilgisini iki basamaklı olarak görüntüler. 00 ile 59 arası değer alır.
- **ss:** Saniye bilgisini iki basamaklı olarak görüntüler. 00 ile 59 arası değer alır.



Kodumuzu çalıştırdığımızda `DateTime.Now` özelliği ile elde edilen 26.07.2014 tarihi ve 21:16:27 zamanı.

Insert Metodu

`Append` metodu kullanılarak `StringBuilder` içerisine eklenen veriler, `StringBuilder` içerisinde en son sıradan itibaren eklenirler. `StringBuilder` içerisinde istediğimiz bir sıraya veri eklemek için `Insert` metodunu kullanabiliriz.

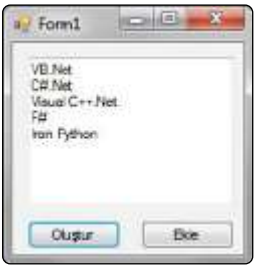
Kullanımı şu şekildedir.

```
StringBuilder.Insert(index numarası, veri, adet)
```

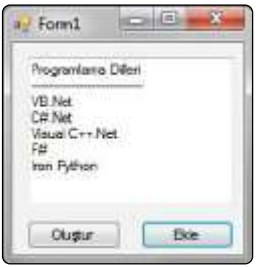
- **Index numarası:** `StringBuilder` içerisine eklenecek olan verinin index numarasını `int32` türünden belirtir. `StringBuilder` içerisinde index numaraları 0'dan itibaren başlar.
- **Veri:** `StringBuilder` içerisine eklenecek olan veridir.
- **Adet:** Eklenecek olan verinin adedini belirtir.

```
using System;
using System.Text;
using System.Windows.Forms;
namespace StringBuilder_Insert
{
public partial class Form1 : Form
{
 public Form1()
 {
  InitializeComponent();
 }
 StringBuilder sb = new StringBuilder();
 private void btnOlustur_Click(object sender, EventArgs e)
 {
  sb.AppendLine("VB.Net");
  sb.AppendLine("C#.Net");
  sb.AppendLine("Visual C++.Net");
  sb.AppendLine("F#");
  sb.AppendLine("Iron Python");
  textBox1.Text = sb.ToString();
 }
 private void btnEkle_Click(object sender, EventArgs e)
 {
  sb.Insert(0, "Programlama Dilleri" + "\r\n");
  sb.Insert(21, "-", 30);
  sb.Insert(51, "\r\n");
  textBox1.Text = sb.ToString();
 }
}
}
```

Kodumuzda **btnOlustur** butonu içerisinde **AppendLine** metodu ile **StingBuilder** içerisine satırlar halinde belirtilen değerleri ekledik. Daha sonra **btnEkle** butonuna tıklandığında ise, **StringBuilder** içerisine eklenmiş olan değerlerden daha önceki konuma başka değerleri ekleyebilmek için **Insert** metodu ile belirtilen değerleri belirtilen konumlara ekledik.



Ekleme öncesi



Ekleme Sonrası

Remove Metodu

`StringBuilder` içerisinde istediğimiz bir veriyi silmek için **`Remove`** metodunu kullanabiliriz.

Kullanımı şu şekildedir:

```
StringBuilder.Remove(başlangıç, uzunluk)
```

- **Başlangıç:** **`StringBuilder`** içerisinden silinecek olan verinin index numarasını belirtir.
- **Uzunluk:** **`StringBuilder`** içerisinden silinecek olan verinin uzunluğunu belirtir.

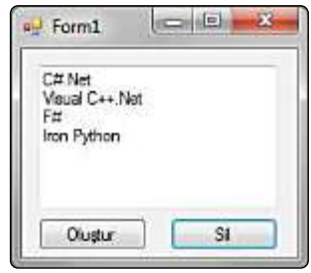
```
StringBuilder sb = new StringBuilder();
private void btnOlustur_Click(object sender, EventArgs e)
{
 sb.AppendLine("VB.Net");
 sb.AppendLine("C#.Net");
 sb.AppendLine("Visual C++.Net");
 sb.AppendLine("F#");
 sb.AppendLine("Iron Python");
 textBox1.Text = sb.ToString();
}
private void btnSil_Click(object sender, EventArgs e)
{
 sb.Remove(0, 7);
 textBox1.Text = sb.ToString();
}
```

Kodumuzda **`btnOlustur`** butonu içerisinde **`AppendLine`** metodu ile

StingBuilder içerisine satırlar halinde belirtilen değerleri ekledik. Daha sonra **btnSil** butonuna tıklandığında ise, **StringBuilder** içerisine eklenmiş olan değerlerden ilk 7 karakteri (satır sonu karakteri de dahil olduğu için 7 karakter) **Remove** metodu ile sildik.



Silme öncesi



Silme Sonrası

Replace Metodu

StringBuilder içerisinde istediğimiz bir veriyi başka bir veri ile değiştirmek için **Replace** metodunu kullanabiliriz.

Kullanımı şu şekildedir:

StringBuilder.Replace (eski değer, yeni değer, başlangıç, adet)

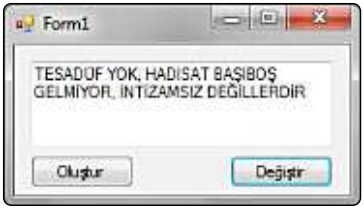
- **Eski değer:** Değiştirilecek olan veriyi belirtir.
- **Yeni değer:** Değiştirilecek olan verinin yerine konulacak veriyi belirtir.
- **Başlangıç:** **StringBuilder** içerisinde değiştirme işleminin başlayacağı index numarasını belirtir.

```
StringBuilder sb = new StringBuilder();
private void btnOlustur_Click(object sender, EventArgs e)
{
 sb.AppendLine("Tesadüf yok, hadisat başıboş gelmiyor, intizamsız
değillerdir");
 textBox1.Text = sb.ToString();
}
private void btnDegistir_Click(object sender, EventArgs e)
{
 sb.Replace(textBox1.Text, textBox1.Text.ToUpper());
 textBox1.Text = sb.ToString();
}
```

Kodumuzda **AppendLine** metodu ile **StingBuilder** içerisine küçük harfli olarak eklediğimiz metni, **Replace** metodu ile ise büyük harfli hali ile değiştirdik.



Değiştirme öncesi



Değiştirme Sonrası

Clear Metodu

StringBuilder içerisindeki tüm karakterleri silmek amacı ile kullanılır.

```
static void Main(string[] args)
{
 string yazi = "KodLab";
 StringBuilder sb = new StringBuilder();
 sb.Append(yazi.ToString());
 Console.WriteLine(sb.Length.ToString());
 sb.Clear();
 Console.WriteLine(sb.Length.ToString());
 Console.ReadKey();
}
```

Örneğimizde, öncelikle oluşturmuş olduğumuz **StringBuilder** nesnesinin ilk boyutunu konsol ekranında yazdırıp, daha sonra **Clear** metodu ile **StringBuilder** nesnesinin içini boşaltıp, **StringBuilder** nesnesinin son boyutunu konsol ekranında yazdırıyoruz. İlk değer olarak **6** değeri görüntülenirken, son değer olarak **0** değeri görüntülenmektedir.

TARİH-ZAMAN METODLARI

13

.NET içerisinde tarih ve zaman değerleri genel olarak **DateTime** veri türü ile ifade edilirler.

Örneğin…

```
DateTime dogum_gunu = new DateTime(2002, 11, 13);
```

… şeklinde **dogum_gunu** adlı tarih değeri içeren bir değişkeni **DateTime** nesnesi türünden tanımlayabiliriz. Bu tarih değeri, **13 Kasım 2002** değerine karşılık gelecektir.

Aynı şekilde tarih ve zaman değerlerini…

```
DateTime zaman = new DateTime(2002, 11, 13, 10, 55, 14)
```

… şeklinde birlikte tanımlayabiliriz. Bu tarih değeri, **13 Kasım 2002**, **saat 10:55:14** değerine karşılık gelecektir.

DateTime Nesnesinin Özellikleri

- **ToLongDateString:** Tarih değerini, uzun tarih formatında görüntülemek amacıyla kullanılır.

Örneğin…

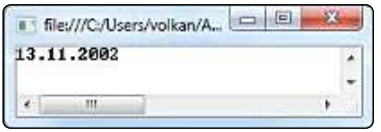
```
DateTime dogum_gunu = new DateTime(2002, 11, 13);
Console.WriteLine( dogum_gunu.ToLongDateString());
Console.ReadKey();
```

… biçiminde yazacağımız kod şu sonucu verecektir.



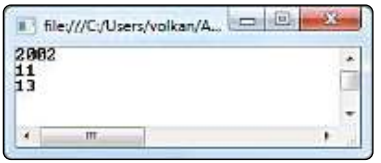
- **`ToShortDateString`:** Tarih değerini, kısa tarih formatında görüntülemek amacıyla kullanılır.

```
DateTime dogum_gunu = new DateTime(2002, 11, 13);
Console.WriteLine( dogum_gunu.ToShortDateString());
Console.ReadKey();
```



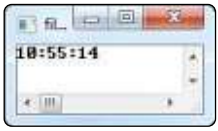
- **`Year`:** Tarih değerinin yıl kısmını verir.
- **`Month`:** Tarih değerinin ay kısmını verir.
- **`Day`:** Tarih değerinin gün kısmını verir.

```
DateTime dogum_gunu = new DateTime(2002, 11, 13);
Console.WriteLine(dogum_gunu.Year);
Console.WriteLine(dogum_gunu.Month);
Console.WriteLine(dogum_gunu.Day);
Console.ReadKey();
```



- **`ToLongTimeString`**: Zaman değerini, uzun zaman formatında görüntülemek amacıyla kullanılır.

```
DateTime zaman = new DateTime(2002, 11, 13,10,55,14);
Console.WriteLine(zaman.ToLongTimeString());
Console.ReadKey();
```



- **`ToShortTimeString`**: Zaman değerini, kısa zaman formatında görüntülemek amacıyla kullanılır.

```
DateTime zaman = new DateTime(2002, 11, 13,10,55,14);
Console.WriteLine(zaman.ToShortTimeString());
Console.ReadKey();
```



- **`Hour`**: Zaman değerinin saat kısmını verir.
- **`Minute`**: Zaman değerinin dakika kısmını verir.
- **`Second`**: Zaman değerinin saniye kısmını verir.
- **`Millisecond`**: Zaman değerinin milisaniye kısmını verir.

```
DateTime zaman = new DateTime(2002, 11, 13,10,55,14);
Console.WriteLine(zaman.Hour);
Console.WriteLine(zaman.Minute);
Console.WriteLine(zaman.Second);
Console.ReadKey();
```



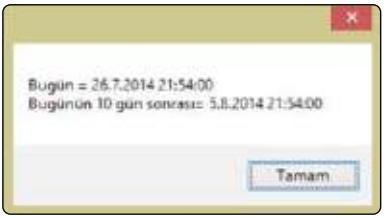
Tarih ve Zaman Üzerinde Ekleme ve Çıkarma Yapmak

DateTime sınıfını kullanarak tarih ve zaman değerleri üzerinde ekleme ve çıkarma işlemleri gerçekleştirebiliriz. Bu amaçla kullanılabilecek metotlar şunlardır:

- **AddDays:** Belirtilen gün değeri kadar, geçerli tarih değeri üzerine ekleme yapar.
- **AddHours:** Belirtilen saat değeri kadar, geçerli zaman değeri üzerine ekleme yapar.
- **AddMilliseconds:** Belirtilen milisaniye değeri kadar, geçerli zaman değeri üzerine ekleme yapar.
- **AddMinutes:** Belirtilen dakika değeri kadar, geçerli zaman değeri üzerine ekleme yapar.
- **AddMonths:** Belirtilen ay değeri kadar, geçerli tarih değeri üzerine ekleme yapar.
- **AddSeconds:** Belirtilen saniye değeri kadar, geçerli zaman değeri üzerine ekleme yapar.
- **AddYears:** Belirtilen yıl değeri kadar, geçerli tarih değeri üzerine ekleme yapar.

Bugünün tarih değerinin üzerine **10 gün** ekleyerek bir mesaj penceresi içerisinde görüntüleyelim.

```
private void button1_Click(object sender, EventArgs e)
{
 DateTime bugun = DateTime.Now;
 DateTime yenitarih;
 yenitarih = bugun.AddDays(10);
 MessageBox.Show("Bugün = " + bugun.ToString() +
  Environment.NewLine + "Bugünün 10 gün sonrası = " +
  yenitarih.ToString());
}
```



Kodumuzda **DateTime.Now** metodu ile bugünün tarihini elde ettik.

```
Bugun.AddDays(10)
```

… satırında ise **AddDays** metodunu kullanarak bugünün tarih değerinin üzerine 10 gün ekledik.

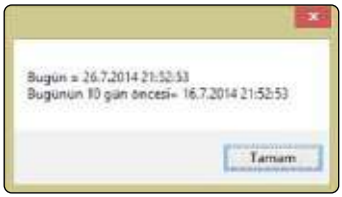
Tarih ve zaman ekleme işlemlerini gerçekleştirirken, eklenen değerin önüne – işareti ekleyerek tarih ve zaman değerlerini arttırmak yerine azaltabiliriz.

Yukarıda gerçekleştirdiğimiz örnek üzerinde değişiklik yaparak…

```
yenitarih=bugun.AddDays(10);
```

… satırını aşağıdaki gibi değiştirelim.

```
yenitarih=bugun.AddDays(-10);
```



Kodumuzu çalıştırdığımızda, programın çalıştırıldığı tarihin 10 gün öncesini görüntüleriz.

TimeSpan Nesnesi

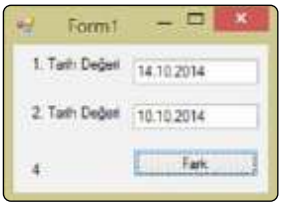
Tarih ve zaman değerleri üzerinde ekleme, çıkarma gibi işlemler yaparken, işlem sonucunu **TimeSpan** nesnesi türünden bir değişkene aktararak işlem sonucunu birimlere ayırabiliriz. **TimeSpan** değişkeni ile elde edilen işlem sonucunu **gün**, **saat**, **dakika**, **saniye** ve **milisaniye** cinsinden elde edebiliriz.

Örneğin iki tarih arasındaki farkı gün cinsinden bulan bir örnek uygulama yapalım.

```
private void btnFark_Click(object sender, EventArgs e)
{
 DateTime tarih1 = Convert.ToDateTime(txtTarih1.Text);
 DateTime tarih2 = Convert.ToDateTime(txtTarih2.Text);
 TimeSpan ts = tarih1 - tarih2;
 lblSonuc.Text = ts.Days.ToString();
}
```



Örneğimizde görüldüğü gibi birinci tarih değeri, ikincisinden küçük olduğu için, tarih değerleri arasındaki fark **-4** olarak gösterildi.



Eğer tam tersi olursa, yani birinci tarih değeri ikincisinden büyük olursa, tarih değerleri arasındaki fark **4** olarak gösterilir.

TimeSpan nesnesinin **TotalDays**, **TotalHours**, **TotalMinutes**, **TotalSeconds**, **TotalMiliseconds** özelliklerini kullanarak, tarih değerleri arasındaki farkı, çeşitli birimler cinsinden ifade edebiliriz.

Yukarıdaki örneğimizde gün cinsinden ifade ettiğimiz tarih değerleri arasındaki farkı saat cinsinden ifade etmek için…

```
lblSonuc.Text = ts.Days.ToString();
```

… satırını aşağıdaki gibi değiştirmemiz yeterlidir.

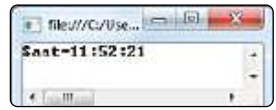
```
lblSonuc.Text = ts.TotalHours.ToString();
```



TimeSpan nesnesini yeni zaman değerleri oluşturmak amacıyla da kullanabiliriz.

```
static void Main(string[] args)
{
 TimeSpan zaman = new TimeSpan(11, 52, 21);
 Console.WriteLine("Saat=" + zaman.ToString());
 Console.ReadKey();
}
```

Örneğimizi çalıştırdığımızda saat, dakika, saniye değerlerini belirttiğimiz zaman isimli yeni bir değişkenin oluşturulduğunu görebiliriz.



Add ve **Subtract** metotlarını kullanarak **TimeSpan** nesneleri üzerinde ekleme ve çıkarma işlemleri gerçekleştirebiliriz. Örneğin; **Add** metodunu kullanarak oluşturacağımız bir zaman değeri üzerine ekleme yapalım.

```
static void Main(string[] args)
{
 TimeSpan zaman = new TimeSpan(11, 52, 21);
 Console.WriteLine("Saat=" + zaman.ToString());
 TimeSpan EklenecekSaat=TimeSpan.FromHours(3);
 Console.WriteLine("Bu saatin 3 saat sonrası = " +
         zaman.Add(EklenecekSaat));
 Console.ReadKey();
}
```

Örneğimizde öncelikle zaman isminde **TimeSpan** türünden bir nesne oluşturduk.

```
TimeSpan EklenecekSaat=TimeSpan.FromHours(3);
```

Daha sonra **EklenecekSaat** isimli bir başka **TimeSpan** nesnesi oluşturduk. Oluşturmuş olduğumuz bu **TimeSpan** nesnesine, saat türünden **3** değerini aktardık.

```
zaman.Add(EklenecekSaat)
```

Sonra da 3 saatlik değeri, zaman isimli değerimizin üzerine ekledik.



Subtract metodunu kullanarak iki tarih değeri arasındaki farkı alan bir örnek yapalım.

```
static void Main(string[] args)
{
 DateTime tarih = new DateTime(2014, 11, 05);
 TimeSpan fark = DateTime.Now.Subtract(tarih);

 Console.WriteLine("Bugün = " + DateTime.Now.ToShortDateString());
 Console.WriteLine("Bugünün tarihi ile " + tarih.ToShortDateString()
+ " arasındaki fark = " + fark.Days +" gün");
 Console.ReadKey();
}
```

Örneğimizde;

```
TimeSpan fark = DateTime.Now.Subtract(tarih);
```

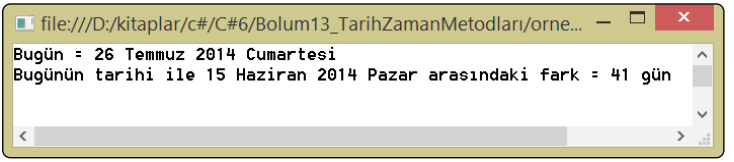
Satırında bugünün tarihinden, tarih isimli değişkene atanan tarih değerini çıkartarak sonucu fark isimli TimeSpan nesnesi türünden değişkene aktardık.

```
fark.Days
```

… ifadesi ile de fark değerinin gün kısmını elde ettik.

```
DateTime.Now.ToShortDateString()
```

Bu kod satırında da **`ToShortDateString()`** ifadesi dikkatinizi çekmiş olmalı. Bu özellik ile belirtilen tarih değerinin kısa tarih formatında görüntülenmesini sağladık. Eğer aynı tarih değerini uzun tarih formatında görüntülemek istersek, **`ToLongDateString()`** özelliğini kullanabiliriz.



Tarih ve Zamanları Karşılaştırmak

Tarih ve zamanları karşılaştırmak için **DateTime** sınıfının **CompareTo** ve **Equals** metotları kullanılabilir.

- **CompareTo**: İki tarih değerini karşılaştırarak geriye **integer** türünden bir değer döndürür. Aşağıdaki gibi kullanılır.

```
sonuç= tarih1.CompareTo(tarih2);
```

Dönen sonuç değeri;

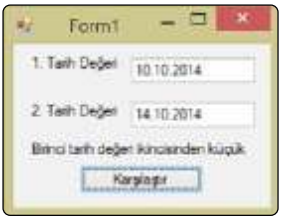
-1 ise, birinci tarih değeri ikincisinden daha küçüktür.

1 ise, birinci tarih değeri ikincisinden daha büyüktür.

0 ise, her iki tarih değeri birbirine eşittir.

Örnek olarak iki ayrı metin kutusuna girilen tarih değerlerini karşılaştıran bir uygulama yapalım.

```
private void btnKarsilastir_Click(object sender, EventArgs e)
{
  string tarih1;
  int sonuc;
  tarih1 = string.Format("{0:d}", txtTarih1.Text);
  string tarih2;
  tarih2 = string.Format("{0:d}", txtTarih2.Text);
  sonuc=tarih1.CompareTo(tarih2);
  switch (sonuc)
  {
   case -1:
    lblSonuc.Text = "Birinci tarih değeri ikincisinden küçük";
    return;
   case 1:
    lblSonuc.Text = "Birinci tarih değeri ikincisinden büyük";
    return;
   case 0:
    lblSonuc.Text = "Her iki tarih değeri birbirine eşit";
    return;
  }
}
```



Örneğimizde öncelikle **TextBox**'lar içerisine **`string`** türünden girilen değerleri

```
string.Format("{0:d}", txtTarih1.Text);
```

şeklinde tarih formatına dönüştürdük. Daha sonra…

```
sonuc=tarih1.CompareTo(tarih2);
```

… satırında **`tarih1`** isimli birinci **TextBox** içerisine girilen tarih değerini, **`tarih2`** isimli ikinci **TextBox** içerisine girilen tarih değeri ile karşılaştırarak, dönen değeri **`sonuc`** isimli değişkene aktardık.

```
switch (sonuc)
```

… satırı ile başlayan yapı içerisinde de **`sonuc`** değişkeninin aldığı değere göre karşılaştırma işleminin sonucunu bir **Label** içerisinde görüntüledik.

- **`Equals`**: İki tarih değerini karşılaştırarak geriye **`boolean`** türünden bir değer döndürür. Aşağıdaki biçimde kullanılır.

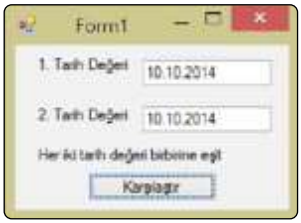
```
sonuç= tarih1.Equals(tarih2);
```

Dönen sonuç değeri **`true`** ise her iki tarih değeri birbirine eşit, **`false`** ise tarih değerleri birbirinden farklıdır.

Örnek olarak iki ayrı metin kutusuna girilen tarih değerlerinin birbirine eşit olup olmadığını bulan bir uygulama yapalım.

```
private void btnKarsilastir_Click(object sender, EventArgs e)
{
 string tarih1;
 bool sonuc;
 tarih1 = string.Format("{0:d}", txtTarih1.Text);
 string tarih2;
 tarih2 = string.Format("{0:d}", txtTarih2.Text);
```

```
 sonuc = tarih1.Equals(tarih2);
 if (sonuc == true)
  lblSonuc.Text = "Her iki tarih değeri birbirine eşit";
 else
  lblSonuc.Text = "Tarih değerleri birbirinden farklı";
}
```



Örneğimizde öncelikle **TextBox**'lar içerisine **`string`** türünden girilen değerleri

```
string.Format("{0:d}", txtTarih1.Text);
```

şeklinde tarih formatına dönüştürdük. Daha sonra…

```
sonuc=tarih1.Equals(tarih2);
```

… **`tarih1`** isimli birinci **TextBox** içerisine girilen **`tarih`** değerinin, **`tarih2`** isimli ikinci **TextBox** içerisine girilen tarih değeri ile eşit olup olmadığı karşılaştırılarak, dönen değeri **`sonuc`** isimli değişkene aktardık. Dönen değerin **`true`** ya da **`false`** olmasına göre sonucu **Label** içerisine yazdırdık.

String Sınıfını Kullanarak Biçimlendirme Yapmak

Tarih ve zaman değeri içeren **`string`**'ler üzerinde biçimlendirme işlemleri yaparken, **`String`** sınıfının **`Format`** metodunu kullanabiliriz. **`Format`** metodu, bir metni formatlı bir biçime dönüştürmek amacıyla kullanılır.

Kullanımı şu şekildedir:

```
String.Format{index[,length][:formatString]}
```

- **`Index`:** String değişkenlerin sırasını belirtir.

- **`FormatString`:** Biçimlendirme işlemi sırasında kullanılacak karakterleri belirtir. Kullanılan tarih ve zaman **`FormatString`** tipleri şunlardır:

  Standart Date and Time Format Strings: Tarih ve zaman değerleri üzerinde biçimlendirme işlemi yapmak amacıyla kullanılan karakterlerdir.

  Custom Date and Time Format Strings: Tarih ve zaman değerleri üzerinde biçimlendirme işlemi yapmak amacıyla kullanılan karakterler bir araya getirilerek özel format biçimleri elde edilebilir.

- **`Length`:** Biçimlendirme işleminin uygulanacağı karakter uzunluğunu belirtir.

Standart Date and Time Format Strings

Karakter	Tanımı	Örnek
d	Kısa tarih	25.11.2014 20:54:39 -> 25.11.2014
D	Uzun tarih	25.11.2014 20:54:39 -> 25 Kasım Salı
f	Uzun tarih/zaman (kısa zaman)	25.11.2014 20:54:39 -> 25 Kasım Salı 20:54
F	Uzun tarih/zaman (uzun zaman)	25.11.2014 20:54:39 -> 25 Kasım Salı 20:54:39
g	Genel tarih/zaman (kısa zaman)	25.11.2014 20:54:39 -> 25/11/2014 20:54
G	Genel tarih/zaman (uzun zaman)	25.11.2014 20:54:39 -> 25/11/2014 20:54:39
M, m	Ay / Gün	25.11.2014 20:54:39 -> 25 Kasım
t	Kısa zaman	25.11.2014 20:54:39 -> 20:54
T	Uzun zaman	25.11.2014 20:54:39 -> 20:54:39
Y, y	Yıl / Ay	25.11.2014 20:54:39 -> Kasım 2014

`DateTime.Now` metodunu kullanarak o anki tarih ve zamanı bulup, yalnızca tarihi görüntületelim.

```
private void btnMesaj_Click(object sender, EventArgs e)
{
 DateTime zaman;
 zaman = DateTime.Now;
 string sonuc;
 sonuc = string.Format("{0:d}", zaman);
 MessageBox.Show(sonuc);
}
```



Kodumuzda `DateTime.Now` ile o anki tarih ve zaman değerini elde ederek, `d` format karakteri ile kısa tarih biçiminde mesaj penceresi içerisinde görüntüledik.

Custom Date and Time Format Strings

- `d`: Gün bilgisini verir.1 ile 31 arası değer alır.
- `dd`: İki basamaklı olarak gün bilgisini verir. 1 ile 31 arası değer alır.
- `ddd`: Gün bilgisini kısaltarak görüntüler. Örneğin; Pazartesi gününü Pzt olarak görüntüler.
- `dddd`: Gün bilgisini tam hali ile görüntüler. Örneğin; Pazartesi gibi.
- `m`: Ay bilgisini verir.1 ile 12 arası değer alır.
- `mm`: İki basamaklı olarak ay bilgisini verir. 1 ile 12 arası değer alır.
- `mmm`: Ay bilgisini kısaltarak görüntüler. Örneğin; Ağustos ayını Ağu olarak görüntüler.
- `mmmm`: Ay bilgisini tam hali ile görüntüler. Örneğin; Ağustos gibi.
- `y`: Yıl bilgisinin son basamağını verir. 2014 yılı için 4 rakamı görüntülenir.
- `yy`: Yıl bilgisinin son iki basamağını verir. 2014 yılı için 14 rakamı görüntülenir.
- `yyy`: Yıl bilgisini tam hali ile görüntüler. Yıl bilgisi 2014 şeklinde görüntülenir.
- `yyyy`: Yıl bilgisini tam hali ile görüntüler. Yıl bilgisi 2014 şeklinde görüntülenir.
- `h`: Saat bilgisini tek basamaklı olarak görüntüler. 1 ile 12 arası değer alır.
- `hh`: Saat bilgisini iki basamaklı olarak görüntüler. 1 ile 12 arası değer alır.
- `H`: Saat bilgisini tek basamaklı olarak görüntüler. 1 ile 24 arası değer alır.
- `HH`: Saat bilgisini iki basamaklı olarak görüntüler. 01 ile 24 arası değer alır.
- `m`: Dakika bilgisini tek basamaklı olarak görüntüler. 0 ile 59 arası değer alır.
- `mm`: Dakika bilgisini iki basamaklı olarak görüntüler. 00 ile 59 arası değer alır.
- `s`: Saniye bilgisini tek basamaklı olarak görüntüler. 0 ile 59 arası değer alır.
- `ss`: Saniye bilgisini iki basamaklı olarak görüntüler. 00 ile 59 arası değer alır.

Bu karakterleri kullanıp örnek formatlar oluşturan bir uygulama yapalım.

```
private void btnislem_Click(object sender, EventArgs e)
{
 string sonuc;

 sonuc = string.Format("{0:hh:mm:ss}", DateTime.Now);
 textBox1.AppendText(sonuc + Environment.NewLine);

 sonuc = string.Format("{0:HH:mm:ss}", DateTime.Now);
 textBox1.AppendText(sonuc + Environment.NewLine);

 sonuc = string.Format("{0:dd MM yyyy}", DateTime.Now);
 textBox1.AppendText(sonuc + Environment.NewLine);

 sonuc = string.Format("{0:dd MMMM yyyy}", DateTime.Now);
 textBox1.AppendText(sonuc + Environment.NewLine);

 sonuc = string.Format("{0:HH:mm:ss dd MMMM yyyy}", DateTime.Now);
 textBox1.AppendText(sonuc);
}
```



26.07.2014 tarihi ve **22:13:17** zamanı için işlemler yaptığımızda sonuç yanda görüldüğü gibi olacaktır.

MATEMATİKSEL METODLAR

14

Visual Studio içerisinde basit matematiksel işlemler gerçekleştirmek için gerekli operatörler mevcut olduğu gibi, detaylı matematiksel işlemler için de **`Math`** sınıfından faydalanılır.

Matematiksel Operatörlerin işlem önceliği sırası aşağıdaki gibidir:

Operatör	Simgesi
Çarpma, Bölme	`* / %`
Ekleme, Çıkarma	`+ -`
Eşitlik (Eşit, Eşit Değil)	`==, !=`
Ve (And)	`&`
Özel Veya (Xor)	`^`
Veya (Or)	`|`
Ve Değil (Nand)	`&&`
Veya Değil (Nor)	`||`
Koşul	`?`

Örneğin;

```
Console.WriteLine(3 * 2 + 3);
```

… şeklinde yazacağımız kod satırı **9** sonucunu üretecektir. Aynı işlemi…

```
Console.WriteLine(3 * (2 + 3));
```

… şeklinde parantez kullanarak yeniden düzenlersek **15** sonucunu üretecektir.

Basit matematiksel işlemler, operatörler bölümünde ayrıntısı ile işlendiği için burada tekrar değinmeyeceğiz.

Math Sınıfı

Math sınıfı detaylı matematikse işlemler ve formüller için kullanılabilen pek çok metoda sahiptir. Bu metotların başlıcaları şunlardır:

- **Abs(sayı)**: Belirtilen sayının mutlak değerini verir.

  Math.Abs(-2) işlemi **2** sonucunu verecektir.

- **Acos(sayı)**: Belirtilen sayının ark kosinüsünü veya ters kosinüsünü verir.

  Math.Acos(0.5) işlemi **1,0471975511966** sonucunu verir.

- **Asin(sayı)**: Belirtilen sayının ark sinüsünü veya ters sinüsünü verir.

  Math.Asin(0.5) işlemi **0,523598775598299** sonucunu verir.

- **Atan(sayı)**: Belirtilen sayının ark tanjantını veya ters tanjantını verir.

  Math.Atan(2) işlemi **1,10714871779409** sonucunu verir.

- **Atan2(sayı1,sayı2)**: Belirlenen **x** ve **y** koordinatlarının arktanjantını veya ters tanjantını verir.

  Math.Atan2(1,1) işlemi **0,785398163397448** sonucunu verir.

- **Ceiling(sayı)**: Belirtilen sayıyı üst sayıya yuvarlar.

  Math.Ceiling(2.4) işlemi **3** sonucunu verir.

- **Cos(sayı)**: Belirtilen sayının kosinüsünü verir.

  Math.Cos(0.5) işlemi **0,877582561890373** sonucunu verir.

- **Cosh(sayı)**: Belirtilen sayının hiperbolik kosinüsünü verir.

  Math.Cosh(4) işlemi **27,3082328360165** sonucunu verir.

- **Exp(sayı)**: e üzeri sayının değerini verir.

  Math.Exp(2) işlemi **7,38905609893065** sonucunu verir.

- **Floor(sayı)**: Belirtilen sayıyı alt sayıya yuvarlar.

  Math.Floor(2.4) işlemi **2** sonucunu verir.

- **Log(sayı)**: Belirtilen sayının e tabanındaki logaritmasını verir

  Math.Log(2) işlemi **0,693147180559945** sonucunu verir.

- **Log(sayı, taban)**: Belirtilen sayının belirlediğiniz tabandaki logaritmasını verir.

  Math.Log(8,2) işlemi **3** sonucunu verir.

- **Log10(sayı)**: Belirtilen sayının on tabanındaki logaritmasını verir.

  Math.Log10(2) işlemi **0,301029995663981** sonucunu verir.

- **Max(sayı 1, sayı 2)**: Belirtilen sayılardan en büyüğünü verir.

  Math.Max(3,5) işlemi **5** sonucunu verir.

- **Min(sayı 1, sayı 2)**: Belirtilen sayılardan en küçüğünü verir.

  Math.Min(3,5) işlemi 3 sonucunu verir.

- **Pow(x,y)**: **x** üzeri **y**'yi verir.

  Math.Pow(2, 3) işlemi **8** sonucunu verir.

- **Round(sayı)**: Belirtilen sayıyı en yakın tam sayıya yuvarlar.

  Math.Round(2.50) işlemi **2** sonucunu verirken, **Math.Round(2.51)** işlemi **3** sonucunu verir.

- **Sign(sayı)**: Belirtilen sayının işaretini bulmak amacıyla kullanılır. Sayı negatif ise **-1**, pozitif ise **1**, **0** ise **0** sonucunu verir.

  Math.Sign(-6) işlemi **-1** sonucunu verirken, **Math.Sign(6)** işlemi **1** sonucunu verir. **Math.Sign(0)** işlemi ise **0** sonucunu verir.

- **Sin(sayı)**: Belirtilen sayının sinüsünü verir.

  Math.Sin(0.5) işlemi **0,479425538604203** sonucunu verir.

- **Sinh(sayı)**: Belirtilen sayının hiperbolik sinüsünü verir.

  Math.Sinh(0.5) işlemi **0,521095305493747** sonucunu verir.

- **Sqrt(sayı)**: Belirtilen sayının karekökünü verir.

  Math.Sqrt(25) işlemi **5** sonucunu verir.

- **Tan(sayı)**: Belirtilen sayının tanjantını verir.

  Math.Tan(0.5) işlemi **0,54630248984379** sonucunu verir.

- **Tanh(sayı)**: Belirtilen sayının hiperbolik tanjantını verir.

  Math.Tanh(0.5) işlemi **0,46211715726001** sonucunu verir.

- **Truncate(sayı)**: Belirtilen sayıyı yuvarlama yapmadan göstermek amacıyla kullanılır.

  Math.Truncate(2.51) işlemi **2** sonucunu verir.

Math sınıfı ile ilgili bir metin kutusuna girilen sayının sinüs ve kosinüsünü radyan ve derece cinsinden bulacak örnek bir uygulama yapalım.

Bir sayının radyan türünden sinüsünü bulmak için **Math.Sin(Sayı)** biçimindeki kullanım yeterli iken, aynı sonucu derece cinsinden **Math.Sin((Math.PI/180)*Sayı)** biçimindeki kullanım ile buluruz.

```
private void btnSin_Click(object sender, EventArgs e)
{
 if (rdDerece.Checked == true)
 {
  textBox1.Text = Convert.ToString(Math.Sin
   ((Convert.ToDouble(Math.PI) / 180) *
    (Convert.ToDouble(textBox1.Text))));
 }
 else
 {
  textBox1.Text = Convert.ToString(Math.Sin
   (Convert.ToDouble(textBox1.Text)));
 }
}
private void btnCos_Click(object sender, EventArgs e)
{
 if (rdDerece.Checked == true)
 {
  textBox1.Text = Convert.ToString(Math.Cos
   ((Convert.ToDouble(Math.PI) / 180) *
    (Convert.ToDouble(textBox1.Text))));
 }
 else
 {
  textBox1.Text = Convert.ToString(Math.Cos
   (Convert.ToDouble(textBox1.Text)));
 }
}
```



Örnek olarak öncelikle `30` değerinin sinüsünü radyan türünden buluyoruz.



Daha sonra aynı değerin sinüsünü, derece türünden buluyoruz.

String Sınıfı Kullanarak Biçimlendirme Yapmak

Matematiksel ifadeler içeren string'ler üzerinde biçimlendirme işlemleri yaparken String sınıfının **`Format`** metodunu kullanabiliriz. **`Format`** metodu, bir metni formatlı bir biçime dönüştürmek amacıyla kullanılır.

Kullanımı şu şekildedir:

```
String.Format{index[,length][:formatString]}
```

- **`Index:`** String değişkenlerin sırasını belirtir.
- **`FormatString:`** Biçimlendirme işlemi sırasında kullanılacak karakterleri belirtir. Kullanılan tarih ve zaman **`FormatString`** tipleri şunlardır:

  `Standart Numeric Format Strings:` Rakamlar üzerinde biçimlendirme işlemi yapmak amacıyla kullanılan karakterlerdir.

  `Custom Numeric Format Strings:` Rakamlar üzerinde biçimlendirme işlemi yapmak amacıyla kullanılan karakterler bir araya getirilerek özel format biçimleri elde edilebilir.
- **`Length:`** Biçimlendirme işleminin uygulanacağı karakter uzunluğunu belirtir.

Standart Numeric Format Strings

Karakter	Tanımı	Örnek
C veya c	Currency	10.495 ("C") -> 10.50 TL
D veya d	Decimal	1234 ("D") -> 1234, -1234 ("D6") -> -001234
E veya e	Exponential (Bilimsel)	1052.0329112756 ("E") -> 1.052033E+003 -1052.0329112756 ("e2") -> -1.05e+003
F veya f	Fixed-point	1234.567 ("F") -> 1234.57 1234 ("F1") -> 1234.0 -1234.56 ("F4",) -> -1234.5600
G veya g	Genel	-123.456 ("G") -> -123.456 123.4546 ("G4") -> 123.5
N veya n	Number	1234.567 ("N") -> 1,234.57 1234 ("N") -> 1,234.0 -1234.56 ("N") -> -1,234.560
P veya p	Yüzde	1 ("P") -> 100.00 % -0.39678 ("P1") -> -39.7 %
X veya x	Hexadecimal	255 ("X") -> FF -1 ("x") -> ff 255 ("x4") -> 00ff -1 ("X4") -> 00FF

Standart Numeric Format String karakterlerini kullanarak örnek bir uygulama yapalım.

```
static void Main(string[] args)
{
 object rakam;
 rakam = 24.495;
 rakam = string.Format("{0:C}", rakam);
 Console.WriteLine(rakam);

 rakam = -12345;
 rakam = string.Format("{0:D4}", rakam);
 Console.WriteLine(rakam);

 rakam = 12345;
 rakam = string.Format("{0:F2}", rakam);
 Console.WriteLine(rakam);

 rakam = 0.24595;
```

```
 rakam = string.Format("{0:P2}", rakam);
 Console.WriteLine(rakam);
 Console.ReadKey();
}
```



Custom Numeric Format Strings

Karakter	Tanımı	Örnek
0	Sıfır	1234.5678 ("00000") -> 01235 0.45678 ("0.00") -> 0.46
#	Basamak	1234.5678 ("#####") -> 1235 0.45678 ("#.##") -> .46
.	Ondalık	0.45678 ("0.00") -> 0.46
,	Gruplandırma	2147483647 ("##,#") -> 2,147,483,647 2147483647 ("#,#,,") -> 2,147
%	Yüzde	0.3697 ("%#0.00") -> %36.97 0.3697 ("##.0 %",) -> 37.0 %
‰	Binde	0.03697 ("#0.00‰") -> 36.97‰

Custom Numeric Format String karakterlerini kullanarak örnek bir uygulama yapalım.

```
static void Main(string[] args)
{
 object rakam;
 rakam = 123456;
 rakam = string.Format("{0:###,###}", rakam);
 Console.WriteLine(rakam);

 rakam = 123.4567;
 rakam = string.Format("{0:###.##}", rakam);
 Console.WriteLine(rakam);

 rakam = 0.24595;
 rakam = string.Format("{0:%#0.00}", rakam);
 Console.WriteLine(rakam);
```

```
 rakam = 0.24595;
 rakam = string.Format("{0:%#0.00}", rakam);
 Console.WriteLine(rakam);
 Console.ReadKey();
}
```



SYSTEM.NUMERICS SINIFI

Byte, **Double** ve **Int32** gibi sayısal veri türlerinin dışında .NET içerisinde **Framework 4.0** ile birlikte kullanılmaya başlanan **System.Numerics** sınıfına ait **BigInteger** ve **Complex** veri türleri de bulunmaktadır. **System.Numerics** sınıfını kullanabilmek için **Add Reference** seçeneği ile projemize **System.Numerics.dll** sınıfını eklememiz gerekmektedir. Ayrıca **using System.Numerics** ifadesi ile **System.Numerics** sınıfını projemize dahil etmeliyiz.

BIGINTEGER YAPISI

Çok büyük boyuttaki verilerin saklanmasında kullanılan yapıdır. .NET içerisindeki diğer veri türlerinin alt ve üst sınırları mevcutken, **BigInteger** için alt ve üst sınır belirlenmemiştir. Ancak gereksiz yere çok büyük boyutta değer belirtilmesi performansı olumsuz etkileyecektir.

```
static void Main(string[] args)
{
 Int64 sonuc;
 sonuc = Int32.MaxValue * Int32.MaxValue;
 Console.WriteLine(sonuc);
 Console.ReadLine();
}
```

Örneğin yukarıdaki kod satırları derleme esnasında **overflow** (*taşma*) hatası vererek çalışmayacaktır.

Aynı kodumuzu **BigInteger** yapısı kullanarak yeniden yazalım.

```
static void Main(string[] args)
{
 BigInteger sonuc;
 sonuc= BigInteger.Multiply(Int32.MaxValue,Int32.MaxValue);
 Console.WriteLine(sonuc);
 Console.ReadLine();
}
```



Kodumuzda **BigInteger** yapısının **Multiply** metodunu kullanarak çarpma işlemini gerçekleştirdiğimizde, uygulamamız herhangi bir **overflow** hatası vermeden doğru bir şekilde sonuç verecektir.

BigInteger yapısı büyük boyuttaki veriler üzerinde işlem yapmayı sağlayan pek çok matematiksel metot ve özelliğe sahiptir.

BigInteger Yapısı Özellikleri

- **IsEven:** Belirtilen sayının çift sayı olup olmadığını geriye döndürür.
- **IsOne:** Belirtilen sayının 1'e eşit olup olmadığını geriye döndürür.
- **IsPowerOfTwo:** Belirtilen sayının 2'nin katı olup olmadığını geriye döndürür.
- **IsZero:** Belirtilen sayının 0'a eşit olup olmadığını geriye döndürür.
- **MinusOne:** Belirtilen sayının -1'e eşit olup olmadığını geriye döndürür.
- **One:** **BigInteger** nesnesine 1 değerine atamak, ya da bir **BigInteger** nesnesi değerini 1 ile karşılaştırmak amacıyla kullanılır.
- **Sign:** Belirtilen sayının pozitif, negatif, ya da sıfır olup olmadığı bilgisini geriye döndürür. Eğer dönen sonuç -1 ise **BigInteger** değeri negatif, 1 ise pozitif, 0 ise sıfırdır.
- **Zero:** **BigInteger** nesnesine 0 değerine atamak ya da bir **BigInteger** nesnesi değerini 0 ile karşılaştırmak amacıyla kullanılır.

```
static void Main(string[] args)
{
 BigInteger sayı;
 sayı = BigInteger.Parse("1267650600228229401496703205376");
 Console.WriteLine("Sayı 2'nin katı mı? = " + sayı.IsPowerOfTwo);
 Console.ReadLine();
}
```

Örneğin; yukarıdaki kod satırında **IsPowerOfTwo** metodu ile **sayı** isimli **BigInteger** değerinin 2'nin katı olup olmadığı sorgulanmakta, sayı ikinin katı olduğu için geriye **true** değerini döndürmektedir. Sayımız çok büyük bir sayı olduğu için string türde yazarak, **Parse** metodu ile **BigInteger** türüne dönüştürdük.

BigInteger Yapısı Metodları

- **Abs:** Belirtilen sayının mutlak değerini verir.
- **Add:** İki adet **BigInteger** değerini toplayarak sonucu geriye döndürür.
- **Compare:** İki adet **BigInteger** değerini karşılaştırarak sonucu geriye döndürür.

  Compare(Sol,Sağ) şeklinde kullanılır.

  Eğer dönen değer 0'dan küçük ise soldaki değer, sağdakinden daha küçüktür.

  Eğer dönen değer 0 ise soldaki değer sağdakine eşittir.

  Eğer dönen değer 0'dan büyük ise soldaki değer, sağdakinden daha büyüktür.
- **Divide:** İki adet **BigInteger** değerini bölerek sonucu geriye döndürür.
- **DivRem:** Bir **BigInteger** değeri ile bir başkasını böler ve sonucu geriye döndürür. Ayrıca bir başka parametre ile bölme işleminden kalanı da geriye döndürür.
- **Equals:** Bir değerin, **BigInteger** değeri ile eşit olup olmadığını gösteren **boolean** türünden bir değer geriye döndürür.
- **GreatestCommonDivisor:** İki adet **BigInteger** değerinin ortak bölenlerinden en büyüğünü bulur.
- **Log(BigInteger):** Belirtilen sayının logaritmasını bulur.
- **Log(BigInteger, Double):** Taban değeri verilen sayının logaritmasını bulur.
- **Log10:** Belirtilen sayının 10 tabanına göre logaritmasını bulur.
- **Max:** İki adet **BigInteger** değerinden büyük olanını geriye döndürür.
- **Min:** İki adet **BigInteger** değerinden küçük olanını geriye döndürür.
- **ModPow:** Belirtilen **BigInteger** değerinin üssünü ve kalanını bulmak amacıyla kullanılır.

```
ModPow(BigInteger değer, BigInteger üs, BigInteger mod)
```

… şeklinde kullanılır.

- **Multiply:** İki adet **BigInteger** değerini çarparak sonucu geriye döndürür.
- **Negate:** Belirtilen **BigInteger** değerinin negatifini alır.
- **Pow:** Belirtilen **BigInteger** değerinin belirtilen değer kadar üssünü alır.
- **Remainder:** İki adet **BigInteger** değerinin bölümünden kalanı geriye döndürür.
- **Subtract:** İki adet **BigInteger** değerini çıkartarak sonucu geriye döndürür.

```
static void Main(string[] args)
{
 BigInteger sayı=3;
 Console.WriteLine(BigInteger.ModPow(sayı,2,2 ));
 Console.ReadLine();
}
```

Örneğin yukarıdaki kod satırında **modPow** metodu ile 3 sayısının 2 üssünü alıp, sonucun 2 ile bölümünden kalan bulunmaktadır. Yani işlemimiz **(2^3)%2** şeklinde olmaktadır. İşlem sonucu 1 olarak görüntülenir.

```
static void Main(string[] args)
{
 BigInteger bolunen=100;
 BigInteger bolen=3;
 BigInteger sonuc = 0, kalan=0;
 sonuc = BigInteger.DivRem(bolunen, bolen, out kalan);
 Console.WriteLine("Sonuç = {0}, Kalan = {1}", sonuc, kalan);
 Console.ReadLine();
}
```

Yukarıdaki örneğimizde ise **DivRem** metodu ile bir **BigInteger** sayısını, belirtilen bir başka **BigInteger** sayısı ile bölerek işlem sonucunu bir değişkene, kalanı ise **out** ifadesi ile bir başka değişkene aktararak konsol ekranında görüntülüyoruz.

100 rakamının 3'e bölünmesiyle işlem sonucu 33, kalan ise 1 olarak konsol ekranında görüntülenecektir.



Complex Yapısı

Complex yapısı karmaşık sayılar tanımlayarak üzerinde işlem gerçekleştirmek amacıyla kullanılır. Bir **gerçek** (*real*) bir de **sanal** (*imaginary*) kısımdan oluşan sayılara karmaşık (sayılar) adı verilmektedir.

a,b $\in$ R (**a** ve **b** gerçek sayı)

ve **i** = $\sqrt{-1}$ ya da **i2 = -1** ise, karmaşık sayılar **z = a + ib** şeklinde ifade edilirler.

a karmaşık sayının gerçek kısmı, **b** sanal kısmı, **i** ise sanal sayı birimidir.

Örneğin; **z = 2-5i ==> R(z) = 2, i(z)=5** şeklindedir.

Real özelliği ile karmaşık sayının gerçek kısmını elde edebilirken, **Imaginary** özelliği ile de karmaşık sayının sanal kısmını elde edebiliriz.

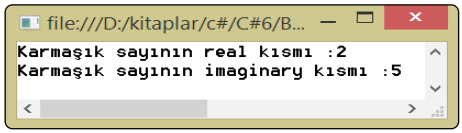
Complex yapısı, karmaşık sayılar üzerinde işlem yapmayı sağlayan pek çok matematiksel metot ve özelliğe sahiptir.

Complex Yapısı Özellikleri

- **Real**: Belirtilen karmaşık sayının gerçek kısmını geriye döndürür.
- **Imaginary**: Belirtilen karmaşık sayının sanal kısmını geriye döndürür.
- **Magnitude**: Belirtilen karmaşık sayının mutlak değerini geriye döndürür.
- **Phase**: Belirtilen karmaşık sayının radyan cinsinden değerini geriye döndürür.

```
static void Main(string[] args)
{
 Complex c = new Complex(2, 5);
 Console.WriteLine("Kompleks sayının real kısmı :" +
c.Real.ToString());
Console.WriteLine("Kompleks sayının imaginary kısmı :" +
c.Imaginary.ToString());
 Console.ReadLine();
}
```

Yukarıdaki örneğimizde **Real** ve **Imaginary** özellikleri ile karmaşık sayının gerçek ve sanal kısımlarını elde ederek konsol ekranını görüntülüyoruz.



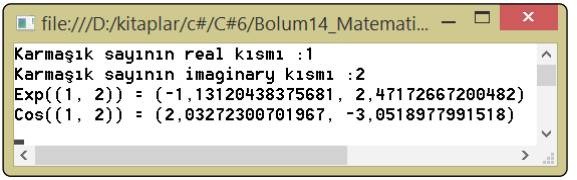
Complex Yapısı Metodları

- **`Abs`**: Belirtilen karmaşık sayının mutlak değerini verir.
- **`Acos`**: Belirtilen karmaşık sayının ark kosinüs değerini verir.
- **`Add`**: İki adet karmaşık sayının değerini toplayarak sonucu geriye döndürür.
- **`Asin`**: Belirtilen karmaşık sayının ark sinüs değerini verir.
- **`Atan`**: Belirtilen karmaşık sayının ark tanjant değerini verir.
- **`Conjugate`**: Belirtilen karmaşık sayının eşleniğini verir. Karmaşık sayının eşleniği demek, karmaşık sayının sanal kısmının işaretinin tersinin alınmasıdır. Örneğin; **`Complex(2,5)`** karmaşık sayısının eşleniği **`(2,-5)`** şeklindedir.
- **`Cos`**: Belirtilen karmaşık sayının kosinüs değerini verir.
- **`Cosh`**: Belirtilen karmaşık sayının hiperbolik kosinüs değerini verir.
- **`Divide`**: İki adet karmaşık sayı değerini bölerek sonucu geriye döndürür.
- **`Equals`**: Bir değerin, karmaşık sayı değeri ile eşit olup olmadığını gösteren **`boolean`** türünden bir değer geriye döndürür.
- **`Log(Complex)`**: Belirtilen karmaşık sayının logaritmasını bulur.
- **`Log(Complex, Double)`**: Taban değeri verilen karmaşık sayının logaritmasını bulur.
- **`Log10`**: Belirtilen karmaşık sayının 10 tabanına göre logaritmasını bulur.
- **`Exp`**: Log metodunun tersi şekilde çalışır e üzeri karmaşık sayının değerini verir
- **`FromPolarCoordinates`**: Bir noktanın polar koordinatlarından bir karmaşık sayı oluşturur.
- **`Multiply`**: İki adet karmaşık sayı değerini çarparak sonucu geriye döndürür.
- **`Negate`**: Belirtilen karmaşık sayı değerinin negatifini alır.
- **`Pow`**: Belirtilen karmaşık sayı değerinin belirtilen değer kadar üssünü alır.
- **`Reciprocal`**: Belirtilen karmaşık sayı değerinin tersini alır. Örneğin Z karmaşık sayısının değerinin tersi 1/|Z| şeklindedir.
- **`Sin`**: Belirtilen karmaşık sayının sinüs değerini verir.
- **`Sinh`**: Belirtilen karmaşık sayının hiperbolik sinüs değerini verir.
- **`Sqrt`**: Belirtilen karmaşık sayının karekökünü verir.
- **`Subtract`**: İki adet karmaşık sayı değerini çıkartarak sonucu geriye döndürür.

- **Tan:** Belirtilen karmaşık sayının tanjant değerini verir.
- **Tanh:** Belirtilen karmaşık sayının hiperbolik tanjant değerini verir.

```
static void Main(string[] args)
{
 Complex c = new Complex(1, 2);
 Console.WriteLine("Karmaşık sayının real kısmı :" +
         c.Real.ToString());
 Console.WriteLine("Karmaşık sayının imaginary kısmı :" +
         c.Imaginary.ToString());
 Console.WriteLine("Exp({0}) = {1}",c, Complex.Exp(c));
 Console.WriteLine("Cos({0}) = {1}", c, Complex.Cos(c));
 Console.ReadLine();
}
```

Kodumuzda **Exp** özelliği ile e üzeri karmaşık sayının değerini ve **Cos** özelliği ile de kosinüsünü bularak konsol ekranına yazdırdık.



REGULAR EXPRESSIONS (DÜZENLİ İFADELER)

15

Bir string ya da string dizisi içerisinden çeşitli ifade kalıplarını kullanarak, bu kalıplara uyan yeni string ya da string dizileri elde etmek amacı ile kullanılır. Örneğin; bir yazı içerisindeki web adreslerini elde etmek ya da bir web sayfasındaki mail adreslerini elde etmek gibi.

Düzenli ifadeleri kullanabilmek için `System.Text.RegularExpressions` sınıfını, proje içerisine dahil etmek gerekmektedir.

Düzenli ifadeler ile birlikte kullanılan ve arama, karşılaştırma ve değiştirme işlemlerinde kullanılan karakterler mevcuttur. Bunlara **metacharacter** adı verilir. Bu karakterleri aşağıdaki gibi gruplandırabiliriz:

Özel Karakterler (Special Characters)

- `$` : Bir metnin bitiş pozisyonunu belirtir. `Multiline` özelliği aktif edildiğinde bir satırın bitiş pozisyonunu ifade eder.
- `()` : Bir alt ifadenin başlangıç ve bitişine işaret eder.
- `*` : Kendisinden önce belirtilen karakterin belirtilen metin içerisinde ya hiç olmadığını ya da bir veya birden fazla mevcut olduğunu belirtir.
- `+` : Kendisinden önce belirtilen karakterin veya alt ifadenin bir veya daha fazla defa tekrarlanabileceğini belirtir.
- `.` Yeni satır karakteri (`\n`) dışındaki herhangi bir karakteri belirtir.
- `[]` Bir aralığın başlangıcını belirtir.

- `?` : Kendisinden önce belirtilen karakterin belirtilen metin içerisinde ya hiç olmadığını ya da mevcut olduğunu belirtir.
- `\` : Kendisinden sonra belirtilen karakterin arama kriterini oluşturan ifade içerisine dahil edilmesini sağlar.
- `/` : Bir düzenli ifadenin başlangıcını veya bitişini belirtir.
- `^` : Karşılaştırma işleminin başlayacağı ilk karakteri belirtir.
- `{}` : Kendisinden önce gelen karakterin ne kadar tekrar edileceğini belirtir.
- `|` : İki eleman arasındaki seçimi belirtir.

Yazdırılmayan Karakterler (Non-Printable Characters)

- `\cx`: **Ctrl** tuşu ile birlikte kullanılan karakterleri belirtir. Örneğin; `\cV` ifadesi **Ctrl+V** tuşlarını belirtir.
- `\n`: Yeni satır başlangıcını belirtir.
- `\r`: Satırbaşı karakterini belirtir.
- `\s`: Boşluk karakterlerini belirtir.
- `\S`: Boşluk olmayan karakterleri belirtir.
- `\t`: Tab karakterini belirtir.

Regex Sınıfı ve Metodları

`Regex` sınıfı arama, karşılaştırma ve değiştirme işlemleri için kullanabileceğimiz metotlara sahip bir sınıftır.

IsMatch Metodu

Bir metnin başka bir metin içerisinde mevcut olup olmadığını bulmak amacı ile kullanılır. Geriye dönen değer **`boolean`**'dır. Eğer geriye dönen değer **`true`** ise metin mevcuttur, **`false`** ise mevcut değildir.

Kullanımı şu şekildedir:

```
bool Regex. IsMatch (string input, string pattern, RegexOptions options)
```

- **`Input`**: Üzerinde işlem yapılacak olan metin.
- **`Pattern`**: Arama işleminde kullanılacak olan kalıp.

- **Options**: Arama seçeneklerini belirtir. Şu değerleri alabilir:

  None: Herhangi bir seçenek uygulanmaz.

  IgnoreCase: Arama işlemi büyük küçük harf duyarlılığı dikkate alınmadan yapılır.

  Multiline: Birden fazla satır üzerinde işlem yapılacağını belirtir. **^** ve **$** karakterlerinin kullanılış biçimini, satır başlangıcı ve bitişi şeklinde değiştirir.

  ExplicitCapture: Aranacak metin içerisindeki aranacak bölümlerin (**?<isim>**…) biçiminde isimlendirildiğini ya da numaralandırıldığını belirtir.

  Compiled: Düzenli ifadenin bir assembly içerisine derlendiğini belirtir. Bu seçenek çalışma hızını arttırır, ancak başlangıç zamanını geciktirir. Bu değer, **CompileToAssembly** metodu çağrıldığı zaman kullanılmamalıdır.

  Singleline: Bir satır üzerinde işlem yapılacağını belirtir. Nokta (**.**) karakterinin kullanılış biçimini değiştirerek herhangi bir karakter yerine geçebilir.

  IgnorePatternWhitespace: Arama kalıbı (*pattern*) içerisindeki boşlukların yok sayılmasını sağlar. **#** karakteri kullanılarak açıklamalar eklenebilir.

  RightToLeft: Arama işleminin soldan sağa doğru değil de sağdan sola doğru yapılmasını sağlar.

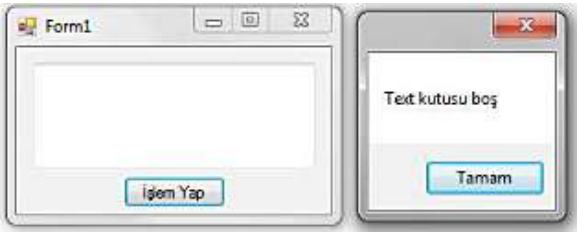
  ECMAScript: ECMAScript uyumluluğunu sağlar. **IgnoreCase**, **Multiline** ve **Compiled** değerleri ile birlikte kullanılır.

  CultureInvariant: Kültürel farklılıkların göz ardı edilmesini sağlar.

Örnek olarak bir text kutusu içeriğinin boş olup olmadığının kontrolünü yaptıralım.

```
using System;
using System.Text.RegularExpressions;
using System.Windows.Forms;
namespace Regex_IsMatch
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
  }
```

```
  private void btnIslem_Click(object sender, EventArgs e)
  {
   if (Regex.IsMatch(textBox1.Text, "^\\s*$"))
    MessageBox.Show("Text kutusu boş");
   else
    MessageBox.Show("Text kutusu boş değil");
  }
 }
}
```



Örneğimizde karşılaştırma kalıbı olarak **^\s*$** ifadesini kullandık.

^\s ile başlangıcın boşluk olabileceğini,

***** ile boşluğun bir ya da birden fazla olabileceğini,

$ ile arama metninin sonuna gelindiği belirtilir.

Bu arama kriterine uygun bir sonuç çıktığında, **IsMatch** metodu geriye **true** değeri döndürecektir. Bu da text kutusunun boş olduğunu belirtir.

Match Metodu

Bir metni başka bir metin içerisinde aramak amacı ile kullanılır. Eğer aranan metin bulunursa geriye aranan metin döndürülür.

Kullanımı şu şekildedir:

```
Match Regex. Match (string input, string pattern, RegexOptions options)
```

Örnek olarak bir text kutusu içerisine girilen bilgilerden ad bilgisini bulan bir uygulama yapalım:

```
private void btnIslem_Click(object sender, EventArgs e)
{
 if (Regex.IsMatch(textBox1.Text, "^\\s*$"))
  MessageBox.Show("Text kutusu boş");
 else
 {
  string ad = Regex.Match(textBox1.Text, "^Adı:(.*)",
              RegexOptions.IgnoreCase).Groups[1].Value;
  MessageBox.Show(ad.ToString());
 }
}
```

Örneğimizde…

^Adı: ile **Adı:** kelimesi ile başlayan metinleri,

(.*) ile de **Adı:** kelimesinden sonra gelebilecek herhangi bir metni arıyoruz.

Groups metodu ile bulunan metinler dizi içerisinde saklanır. Örneğimizde **Groups[1].Value** şeklindeki kullanım ile bulunan değerlerden ilk sıradaki değeri görüntülüyoruz.



Text kutularına girilen isim, numara, kimlik no ve telefon numaralarını kontrol ederek yanlış bir değer girişini önleyen bir uygulama yapalım.

```
private void txtAdSoyad_Leave(object sender, EventArgs e)
{
 Regex regex = new Regex("^[a-zA-Z\\s]{1,30}$");
 if (!regex.Match(txtAdSoyad.Text).Success)
 {
```

```
  MessageBox.Show
   ("30 karakter uzunluğunda bir string girebilirsiniz!");
  txtAdSoyad.Focus();
 }
}
private void txtTCKimlik_Leave(object sender, EventArgs e)
{
 Regex regex = new Regex("^\\d{11}$");
 if (!regex.Match(txtTCKimlik.Text).Success)
 {
  MessageBox.Show("T.C. Kimlik numarasında yanlışlık var!");
  txtTCKimlik.Focus();
 }
}
private void txtTelefon_Leave(object sender, EventArgs e)
{
Regex regex = new Regex
 ("^[01]?[- .]?(\\([2-9]\\d{2}\\)|[2-9]\\d{2})[- .]?\\d{3}[- .]?\\d{4}$");
 if (!regex.Match(txtTelefon.Text).Success)
 {
  MessageBox.Show("Telefon numarasında yanlışlık var!");
  txtTelefon.Focus();
 }
}
private void txtNumara_Leave(object sender, EventArgs e)
{
 Regex regex = new Regex("^\\d*$");
 if (!regex.Match(txtNumara.Text).Success)
 {
  MessageBox.Show("Yalnızca rakam girebilirsiniz!");
  txtNumara.Focus();
 }
}
```

Örneğimizde **`Regex regex = new Regex ("")`** satırları ile yeni bir **`regex`** deseni tanımlayarak, **`Regex.Match`** metodu ile **TextBox** içerisindeki içeriğin düzenli ifade kalıbına uyup uymadığı kontrol edilmekte ve **`Success`** metodu ile eğer dönen değer **`true`** ise **TextBox** içerisindeki değerin düzenli ifade kalıbına uygun olduğu belirlenmektedir. Örneklerimizde **`Not (!)`** operatörünü kullanarak

işlemin tersi alınmakta ve **TextBox** içerisindeki değer, düzenli ifade kalıbına uygun olmadığında kullanıcı bilgilendirilmektedir.



Matches Metodu

`Match` metodunda olduğu gibi bir metni başka bir metin içerisinde aramak amacı ile kullanılır. Ancak bulunan tek bir sonucu değil tüm sonuçları içerir.

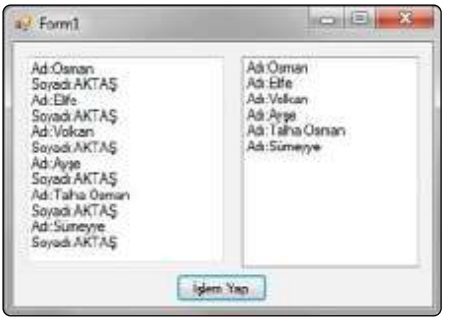
Kullanımı şu şekildedir:

```
MatchCollection Regex.Matches (string input, string pattern,
RegexOptions options)
```

Bir text kutusu içerisinde yazılan metin içerisindeki adları bularak, bir **ListBox** içerisine ekleyen bir uygulama yapalım:

```
private void btnIslem_Click(object sender, EventArgs e)
{
 if (Regex.IsMatch(textBox1.Text, "^\\s*$"))
  MessageBox.Show("Text kutusu boş");
 else
 {
  foreach (Match isim in Regex.Matches(textBox1.Text,
     "^Adı:(.*)", RegexOptions.Multiline))
   listBox1.Items.Add(isim.Value);
 }
}
```

Örneğimizde arama kriterine uyan metin sayısı kadar bir döngü kurarak bu kritere uyan metinleri **ListBox** içerisine ekliyoruz. **`Matches`** ifadesine dikkat edecek olursak, arama yapacağımız metin birden çok satırdan oluştuğu için **`RegexOptions.Multiline`** seçeneğini kullandık. Aksi takdirde yalnızca ilk satırdaki metin üzerinde işlem yapılacaktır.



Bir internet sayfasındaki mail adreslerini ayıklayarak, bulunan mail adreslerini **ListBox** içerisine ekleyen örnek uygulama yapalım.

```
using System;
using System.Text.RegularExpressions;
using System.Net;
using System.IO;
using System.Windows.Forms;
namespace Regex_Mail
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
  }
  private void btnislem_Click(object sender, EventArgs e)
  {
   lbldurum.Text = "Siteye Bağlanıyor...";
   WebResponse resp = null;
   try
   {
    WebRequest WebReq = WebRequest.Create(textBox1.Text);
    resp = WebReq.GetResponse();
    lbldurum.Text = "Siteye Bağlandı";
    Stream str = resp.GetResponseStream();
    StreamReader reader = new StreamReader(str);
    string kaynak = reader.ReadToEnd();

    Regex mailDeseni = new Regex
```

```
      ("[\\w-]+(?:\\.[\\w-]+)*@(?:[\\w-]+\\.)+[a-zA-Z]{2,7}");
     listBox1.Items.Clear();
     foreach (Match isim in mailDeseni.Matches(kaynak))
      listBox1.Items.Add(isim.Value);
     lbldurum.Text = "Bulunan mailler listelendi";
    }
    catch
    {
     lbldurum.Text = "Siteye Bağlanamıyor.";
    }
   }
  }
}
```



Örneğimizde…

```
WebRequest WebReq = WebRequest.Create(textBox1.Text);
```

… satırında **`WebRequest.Create()`** metodu ile **`textBox1`** içerisinde yazılan web adresine bağlanmak için istekte bulunuyoruz.

```
resp = WebReq.GetResponse();
```

`GetResponse()` metodu ile siteden yanıt alıyoruz.

```
Stream str = resp.GetResponseStream();
```

`GetResponseStream()` metodu ile sitenin html kodunu okuyoruz.

```
StreamReader reader = new StreamReader(str);
```

Yeni bir **StreamReader** nesnesi oluşturarak, içeriğine **str** değişkeni içerisinde tutulan sitenin html kodunu aktarıyoruz.

```
string kaynak = reader.ReadToEnd();
```

kaynak adlı string bir değişken tanımlayarak, içeriğine **StreamReader** nesnesi içerisinde tutulan sitenin html kodunu, başından sonuna kadar aktarıyoruz.

```
Regex mailDeseni = new Regex ("[\\w-]+(?:\\.[\\w-]+)*@
(?:[\\w-]+\\.)+[a-zA-Z]{2,7}");
```

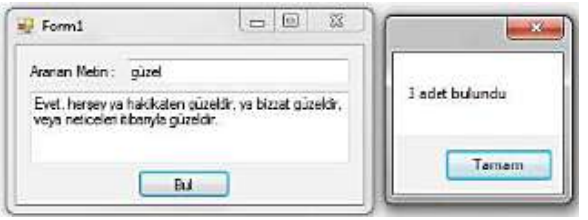
Bu satırda, mail deseni adlı yeni bir **Regex** nesnesi oluşturuyoruz.

```
foreach (Match isim in mailDeseni.Matches(kaynak))
{
   listBox1.Items.Add(isim.Value);
}
```

… satırlarında ise, **kaynak** değişkeni içerisinde tutulan sitenin **html** kodunu taratarak, içerisinde **maildeseni** isimli düzenli ifade kalıbımıza uyan değerleri **ListBox** içerisine ekliyoruz.

Bir text kutusu içerisinde aranan bir metnin kaç defa tekrarlandığını bulan örnek bir uygulama yapalım. Bunun için **Regex.Matches.Count** metodunu kullanacağız.

```
private void btnBul_Click(object sender, EventArgs e)
{
 if (Regex.IsMatch(txticerik.Text, "^\\s*$"))
  MessageBox.Show("Text kutusu boş");
 else
  MessageBox.Show(Regex.Matches(txticerik.Text, txtaranan.Text,
RegexOptions.Multiline).Count + " adet bulundu");
}
```



Replace Metodu

Bir metnin içerisinde geçen bir ya da birden fazla ifadeyi başka bir ifade ile değiştirmek amacı ile kullanılır. Kullanımı şu şekildedir:

```
Regex.Replace(Üzerinde işlem yapılan metin, Değiştirilecek metin)
```

Bir text kutusu içerisinde yazılan metin içerisindeki belirtilen metni, başka bir metin ile değiştiren bir uygulama yapalım.

```
private void btnDegistir_Click(object sender, EventArgs e)
{
 if (Regex.IsMatch(txticerik.Text, "^\\s*$"))
  MessageBox.Show("Text kutusu boş");
 else
 {
  Regex regex = new Regex(txtAranan.Text);
  txticerik.Text = regex.Replace(txticerik.Text, txtDegistirilecek.Text);
 }
}
```



Değiştirme işlemi öncesi



Değiştirme işlemi sonrası

Split Metodu

Bir metni belirtilen bir tasarım kalıbına göre parçalara bölmek amacı ile kullanılır. Kullanımı şu şekildedir:

```
Regex.Split(Üzerinde işlem yapılan metin, "Bölme işleminde
kullanılacak karakter")
```

Bir text kutusu içerisinde yazılan metni belirtilen bir karaktere göre bölerek, bir **ListBox** içerisine aktaran bir uygulama yapalım.

```
private void btnBol_Click(object sender, EventArgs e)
{
 byte i;
 string[] dizi = Regex.Split(txticerik.Text, ",");
 for (i = 0; i <= dizi.Length - 1; i++)
  listBox1.Items.Add(dizi[i]);
}
```



Örneğimizde **TextBox** içerisine yazılan metni , (*virgül*) karakterine göre parçalara bölerek, bölünen parçaları `dizi` isimli bir dizi değişkeni içerisine aktardık ve bir döngü içerisine dizi elemanlarını sırası ile okutarak **ListBox** içerisine aktardık.

RESOURCE DOSYALARI

16

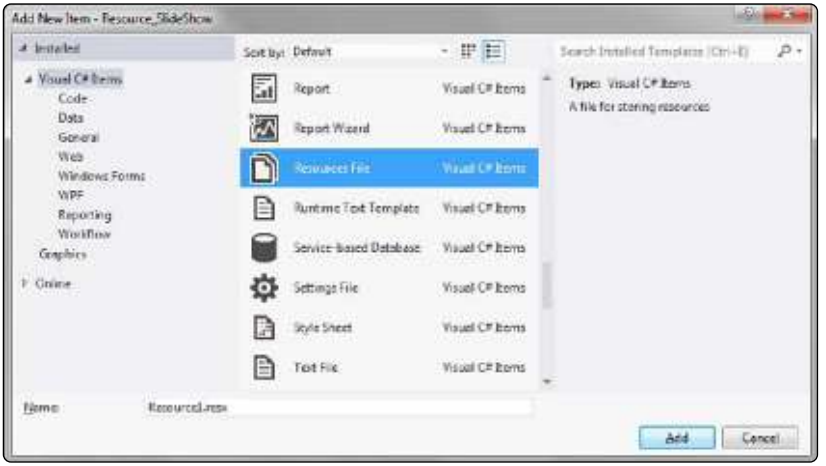
.Net içerisinde resim, müzik, video gibi dosyaları haricen projelerimize eklemek amacıyla **resource** dosyaları kullanılır. Resource dosyaları içerisine iki yöntemle dosya ekleyebiliriz.

Birinci yöntem, **local resource** olarak eklemektir. Form ya da Form üzerindeki **PictureBox** gibi bir resim kontrolüne resim yüklemek istediğimizde karşımıza **Select Resource** penceresi gelecektir. Bu penceredeki **Local Resource** seçeneği işaretlenip resim dosyaları seçilip **Import** butonu tıklandığında, proje içerisinde kullanacağımız dosyaları, proje başka bilgisayarlarda çalıştığında oraya taşımamız gerekir.

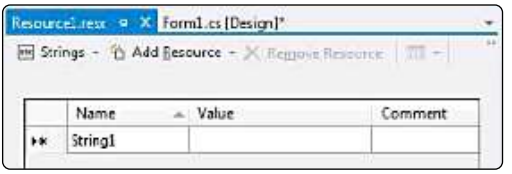


İkinci yöntem ise, **resource** dosyası oluşturarak içerisine eklemektir. Bu yöntemde dosyalar proje içerisine eklenmiş olur. Resource dosyaları **resx** uzantısı ile kaydedilirler. Resource dosyası içerisine eklenen tüm resim, yazı, ses, video gibi dosyalara tek bir resource dosyası içerisinden erişilebilir.

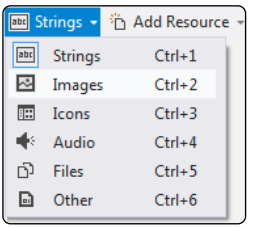
Örnek bir resource dosyası oluşturarak içerisine resim dosyaları ekleyelim. **Project / Add New Item** seçeneği ile yeni bir **Resources File** oluşturuyoruz.



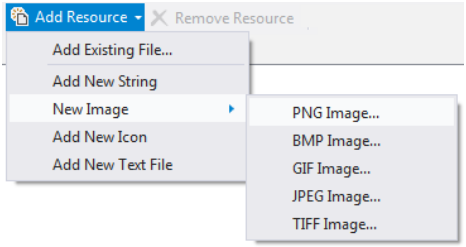
Gelen **Resources** ekranında **Resources** dosyası içerisine ekleme ve silme işlemleri gerçekleştirebiliriz.



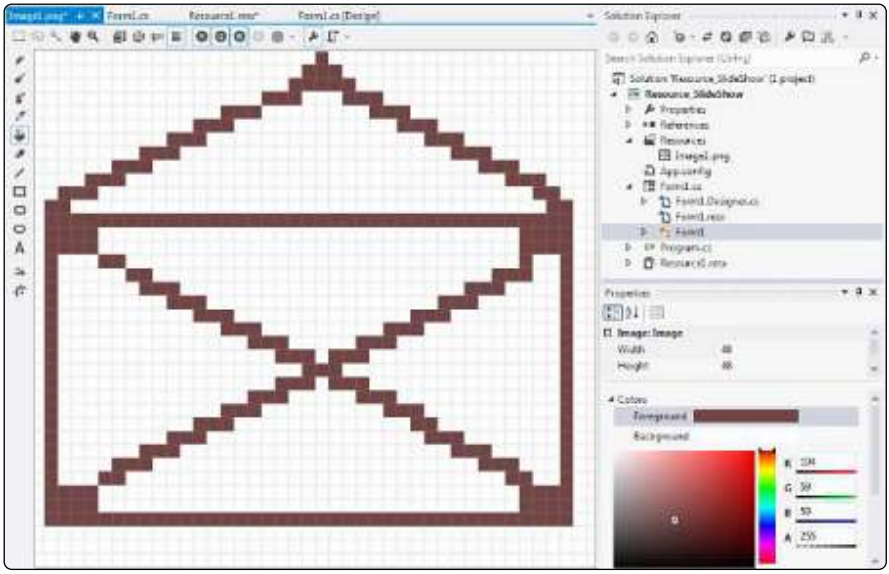
Strings yazısının yan tarafında bulunan kulakçığa tıklayarak **Resources** dosyası içerisine eklenebilecek dosya türünü değiştirebiliriz. Biz resmi dosyası ekleyeceğimiz için **Images** seçeneğini işaretliyoruz.



Add Resource menüsü altındaki **Add New Image** seçeneği ile yeni bir resim dosyası oluşturulur.

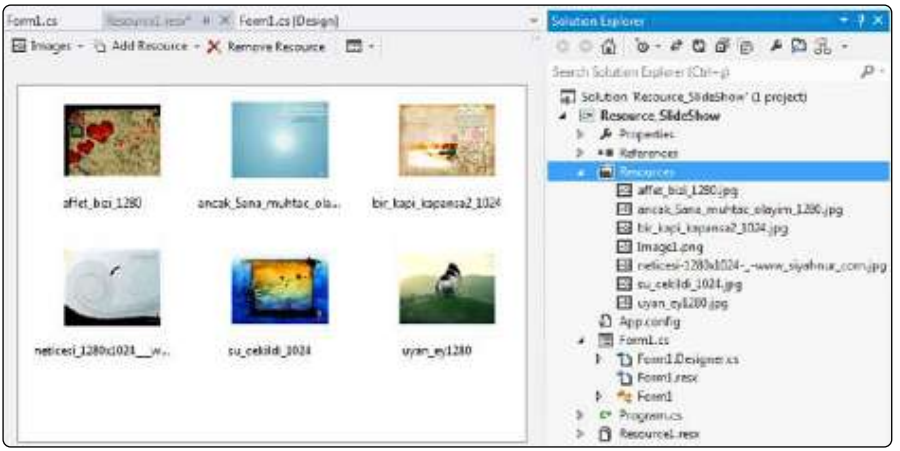


Yeni bir resim dosyası oluşturmak için karşımıza bir tasarım ekranı gelecektir. Bu ekran vasıtası ile yeni bir resim dosyası oluşturabiliriz. Ayrıca gerekli olan temel araçlar **Visual Studio IDE** içerisine eklenen bir araç çubuğu vasıtası ile sağlanır. Burada temel çizim ve boyama işlemleri, şekiller oluşturma gibi işlemleri gerçekleştirebiliriz.



Properties penceresi yardımı ile de resim boyutu, renk düzeni, sıkıştırma gibi ayarlamaları yapabilirsiniz.

Daha önceden var olan resim dosyalarını **Resources File** içerisine eklemek için ise **Add Existing File** seçeneğini kullanabileceğimiz gibi resim dosyalarını fare ile sürükleyip **Resource File** paneli içerisine bırakabiliriz.



Resource dosyası ile ilgili olarak **Properties Window** içerisinden şu özellikleri değiştirebiliriz.

- **`FileName`:** Resource dosyasının ismini değiştirebiliriz.
- **`Build Action`:** Resource dosyasının derlenmesi ile ilgili ayarlar yapılır. Şu değerleri alır:

  `None`: Hiçbir işlem yapılmaz. Belgeleme işlemleri ve derlenmemiş dosyaların proje içerisine eklenmesi amacıyla kullanılabilir.

  `Compile`: Resource dosyasını derler.

  `Content`: Setup projesinde Resource dosyası içerisindeki dosyalara referans verilebilir.

  `Embedded Resource`: Kaynak dosyaları resource dosyasının içerisine yerleştirir.

Örnek olarak önceden oluşturulmuş bir resource dosyasına eklenmiş olan resimleri kullanarak slayt gösterisi yapalım.

Form üzerine 1 adet **Timer** nesnesi (**`interval`** değeri **`1000`** olacak), 1 adet **PictureBox** nesnesi, 1 adet de **`numericUpDown`** nesnesi (**`minimum`** değeri **`1`**, **`maximum`** değeri **`100`** olacak) ekliyoruz. Projemiz içerisinde kullanılacak resimler için **Project / Add New Item** menüsünden **Resource File** seçeneği ile yeni bir **Resource** dosyası ekleyelim. **Reseource File** dosyasını seçerek de **Add Existing File** menüsünden 5 adet örnek resim ekleyelim. **PictureBox** nesnesinin **`SizeMode`** özelliğini **`StretchImage`** olarak belirleyerek resmin **PictureBox** boyutuna göre otomatik olarak ayarlanmasını sağlıyoruz.

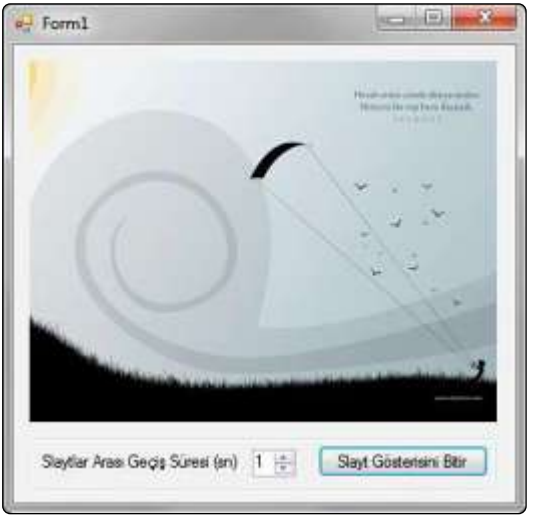
```
Byte i;
private void btnSlayt_Click(object sender, EventArgs e)
{
  if (btnSlayt.Text == "Slayt Gösterisini Başlat")
  {
    btnSlayt.Text = "Slayt Gösterisini Bitir";
    timer1.Enabled = true;
    timer1.Interval = Convert.ToInt32(numericUpDown1.Value) * 1000;
  }
  else
  {
    btnSlayt.Text = "Slayt Gösterisini Başlat";
    timer1.Enabled = false;
  }
}
private void timer1_Tick(object sender, EventArgs e)
{
  i++;
  if (i == 1)
    pictureBox1.Image = Resource1.r1;
  else if (i == 2)
    pictureBox1.Image = Resource1.r2;
  else if (i == 3)
    pictureBox1.Image = Resource1.r3;
  else if (i == 4)
    pictureBox1.Image = Resource1.r4;
  else if (i == 5)
  {
    pictureBox1.Image = Resource1.r5;
    i = 0;
  }
}
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
{
  timer1.Interval = Convert.ToInt32(numericUpDown1.Value) * 1000;
}
```

Kodumuzda **`timer`** nesnesi içerisinde `1`'den `5`'e kadar sırasıyla **PictureBox** nesnesinin **`image`** özelliğine **Resource** dosyası içerisindeki resimleri aktarıyoruz.

Resource dosyası içerisindeki resimlere **Resource** dosyası ismini yazıp noktaya bastıktan sonra, erişebileceğiniz için kolayca yazabilirsiniz.



Slaytlar arası geçiş süresini saniye cinsinden bir **NumericUpDown** nesnesi yardımı ile ayarlayabiliyoruz. **NumericUpDown** nesnesinin `Value` özelliğinde kullanıcı tarafından seçilmiş olan değeri `1000` ile çarparak **Timer** nesnesinin `Interval` özelliğine aktarıyoruz. Bu sayede **NumericUpDown** nesnesinde seçili olan değer ile slaydımızın geçiş süresini belirliyoruz.



Resource Dosyası İşlemleri

`System.Resources` sınıfını kullanarak `Resources` dosyaları üzerinde işlem yapabiliriz. Projemize `System.Resources` sınıfını dahil ettikten sonra ise, Assembly nesnesine erişebilmemiz için ayrıca `System.Reflection` sınıfını projenize dahil etmeniz gerekmektedir.

System.Resources sınıfı ile kullanılan metotlar şunlardır:

ResourceManager

Yeni bir **Resource** dosyası oluşturmak için **Resource Manager** nesnesi kullanılır.

Kullanımı şu şekildedir:

```
ResourceManager değişkenadı = new ResourceManager("Resources dosyası adı",
Assembly.GetExecutingAssembly( ));
```

- **Assembly.GetExecutingAssembly**: O an üzerinde çalıştığımız Assembly nesnesini verir.

ResXResourceReader

Resource dosyası içerisinden okuma yapmak amacıyla kullanılır. Kullanımı şu şekildedir:

```
ResXResourceReader değişkenadı
= new ResXResourceReader("Resources dosyası adı");
```

GetEnumerator Metodu

Bir dizi içerisindeki elemanlara tek tek erişebilmek amacı ile kullanılır. Bu metodu kullanmak için kullanacağımız değişkeni **IDictionaryEnumerator** türünden tanımlamamız gerekir.

ResXResourceWriter

Bir resource dosyası içerisinde yazma işlemi gerçekleştirmek amacı ile kullanılır. Kullanımı şu şekildedir:

```
ResXResourceWriter değişkenadı
= new ResXResourceWriter("Resources dosyası adı ");
```

AddResource Metodu

Resource dosyası içerisine ekleme yapmak amacı ile kullanılır. Kullanımı şu şekildedir:

```
ResXResourceNesnesi.AddResource(İsim,Değer);
```

Bir **resource** dosyası içerisine eklenen nesnenin iki parametresi vardır. Biri isim diğeri ise değer parametresidir.

Örneğin; bir resim dosyası ekliyorsak:

```
rdy.AddResource("Resim1" , "C:\\ornekresim.jpg");
```

Bir yazı ekliyorsak…

```
rdy.AddResource(isim , "Volkan");
```

… şeklinde ekleme yapabiliriz.

Örnek olarak **Resource** dosyası üzerinde ekleme ve okuma işlemleri yapan bir uygulama yapalım. **Form** üzerine 2 adet **Button**, 1 adet **PictureBox**, 1 adet **listBox**, 1 adet de **OpenFileDialog** nesnesi ekliyoruz. **PictureBox** nesnesinin **`SizeMode`** özelliğini **`StretchImage`** olarak belirleyerek, resmin **PictureBox** boyutuna göre otomatik olarak ayarlanmasını sağlıyoruz.

```
using System;
using System.Resources;
using System.Reflection;
using System.Collections;
using System.Windows.Forms;
using System.Drawing;
namespace resources_file
{
  public partial class Form1 : Form
  {
    public Form1()
    {
         InitializeComponent();
    }
    private void btnDosyaEkle_Click(object sender, EventArgs e)
    {
       if (openFileDialog1.ShowDialog() == DialogResult.OK)
           listBox1.Items.Add(openFileDialog1.FileName);
    }
    private void Form1_Load(object sender, EventArgs e)
    {
```

```
        try
        {
            ResXResourceReader rdo =
               new ResXResourceReader("resources.resx");
            IDictionaryEnumerator nmr = rdo.GetEnumerator();
            while (nmr.MoveNext())
            {
               listBox1.Items.Add(nmr.Key);
            }
            rdo.Close();
          }
          catch { }
    }
    private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
        try
        {
          ResXResourceReader rdo =
             new ResXResourceReader("resources.resx");
          IDictionaryEnumerator nmr = rdo.GetEnumerator();
          while (nmr.MoveNext())
          {
            if (nmr.Key.ToString() == listBox1.Text.ToString())
                pictureBox1.Image = new Bitmap(nmr.Value.ToString());
          }
            rdo.Close();
          }
          catch { }
    }
    private void btnKaydet_Click(object sender, EventArgs e)
    {
        ResXResourceWriter rdy = new ResXResourceWriter("resources.resx");
        Byte i;
        for(i = 0;i<=listBox1.Items.Count - 1;i++)
           rdy.AddResource("Resim" + i, listBox1.Items[i].ToString());
        rdy.Close();
        MessageBox.Show
          ("Tüm dosyalar resources dosyası içerisine kaydedildi!");
     }
  }
}
```

Form'un **Load** olayı içerisinde eğer daha önceden oluşturulmuş bir **Resource** dosyası mevcutsa, bunun içerisinde mevcut olan dosyalar varsa, onları okuyarak **ListBox** içerisine aktarıyoruz.

```
ResXResourceReader rdo = new ResXResourceReader("resources.resx");
```

Satırında **rdo** adlı yeni bir **ResXResourceReader** nesnesi oluşturarak **resources.resx** adlı **resource** dosyasını okumak üzere açıyoruz.

```
IDictionaryEnumerator nmr = rdo.GetEnumerator();
```

Satırında **nmr** adlı bir nesneyi **IDictionaryEnumerator** türünden tanımlayarak **rdo** nesnesinin **GetEnumerator** metodu ile resources dosyasının elemanlarına erişim sağlıyoruz.

```
while (nmr.MoveNext())
```

Bir **while** döngüsü oluşturarak sıra ile resource dosyasının elemanları arasında dolaşıyoruz.

```
listBox1.Items.Add(nmr.Key);
```

ListBox nesnesi içerisine okuduğumuz elamanları **Key** değerini aktarıyoruz.

```
rdo.Close();
```

Okuma işleminin bitiminde **ResxResourceReader** nesnemizi kapatıyoruz.

btnKaydet nesnesinin **Click** olayı içerisinde **ListBox** içerisine eklediğimiz dosyaları, **Resources** dosyası içerisine kaydetme işlemi yapacağız.

```
ResXResourceWriter rdy = new ResXResourceWriter("resources.resx");
```

Satırında **rdy** adlı yeni bir **ResXResourceWriter** nesnesi oluşturarak **resources.resx** adlı resource dosyasını yazmak üzere açıyoruz.

```
for(i = 0;i<=listBox1.Items.Count - 1;i++)
```

ListBox nesnesi içerisinde eleman sayısı kadar bir döngü kurarak bu döngü içerisinde…

```
rdy.AddResource("Resim" + i, listBox1.Items[i].ToString());
```

… **`AddResource`** metodu ile **Resource** dosyası içerisine isim, değer şeklinde **ListBox** içerisindeki dosyaları ekliyoruz.

```
rdy.Close();
```

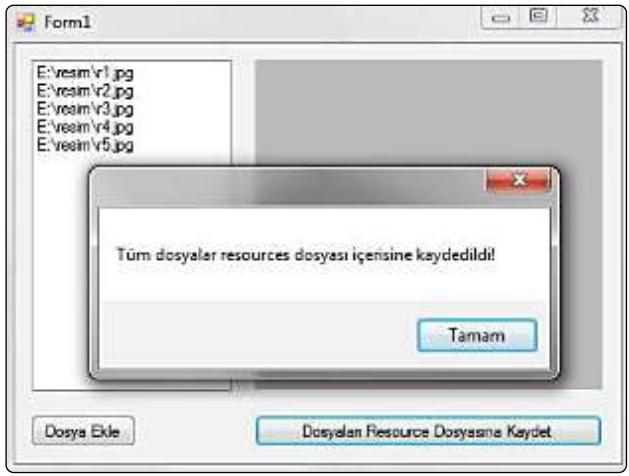
Yazma işleminin bitiminde **`ResxResourceWriter`** nesnemizi kapatıyoruz.

ListBox nesnesinin **`SelectedIndexChanged`** olayında ise **ListBox** içerisindeki resim dosyası isimlerine tıklandıkça…

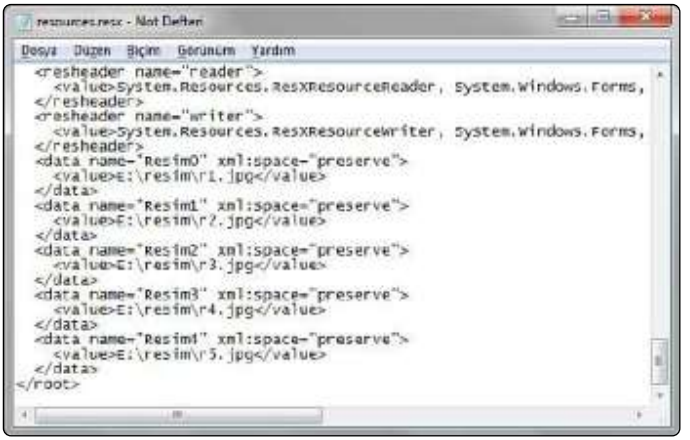
```
if (nmr.Key.ToString() == listBox1.Text.ToString())
    pictureBox1.Image = new Bitmap(nmr.Value.ToString());
```

Eğer **Resource** dosyası içerisinde okunan isim ile **ListBox** nesnesi içerisinde okunan isim birbirine eşit ise, **PictureBox** içerisinde **resource** dosyası içerisinde okunan değer ismi ile yeni bir **Bitmap** nesnesi oluşturularak görüntülenir.

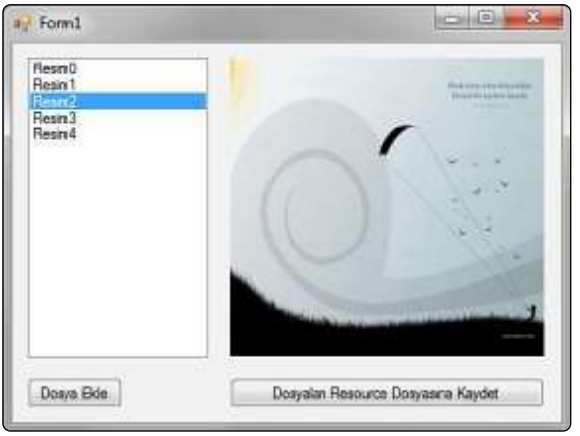
Örneğimizi çalıştırarak **Dosya Ekle** düğmesine basalım. Dosya açma penceresi yardımı ile resim dosyalarının yollarını ve isimlerini **ListBox** içerisine ekleyelim ve **Dosyaları Resources Dosyasına Kaydet** düğmesine tıklayarak **ListBox** içerisinde dosyaları **Resources** dosyası içerisine ekleyelim.



Dosyaların eklendiğini uygulamamızın **Bin\Debug** klasörü içerisine kaydettiğimiz **resources.resx** dosyası içerisine bakarak da görebiliriz. Burada **`data name`** tag'ında dosya ismini, **`<value>`** tag'ında ise dosya yolunu görebiliriz. İstediğimiz takdirde bu dosyayı elle düzenleyerek de ekleme yapabiliriz.



Projeyi kapatıp yeniden açtığımızda ise **resources.resx** dosyasının mevcut olup olmadığının kontrolü yapılmaktadır. Dosyamız mevcut olduğundan içerisindeki resim dosyalarının ismi **ListBox** nesnesi içerisine aktarılmaktadır. **ListBox** içerisindeki resim dosyası isimlerine tıklandığında ise, **PictureBox** içerisinde ilgili resim dosyası görüntülenmektedir.



SYSTEM.DRAWING SINIFI

17

.NET içerisinde temel seviyede resim ve grafik işlemleri için **System.Drawing** sınıfı kullanılır.

System.Drawing sınıfı altındaki **Graphics** sınıfı, **Form** üzerine yazı ya da grafikleri çizdirmek amacıyla kullanılırken, **Image** sınıfı ise resimler üzerinde işlem yapmak amacıyla kullanılır.

Graphics Sinifi

Graphics sınıfı basit seviyede yapılacak grafiksel işlemler için yeterli özellik ve metodlara sahiptir. Form üzerine yazı ve grafik çizimi ve bu çizimlere uygulanacak temel seviyedeki efektler Graphics sınıfı ile kolayca uygulanabilir.

Form Üzerine Yazı Çizimi

Graphics sınıfının **DrawString** metodu ile **Form** üzerine yazdıracağımız yazıyı çizdirip yazıma hazırlarız.

DrawString metodunun kullanımı aşağıdaki gibidir:

```
DrawString (s As String, font As Font, brush As Brush, x As Single,
y As Single, format As StringFormat)
```

- **s:** Çizdirilecek olan string'in ismi.
- **font:** Çizdirilecek olan metnin yazı tipi.
- **brush:** Çizdirilecek olan metnin renk ve görünüm ayarlarını içerir.

- **x**: Çizdirilecek olan metnin yatay koordinatını belirtir.
- **y**: Çizdirilecek olan metnin dikey koordinatını belirtir.
- **format**: Çizdirilecek olan metnin görünüm ve yerleşim ayarlarını belirtir. Bu ayarlar **StringFormatFlags** altında belirlenir. Yapılabilecek bazı ayarlar aşağıdaki gibidir:

  DirectionRightToLeft: Yazıyı sağdan sola doğru olacak biçimde görüntüler.

  DirectionVertical: Yazıyı dikey biçimde görüntüler.

  NoWrap: Bu seçenek kullanıldığında çizdirilen yazının, çizdirilen şekil tarafından otomatik olarak çevrelenmesi iptal edilir.

  LineLimit: Çizdirilen bir şekil içerisine yazdırılan satırlar, şeklin çizgileri üzerine denk gelmeyecek biçimde kaydırılır.

  NoClip: Çizdirilen bir şekil içerisine yazdırılan satırlar, şeklin içerisine sığmadığı takdirde şeklin dışında devam eder.

Örnek olarak form üzerine örnek bir yazı yazdırıp yazı fontunu ve rengini belirleyelim.

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace Form_Draw_Yazı
{
  public partial class Form1 : Form
  {
   public Form1()
   {
    InitializeComponent();
   }
   private void btnCiz_Click(object sender, EventArgs e)
   {
    Graphics formGraphics = this.CreateGraphics();
    string drawString = "Örnek yazı";
    Font drawFont = new Font("Comic Sans MS", 20);
    SolidBrush drawBrush = new SolidBrush(Color.Black);
    formGraphics.DrawString(drawString, drawFont, drawBrush, 50, 30);
    drawFont.Dispose();
```

```
    drawBrush.Dispose();
    formGraphics.Dispose();
   }
  }
}
```

drawFont değişkeni ile **20 punto** yüksekliğinde ve **Comic Sans MS** tipinde bir font tanımladık.

drawBrush değişkeni ile siyah rengi tanımladık.



DrawString metodu ile tanımladığımız font ve renk özelliklerinde **50,30** koordinatlarında belirlediğimiz örnek metni yazdırdık.

Dispose metodunu kullanarak da **Form** üzerine çizim yaparken kullandığımız hafıza kaynaklarının boşaltılmasını sağladık.

Örneğimiz üzerinde değişiklik yaparak **DrawString** metodunun format parametresini kullanıp, yazımızı dikey olarak çizelim.

```
private void btnCiz_Click(object sender, EventArgs e)
{
 Graphics formGraphics = this.CreateGraphics();
 string drawString = "Örnek yazı";
 Font drawFont = new Font("Comic Sans MS", 20);
 SolidBrush drawBrush = new SolidBrush(Color.Black);

 StringFormat drawFormat = new StringFormat
(StringFormatFlags.DirectionVertical);

 formGraphics.DrawString(drawString, drawFont, drawBrush, 60, 10, drawFormat);

 drawFont.Dispose();
 drawBrush.Dispose();
 formGraphics.Dispose();
}
```

Yazımızı dikey yazdırmak amacıyla öncelikle…

```
System.Drawing.StringFormat drawFormat =  new
 System.Drawing.StringFormat(StringFormatFlags.DirectionVertical);
```

… kod satırlarında görüldüğü gibi **`drawFormat`** adıyla bir değişken tanımlayarak, içerisine **`StringFormatFlags.DirectionVertical`** değerini aktardık.



Tanımladığımız bu format biçimini…

```
formGraphics.DrawString(drawString, drawFont, drawBrush, 60, 10, drawFormat);
```

… kod satırlarında görüldüğü gibi son parametre olarak kullandık.

Windows işletim sistemi içerisinde yüklü olan yazı tiplerinin listesine **`FontFamily`** sınıfının **`Families`** özelliği ile erişebiliriz.

```
private void listBox1_Click(object sender, EventArgs e)
{
 Graphics formGraphics = this.CreateGraphics();
 formGraphics.Clear(SystemColors.Control);
 string drawString = "Örnek yazı";
 SolidBrush drawBrush = new SolidBrush(Color.Black);

 Font drawFont = new Font(listBox1.Text, 20,
  FontStyle.Bold | FontStyle.Italic);

 formGraphics.DrawString(drawString,
  drawFont, drawBrush, 20, 30);

 drawFont.Dispose();
 drawBrush.Dispose();
 formGraphics.Dispose();
}
```

```
private void Form1_Load(object sender, EventArgs e)
{
 foreach (FontFamily fonts in FontFamily.Families)
 {
  listBox1.Items.Add(fonts.Name);
 }
}
```

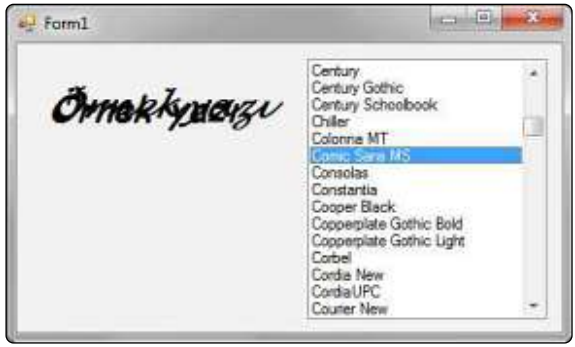
Örneğimizde…

```
formGraphics.Clear(SystemColors.Control);
```

…. satırı ile **Form** üzerine herhangi bir çizim işlemi yapılmadan önce, **Form** üzerini belirttiğimiz renk ile temizledik.



Aksi takdirde form üzerine yapacağımız çizimler aşağıda görüldüğü gibi üst üste gelecektir.



Örneğimizde…

```
Font drawFont = new Font(listBox1.Text, 20, FontStyle.Bold |
FontStyle.Italic);
```

… satırlarında **drawFont** değişkeni ile yeni bir yazı tipi oluştururken, son parametreyi kullanarak yazı tipini kalın ve eğik olarak belirttik.

Form Üzerine Şekil Çizimi

Graphics sınıfının metotlarını kullanarak form üzerine çeşitli şekiller çizdirebiliriz. Bu şekillerin en önemlileri şunlardır:

- **DrawLine:** İki nokta arasını birleştiren çizgi çizmek amacıyla kullanılır. Kullanımı aşağıdaki gibidir:

```
DrawLine(Pen, Point 1, Point 2)
```

  Pen: Form üzerine yazı ya da şekil çizmek amacıyla kullanılan nesnedir. Çizgi rengi, kalınlığı ve stil gibi özelliklere sahiptir.

  Point 1: Başlangıç noktası.

  Point 2: Bitiş noktası.

- **DrawRectangle:** Dikdörtgen çizmek amacı ile kullanılır. Kullanımı aşağıdaki gibidir:

```
DrawRectangle(Pen, Rectangle)
```

  Rectangle(x, y, genişlik, yükseklik) biçiminde tanımlanır.

- **DrawArc:** Yapısı bir dikdörtgen tarafından belirlenen bir elipsin, bir kısmını içeren yay çizmek amacıyla kullanılır. Kullanımı aşağıdaki gibidir:

```
DrawArc(Pen, Rectangle, StartAngle, SweepAngle)
```

  Rectangle(x, y, genişlik, yükseklik) biçiminde tanımlanır.

  StartAngle: Yayın x ekseninde saat yönündeki başlangıç açısını belirtir.

  SweepAngle: Başlangıç açısından yayın bitiş noktasına ölçülen açıyı belirtir.

- **DrawBezier:** Belirtilen dört nokta vasıtasıyla bir eğri çizmek amacıyla kullanılır. Kullanımı aşağıdaki gibidir:

```
DrawBezier(Pen, Point 1, Point 2, Point 3, Point 4)
```

Point 1: Eğrinin başlangıç noktasını belirtir.

Point 2: Eğrinin ilk kontrol noktasını belirtir.

Point 3: Eğrinin ikinci kontrol noktasını belirtir.

Point 4: Eğrinin bitiş noktasını belirtir.

- **DrawCurve:** Bir nokta dizisi yardımıyla eğri çizmek amacıyla kullanılır. Kullanımı aşağıdaki gibidir:

```
DrawCurve(Pen, Point())
```

Point(): Nokta dizisi.

- **DrawClosedCurve:** Bir nokta dizisi yardımıyla kapalı eğri çizmek amacıyla kullanılır. Kullanımı aşağıdaki gibidir:

```
DrawClosedCurve(Pen, Point())
```

Point(): Nokta dizisi.

- **DrawEllipse:** Yapısı bir dikdörtgen tarafından belirlenen bir elips çizmek amacıyla kullanılır. Kullanımı aşağıdaki gibidir:

```
DrawEllipse(Pen, x, y, genişlik, yükseklik)
```

- **DrawPie:** Pasta grafiği çizmek amacıyla kullanılır. Kullanımı aşağıdaki gibidir:

```
DrawPie(Pen, x,y, genişlik, yükseklik, StartAngle, SweepAngle)
```

Rectangle(x, y, genişlik, yükseklik) biçiminde tanımlanır.

StartAngle: Yayın x ekseninde saat yönündeki başlangıç açısını belirtir.

SweepAngle: Başlangıç açısından yayın bitiş noktasına ölçülen açıyı belirtir.

- **DrawPolygon:** Yapısı bir nokta dizisi tarafından belirlenen çokgen çizmek amacıyla kullanılır.

```
DrawPolygon(Pen, Point())
```

`Point()`: Nokta dizisi.

Örnek olarak bu şekillerin hepsini bir form üzerine isimleri ile birlikte çizdiren bir uygulama yapalım:

```
private void btnCiz_Click(object sender, EventArgs e)
{
  int x, y;
  x = 10;
  y = 10;
  Pen myPen = new Pen(Color.Black, 2);
  Graphics formGraphics = null;
  formGraphics = this.CreateGraphics();
  SolidBrush drawBrush = new SolidBrush(Color.Black);
  Font drawFont = new Font("Times New Roman", 12, FontStyle.Bold);

  formGraphics.DrawRectangle(myPen, new Rectangle (x, y, 120, 60));
  formGraphics.DrawString("Rectangle", drawFont, drawBrush, x+30, y + 60);

  formGraphics.DrawArc(myPen, new Rectangle
      (x+150, y, 100, 120),180,180);
  formGraphics.DrawString("Arc", drawFont, drawBrush, x+180, y+60);

  formGraphics.DrawLine(myPen, new Point(x + 300, y+40 ),
      new Point(x + 400, y + 40) );
  formGraphics.DrawString("Line", drawFont, drawBrush, x + 330, y + 60);

  formGraphics.DrawCurve(myPen, new Point[]
      { new Point(x, y + 120), new Point(x + 60, y + 120),
        new Point(x + 60, y + 150), new Point(x + 90, y + 150) });
  formGraphics.DrawString("Curve", drawFont, drawBrush, x+30 , y + 160);

  formGraphics.DrawClosedCurve(myPen, new Point[]
      { new Point(x+150, y + 120), new Point(x + 210, y + 120),
        new Point(x + 210, y + 150), new Point(x + 240, y + 150) });
  formGraphics.DrawString("Closed Curve", drawFont, drawBrush,
      x + 180, y + 160);

  formGraphics.DrawBezier(myPen, new Point(300, y + 140),
      new Point(325, y + 120), new Point(375, y + 160),
      new Point(x + 400, y + 140));
```

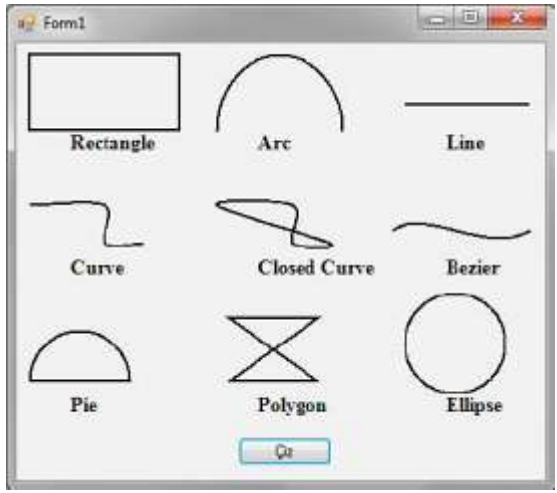
```
  formGraphics.DrawString("Bezier", drawFont, drawBrush, x + 330, y + 160);

  formGraphics.DrawPie(myPen, x, y + 220, 80, 80, 180, 180);
  formGraphics.DrawString("Pie", drawFont, drawBrush, x + 30, y + 270);

  formGraphics.DrawPolygon(myPen, new Point[]
      { new Point(x + 160, y + 210), new Point(x + 230, y + 210),
        new Point(x + 160, y + 260), new Point(x + 230, y + 260) });
  formGraphics.DrawString("Polygon", drawFont, drawBrush, x + 180, y + 270);

  formGraphics.DrawEllipse(myPen, x + 300, y + 190, 80, 80);
  formGraphics.DrawString("Ellipse", drawFont, drawBrush, x + 330, y + 270);

  myPen.Dispose();
  formGraphics.Dispose();
}
```



Kodumuz içerisinde

```
Pen myPen = new Pen(Color.Black,2);
```

Satırında çizgi rengini siyah ve kalınlığını 2 olarak belirledik.

Örneğimizi çalıştırdığımızda form üzerinde çizgiler üzerindeki bazı bozulmalara dikkat etmişsinizdir. Bu bozulmaları **`SmoothingMode`** işlemi ile gidermek mümkün. Ancak bunun için öncelikle **`using`** ifadesini kullanarak

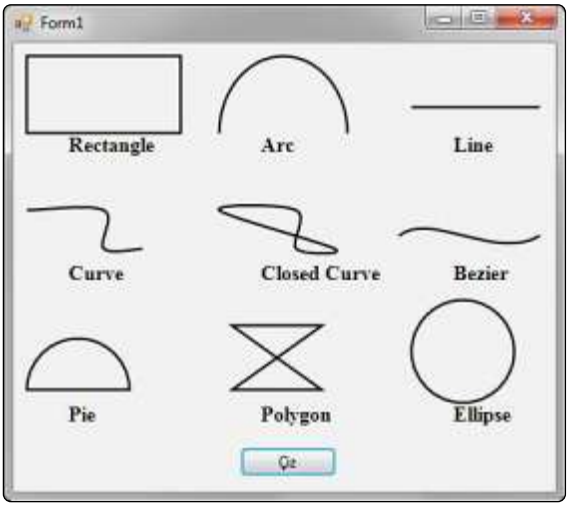
System.Drawing.Drawing2D sınıfını projemize dahil etmemiz gerekmektedir. Bu işlemin ardından kodumuzda…

```
formGraphics = this.CreateGraphics();
```

…. satırından sonra…

```
formGraphics.SmoothingMode = SmoothingMode.AntiAlias;
```

… satırını ekleyerek tekrar çalıştıralım.



Görüldüğü gibi çizgilerdeki bozulmalar giderildi.

Form üzerine çizdirdiğimiz şekillerin içlerini doldurmak için **FillRectangle**, **FillEllipse**, **FillPie**, **FillPolygon**, **FillClosedCurve**, **FillRegion** metotları kullanılır.

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace Form_Draw_FillShape
{
 public partial class Form1 : Form
 {
```

```
  public Form1()
  {
   InitializeComponent();
  }

  private void btnCiz_Click(object sender, EventArgs e)
  {
   Pen myPen = new Pen(Color.Black, 2);
   Graphics formGraphics = null;
   formGraphics = this.CreateGraphics();
   SolidBrush drawBrush = new SolidBrush(Color.Black);
   Brush fillBrush = Brushes.Gray;
   formGraphics.SmoothingMode = SmoothingMode.AntiAlias;

   formGraphics.FillRectangle(fillBrush, new Rectangle
    (10, 10, 250, 80));
   formGraphics.DrawRectangle(myPen, new Rectangle
    (10, 10, 250, 80));

   myPen.Dispose();
   formGraphics.Dispose();
  }
 }
}
```

Örneğimizde **`FillRectangle`** metodunda **`DrawRectangle`** metodu ile çizeceğimiz dikdörtgenle aynı koordinatları kullanarak dikdörtgenin içerisini dolduruyoruz.



Tek bir renk ile şekillerin içerisini doldurabileceğimiz gibi farklı brush seçenekleri ile farklı biçimlerde de şekillerin içerisini doldurabiliriz.

- **`HatchBrush`**: Bu seçeneği kullanarak düz çizgileri kesik çizgiler, noktalar gibi farklı biçimlerle şeklin içerisini bölümleyebiliriz.

Kullanım biçimi aşağıdaki gibidir:

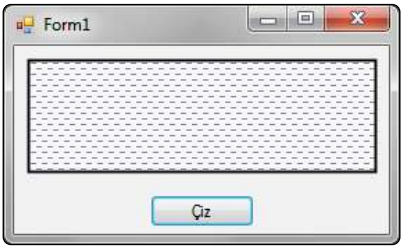
```
HatchBrush(Brush biçimi, Çizim rengi, Zemin Rengi)
```

```
private void btnCiz_Click(object sender, EventArgs e)
{
 Pen myPen = new Pen(Color.Black, 2);
 Graphics formGraphics = null;
 formGraphics = this.CreateGraphics();
 SolidBrush drawBrush = new SolidBrush(Color.Black);
 HatchBrush hBrush = new HatchBrush(HatchStyle.DashedHorizontal,
   Color.Gray,Color.GhostWhite);

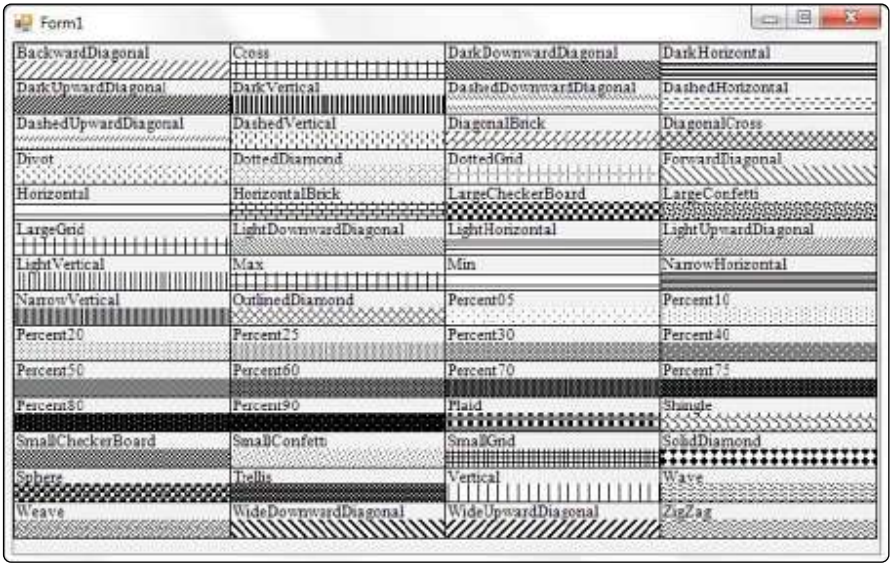
 formGraphics.SmoothingMode = SmoothingMode.AntiAlias;

 formGraphics.FillRectangle(hBrush, new Rectangle (10, y, 250, 80));
 formGraphics.DrawRectangle(myPen, new Rectangle (10, y, 250, 80));

 myPen.Dispose();
 formGraphics.Dispose();
}
```



Kullanabileceğimiz çizgi stillerinin isimleri ve görünümleri aşağıdaki gibidir:



- **`LinearGradientBrush`**: Bu seçeneği kullanarak renk tonlamaları arasında geçişler gerçekleştirebiliriz. **`System.Drawing.Drawing2D`** namespace'i altında yer alır. Kullanım biçimi aşağıdaki gibidir:

```
LinearGradientBrush (Point1 , Point 2,  Renk 1, Renk 2, LinearGradientMode)
```

Renk tonlamasında brush seçeneği olarak 2 seçenek kullanılabilir:

`SetSigmaBellShape`: Çan eğrisi şekli temel alınarak renk tonlaması gerçekleştirilir.

`SetBlendTriangularShape`: Bir merkez renk ve merkezin her iki ucundaki tek bir renk için doğrusal bir renk tonlaması gerçekleştirilir.

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;
namespace LinearGradientBrushes
{
 public partial class Form1 : Form
 {
```

```
    public Form1()
    {
     InitializeComponent();
    }
    private void btnCiz_Click(object sender, EventArgs e)
    {
     int x, y;
     x = 10;
     y = 10;
     Pen myPen = new Pen(Color.Black, 2);
     Graphics formGraphics = null;
     formGraphics = this.CreateGraphics();

     formGraphics.SmoothingMode = SmoothingMode.AntiAlias;

     LinearGradientBrush LGBrush = new LinearGradientBrush
      (ClientRectangle, Color.White, Color.Gray,
       LinearGradientMode.Horizontal);

     formGraphics.FillRectangle(LGBrush, new Rectangle (x, y, 300, 80));
     formGraphics.DrawRectangle(myPen, new Rectangle (x, y, 300, 80));

     LGBrush.SetSigmaBellShape(0.5f);
     formGraphics.FillRectangle(LGBrush, new Rectangle
       (x, y+100, 300, 80));
     formGraphics.DrawRectangle(myPen, new Rectangle
       (x, y+100, 300, 80));

     LGBrush.SetBlendTriangularShape(1);
     formGraphics.FillRectangle(LGBrush, new Rectangle
       (x, y + 200, 300, 80));
     formGraphics.DrawRectangle(myPen, new Rectangle
       (x, y + 200, 300, 80));

     myPen.Dispose();
     formGraphics.Dispose();
   }
  }
}
```



- **PathGradientBrush**: Bu seçenek ile kontrol noktaları vasıtasıyla renk geçişleri gerçekleştirebiliriz. **System.Drawing.Drawing2D** namespace'i altında yer alır.

```
private void btnCiz_Click(object sender, EventArgs e)
{
 Pen myPen = new Pen(Color.Black, 2);
 Graphics formGraphics = null;
 formGraphics = this.CreateGraphics();

 formGraphics.SmoothingMode = SmoothingMode.AntiAlias;

 Point[] points= new Point[] { new Point(10, 10),
                               new Point(310, 10),
                               new Point(310, 90),
                               new Point(10, 90) };

 PathGradientBrush PGBrush = new PathGradientBrush(points);

 formGraphics.FillRectangle(PGBrush, new Rectangle
     (10, 10, 300, 80));
 formGraphics.DrawRectangle(myPen, new Rectangle
     (10, 10, 300, 80));
 }
```

Örneğimizde çizdirdiğimiz dikdörtgen koordinatları ile yaklaşık olarak aynı koordinatlara **PathGradient** biçimi uyguladık. **FillRectangle** metodu ile hazırladığımız gradient biçimi ile dikdörtgenin içini doldurduk.



Şekil ve Yazının Birlikte Kullanımı

Form üzerinde oluşturduğumuz şeklin içerisinde yazı oluşturmak için **DrawString** metodunu kullanırken, tanımladığımız x ve y koordinatları yerine oluşturduğumuz şekli belirtmeliyiz.

Örneğin; **Form** üzerinde bir dikdörtgen oluşturup içerisine yazı ekleyelim.

```
private void btnCiz_Click(object sender, EventArgs e)
{
 Graphics formGraphics = CreateGraphics();
 Font drawFont = new Font("Arial", 20);
 SolidBrush drawBrush = new SolidBrush (System.Drawing.Color.Black);
 StringFormat drawFormat = new StringFormat
           (StringFormatFlags.LineLimit);

 Rectangle r = new Rectangle(new Point(10, 10), new Size(250, 70));
 formGraphics.DrawRectangle(Pens.Black, r);

 formGraphics.DrawString("Birinci Satır" + "\n" +
  "İkinci Satır", drawFont, drawBrush, (Rectangle)r, drawFormat);

 drawFont.Dispose();
 drawBrush.Dispose();
 formGraphics.Dispose();
}
```

Örneğimizde **`DrawString`** metodunu kullanırken x ve y koordinatları parametresi yerine (*rectangle*) **`r`** ifadesini kullanarak karakterlerin…

```
Rectangle r = new Rectangle(new Point(10, 10), new Size(250, 70));
 formGraphics.DrawRectangle(Pens.Black, r);
```

… satırları ile çizdirdiğimiz dikdörtgen içerisine çizdirilmesini sağlıyoruz.

```
StringFormat drawFormat = new StringFormat
     (StringFormatFlags.LineLimit);
```

Satırlarında ise çizdirdiğimiz dikdörtgen içerisine eklenen satırların, dikdörtgenin çizgileri üzerine denk gelmeyecek biçimde kaydırılmasını sağlıyoruz.



Image Sinifi

Image sınıfı resim dosyaları oluşturma, oluşturulan resim dosyaları hakkında detaylı bilgi alma ve resim dosyaları üzerinde temel seviyede işlemler gerçekleştirme amacıyla kullanılan özellik ve metotlara sahiptir.

- **FromFile Metodu:** Klasör yolu belirtilen resim dosyası ile yeni bir **Image** nesnesi oluşturmak amacıyla kullanılır.

Örneğin; **OpenFileDialog** nesnesi ile seçilen bir resim dosyası ile yeni bir **Image** nesnesi oluşturarak bunu **PictureBox** içerisinde görüntüleyelim.

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace Image_From_File
{
  public partial class Form1 : Form
  {
   public Form1()
   {
    InitializeComponent();
   }
   private void btnSec_Click(object sender, EventArgs e)
   {
    DialogResult buton= openFileDialog1.ShowDialog();
    if (buton == DialogResult.OK)
    {
     Image image = Image.FromFile(openFileDialog1.FileName);
     pictureBox1.Image = image;
    }
   }
  }
}
```

Kodumuzda **OpenFileDialog** penceresini görüntüleyerek, pencerede **OK** butonu tıklandığında seçili olan dosya ismi ve yolu ile **FromFile** metodunu kullanarak yeni bir **Image** nesnesi oluşturduk. **PictureBox** kontrolünün **Image** özelliğine oluşturmuş olduğumuz **Image** nesnesini atayarak resim dosyasını görüntüledik.

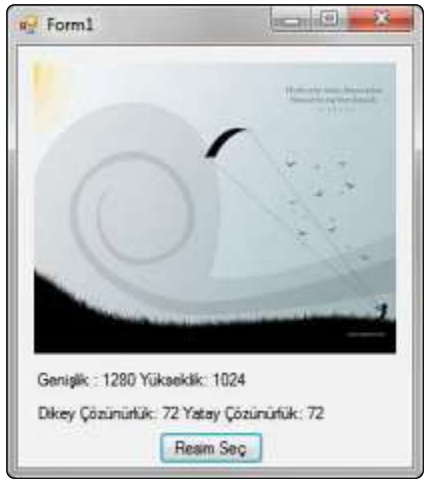


- **`width`, `height` Özellikleri:** width özelliği **`Image`** nesnesinin genişlik değerini verirken, height özelliği ise yükseklik değerini verir.
- **`HorizontalResolution`, `VerticalResolution` Özellikleri:** **`HorizontalResolution`** özelliği Image nesnesinin yatay çözünürlük değerini verirken, **`VerticalResolution`** özelliği ise Image nesnesinin dikey çözünürlük değerini verir.
- **`RawFormat` Özelliği:** **`Image`** nesnesinin dosya formatını verir.

Bir önceki örneğimizi düzenleyerek **Form** üzerine 2 adet **Label** ekleyelim. Bu **Label** kontrolleri içerisinde genişlik, yükseklik ve çözünürlük değerlerini görüntüleyeceğiz.

```
private void btnSec_Click(object sender, EventArgs e)
{
  DialogResult buton= openFileDialog1.ShowDialog();
  if (buton == DialogResult.OK)
  {
   Image image = Image.FromFile(openFileDialog1.FileName);
         pictureBox1.Image = image;
   lblGenislikYukseklik.Text = "Genişlik : " +
         image.Width + " Yükseklik: " + image.Height;
   lblCozunurluk.Text = "Dikey Çözünürlük: " +
         image.VerticalResolution + " Yatay Çözünürlük: " +
         image.HorizontalResolution;
  }
}
```

Kodumuzda **`Width`** özelliği ile genişlik değerini, **`Height`** özelliği ile yükseklik değerini, **`VerticalResolution`** özelliği ile dikey yükseklik değerini, **`HorizontalResolution`** özelliği ile de yatay çözünürlük değerini elde ederek **Label** kontrolleri içerisinde görüntüledik.



- **`PixelFormat` Özelliği:** **`Image`** nesnesinin piksel formatını verir.

Bir resmin piksel renklendirmesini değiştirerek, **`PixelFormat`** özelliği ile piksel formatını görüntüleyen örnek bir uygulama yapalım.

```
Bitmap bmp;
Image image;
private void btnSec_Click(object sender, EventArgs e)
{
  DialogResult buton = openFileDialog1.ShowDialog();
  if (buton == DialogResult.OK)
  {
   image = Image.FromFile(openFileDialog1.FileName);
   bmp = (Bitmap)image;
   pictureBox1.Image = bmp;
   label1.Text = image.PixelFormat.ToString();
  }
}
private void btnislemYap_Click(object sender, EventArgs e)
{
```

```
    int x, y;
    for (x = 0; x < bmp.Width; x++)
    {
     for (y = 0; y < bmp.Height; y++)
     {
      Color pixelColor = bmp.GetPixel(x, y);
      Color newColor = Color.FromArgb(pixelColor.R, 0, 0);
      bmp.SetPixel(x, y, newColor);
     }
    }
    pictureBox2.Image = bmp;
    label2.Text = bmp.PixelFormat.ToString();
}
```

Kodumuzda bir resmin piksellerini yeniden renklendirmek için **`SetPixel`** metodunu kullandık. **`SetPixel`** metodu Bitmap sınıfına ait bir metot olduğu için **`FromFile`** metodu ile oluşturmuş olduğumuz **`Image`** nesnesini **`Bitmap`** nesnesine dönüştürdük.

Biri resmin genişliği, diğeri de yüksekliği adedince iç içe iki döngü kurarak resmi piksel piksel okuyup, **`ColorFromArgb`** özelliği ile belirtilen renk formatına **`SetPixel`** metodu ile dönüştürdük.

Resmin piksel formatını ise **`PixelFormat`** özelliği ile elde ederek **Label** kontrolü içerisinde görüntüledik.



- **`Image.FromStream` Metodu:** **`FromFile`** metodunda olduğu gibi resim dosyalarını **`Image`** nesnesine aktarmak amacıyla kullanılır. Ancak farklı olarak

resim dosyası ile ilgili okunan verileri **MemoryStream** ya da **FileStream** nesnesi gibi bir stream içerisinde saklar.

```
private void btnSec_Click(object sender, EventArgs e)
{
 DialogResult buton = openFileDialog1.ShowDialog();
 if (buton == DialogResult.OK)
 {
  using (FileStream stream = File.Open
        (openFileDialog1.FileName, FileMode.Open))
  using (Image image = Image.FromStream(stream))
  {
   pictureBox1.Image = Bitmap.FromStream(stream);
   lblBoyut.Text = image.Size.ToString();
  }
 }
}
```

Kodumuzda **ImageFromStream** metodu ile **openFileDialog** penceresi içerisinde seçili olan resmi **FileStream** nesnesi içerisine aktardık. **FileStream** nesnesi içerisinde bulunan resim dosyasını okuyarak, **PictureBox** içerisinde görüntüleyebilmek için **Bitmap.FromStream** metodunu kullandık. Ayrıca **Size** özelliği ile resim dosyasının genişlik (**Width**) ve yükseklik (**Height**) değerlerini okuyarak **Label** içerisinde görüntüledik.



ImageFromStream metodunu kullanarak bir web adresindeki resim dosyasını okuyabiliriz.

```
private void btnResimGoruntule_Click (object sender, EventArgs e)
{
   var wr = WebRequest.Create(textBox1.Text);
   using (var response = wr.GetResponse())
   using (var stream = response.GetResponseStream())
   {
    pictureBox1.Image = Bitmap.FromStream(stream);
   }
}
```



Kodumuzda **WebRequest.Create** metodunu kullanarak **TextBox** içerisinde yazılı olan web adresi ile bir web isteği oluşturduk. **GetResponse** metodu ile web isteğinin sonucunu aldık. **GetResponseStream** metodu ile de sonucu **stream** değişkenine aktardık. **stream** değişkeni içerisinde bulunan resim dosyasını okuyarak **PictureBox** içerisinde görüntüleyebilmek için **Bitmap.FromStream** metodunu kullandık.

NOT

Aynı işlemi **PictureBox** nesnesinin **ImageLocation** özelliği ile de gerçekleştirebiliriz. Buna göre aynı işlemi yapan kodu…

```
pictureBox1.ImageLocation = textBox1.Text;
```

… şeklinde yazabiliriz.

- **Tag Özelliği:** Resim dosyasının tag bilgisini görüntülemek ve değiştirmek amacıyla kullanılır.

```
using (Image image = Image.FromFile("C:\\ornekresim.gif"))
{
 image.Tag = "Örnek Tag Metni" ;
}
```

Örneğin; yukarıdaki kod satırlarında belirtilen resim dosyasının tag bilgisi "*Örnek Tag Metni*" şeklinde değiştirilmektedir.

- **`RotateFlip` Metodu:** Resim dosyasının döndürmek ve yönünü çevirmek amacıyla kullanılır. Şu şekilde kullanılır:

```
ResimDosyası.RotateFlip(RotateFlipType);
```

- **`RotateFlipType`:** Aşağıdaki değerleri alır:
- **`RotateNoneFlipNone`:** Herhangi bir döndürme ve çevirme işleminin yapılmayacağını belirtir.
- **`Rotate90FlipNone`:** Çevirme işlemi yapılmadan, 90 derece saat yönünde döndürür.
- **`Rotate180FlipNone`:** Çevirme işlemi yapılmadan, 180 derece saat yönünde döndürür.
- **`Rotate270FlipNone`:** Çevirme işlemi yapılmadan, 270 derece saat yönünde döndürür.
- **`RotateNoneFlipX`:** Herhangi bir döndürme işlemi yapmadan yatay yönde çevirme yapar.
- **`Rotate90FlipX`:** Yatay yönde çevirme, 90 derece saat yönünde de döndürme yapar.
- **`Rotate180FlipX`:** Yatay yönde çevirme, 180 derece saat yönünde de döndürme yapar.
- **`Rotate270FlipX`:** Yatay yönde çevirme, 270 derece saat yönünde de döndürme yapar.
- **`RotateNoneFlipY`:** Herhangi bir döndürme işlemi yapmadan dikey yönde çevirme yapar.
- **`Rotate90FlipY`:** Dikey yönde çevirme, 90 derece saat yönünde de döndürme yapar.
- **`Rotate180FlipY`:** Dikey yönde çevirme, 180 derece saat yönünde de döndürme yapar.
- **`Rotate270FlipY`:** Dikey yönde çevirme, 270 derece saat yönünde de döndürme yapar.
- **`RotateNoneFlipXY`:** Herhangi bir döndürme işlemi yapmadan yatay ve dikey yönde çevirme yapar.

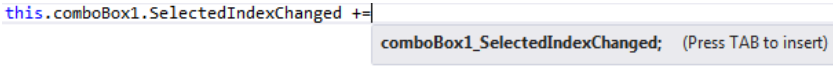
- **Rotate90FlipXY**: Yatay ve dikey yönde çevirme, 90 derece saat yönünde de döndürme yapar.
- **Rotate180FlipXY**: **Rotate90FlipXY**: Yatay ve dikey yönde çevirme, 180 derece saat yönünde de döndürme yapar.
- **Rotate270FlipXY**: **Rotate90FlipXY**: Yatay ve dikey yönde çevirme, 270 derece saat yönünde de döndürme yapar.

Örnek olarak bir **ComboBox** içerisinde listelenen **RotateFlipType** biçimlerinden biri seçildikçe, seçili döndürme ve çevirme işlemini gerçekleştiren bir uygulama yapalım.

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace Image_Rotate_And_Flip
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
   foreach (RotateFlipType degerler in
            Enum.GetValues(typeof(RotateFlipType)))
   {this.comboBox1.Items.Add(degerler);}
   this.comboBox1.SelectedIndex = 0;
   this.comboBox1.SelectedIndexChanged +=
            comboBox1_SelectedIndexChanged;
 }
 void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
 {
  if (pictureBox1.Image != null)
  {
   RotateFlipType rt = (RotateFlipType)comboBox1.SelectedItem;
   Image image = pictureBox1.Image;
   image.RotateFlip(rt);
   pictureBox1.Refresh();
  }
 }
 private void btnSec_Click(object sender, EventArgs e)
```

```
 {
   DialogResult buton = openFileDialog1.ShowDialog();
   if (buton == DialogResult.OK)
   {
    image image = Image.FromFile(openFileDialog1.FileName);
    pictureBox1.Image = image;
   }
  }
 }
}
```

Kodumuzda **comboBox1_SelectedIndexChanged** olayını elle yazmayıp, **event handler** ile yazıyoruz. Bunun için **public Form1() { }** içerisinde aşağıda görüldüğü gibi **this.comboBox1.SelectedIndexChanged** yazdıktan sonra **TAB** tuşuna basmalıyız.



SelectedIndexChanged olayı otomatik olarak **event handler** ile eklenecektir.

```
foreach (RotateFlipType degerler in
  Enum.GetValues(typeof(RotateFlipType)))
  {this.comboBox1.Items.Add(degerler);}
```

Satırlarında **RotateFlipType** içerisindeki değerleri **foreach** döngüsü ile **ComboBox** içerisine ekliyoruz.

```
RotateFlipType rt = (RotateFlipType)comboBox1.SelectedItem;
Image image = pictureBox1.Image;
image.RotateFlip(rt);
```

… satırlarında **ComboBox** içerisinde seçili olan değeri **RotateFlipType** türüne dönüştürerek, **PictureBox** içerisinde bulunan resme **RotateFlip** metodu ile seçili olan döndürme ve çevirme işlemini uyguluyoruz.



Örneğin; yukarıdaki kod satırlarında belirtilen resim dosyasının tag bilgisi "*Örnek Tag Metni*" şeklinde değiştirilmektedir.

- **`Save` Metodu:** Resim dosyasını kaydetmek amacıyla kullanılır. Aşağıdaki şekilde kullanılır:

```
Image.Save (Dosya Adı, ImageCodecInfo, EncoderParameters)
```

- **`ImageCodecInfo`:** Bilgisayarda yüklü codec bilgilerini elde etmek amacıyla kullanılan bir sınıftır.
- **`EncoderParameters`:** Kaydedilecek dosya için encode parametrelerini belirtir.

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Windows.Forms;
namespace Image_Save
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
  }
  Image image;
 private void btnSec_Click(object sender, EventArgs e)
```

```
 {
  DialogResult buton = openFileDialog1.ShowDialog();
  if (buton == DialogResult.OK)
  {
   image = Image.FromFile(openFileDialog1.FileName);
   pictureBox1.Image = image;
   FileInfo fInfo = new FileInfo(openFileDialog1.FileName);
   label1.Text = "İlk boyut: " + (fInfo.Length/(1024)).ToString() + " kb";
  }
 }
 private void btnKaydet_Click(object sender, EventArgs e)
 {
  if (saveFileDialog1.ShowDialog() == DialogResult.OK)
  {
   ImageCodecInfo ci;
   Encoder enc;
   EncoderParameter ep;
   EncoderParameters eps;

   ci = EncoderInfo("image/jpeg");
   enc = Encoder.Quality;
   eps = new EncoderParameters(1);

   ep = new EncoderParameter(enc, 25L);
   eps.Param[0] = ep;
   pictureBox1.Image = image;
   image.Save(saveFileDialog1.FileName, ci, eps);
   FileInfo fInfo = new FileInfo(saveFileDialog1.FileName);
   label1.Text = "Yeni boyut: " + (fInfo.Length / (1024)).ToString() + " kb";
   }
 }
 private static ImageCodecInfo GetEncoderInfo(String mimeType)
 {
   int j;
   ImageCodecInfo[] encoders;
   encoders = ImageCodecInfo.GetImageEncoders();
   for (j = 0; j < encoders.Length; ++j)
   {
    if (encoders[j].MimeType == mimeType)
     return encoders[j];
   }
   return null;
  }
 }
}
```

Kodumuzda **`ImageCodecInfo`** türünden tanımladığımız **`GetEncoderInfo`** metodu ile sistemde yüklü olan codec bilgilerini elde ediyoruz. Geriye dönene encoder parametreleri dizisi içerisindeki **`EncoderParameters(1)`** değerini okuyarak **`Save`** metodu içerisinde eps parametresi içerisinde kullanıyoruz. Encode türünü ise **`EncoderInfo("image/jpeg")`** şeklinde belirterek resim dosyasını **jpeg** dosyası türüne çevirerek, üzerinde işlem yapacağımızı belirtiyoruz. **`EncoderInfo`** bilgisini **`ci`** parametresi ile **`Save`** metodu içerisinde kullanıyoruz.

Jpeg dosyasının kayıt kalitesini ise…

```
ep = new EncoderParameter(enc, 25L);
```

… satırında belirtiyoruz. Bu değer düştükçe resmin çözünürlüğü düşer, arttıkça çözünürlük değeri de artar. Resim boyutu ise çözünürlükle ters orantılıdır.



Kodumuzda resim ilk **`openFileDialog`** nesnesi içerisinde seçilip yüklendi **`FileInfo`** sınıfının **`Length`** özelliği ile resmin boyutu elde edilerek görüntülenmektedir. **Kaydet** butonuna tıklanıp resmimiz farklı biçimde kaydedildiğinde, yeni boyut görüntülenmektedir.



YAZDIRMA İŞLEMLERİ

18

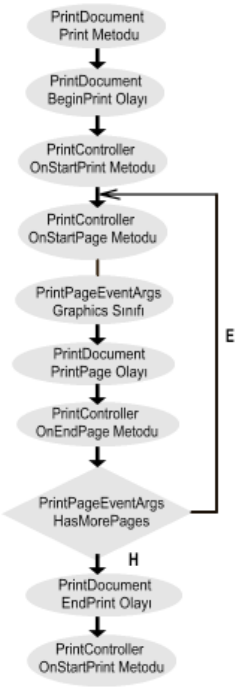
C# içerisinde bir belgeyi direkt olarak yazdırabileceğimiz gibi yazdırmadan önce sayfa ayarı, baskı önizleme gibi işlemler de gerçekleştirebiliriz. Yazdırma işlemi için **`PrintDocument`** sınıfını, baskı önizleme, yazıcı ayarları ve sayfa ayarları işlemleri için de **`PrintDialog`** sınıfını kullanabiliriz.

PrintDocument Sınıfı

`PrintDocument` sınıfı **`System.Drawing.Printing`** sınıfı içerisinde yer almaktadır.

PrintDocument Sınıfının Özellik ve Metotları

- **`Print`**: Yazdırma işlemini başlatmak amacı ile kullanılan metottur. **`Print`** olayı gerçekleşirken, **`PrintController`** sınıfının **`OnStartPrint`**, **`OnEndPrint`**, **`OnStartPage`** ve **`OnEndPage`** olayları gerçekleşir. **`PrintController`** sınıfı bir dokümanın nasıl yazdırılacağını belirler. **`PrintController`** sınıfı **`PrintDocument`** sınıfı tarafından kullanılıp, tek başına kullanılamaz. **`PrintController`** ve **`PrintDocument`** sınıfları arasındaki olaylar yandaki şekilde görüldüğü sıralamada gerçekleştirilir.
- **`PrintPage`**: **`Print`** metodu başlatıldığı sırada **`PrintPage`** olayı içerisinde tanımlanan işlemler gerçekleştirilir. Bu



olaya dair argümanlar **PrintPageEventArgs** sınıfı içerisinde tanımlanır. Bu sınıfın elemanları aşağıdaki gibidir:

- **Cancel: Boolean** değer alır. **true** değerini aldığında yazma işlemi iptal edilir.
- **Graphics:** Yazdırılacak içeriklerin oluşturulmasında kullanılır.
- **HasMorePages: Boolean** değer alır. Eğer yazdırılacak birden fazla sayfa varsa, **true** değerini alır. Varsayılan olarak **false** değerini alır.
- **MarginBounds:** Sayfanın kenarlarının içerisinde kalan kısmını temsil eder. Her bir sayfanın yazdırılmasının nerede başlayacağı ve nerede bitirileceğinin belirlenmesi için **Rectangle** nesnesinden faydalanılır.
- **PageBounds:** Kenarlar dahil olmak üzere sayfanın toplam alanını temsil eder.
- **PageSettings:** Yazdırılacak olan sayfa ile ilgili ayarları içerir.

Bu bilgilere göre yukarıdaki şekilde belirtilen yazdırma döngüsünü incelelim…

Öncelikle yazdırılacak olan belgenin seçimi yapılır. **PrintPageEventArgs** sınıfının **Graphics** özelliği kullanılarak, yazdırılacak belge içerisindeki sayfa başına satır sayısı hesaplanır, belge içerisindeki yazı fontu, rengi gibi özellikler okunur. Her bir sayfa **Graphics** nesnesinin **DrawString** metodu çizdirilip, yazdırılmaya hazırlanır ve eğer son sayfaya gelinmişse, **PrintPageEventArgs** sınıfının **HasMorePages** özelliği kontrol edilir. Eğer true değerini içeriyorsa **PrintPage** olayı yeniden gerçekleştirilir. **PrintPage** olayı **HasMorePages** özelliği false değerini alıncaya kadar devam eder. **HasMorePages** özelliğinin **false** olması yazdırılacak olan belgenin başka sayfası olmadığını belirtir. Dolayısı ile eğer yazdırılacak olan belgenin birden fazla sayfası mevcutsa, **HasMorePages** özelliğine **true** değeri aktarılmalıdır.

Örnek olarak **Form** üzerindeki bir **TextBox** içerisindeki yazıları yazıcıdan çıkartan bir uygulama yapalım.

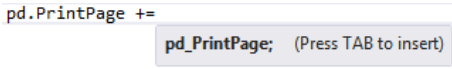
```
using System;
using System.Windows.Forms;
using System.Drawing.Printing;
using System.Drawing;
namespace print_textbox
{
 public partial class Form1 : Form
 {
```

```
  public Form1()
  {
    InitializeComponent();
  }
  private void btnYazdir_Click(object sender, EventArgs e)
  {
   PrintDocument pd = new PrintDocument();
   pd.PrintPage += new PrintPageEventHandler(pd_PrintPage);
   pd.Print();
  }
  private void pd_PrintPage(Object sender, PrintPageEventArgs e)
  {
   Font printFont = new Font("Times New Roman", 12);
   e.Graphics.DrawString(textBox1.Text, printFont, Brushes.Black, 0, 0);
  }
 }
}
```

Yazdırma işlemi için **`System.Drawing.Printing`**, yazdırılacak olan belgenin içeriğinin hazırlanması için ise **`System.Drawing`** sınıfını projemize dahil ettik. Örneğimizde herhangi bir yazıcı ayarı yapmadığımız için varsayılan yazıcı üzerinden yazdırma işlemi gerçekleştirilecektir.

```
PrintDocument pd = new PrintDocument( )
```

Satırı ile yeni bir **`PrintDocument`** nesnesi tanımladık.



`PrintPage` olayı içerine **event handler** ile **`pd_PrintPage`** adındaki metodumuzu bağladık.

```
pd.PrintPage += new PrintPageEventHandler(pd_PrintPage);
```

`pd_PrintPage` metodu içerisinde yazdıracağımız belgenin içeriği ile ilgili düzenlemeleri gerçekleştiriyoruz.

```
Font printFont = new Font("Times New Roman", 12);
```

printFont adında bir değişken tanımlayarak bu değişken içerisinde, yazdıracağımız yazının font ayarını saklarız.

```
e.Graphics.DrawString(textBox1.Text, printFont, Brushes.Black, 0, 0);
```

... satırında **Graphics** sınıfının **DrawString** metodu ile, yazdıracağımız yazıyı çizdirip yazıma hazırlarız.

DrawString metodunun kullanımı aşağıdaki gibidir:

```
DrawString ( s As String, font As Font, brush As Brush, x As Single,
y As Single, format As StringFormat )
```

- **s:** Çizdirilecek olan string'in ismi.
- **font:** Çizdirilecek olan metnin yazı tipi.
- **brush:** Çizdirilecek olan metnin renk ve görünüm ayarlarını içerir.
- **x:** Çizdirilecek olan metnin yatay koordinatını belirtir.
- **y:** Çizdirilecek olan metnin dikey koordinatını belirtir.
- **format:** Çizdirilecek olan metnin görünüm ve yerleşim ayarlarını belirtir. Bu ayarlar **StringFormatFlags** altında belirlenir. Yapılabilecek bazı ayarlar aşağıdaki gibidir:

  DirectionRightToLeft: Yazıyı sağdan sola doğru olacak biçimde görüntüler.

  DirectionVertical: Yazıyı dikey biçimde görüntüler.

  NoWrap: Bu seçenek kullanıldığında çizdirilen yazının, çizdirilen şekil tarafından otomatik olarak çevrelenmesi iptal edilir.

  LineLimit: Çizdirilen bir şekil içerisine yazdırılan satırlar, şeklin çizgileri üzerine denk gelmeyecek biçimde kaydırılır.

  NoClip: Çizdirilen bir şekil içerisine yazdırılan satırlar, şeklin içerisine sığmadığı takdirde şeklin dışında devam eder.

Örneğin; aşağıdaki kodumuz yazdırılacak olan metni, dikey biçimde yazıcıya gönderir.

```
private void btnYazdir_Click(object sender, EventArgs e)
{
  PrintDocument pd = new PrintDocument();
  pd.PrintPage += new PrintPageEventHandler(pd_PrintPage);
  pd.Print();
}
private void pd_PrintPage(Object sender, PrintPageEventArgs e)
{
  Font printFont = new Font("Times New Roman", 12);
  StringFormat f = new StringFormat(StringFormatFlags.DirectionVertical);
  e.Graphics.DrawString(textBox1.Text, printFont, Brushes.Black, 0, 0,f);
}
```

`Print` olayı gerçekleşirken **`PrintController`** sınıfının **`OnStartPrint`**, **`OnEndPrint`**, **`OnStartPage`** ve **`OnEndPage`** olaylarının gerçekleştiğini söylemiştik. Buna dair bir örnek yapalım:

```
Font printFont;
private void btnYazdir_Click(object sender, EventArgs e)
{
  PrintDocument pd = new PrintDocument();
  pd.PrintPage += new PrintPageEventHandler(pd_PrintPage);
  pd.BeginPrint += new PrintEventHandler(pd_BeginPrint);
  pd.EndPrint += new PrintEventHandler(pd_EndPrint);
  pd.Print();
}
private void pd_PrintPage(Object sender, PrintPageEventArgs e)
{
  e.Graphics.DrawString(textBox1.Text,
    printFont, Brushes.Black, 0, 0);
}
private void pd_BeginPrint(object sender, PrintEventArgs e)
{
  MessageBox.Show("Yazdırma işlemi başladı");
    printFont = new Font("Times New Roman", 12);
}
private void pd_EndPrint(object sender, PrintEventArgs e)
```

```
{
  MessageBox.Show("Yazdırma işlemi bitti");
  printFont.Dispose();
  printFont = null;
}
```

Event handler ile **BeginPrint** ve **EndPrint** olaylarını yakalayarak, yazdırma işlemi başladığında ve bittiğinde belirttiğimiz işlemlerin gerçekleştirilmesini sağlıyoruz. **pd_BeginPrint** metodunda yazdırma işlemi başlangıcında, yazdırılacak olan metnin font özelliğini ayarlıyoruz. **Pd_EndPrint** metodunda ise yazdırma işleminin bitmesinin ardından sistem kaynaklarını temizliyoruz.

- **PrintDocument** sınıfının **PrinterSettings** özelliğini kullanarak yazıcı ile ilgili ayarlar yapmamız mümkündür. Yapabileceğimiz ayarların önemli bir kısmı aşağıdaki gibidir:

  CanDuplex: Yalnız okunabilir bir özelliktir. Yazıcının çift taraflı yazdırmayı destekleyip desteklemediğini belirtir. **Boolean** değer alır. Eğer **true** değerini alırsa yazıcı çift taraflı yazdırmayı destekliyor, **false** değerini alıyorsa desteklemiyordur.

  Collate: Yazıcının harmanlama işlemi yapıp yapmadığını belirlemek ya da harmanlama işlemi yapmasını sağlamak amacıyla kullanılır. Boolean değer alır. Eğer **true** değerini alırsa harmanlama işlemi yapılır, false değerini alırsa yapılmaz. Harmanlama, **Copies** değeri 1 rakamından fazla bir değer alırsa kullanılır.

  Copies: Yazdırılacak olan dokümanın kopya sayısını bulmak ya da belirlemek amacıyla kullanılır.

  DefaultPageSettings: Yazıcının varsayılan sayfa ayarlarını bulmak amacıyla kullanılır.

  Duplex: Yazıcının çift taraflı yazdırma seçeneğinin aktif olup olmadığını bulmak ya da bu özelliği aktifleştirmek amacıyla kullanılır.

  FromPage: Yazdırılan belgenin ilk sayfasının numarasını bulmak ya da belirlemek amacıyla kullanılır.

  InstalledPrinters: Bilgisayarda yüklü olan yazıcıların isimlerini bulmak amacıyla kullanılır.

  IsDefaultPrinter: PrinterName özelliğinin varsayılan yazıcıyı gösterip göstermediğini belirtir.

IsPlotter: Yazıcının bir **plotter** (*çizici*) olup olmadığını belirtir.

IsValid: **PrinterName** özelliği ile belirtilen yazıcı isminin geçerli bir yazıcı olup olmadığını belirtir.

MaximumCopies: Yazdırma anında yazıcının erişilen maksimum kopya sayısını belirtir.

MaximumPage: **FromPage** veya **ToPage** özelliğinde belirtilen en büyük sayfa değerini bulmak ya da belirlemek amacıyla kullanılır.

MinimumPage: **FromPage** veya **ToPage** özelliğinde belirtilen en küçük sayfa değerini bulmak ya da belirlemek amacıyla kullanılır.

PaperSizes: Yazıcının desteklediği kağıt ölçüsü değerlerini döndürür.

PaperSources: Yazıcının desteklediği kağıt besleme seçeneklerini döndürür.

PrinterName: Kullanımda olan yazıcı ismini döndürür ya da belirler.

PrinterResolutions: Yazıcının desteklediği çözünürlük değerlerini döndürür.

PrintRange: Kullanıcı tarafından özel olarak belirtilen yazdırılacak sayfa aralığını bulmak ya da belirleme amacıyla kullanılır.

PrintToFile: Yazdırma işleminin yazıcıya değil de disk üzerinde bir dosya biçiminde yapılıp yapılmadığını bulmak ya da yazdırma işleminin bu şekilde yapılmasını belirlemek amacıyla kullanılır.

SupportsColor: Yazıcının desteklediği renk değerlerini döndürür.

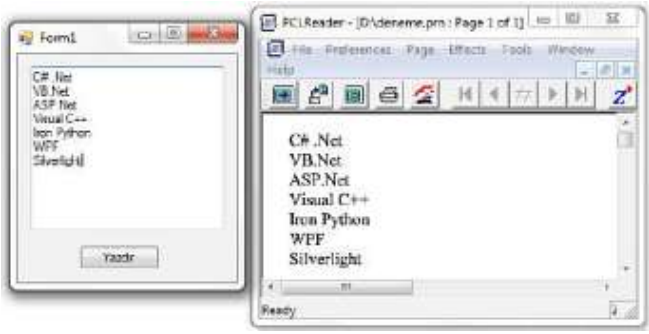
ToPage: Yazdırılan belgenin son sayfasının numarasını bulmak yada belirlemek amacıyla kullanılır.

Örnek olarak **TextBox** içerisindeki yazıyı yazıcıdan çıkarmak yerine, disk üzerinde dosyaya yazdıran bir uygulama yapalım.

```
private void btnYazdir_Click(object sender, EventArgs e)
{
  PrintDocument pd = new PrintDocument();
  pd.PrintPage += new PrintPageEventHandler(pd_PrintPage);
  pd.PrinterSettings.PrintToFile = true;
  pd.PrinterSettings.PrintFileName = "d:\\deneme.prn";
  pd.Print();
}
private void pd_PrintPage(Object sender, PrintPageEventArgs e)
```

```
{
  Font printFont = new Font("Times New Roman", 12);
    e.Graphics.DrawString(textBox1.Text, printFont,
  Brushes.Black, 0, 0);
}
```

PrinterSettings.PrintToFile özelliğine **true** değerini atayarak dosyaya yazdırma işleminin aktif olmasını sağlıyoruz. **PrinterSettings.PrintFileName** özelliğine ise; disk üzerinde oluşturulacak olan dosyanın yolunu ve ismini yazıyoruz.



Yukarıdaki resimde görüldüğü gibi **TextBox** içerisine girdiğimiz metin, **deneme.prn** dosyası içerisine yazdırılmıştır.

Şu ana kadar yaptığımız yazdırma işlemlerinde bir **TextBox** içerisindeki içeriği yazdırdık. Disk üzerinde mevcut olan bir Word belgesini yazdıran örnek bir uygulama yapalım.

```
using System;
using System.Drawing;
using System.Drawing.Printing;
using System.Windows.Forms;
using System.IO;
namespace Print_Files
{
 public partial class Form1 : Form
 {
  public Form1()
  {
    InitializeComponent();
  }
```

```
  Font printFont;
  StreamReader sr;
  byte sayfa_no;
  private void btnDosyaSec_Click(object sender, EventArgs e)
  {
    openFileDialog1.FileName = "dosya seç";
    openFileDialog1.Filter = "Word belgeleri (*.doc) |*.doc";
    openFileDialog1.Title = "Lütfen Bir Word Belgesi Seçin";
    DialogResult basilan_tus;
    basilan_tus = openFileDialog1.ShowDialog();
    if (basilan_tus == DialogResult.OK)
      txtDosyaAdı.Text = openFileDialog1.FileName;
  }
  private void btnYazdir_Click(object sender, EventArgs e)
  {
    PrintDocument pd = new PrintDocument();
    printFont = new Font("Times New Roman", 12);
    pd.PrintPage += new PrintPageEventHandler(pd_PrintPage);
    pd.PrintPage += new PrintPageEventHandler(pd_OnPrintPage);
    pd.EndPrint += pd_EndPrint;

    sr = new StreamReader(txtDosyaAdı.Text);

    pd.PrinterSettings.PrintToFile = true;
    pd.PrinterSettings.PrintFileName = "d:\\deneme.prn";
    pd.Print();
  }
  private void pd_PrintPage(Object sender, PrintPageEventArgs e)
  {
    Single SolMargin = e.MarginBounds.Left;
    Single UstMargin = e.MarginBounds.Top;
    int sayi = 0;
    string satir = null;
    Single yPozisyonu = 0;
    Single SayfaBasiSatirAdedi = e.MarginBounds.Height / printFont.
GetHeight(e.Graphics);

    while (sayi < SayfaBasiSatirAdedi && ((satir = sr.ReadLine()) != null))
    {
      yPozisyonu = UstMargin + (sayi * printFont.GetHeight(e.Graphics));
      e.Graphics.DrawString(satir, printFont, Brushes.Black,
SolMargin, yPozisyonu, new StringFormat());
```

```
        sayi++;
      }

      if (satir != null)
        e.HasMorePages = true;
      else
        e.HasMorePages = false;
    }
    private void pd_OnPrintPage(Object sender, PrintPageEventArgs e)
    {
      sayfa_no++;
      if (sayfa_no == 3)
        MessageBox.Show("Üçüncü sayfa yazdırılıyor");
    }
    private void pd_EndPrint(object sender, PrintEventArgs e)
    {
      printFont.Dispose();
      printFont = null;
      sr.Close();
    }
  }
}
```

Örneğimizde…

```
sr = new StreamReader(txtDosyaAdı.Text);
```

… satırı ile **`OpenFileDialog`** nesnesi yardımıyla seçilip ismi **TextBox** içerisine atılan dosyayı **`StreamReader`** nesnesi ile okuyup **`sr`** isimli string değişkene aktarıyoruz.

`PrintPage` olayı içerisinde…

```
satir = sr.ReadLine()
```

… **`ReadLine`** metodunu kullanarak **`StreamReader`** nesnesi ile okuyup **`sr`** değişkeni içerisine aktardığımız dosyayı satır satır okuyarak **`satir`** değişkenine aktarıyoruz.

```
e.Graphics.DrawString(satir, printFont, Brushes.Black,
     SolMargin, yPozisyonu, new StringFormat());
```

Okuduğumuz satırı yukarıdaki kod satırında görüldüğü gibi yazdırıyoruz. Yazdıracağımız metin, **satir** değişkeni içerisinde saklanmakta. Yazı fontu **printFont** değişkeni içerisinde saklanmakta ve **btnYazdir** olayı içerisinde…

```
printFont = new Font("Times New Roman", 12);
```

… biçiminde tanımlanmaktadır.

Yazı rengi **Brushes.Black** özelliği ile siyah renk olarak belirlenmektedir. Yazılan yazının **x** pozisyonu **SolMargin** değişkeni ile y pozisyonu ise **yPozisyonu** değişkeni ile belirlenmektedir.

SolMargin değişkenine x pozisyonu **e.MarginBounds.Left** ifadesi ile aktarılmakta ve sol kenar boşluğu değeri ne kadar olarak belirlenmişse, yazdırma işleminin belirtilen sol kenar boşluğundan itibaren yapılması sağlanmaktadır.

```
yPozisyonu = UstMargin + (sayi * printFont.GetHeight(e.Graphics));
```

y pozisyonu ise yukarıdaki kod satırında görüldüğü gibi **UstMargin** (sol üst kenar boşluğu değeri) değişkeninin değerinin, **sayi** değişkeni değerinin aktif font boyutu ile çarpımı ile toplanmasından elde edilmektedir.

Örneğin; üst kenar boşluğu değerinin **100** olduğunu varsayalım. **sayi** değişkeni ilk satırın yazdırılması sırasında **1** değerini içeriyordur. Yazdıracağım belgenin font yüksekliği ise; **19** olarak görünmekte olduğundan…

```
yPozisyonu=100+(1*19) = 119
```

… değerini alacaktır.

sayi değişkeni **sayi++** ifadesi ile dosya içerisinden her satır okumasında bir arttırılmaktadır. Ancak…

```
while (sayi < SayfaBasiSatirAdedi && ((satir = sr.ReadLine()) != null))
```

… döngüsünde görüldüğü gibi **sayi** değişkeninin değeri **SayfaBasiSatirAdedi** değişkeninin değerinden küçükse ve okunan satır değeri boş değilse işlem yapılmaktadır.

SayfaBasiSatirAdedi değişkeninin değeri…

```
Single SayfaBasiSatirAdedi = e.MarginBounds.Height /
printFont.GetHeight(e.Graphics);
```

… satırı ile tanımlanıp elde edilmektedir. Örneğin; yazdıracağım bir belgenin **e.MarginBounds.Height** değerini kontrol ettiğimde 969 değerine sahip olduğunu gördüm.

NOT A4 boyutundaki kağıdın genişlik ve yükseklik değerleri, **Width = 826**, **Heigth = 1169** biçimindedir.

Yazdıracağım belgenin font yüksekliği ise; 19 olarak görünmekte olduğundan sayfa başı satır adedi yaklaşık olarak 51 satıra denk gelmektedir.

Satır adedi aşıldığında…

```
if (satir != null)
      e.HasMorePages = true;
    else
      e.HasMorePages = false;
```

…. satırlarında görüldüğü gibi **e.HasMorePages** özelliğini kullanarak, eğer **true** değerini almışsa, yazdırma işlemlerini tekrarlıyoruz.

Yazdırma işlemi esnasında gerçekleşen olayların kontrolünü, **OnPrintPage** olayı içerisinde kontrol ederek…

```
sayfa_no++;
if (sayfa_no == 3)
      MessageBox.Show("Üçüncü sayfa yazdırılıyor");
```

… satırı ile eğer 3 nolu sayfa yazdırılmışsa, ekrana bir mesaj penceresi çıkmasını sağlıyoruz. Burada istenirse yazı biçimi, yönü gibi değişik işlemler de gerçekleştirilebilir.



PrintDialog Sınıfı

PrintDialog sınıfı; **System.Windows.Forms** sınıfı içerisinde yer almaktadır. **System.Windows.Forms** sınıfı içerisinde 3 farklı **PrintDialog** nesnesi bulunmaktadır. Bunlar; **PrintDialog**, **PrintPreviewDialog** ve **PageSetupDialog** nesneleridir.



Yazdırma işlemleri için **PrintDialog** sınıfını kullanabileceğiniz gibi, bu işlemler için oluşturulmuş olan .NET kontrollerini de kullanabilirsiniz. Bu kontroller toolBox içerisinde **Printing** başlığı altında yer almaktadır.



PrintDialog Nesnesi

Dosya yazdırma diyalog penceresini görüntülemek amacıyla kullanılır.

Örnek olarak dosya yazdırma örneğimizi, yazdırma işleminden önce **PrintDialog** nesnesi görüntülenecek şekilde yeniden yazalım. Aşağıya yalnızca yazdırma bölümünün kodunu yazıyorum:

```
private void btnYazdir_Click(object sender, EventArgs e)
{
  PrintDocument pd = new PrintDocument();
  printFont = new Font("Times New Roman", 12);
  pd.PrintPage += new PrintPageEventHandler(pd_PrintPage);

  sr = new StreamReader(txtDosyaAdı.Text);

  PrintDialog dlgPrint = new PrintDialog();
  dlgPrint.Document = pd;
```

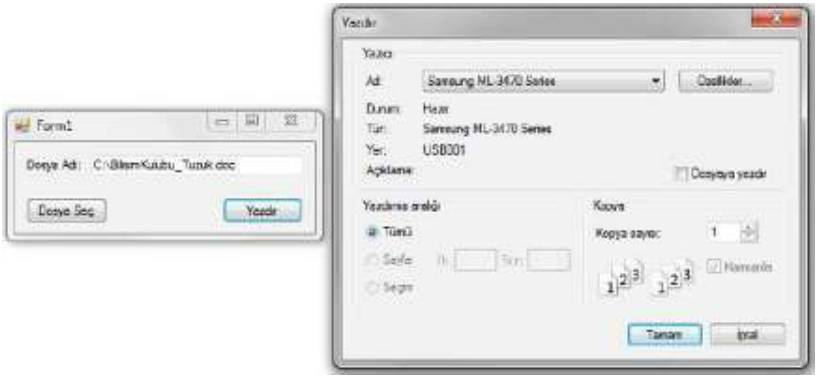
```
  if (dlgPrint.ShowDialog() == DialogResult.OK)
    pd.Print();
}
```

`PrintDialog` nesnesini…

```
PrintDialog dlgPrint = new PrintDialog();
```

… biçiminde **`PrintDialog`** penceresini görüntüleyecek biçimde tanımladık.

Diyalog penceresinin **`Document`** özelliğine **`pd`** isimli **`PrintDocument`** nesnesini bağladık.



NOT Aynı işlemi yapan kodu **ToolBox** üzerinde yer alan **`PrintDialog`** kontrolü ile yapmak için **Form** üzerine 1 adet **`PrintDialog`** kontrolü, 1 adet **`PrintDocument`** kontrolü ekliyoruz. Diğer kontroller örneğimizdeki ile aynı. **`PrintDialog`** kontrolü yazdırma penceresini görüntülemek için kullanılırken, **`PrintDocument`** kontrolü ise yazdırma olaylarını içerisinde barındırır.

Yazdırma diyalog penceresini görüntüleyen kodumuz aşağıdaki gibi olacak:

```
private void btnYazdir_Click(object sender, EventArgs e)
{
  sr = new StreamReader(txtDosyaAdı.Text);
  DialogResult basilan_tus;
  basilan_tus = printDialog1.ShowDialog();
  if (basilan_tus == DialogResult.OK)
      printDocument1.Print();
}
```

printPage, **beginPrint**, **endPrint** olayları için **PrintDocument** nesnesi seçili iken, **Properties Window** içerisinde ilgili olayları seçip çift tıklayıp kodumuzu yazabiliriz.

PrintPreviewDialog Nesnesi

Baskı önizleme diyalog penceresini görüntülemek amacıyla kullanılır.

PrintPreviewDialog penceresini görüntülemek için **PrintDialog** örneğindeki kod satırlarımızı aşağıdaki şekilde değiştirelim.

```
private void btnBaskiOnizleme_Click(object sender, EventArgs e)
{
  PrintDocument pd = new PrintDocument();
  printFont = new Font("Times New Roman", 12);
  pd.PrintPage += new PrintPageEventHandler(pd_PrintPage);
  pd.EndPrint += pd_EndPrint;
  sr = new StreamReader(txtDosyaAdı.Text);

  PrintPreviewDialog dlgPrintPreview = new PrintPreviewDialog();
  dlgPrintPreview.Document = pd;

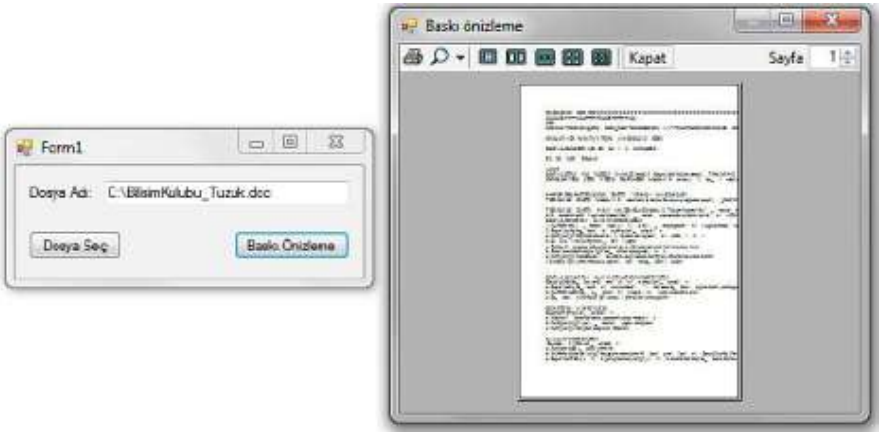
  if (dlgPrintPreview.ShowDialog() == DialogResult.OK)
     pd.Print();
}
```

Görüldüğü gibi **PrintDialog** nesnesi yerine, **PrintPreview** nesnesini oluşturmak için…

```
PrintDialog dlgPrint = new PrintDialog();
```

… tanımlamasını aşağıdaki gibi tanımlamak yeterlidir.

```
PrintPreviewDialog dlgPrintPreview = new PrintPreviewDialog();
```



NOT Aynı işlemi yapan kodu **ToolBox** üzerinde yer alan **`PrintPreview`** kontrolü ile yapmak için **Form** üzerine bir adet **`printPreview`** kontrolü, bir adet **`PrintDocument`** kontrolü ekliyoruz. Diğer kontroller örneğimizdeki ile aynı. **`PrintPreview`** kontrolü diğer yazıcı kontrollerinin aksine, **Form** üzerine eklenerek içerisinde eklenmiş olan belgeyi önizleme biçiminde görüntülemek için kullanılırken, **`PrintDocument`** kontrolü ise yazdırma olaylarını içerisinde barındırır.

Kodumuz aşağıdaki gibi olacak:

```
using System;
using System.Drawing;
using System.Drawing.Printing;
using System.IO;
using System.Windows.Forms;
namespace printpreviewdialog_kontrol_form_uzerinde
{
 public partial class Form1 : Form
 {
   public Form1()
   {
     InitializeComponent();
   }
   string icerik;
   PrintDocument printDocument1 = new PrintDocument();
   string stringToPrint;
   private void btnDosyaSec_Click(object sender, EventArgs e)
   {
```

```
    DialogResult basilan_tus;
    basilan_tus = openFileDialog1.ShowDialog();
    if (basilan_tus == DialogResult.OK)
      txtDosyaAdı.Text = openFileDialog1.FileName;
   }
   private void printDocument1_PrintPage(object sender,
   System.Drawing.Printing.PrintPageEventArgs e)
   {
    int karaktersayısı = 0;
    int satırsayısı = 0;

    e.Graphics.MeasureString(stringToPrint, this.Font,
      e.MarginBounds.Size, StringFormat.GenericTypographic,
      out karaktersayısı, out satırsayısı);

    e.Graphics.DrawString(stringToPrint, this.Font, Brushes.Black,
            e.MarginBounds, StringFormat.GenericTypographic);

    stringToPrint = stringToPrint.Substring(karaktersayısı);

    e.HasMorePages = (stringToPrint.Length > 0);
    if (!e.HasMorePages)
      stringToPrint = icerik;
   }
   private void btnBaskiOnizleme_Click(object sender, EventArgs e)
   {
    FileInfo fileInfo = new FileInfo(txtDosyaAdı.Text);
    string dosya_adı = fileInfo.Name;
    string dosya_yolu = fileInfo.DirectoryName;
    printDocument1.DocumentName = dosya_adı;
    using (FileStream stream = new FileStream(dosya_yolu +
        dosya_adı, FileMode.Open))
    using (StreamReader reader = new StreamReader(stream))
    {
      icerik = reader.ReadToEnd();
    }
    stringToPrint = icerik;
    printPreviewControl1.Document = printDocument1;
   }
  }
}
```

printPreview kontrolü içerisinde yazıları görüntülemek için **SystemDrawing.Graphics** sınıfı altında bulunan **DrawString** metodunu kullanarak **printPreview** kontrolü içerisine karakterleri çiziyoruz. Bu nedenle Word ve Excel gibi dosya içeriklerini **printPreview** kontrolü içerisinde görüntüleyemiyoruz. Yalnızca **txt**, **rtf** gibi uzantılara sahip **text** dosyalarını görüntüleyebiliyoruz. Dolayısı ile Word ve Excel gibi belgeleri görüntülemek için **PrintDialog** sınıfına ait olan **PrintPreviewDialog** nesnesini kullanmalısınız.

```
using (FileStream stream = new FileStream(dosya_yolu +
dosya_adı, FileMode.Open))
using (StreamReader reader = new StreamReader(stream))
{
   icerik = reader.ReadToEnd();
}
```

Satırlarında belirtilen **text** dosyayı okuyarak **StreamReader** içerisine aktarıyoruz. **StreamReader** içerisine aktarılan metni de, **icerik** adlı string değişkene aktarıyoruz.

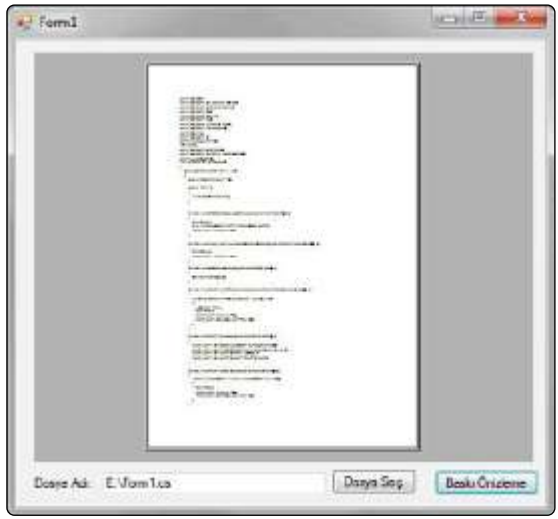
Okunan string değişkeni **PrintPreviewControl** nesnesinin **Document** olayına aktararak okunan string dosyası içeriğini **PrintPreviewControl** nesnesine aktarmış oluyoruz.

```
e.HasMorePages = (stringToPrint.Length > 0);
```

… satırında yazdırılacak olan içeriğin uzunluğu 0'dan büyük ise; yani yazdırılacak bir metin içeriği mevcutsa, birden fazla sayfa yazdırma özelliğini aktif ediyoruz.

```
if (!e.HasMorePages)
   stringToPrint = icerik;
```

… satırında ise, eğer yazdırılacak sayfa kalmamışsa yazdırma işlemini sıfırlıyoruz.



Yukarıdaki örneğimizi **PrintPreview** kontrolünün **Form** üzerine eklenmesi ile yapmıştık. Aynı kodu **PrintPreviewDialog** kontrolü için yazalım. Uygulamamızda baskı önizleme olayı form üzerinde değil, **PrintPreviewDialog** penceresi üzerinde gerçekleştirilmektedir.

```
using System;
using System.Drawing;
using System.Drawing.Printing;
using System.IO;
using System.Windows.Forms;
namespace printpreviewdialog_kontrol_form_uzerinde
{
 public partial class Form1 : Form
 {
   public Form1()
   {
     InitializeComponent();
     printDocument1.PrintPage += new
       PrintPageEventHandler(printDocument1_PrintPage);
   }
   string icerik;
   PrintDocument printDocument1 = new PrintDocument();
   string stringToPrint;
   private void btnDosyaSec_Click(object sender, EventArgs e)
```

```
    {
     DialogResult basilan_tus;
     basilan_tus = openFileDialog1.ShowDialog();
     if (basilan_tus == DialogResult.OK)
       txtDosyaAdı.Text = openFileDialog1.FileName;
    }
    private void printDocument1_PrintPage(object sender,
    System.Drawing.Printing.PrintPageEventArgs e)
    {
     int karaktersayısı = 0;
     int satırsayısı = 0;

     e.Graphics.MeasureString(stringToPrint, this.Font,
       e.MarginBounds.Size, StringFormat.GenericTypographic,
       out karaktersayısı, out satırsayısı);

     e.Graphics.DrawString(stringToPrint, this.Font, Brushes.Black,
             e.MarginBounds, StringFormat.GenericTypographic);

     stringToPrint = stringToPrint.Substring(karaktersayısı);

     e.HasMorePages = (stringToPrint.Length > 0);
     if (!e.HasMorePages)
       stringToPrint = icerik;
    }
    private void btnBaskiOnizleme_Click(object sender, EventArgs e)
    {
     FileInfo fileInfo = new FileInfo(txtDosyaAdı.Text);
     string dosya_adı = fileInfo.Name;
     string dosya_yolu = fileInfo.DirectoryName;
     printDocument1.DocumentName = dosya_adı;
     using (FileStream stream = new FileStream(dosya_yolu +
         dosya_adı, FileMode.Open))
     using (StreamReader reader = new StreamReader(stream))
     {
       icerik = reader.ReadToEnd();
     }
     stringToPrint = icerik;
     printPreviewDialog1.Document = printDocument1;
     printPreviewDialog1.ShowDialog();
    }
   }
}
```

Kodumuzda **event handler** ile **PrintPage** olayına **PrintDocument1_PrintPage** metodunu bağladık. **PrintPreview** kontrolünün **Document** özelliğine **PrintDocument** nesnesini bağladık. **ShowDialog** metodu ile **PrintPreviewDialog** penceresini görüntüledik.



Kodumuzda bir önceki **PrintPreview** kontrolü örneğimizle kıyaslama yapabilmek için farklı olan kısımları koyu renkli olarak yazdık.

PageSetupDialog Nesnesi

Sayfa ayarları diyalog penceresini görüntülemek amacıyla kullanılır.

PageSetupDialog penceresini görüntülemek için **PrintPreviewDialog** nesnesi örneği kodumuzda gerekli değişiklikleri yapalım.

```
private void btnSayfaAyari_Click(object sender, EventArgs e)
{
  PrintDocument pd = new PrintDocument();
  printFont = new Font("Times New Roman", 12);
  pd.PrintPage += new PrintPageEventHandler(pd_PrintPage);
  pd.EndPrint += pd_EndPrint;

  sr = new StreamReader(txtDosyaAdı.Text);

  PageSetupDialog dlgPageSetup = new PageSetupDialog();
  dlgPageSetup.Document = pd;
  dlgPageSetup.PageSettings.Landscape = true;
  dlgPageSetup.ShowDialog();
}
```

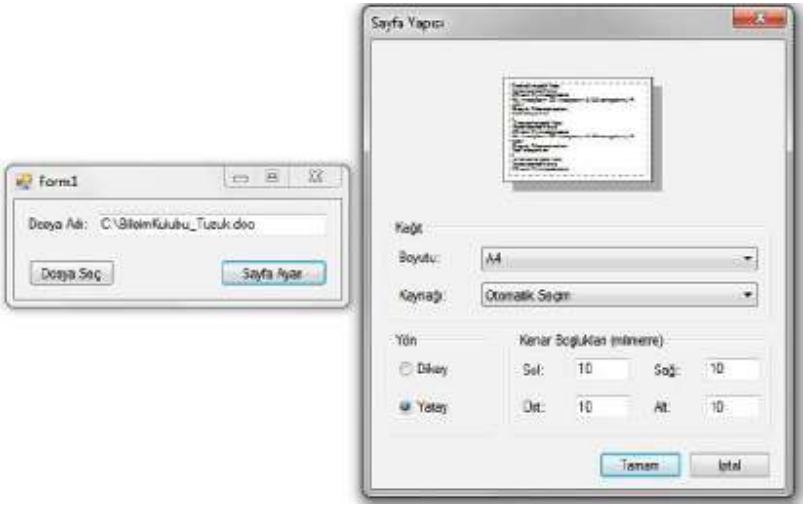
PageSetupDialog nesnesini aşağıdaki biçimde tanımlıyoruz.

```
PageSetupDialog dlgPageSetup = new PageSetupDialog();
```

PageSetupDialog nesnesi altında yazıcı ayarlarına **PrinterSettings**, sayfa ayarlarına ise **PageSettings** özelliği ile erişebiliriz. Kodumuzda…

```
dlgPageSetup.PageSettings.Landscape = true;
```

… satırında **Landscape** özelliğine **true** değerini aktararak belgenin yönünün yatay biçimde ayarlanmasını sağlıyoruz.



NOT Aynı işlemi yapan kodu **ToolBox** üzerinde yer alan **`PageSetupDialog`** kontrolü ile yapmak için **Form** üzerine 1 adet **`PageSetupDialog`** kontrolü, 1 adet **`PrintDocument`** kontrolü ekliyoruz. Diğer kontroller örneğimizdeki ile aynı. **`PageSetupDialog`** kontrolü sayfa ayarları penceresini görüntülemek için kullanılırken, **`PrintDocument`** kontrolü ise yazdırma olaylarını içerisinde barındırır.

Sayfa ayarları diyalog penceresini görüntüleyen kodumuz aşağıdaki gibi olacak:

```
private void btnSayfaAyari_Click(object sender, EventArgs e)
{
 pageSetupDialog1.PageSettings = printDocument1.DefaultPageSettings;
 if (pageSetupDialog1.ShowDialog() == DialogResult.OK)
 {
   printDocument1.DefaultPageSettings = pageSetupDialog1.PageSettings;
   printDocument1.Print();
  }
}
```

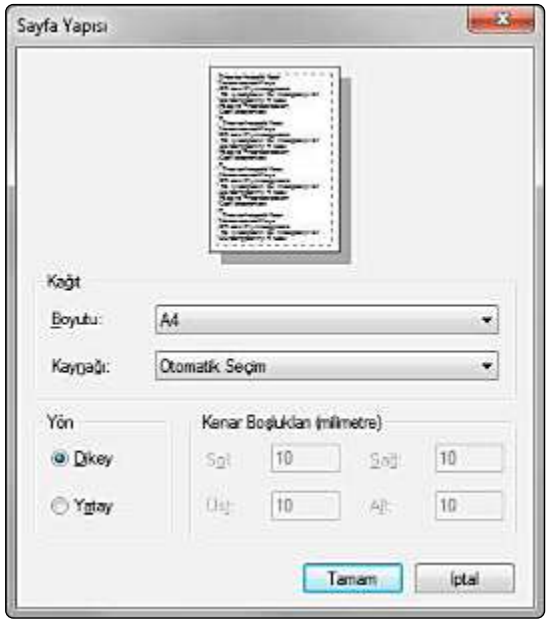
Kodumuzda **`PageSetupDialog`** penceresinin **`PageSettings`** özelliğine **`DefaultPageSettings`** değerini atayarak, pencerenin varsayılan ayarları ile açılmasını sağlıyoruz.

Uygulama sayfa ayarları penceresindeki ayarların aktif ya da pasif olması için aşağıdaki özelliklere **true** ya da **false** değerini vermeliyiz.

- **AllowMargins:** Kenar boşlukları ayarlama bölümünü aktif ya da pasif yapmak için kullanılır.
- **AllowOrientation:** Sayfa yönünü ayarlama bölümünü aktif ya da pasif yapmak için kullanılır.
- **AllowPaper:** Kağıt boyutu ayarlama bölümünü aktif ya da pasif yapmak için kullanılır.

Örneğin kenar boşlukları ayarlama bölümünün pasif olması için aşağıdaki komutu kodumuza eklemeliyiz.

```
pageSetupDialog1.AllowMargins = false;
```



METOTLAR

19

Program yazarken bazı durumlarda bir işi gerçekleştiren kod yapılarını tekrar tekrar yazmak durumunda kaldığımız olur. Bu kodları yeniden yazmak yerine metotları kullanabiliriz. **C#** içerisinde geriye değer döndüren ve değer döndürmeyen metotlar olmak üzere iki çeşit metot yapısı kullanılır.

DEĞER DÖNDÜRMEYEN METOTLAR

Kendi ismi ile çağırılarak içerisinde yazılmış olan kod satırlarını çalıştıran ve geriye herhangi bir değer döndürmeyen kod bloğudur. Değer döndürmeyen metotlar **void** anahtar sözcüğü ile başlarlar.

Kullanımı şu şekildedir:

```
<erişim biçimi>void<metod ismi>(parametre)
{
Metodun içerisinde çalışacak kodlar;
}
```

Uygulama içerisinde bir kez yazacağımız kod, bir daha yazılmasına gerek kalmadan sadece ismi ile çağırılarak defalarca çalıştırılabilir.

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace Renk_Degistir_Metod
{
  public partial class Form1 : Form
  {
    public Form1()
    {
        InitializeComponent();
    }
    void Renk_Degistir()
    {
        foreach (Control nesne in this.Controls)
        {
            if ((nesne) is TextBox)
            {
                nesne.Text = "";
                nesne.BackColor = Color.SteelBlue;
                nesne.ForeColor = Color.Red;
            }
        }
    }
    private void button1_Click(object sender, EventArgs e)
    {
        Renk_Degistir();
    }
  }
}
```

Örneğimizde **Renk_Degistir** adlı parametresiz bir metot tanımlanarak butona tıklandığı anda bu metot ismi ile çağırılmış ve form üzerindeki **textBox** nesnelerinin renginin değiştirilmesi sağlanmıştır. **Renk_Degistir** metodu içerisinde **foreach** döngümüzde **this.Controls** sınıfı içerisinde yer alan nesnelere erişerek:

```
if ((nesne) is TextBox)
```

Satırı ile eğer erişilen nesne bir **textBox** nesnesi ise üzerinde gerekli değişikliklerin yapılmasını sağlıyoruz.

Metot İçerisinde Parametre Kullanımı

Metotlar parametresiz bir biçimde kullanılabildiği gibi metot içerisine, dışarıdan parametreler ile bilgi göndermek de mümkündür.

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace Renk_Degistir_Metod_Parametreli
{
  public partial class Form1 : Form
  {
    public Form1()
    {
        InitializeComponent();
    }
    void Renk_Degistir(Control nesne)
    {
        if ((nesne) is TextBox)
        {
            nesne.Text = "";
            nesne.BackColor = Color.SteelBlue;
            nesne.ForeColor = Color.Red;
        }
        else if ((nesne) is Label)
        {
            nesne.BackColor = Color.DimGray;
            nesne.ForeColor = Color.White;
        }
    }
```

```
    private void button1_Click(object sender, EventArgs e)
    {
        Renk_Degistir(textBox1);
        Renk_Degistir(label1);
    }
  }
}
```

Örneğimizde **`Renk_Degistir`** metoduna parametre olarak kontrol türünden olan nesne değerini gönderiyoruz. Gönderilen **`nesne`** parametresinin türüne göre metot içerisinde farklı işlemler yapılıyor.

Yukarıdaki örneğimizde **`nesne`** parametresinin kullanımı zorunludur. **C#** içerisinde **Optional Parameters** kullanarak, kullanımı zorunlu olmayan parametreleri tanımlayabiliriz. **Optional** parametreleri tanımlarken ilk değer vermek zorunludur.

```
void Renk_Degistir(Control nesne, string yazi="label", Boolean
bold=false )
{
    if ((nesne) is TextBox)
    {
        nesne.Text = "";
        nesne.BackColor = Color.SteelBlue;
        nesne.ForeColor = Color.Red;
        if (bold==true)
            nesne.Font = new Font(Font, FontStyle.Bold);
    }
    else if ((nesne) is Label)
    {
        nesne.BackColor = Color.DimGray;
        nesne.ForeColor = Color.White;
    }
    nesne.Text = yazi;
}
private void button1_Click(object sender, EventArgs e)
{
    Renk_Degistir(label1, "Adı Soyadı");
```

```
    Renk_Degistir(nesne:textBox1, yazi:”Volkan AKTAŞ”,bold:true);
    Renk_Degistir(label2, “Görev Yeri”);
    Renk_Degistir(nesne: textBox2, yazi: “MERSİN”);
}
```

Metodumuzu tanımlarken:

```
Renk_Degistir(Control nesne, string yazi=””, Boolean bold=false )
```

Yukarıda görüldüğü gibi optional olarak kullanacağımız parametrelere ilk değer ataması yapıyoruz.

Optional parametre olarak tanımladığımız **yazi** değişkenine ilk değer olarak “**label**” metnini aktardık. **button1** içerisinden ise **“Adı Soyadı”** değerini göndererek **Label1** nesnesinin **text** değerinin değiştirilmesini sağladık. Görüldüğü gibi:

```
Renk_Degistir(label1);
```

Şeklinde **yazi** parametresini kullanmadan **Renk_Degistir** metodunu çağırabildiğimiz gibi:

```
Renk_Degistir(label1, “Adı Soyadı”);
```

Şeklinde **yazi** parametresini kullanarak da **Renk_Degistir** metodunu çağırabiliriz.

Eğer optional parametreye değer ataması yapılmazsa varsayılan değer kullanılır.

Yani:

```
Renk_Degistir(label1);
```

Satırında **yazi** parametresini kullanmadan **Renk_Degistir** metodunu çağırdığımızda **label1** nesnesinin **Text** özelliğine **yazi** parametresinin varsayılan değeri olan “**label**” değeri aktarılacaktır.

Optional parametrelere değer aktarmak için:

```
Parametreadı:değer
```

Biçiminde değer aktarması yapıyoruz.

```
Renk_Degistir(nesne:TextBox1, yazi:"Volkan AKTAŞ",bold:true);
```

Örneğinde görüldüğü gibi optional olan **yazi** ve **bold** değişkenlerine değer aktarması yaptık.

Kullanmadığımız parametreleri yazmadan diğer parametrelere değer aktarımı yapabiliriz.

```
Renk_Degistir(Label2, "Görev Yeri");
```

Örneğinde olduğu gibi.

C# 7.2 versiyonundan itibaren metoda gönderilecek parametrelerin değerlerini, parametrelerin bir kısmının ismini yazıp bir kısmını yazmadan gönderebiliriz. Örneğin yukarıdaki kod satırlarındaki;

```
Renk_Degistir(nesne:textBox1, yazi:"Volkan AKTAŞ",bold:true);
```

Satırını artık:

```
Renk_Degistir(nesne: textBox1, "Volkan AKTAŞ", bold: true);
```

Şeklinde sırasız bir biçimde, **yazi** parametresinin ismini yazmayıp diğerlerinin ismini yazarak kullanabiliyoruz.

Bir optional parametreden sonra başka bir optional parametre kullanılmalıdır. Dolayısı ile optional parametreler diğer parametrelerden sonra kullanılmalıdır.

Örneğin:

```
void Renk_Degistir(Control nesne, Boolean bold=false, string yazi)
```

şeklindeki bir kullanım **hata** verecektir.

Parametre sayısının belli olmadığı durumlarda **params** deyimi ile metot içerisinde bu parametreleri, bir parametre dizisi şeklinde tanımlayabiliriz. **params** ile tanımladığımız **dizi** değişkeni metodun son parametresi olarak tanımlanmalıdır.

Örnek olarak bir form üzerindeki butonları **pasif** yapacak bir **sub** metodu yazalım. Buton sayısı belirsiz olduğu için **btn** şeklinde bir dizi değişkenini **params** ile parametre dizisi olarak tanımladık.

```
public void pasif(bool onay, params Button[] btn)
{
    byte i = 0;
    if (onay == true)
    {
        for (i = 0; i <= btn.Length - 1; i++)
            btn[i].Enabled = false;
    }
}
private void Form1_Load(object sender, EventArgs e)
{
    pasif(true, btnGuncelle, btniptal, btnKaydet, btnSil);
}
```

Kodumuzu çalıştırdığımızda "**Yeni**" dışındaki bütün butonların **pasif** yapıldığını görürüz. "**Pasif**" alt programının ilk parametresine "**true**" değerini göndererek daha sonra parametre dizisi şeklinde gelen buton isimlerinin pasif yapılmasını sağladık.



Veritabanı uygulamalarında sıklıkla kullanılan **kayıt ekleme**, **silme**, **güncelleme** gibi işlemlerde sıklıkla bu yöntemi kullanarak kod satırlarını kısaltabilirsiniz.

Değer Döndüren Metotlar

Kendi ismi ile çağırılarak içerisinde yazılmış olan kod satırlarını çalıştıran ve geriye değer döndüren kod bloğudur.

Kullanımı şu şekildedir:

```
<erişim biçimi><geri dönüş değeri> <metod ismi>(parametre)
{
Metodun içerisinde çalışacak kodlar;
}
```

Örnek olarak kendisine gönderilen sınav notunun 5'lik not sistemindeki karşılığını hesaplayıp geriye döndüren bir metot yazalım.

```
using System;
using Microsoft.VisualBasic;
using System.Windows.Forms;
namespace NotHesaplama
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        public byte NotHesapla(byte notu)
        {
            byte puan = 0;
            if (notu < 25)
                puan=0;
            else if (notu < 45)
                puan = 1;
            else if (notu < 55)
                puan = 2;
            else if (notu < 70)
                puan = 3;
            else if (notu < 85)
                puan = 4;
            else if (notu <= 100)
                puan = 5;
```

```
            else
                MessageBox.Show("Yanlış Not Girdiniz");
            return puan;
        }
        private void button1_Click(object sender, EventArgs e)
        {
            byte notdegeri = 0;
            notdegeri = Convert.ToByte(Interaction.
InputBox("Notunuzu Girin"));
            MessageBox.Show("Puanınız= " + NotHesapla(notdegeri));
        }
    }
}
```

Kodumuzda:

```
NotHesapla(notdegeri)
```

Satırında **NotHesapla** metodunun notu parametresine, **notdegeri** değişkeni içerisinde saklanan **inputBox** içerisinde girilen not değeri gönderilmekte ve **NotHesapla** metodumuz gönderilen değeri 5'lik not sistemine çevirip sonucu geriye göndermektedir.



Metot içerisinde kullanılan parametre türü ile metoda gönderilen değerin türü aynı olmalıdır. Örneğimizde her ikisi de **byte** türündendir.

Aynı örneği optional parametre kullanacak biçimde düzenleyelim.

```
using System;
using Microsoft.VisualBasic;
using System.Windows.Forms;
namespace NotHesaplama_Optional
{
    public partial class Form1 : Form
    {
      public Form1()
      {
         InitializeComponent();
      }
      public byte NotHesapla(byte yazılı, byte sozlu= 0)
      {
          byte puan = 0;
          int notu = 0;
          if (sozlu > 0)
              notu = (yazılı + sozlu) / 2;
          else
              notu = yazılı;

          if (notu < 25)
              puan = 0;
          else if (notu < 45)
              puan = 1;
          else if (notu < 55)
              puan = 2;
          else if (notu < 70)
              puan = 3;
          else if (notu < 85)
              puan = 4;
          else if (notu <= 100)
              puan = 5;
          else
              MessageBox.Show("Yanlış Not Girdiniz");
          return puan;
      }
```

```
      private void button1_Click(object sender, EventArgs e)
      {
          byte yazılınotu = 0;
          byte sozlunotu = 0;
          yazılınotu = Convert.ToByte(Interaction.
InputBox("Yazılı Notunuzu Girin"));
          sozlunotu = Convert.ToByte(Interaction.
InputBox("Sözlü Notunuzu Girin"));

          if (sozlunotu == 0)
                MessageBox.Show("Puanınız= " +
NotHesapla(yazılınotu,0));
          else
                MessageBox.Show("Puanınız= " +
NotHesapla(yazılınotu, sozlunotu));
      }
    }
}
```

NotHesapla metodumuzda **sozlu** değişkenini **optional** parametre olarak tanımladık. Eğer sözlü notu girişi yapılmazsa metodumuzu **NotHesapla(yazılınotu,0)** şeklinde, sözlü notu girişi yapılırsa **NotHesapla(yazılınotu, sozlunotu)** şeklinde kullanıyoruz.

Aynı örneğimizi **params** deyimi ile parametreleri bir parametre dizisi şeklinde tanımlayarak yeniden düzenleyelim. Yukarıda yaptığımız örnekteki sözlü notlarını **params** şeklinde tanımlayarak kullanıcının istediği kadar sözlü notu girmesine imkân sağlayalım.

```
public byte NotHesapla(byte yazılı, params byte[] sozlu)
{
    byte puan = 0;
    int notu = 0;
    byte i = 0;
    int sozlutoplam = 0;

    for (i = 0; i <= sozlu.Length - 1; i++)
        sozlutoplam += sozlu[i];
```

```
    if (i > 0)
        notu = (yazılı + sozlutoplam) / (i + 1);

    if (notu < 25)
        puan = 0;
    else if (notu < 45)
        puan = 1;
    else if (notu < 55)
        puan = 2;
    else if (notu < 70)
        puan = 3;
    else if (notu < 85)
        puan = 4;
    else if (notu <= 100)
        puan = 5;
    else
        MessageBox.Show("Yanlış Not Girdiniz");
    return puan;
}
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show(NotHesapla(45, 80, 90, 100).ToString());
}
```

Kodumuzu çalıştırdığımızda **`NotHesapla(45, 80, 90, 100)`** şeklinde metodumuza yazılı notu değeri olarak **`45`**'i, sözlü notu değerleri olarak da **`80,90,100`** değerlerini gönderiyoruz. Bu değerlerin ortalaması alınarak (**`78,75`**) bunun not karşılığı olan **`4`** değeri görüntüleniyor.

Params IEnumerable

C# 6 sürümünden itibaren, **`IEnumerable`** arayüzünü **`params`** deyimi ile birlikte metotlarda kullanabiliriz. Bu sayede parametre dizisi içerisindeki elemanlara sıra ile erişebilmekteyiz. **`Params`** deyimi için yaptığımız **`NotHesapla`** metodu örneği üzerinde **`Params IEnumarable`** kullanarak değişiklik yapalım.

```
public static int NotHesapla(int yazılı,params IEnumerable<int>[]
sozlu)
{
    byte puan = 0;
```

```
    int i,notu = 0,sozlusayisi=0;
    var sozlutoplam = 0;

    sozlusayisi = sozlu[0].Count();
    for (i=0;i<sozlusayisi;i++)
        sozlutoplam += sozlu[0].ElementAt(i);

    if (sozlusayisi > 0)
        notu = (yazılı + sozlutoplam) / (sozlusayisi+1);

    if (notu < 25)
        puan = 0;
    else if (notu < 45)
        puan = 1;
    else if (notu < 55)
        puan = 2;
    else if (notu < 70)
        puan = 3;
    else if (notu < 85)
        puan = 4;
    else if (notu <= 100)
        puan = 5;
    else
        Console.WriteLine("Yanlış Not Girdiniz");
    return puan;
}
static void Main(string[] args)
{
    Console.WriteLine(NotHesapla(45,new int[] { 80, 90, 100 }));
    Console.ReadLine();
}
```

`NotHesapla` metodumuzda **`sozlu`** notlarını **`params IEnumarable`** kullanarak **`int`** dizisi şeklinde tanımladık. Sözlü notlarına indeks numaralarını kullanarak erişebilmek için **`ElementAt`** metodunu kullandık.

Metodumuzu çağırırken ise:

```
new int[] { 80, 90, 100 }
```

Şeklinde parametre dizisi olarak sözlü notlarını belirttik.

Tuple Deconstruction

Tuple sınıfını kullanarak bir metot içerisinden geriye birden fazla değer döndürebiliriz. **Tuple** asıl olarak farklı tipten verilerin saklanması amacıyla kullanılmaktadır.

Örneğin:

```
var bilgiler = new Tuple<int, string,DateTime>(1, "Volkan",DateTime.
Now);
```

Satırında görüldüğü gibi bilgiler isimli **Tuple** içerisinde, farklı türden değerleri saklayabiliyoruz.

Tuple oluşturmak için **Create** metodunu da kullanabiliriz. Yukarıdaki örnekte yaptığımız tanımlamayı:

```
var bilgiler = Tuple.Create<int, string, DateTime>(1, "Volkan",
DateTime.Now);
```

Şeklinde **Create** metodunu kullanarak yapabiliriz.

Tuple içerisindeki elemanlara **Item** özelliğini kullanarak erişebiliriz. Örneğin bilgiler isimli **Tuple** içerisindeki ilk elemana **bilgiler.Item1** şeklinde erişebilirken ikinci elemana **bilgiler.Item2** şeklinde erişebiliriz.

Buna göre:

```
Console.WriteLine(bilgiler.Item2);
```

Satırının çıktısı "**Volkan**" şeklinde olacaktır.

C# 7.3 sürümünden itibaren **tuple** değerlerini karşılaştırmak için eşitlik **(==)** ve farklılık **(!=)** operatörlerini kullanabiliriz. Örneğin ilk olarak bu operatörleri kullanmadan bir uygulama yapalım.

```
static void Main(string[] args)
{
  Tuple<string, int> sifre = new Tuple<string,
      int>(Console.ReadLine(), int.Parse(Console.ReadLine()));
  (string, int) admin = ("admin",12345);
  if (sifre.Item1 == admin.Item1 && sifre.Item2 == admin.Item2)
      Console.WriteLine("Admin olarak giriş yaptınız!");
  else
      Console.WriteLine("Kullanıcı olarak giriş yaptınız!");
  Console.ReadLine();
}
```

Kodumuzda **sifre** isimli **tuple** değerlerini **admin** değişkeni ile karşılaştırmak için ve **(&&)** mantıksal operatörünü kullandık. Bunun yerine:

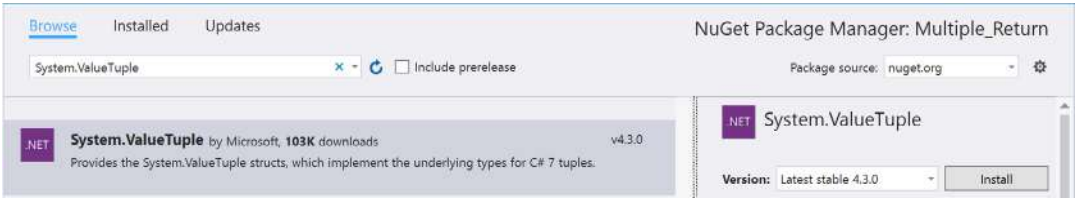
```
if (sifre.Item1 == admin.Item1 && sifre.Item2 == admin.Item2)
```

Satırını **C# 7.3** versiyonundan itibaren, eşitlik **(==)** operatörünü kullanarak aşağıdaki gibi yazabiliriz:

```
if ((sifre.Item1,sifre.Item2) == (admin.Item1,admin.Item2))
```

C# 7 sürümünden itibaren, **Tuple** sınıfı içerisinde tanımlanan alanlara ait değerleri alarak değişkenlere aktarmak mümkündür. Bu işleme "**Tuple Deconstruction**" adı verilmektedir.

Bu özelliği kullanabilmek için **System.ValueTuple** paketini **NuGet** üzerinden yüklememiz gerekiyor. "**Project**" menüsünden "**Manage NuGet Packages**" seçeneğine tıklayarak **System.ValueTuple** referansını projemize ekliyoruz.



Kodumuz aşağıdaki gibi olacak:

```
public static(string deger,bool onay) SifreKontrol(string sifre)
{
    bool sifreOnay = false;
    if (sifre == "123")
        sifreOnay = true;
    var donenDeger = (d: sifre, o: sifreOnay);
    return donenDeger;
}
static void Main(string[] args)
{
    Console.WriteLine("Şifreyi Giriniz:");
    string girilensifre = Console.ReadLine();
    var sifre = SifreKontrol(girilensifre);
    Console.WriteLine
($"Girilen Şifre={sifre.Item1}. İşlem Sonucu={sifre.Item2}.");
    Console.ReadLine();
}
```

Kodumuzda **`SifreKontrol`** isimli **`Tuple`** içerisine parametre olarak konsol ekranında girilen şifre değerini gönderiyoruz. **`Tuple`** metot içerisinde alanlara ait değerleri **`donenDeger`** isimli değişkene aktararak metottan geriye döndürüyoruz.

`Metod` parametrelerini direkt olarak kendi isimleriyle kullanmak da mümkündür.

Bunun için **`Main`** metodumuzu aşağıdaki gibi düzenliyoruz:

```
static void Main(string[] args)
{
    Console.WriteLine("Şifreyi Giriniz:");
    string girilensifre = Console.ReadLine();
    (string deger, bool onay) = SifreKontrol(girilensifre);
    Console.WriteLine
($"Girilen Şifre={deger}. İşlem Sonucu={onay}.");
    Console.ReadLine();
}
```

Kodumuzda **`SifreKontrol`** metodumuzdaki alanlara ait değerleri "**`değer`**" ve "**`onay`**" ismli değişkenlere aktararak çektik.

C# 7.1 versiyonundan itibaren **tuple** isimlendirmelerinde yapılan geliştirme ile artık haricî bir değişken tanımlamaya gerek kalmadan **tuple** içerisindeki alanlara erişmek mümkün olmaktadır. Yukarıda yaptığımız örnek üzerinde bir düzenleme yaparak şifre alanına boş değer girilip girilmediğini kontrol edelim. Kodumuzda yalnızca **SifreKontrol** isimli metodumuzda düzenleme yapıyoruz.

```
public static(string deger,bool onay) SifreKontrol(string sifre)
{
    bool sifreOnay = false;
    if (sifre == "123")
        sifreOnay = true;
    var donenDeger = (d: sifre, o: sifreOnay);
    if (donenDeger.d.Length == 0)
        donenDeger.d = "Şifre alanı boş bırakıldı!";
    return donenDeger;
}
```

Kodumuzda:

```
if (donenDeger.d.Length == 0)
        donenDeger.d = "Şifre alanı boş bırakıldı!";
```

Satırılarında eğer şifre alanı boş bırakılmışsa **donendeger** isimli **tuple** değişkenin **d** alanına "**Şifre alanı boş bırakıldı!**" değerinin aktarılmasını sağlıyoruz.

C# 7.1 ile birlikte kodumuzu aşağıdaki gibi yazabiliyoruz:

```
public static(string deger,bool onay) SifreKontrol(string sifre)
{
    bool sifreOnay = false;
    if (sifre == "123")
        sifreOnay = true;
    var donenDeger = (sifre, sifreOnay);
    if (donenDeger.sifre.Length == 0)
        donenDeger.sifre = "Şifre alanı boş bırakıldı!";
    return donenDeger;
}
```

Kodumuzda görüldüğü gibi artık **d** adlı değişkeni kullanmadan, doğrudan **sifre** isimli alanı kullanarak boş değer kontrolü yapabiliyoruz.

Tuple İle Geriye Çoklu Değer Döndürme

IEnumerable arayüzünü kullanarak **Tuple** ile istediğiniz sayıda parametre tanımlayarak **return** ile bunları geriye döndürebiliriz.

```
public static Tuple<int, int> NotTopla(IEnumerable<int> notlar)
{
    int i = 0, t = 0;
    foreach (var not in notlar)
    {
        i++;
        t += not;
    }
    return Tuple.Create(i, t);
}
static void Main(string[] args)
{
    var notlar = new[] { 90, 60, 80, 70 };
    var nothesapla = NotTopla(notlar);
    Console.WriteLine("Ortalama="+(nothesapla.Item2/nothesapla.
Item1));
    Console.ReadLine();
}
```

Kodumuzda **NotTopla** isimli **Tuple** içerisine **notlar** ismiyle tanımladığımız dizi içerisindeki değerleri gönderdik. **Tuple** içerisinde:

```
return Tuple.Create(i, t);
```

Satırında **i** ve **t** parametrelerini geriye döndürdüğümüz için **NotTopla** metodundan dönen bu değişkenlerin değerlerini kullanarak not ortalamasını hesapladık.

C# 7 sürümünden itibaren gelen yeniliklerden birisi de tek bir **return** ifadesi ile geriye birden fazla değer döndürmenin mümkün olmasıdır. Yukarıda not ortalaması için yaptığımız örneği **C# 7** sürümünden itibaren aşağıdaki şekilde gerçekleştirebiliriz.

Yeni kodumuz aşağıdaki gibi olacak:

```
public static (int i, int t) NotTopla(IEnumerable<int> notlar)
{
    int i = 0, t = 0;
    foreach (var not in notlar)
    {
        i++;
        t += not;
    }
    return (i, t);
}
static void Main(string[] args)
{
    var notlar = new[] { 90, 60, 80, 70 };
    var nothesapla = NotTopla(notlar);
    Console.WriteLine("Ortalama=" + (nothesapla.t / nothesapla.i));
    Console.ReadLine();
}
```

Kodumuzda:

```
return (i, t);
```

Satırında görüldüğü gibi metot içerisinde direkt olarak **`return`** ile birden fazla parametreyi geriye döndürebiliyoruz.

```
Console.WriteLine("Ortalama=" + (nothesapla.t / nothesapla.i));
```

Satırında da görüldüğü gibi metot parametrelerini direkt olarak kullanıp ortalamayı hesapladık.

REF RETURN

Kitabımızda **Değişkenler** bölümünde referans türü değişkenleri anlatmıştık. Bir metoddan dönen ya da bir yapıya (*struct*) geçilen değişken, değer türünde olduğunda, içeriği kopyalanır. Eğer referans türünde (*ref anahtar sözcüğünü kullanarak*) kullanılırsa yeni değişken aynı verilere başvurur. Referans tipi değişkenler direkt olarak veriyi saklamak yerine verinin hafızada saklanacağı adresi saklarlar. **C# 7** sürümünden itibaren değişkenlerimizi metoddan referans olarak döndürerek bir başka değişkene aktarıp kullanabiliyoruz.

```
static void Main(string[] args)
{
    string[] diller = { "C++", "C#", "Python" , "Java"};
    ref string dil = ref StringYaz(3, diller);
    dil = "Go";
    Console.WriteLine(diller[3]);
    Console.ReadKey();
}
public static ref string StringYaz(int indeks, string[] degerler)
{
    foreach (string deger in degerler)
    {
        if (degerler[indeks] == deger)
            return ref degerler[indeks];
    }
    throw new Exception("Aranan değer bulunamadı");
}
```

Kodumuzda metodumuzun içerisinde bir dizi içerisinde indeks numarası verilen değeri referans türünde geriye döndürüyoruz. Geriye referans türünden değer döndüğü için verinin kendisi değil de hafızadaki adresi dönmektedir. **`Main`** metot içerisinde metodumuzun dönüş değerini **`dil`** isimli referans türdeki değişkene aktardık. Değişkenimiz referans türünden olduğu için değişkenin içeriğinde yaptığımız değişiklik, **`diller`** dizisinin içerisindeki değeri de etkilemekte ve değişkenin yeni değeri olan "**`Go`**" değeri görüntülenmektedir.

ref Conditional

C# 7.2 sürümünden itibaren **?** (*Conditional - Koşul*) operatörünü **ref** ifadesi ile birlikte kullanarak referansı iki ifadeden birinin sonucuna göre döndürebiliriz. Bu referansı lokal ya da yalnız okunabilir (*read only*) bir değişkene atayabilir, bir metodun parametresi ya da dönüş değeri olarak kullanabiliriz. Yapacağımız örnek uygulamada öncelikle bir metoda gönderilen parametrelerdeki değerleri, metota verilen koşula bağlı olarak karşılaştırıp geriye referans türde bir sonuç döndürelim.

```
static void Main(string[] args)
{
  var dizi1 = new int[] { 1, 3, 5, 7, 9 };
  var dizi2 = new int[] { 2, 4, 6, 8, 10 };

  Console.WriteLine("Bir sayi giriniz=");
  int sayi = Convert.ToInt32(Console.ReadLine());
  int index = dizi1.Length - 1;
  ref int refValue = ref sec(sayi % 2 == 1, ref dizi1[index],
     ref dizi2[index]);
  refValue = dizi1.Length;

  Console.WriteLine(string.Join(" ", dizi1));
  Console.WriteLine(string.Join(" ", dizi2));
  Console.ReadLine();
}
public static ref T sec<T>(bool kosul, ref T deger1, ref T deger2)
{
  if (kosul)
  {
    return ref deger1;
  }
  else
  {
    return ref deger2;
  }
}
```

Kodumuzda **Sec** isimli metodumuzun içerisinde **deger1** ve **deger2** isimli parametrelere gelen değerler koşul parametresine gelen karşılaştırma ifadesine göre karşılaştırılarak sonuç geriye döndürülmektedir.

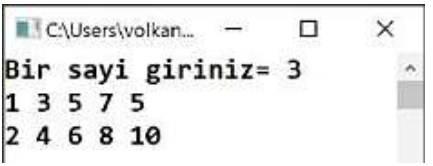
Metodumuzu çağırırken:

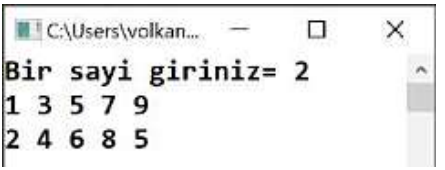
```
int index = dizi1.Length - 1;
ref int refValue = ref sec(sayi % 2 == 1, ref dizi1[index],
ref dizi2[index]);
refValue = dizi1.Length;
```

Satırlarında görüldüğü gibi belirttiğimiz koşula göre eğer sayı **tek** ise **`dizi1`** isimli dizinin, **çift** ise **`dizi2`** isimli dizinin son elemanını:

```
refValue = dizi1.Length;
```

Satırı ile dizinin eleman sayısını bularak değiştiriyoruz. **`refValue`** isimli değişkenin değerinde yapılan değişiklik dizinin belirttiğimiz elemanında güncellenerek uygulanmaktadır. Uygulamayı çalıştırdığımızda görüldüğü gibi girilen sayı tekse birinci dizinin son elemanı, dizinin eleman sayısını gösterir şekilde güncellenirken; çift sayı girildiğinde ise ikinci dizinin son elemanı dizinin son elemanını gösterir şekilde güncellenmektedir.





Yukarıda metot tanımlayarak yaptığımız aynı işlemi **C# 7.2** sürümünden itibaren metot kullanmadan,

```
ref int refValue = ref (sayi % 2 == 1 ? ref dizi1[index] : ref
dizi2[index]);
```

şeklinde gerçekleştirebiliyoruz.

in Parametreler

C# 7.2 sürümünden itibaren metotlarda kullandığımız parametreleri **read only** (*değiştirilemez*) olarak belirleyerek ilgili değerlerin sabit tutulup değiştirilmemesini sağlayabiliriz. (*In parametreler, asenkron metotlarda kullanılamaz.*)

```
static void Main(string[] args)
{
    int d = 3;
    ekle(d);
    Console.WriteLine(d);
    Console.ReadLine();
}
public static int ekle(in int s)
{
    int sayi = s++;
    return sayi;
}
```

Kodumuzda:

```
int sayi = s++;
```

Satırında "**Cannot assign to variable 'in int' because it is a readonly variable**" şeklinde bir hata mesajı oluşacaktır. Çünkü metodun tanımlanmasında **`s`** parametresi **`in`** ifadesi ile **`readonly`** olarak belirtilmiştir. Böylece metot içerisinde ilgili parametrenin değeri değiştirilmek istenildiğinde oluşacak derleme zamanı hatasıyla, parametrenin **`readonly`** olduğu konusunda bilgilendirme yapılacaktır.

REF READONLY RETURN

in parametrelerle bir metoda ait değer türünde parametrelerin değiştirilmelerine izin vermeden referansa göre aktarılabildiğini yukarıda yazmıştık. Ancak değişkenlerimizi metoddan referans olarak döndürerek bir başka değişkene aktarıp kullanabiliyoruz. Değer türünde değişkenler kullanıldıkça içeriklerinin kopyalanması nedeniyle performansın düşmesine neden olur. Referans türlerini kullanarak hafıza üzerinde aynı verilere başvurmak için ise farklı değişkenler gerekir ve muhtemelen bu verilerin de kopyalanması gerekecektir. Örneğin **string** metotlardan **ToUpper** metodu gibi metotlar dizinin kendisini değiştirmezken bunun yerine yeni bir dizi döndürür. Bu tip işlemlerde kullanılmayan nesnelerin **garbage collector** ile temizlenmesi gerekmektedir. Diğer yandan garbage collector'un çok fazla kullanılması da performansı **olumsuz** etkileyecektir. Hem performansı arttırmak hem de kod güvenliğini sağlayarak veri kopyalamak amacıyla **C# 7.2** versiyonundan itibaren **ref readonly return** yapısını kullanarak tek bir değişken kullanıp sürekli olarak bunun referansının döndürülmesini sağlayabiliriz.

```
public static int sayac = 0;
public static void arttir()
{
  sayac++;
}
public static ref readonly int SayacDegeri()
{
  return ref sayac;
}
static void Main(string[] args)
{
  ref readonly int sayi = ref SayacDegeri();
  Console.WriteLine(sayi);
  arttir();
  Console.WriteLine(sayi);
  Console.ReadLine();
}
```

Kodumuzda **Main** metodumuzun içerisinden **SayacDegeri()** isimli metodumuzu çağırıyoruz. Bu metot **readonly ref int** türünde **sayac** değişkenini geriye döndürecektir. **Main** metot içerisinde **arttir()** isimli metot çağrıldığında **sayac** değişkeninin hafızadaki kopyası alınarak değeri bir arttırılır.

Kodumuzun ekran çıktısı



Şeklinde olacaktır.

sayac değişkenin değerini arttırdığımızda, **SayacDegeri()** isimli metodun içerisindeki değişkenin değeri değişmez.

Kodumuzda **Main** metot içerisinde:

```
sayi++;
```

şeklinde **sayi** değişkeninin değerini arttırmaya kalktığımızda değişkenimiz **ref readonly** (*referans türünde yalnız okunabilir*) şeklinde olduğundan dolayı derleme esnasında hata oluşacaktır.

Kontrollere Metot Bağlamak (Event Handler)

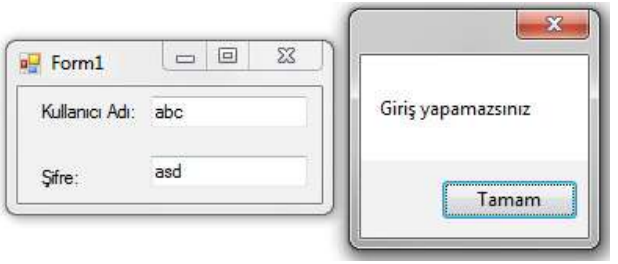
Bir kontrole herhangi bir metodu bağlamak için **+=** operatöründen faydalanabiliriz. Bir kontrole eklenen metodu çıkartmak için ise **-=** operatörü kullanılabilir.

```
public void sorgula(object sender, KeyEventArgs e)
{
  if (txtKullanıcıAdı.Text == "volkan" & txtSifre.Text == "123")
      MessageBox.Show("Giriş yapabilirsiniz");
  else
      MessageBox.Show("Giriş yapamazsınız");
}
private void txtKullanıcıAdı_KeyDown(object sender, KeyEventArgs e)
{
  if (e.KeyData == Keys.Enter)
      this.txtKullanıcıAdı.KeyDown += new
          System.Windows.Forms.KeyEventHandler(this.sorgula);

}
```

```
private void txtSifre_KeyDown(object sender, KeyEventArgs e)
{
  if (e.KeyData == Keys.Enter)
      this.txtSifre.KeyDown += new
          System.Windows.Forms.KeyEventHandler(this.sorgula);
}
```

Örneğimizde **`txtKullanıcıAdı`** isimli **`textBox`** ve **`txtSifre`** isimli **`textBox`**'ların **`KeyDown`** olaylarına **event handler** ile **`sorgula`** metodunu bağlıyoruz. Böylece **`textBox`** içerisinde **enter** tuşuna basıldığında **`sorgula`** isimli metodu çağırarak çalıştırılmasını sağlıyoruz.



Generic Metotlar

Generic koleksiyonlar, içerisinde saklanabilecek veri türüne karar verilebilen, aynı türdeki verilerin bir arada saklandığı koleksiyon türleridir. Veriler **`object`** türden olmadığı için **boxing** ve **unboxing** işlemleri yapılmaz. Dolayısıyla daha performanslı çalışırlar. Generic koleksiyonları metotların tanımlanmasında da kullanabiliriz. Generic metotları kullanabilmek için **System.Collections.Generic** namespace'ini projemize ekliyoruz. Örnek olarak, kendisine parametre olarak gönderilen bir diziyi sıralayarak geri döndüren bir metot yazalım.

```
public static int[] Sirala(int[] dizi)
{
  IEnumerable<int> siralidizi = dizi.OrderByDescending(n => n);
  return (siralidizi.ToArray());
}
static void Main(string[] args)
{
  int[] d = { 5, 3, 8, 2, 9 };
  Console.WriteLine("[{0}]", string.Join(", ", Sirala(d)));
  Console.ReadLine();
}
```

Kodumuzda **`Sirala`** metoduna **`int`** türünde gelen bir dizi:

```
IEnumerable<int> siralidizi = dizi.OrderByDescending(n => n);
```

Satırında **Linq** sorgusu ile **`OrderByDescending`** metodu kullanılarak büyükten küçüğe doğru sıralanmaktadır. Sorgumuzdan geriye dönen değeri **`IEnumerable<int>`** türündeki **`siralidizi`** isimli değişkene aktarıyoruz. Dönen değer, bir dizi değil de sorgulanabilir bir nesnedir. Dolayısıyla **`siralidizi`** isimli değişkenimizi:

```
return (siralidizi.ToArray());
```

Satırında **`ToArray()`** metoduyla diziye dönüştürerek geriye döndürüyoruz. Farklı veri tiplerinden oluşan diziler için farklı metotlar yazmamız gerekecektir. Farklı veri türleri yerine bir tek **`Type`** parametresi kabul eden bir generic metot yazarak bu işlemi tek bir metotla gerçekleştirebiliriz.

Metodumuzu aşağıdaki gibi yeniden yazıyoruz.

```
public static IEnumerable<T> Sirala<T>(T[] dizi)
{
  return dizi.OrderByDescending(a => a);
}
static void Main(string[] args)
{
  int[] d = { 5, 3, 8, 2, 9 };
  Console.WriteLine("[{0}]", string.Join(", ", Sirala(d)));
  Console.ReadLine();
}
```

Kodumuzda metodumuzu:

```
public static IEnumerable<T> Sirala<T>(T[] dizi)
```

Şeklinde generic olarak tanımladığımız için **T** yerine:

```
int[] d = { 5, 3, 8, 2, 9 };
```

ya da;

```
double[] d = { 5, 3.5, 8.2, 2, 9 };
```

gibi istediğimiz türde değer ataması yapabiliriz.

Metotların Aşırı Yüklenmesi (Overloading)

Bir metotla birden fazla işlemi gerçekleştirmek için o metotla aynı isme sahip yeni metotlar tanımlanabilir. Ancak tanımlanan yeni metotlar farklı parametrelere sahip olmalıdır ya da tanımlanan parametrelerin isimleri aynı ise türleri farklı olmalıdır. Bu işleme **metotların aşırı yüklenmesi** (*overloading*) adı verilir.

Örnek olarak kendisine parametre olarak gönderilen yazılı, sözlü ve ödev notlarının ortalamasını hesaplayarak geriye döndüren bir uygulama yapalım. Ortalama işlemi için aynı isimli iki tane metot tanımlayarak **overloading** işlemi gerçekleştireceğiz. İlk metodumuz yazılı ve sözlü notlarının ortalamasını hesaplarken ikinci metodumuz ek olarak ödev notlarını da ortalamaya dâhil ederek hesaplayacak.

```
private int Ortalama(int yazılı1, int yazılı2, int sozlu)
{
    return(yazılı1 + yazılı2 + sozlu) / 3;
}
private int Ortalama(int yazılı1, int yazılı2, int sozlu, int odev)
{
    return(yazılı1 + yazılı2 + sozlu + odev) / 4;
}
private void btnHesapla_Click(object sender, EventArgs e)
{
    if (txtOdev.Text == "")
    {
        MessageBox.Show("Ortalama= " +
Ortalama(Convert.ToInt32(txtYazılı1.Text),
            Convert.ToInt32(txtYazılı2.Text),
Convert.ToInt32(txtSozlu.Text)));
    }
    else
        MessageBox.Show("Ortalama= " +
Ortalama(Convert.ToInt32(txtYazılı1.Text),
            Convert.ToInt32(txtYazılı2.Text),
Convert.ToInt32(txtSozlu.Text),
            Convert.ToInt32(txtOdev.Text)));
    }
}
```

Overloading metotlar arasında yukarı aşağı ok tuşları ile dolaşabiliriz. Metodumuzun ilk kullanım şeklinde **yazılı1**, **yazılı2** ve **sozlu** adlı parametrelerin tanımlandığı ilk metodumuz görüntülenir.

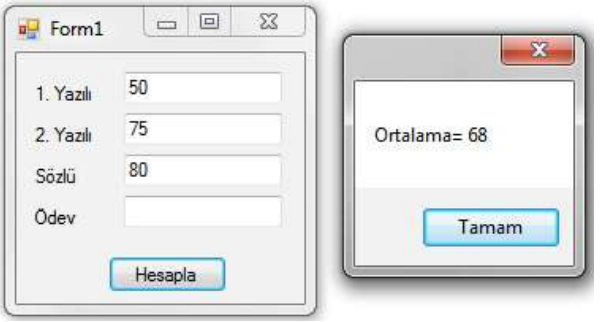
```
Ortalama()
▲ 1 of 2 ▼ int Form1.Ortalama(int yazılı1, int yazılı2, int sozlu)
```

Diğer kullanım şeklinde ise **odev** parametresinin de tanımlandığı ikinci metodumuz görüntülenir.

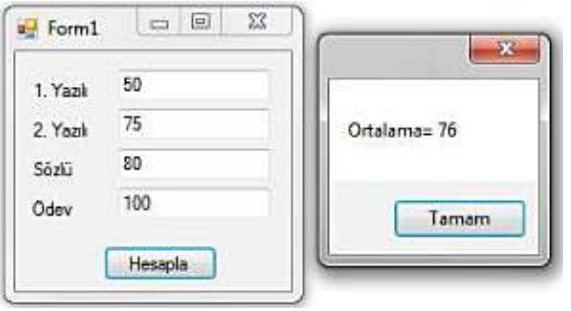
```
Ortalama()
▲ 2 of 2 ▼ int Form1.Ortalama(int yazılı1, int yazılı2, int sozlu, int odev)
```

Ödev değerinin text kutusuna girilip girilmediğine göre ortalamayı hesaplatıp sonucu mesaj penceresi içerisinde görüntülüyoruz.

Örneğimizi çalıştırıp ödev notunu boş girdiğimizde metodumuz ilk kullanım şekli ile çalıştırılacaktır.



Ödev notunu yazdığımızda ise **overloading** metodumuz çalıştırılacaktır.



Temsilciler (Delegates)

Temsilciler en genel ifadesi ile metotların hafızadaki adreslerini tutarlar. Bu yönü ile **C++** içerisinde kullanılan pointer'lara benzerler. Ancak pointer'lara göre daha **güvenli**dirler. **Temsilciler,** metotların çağırılmasında **aracılık** ederler.

Örnek olarak kendisine gönderilen mesaj içeriği ve mesaj penceresi başlığı parametrelerini alarak bir mesaj penceresi içerisinde görüntüleyen bir metot hazırlayalım.

```
public void DelegateMetod(string mesaj, string baslık)
{
    MessageBox.Show(baslık,mesaj);
}
```

Bu metodu çağırmak için bir **`delegates`** hazırlayalım:

```
public delegate void Del_Metod(string mesaj, string baslık);
```

`Delegates` ile yazdığımız tanımlamada parametreler çağıracağımız metoddaki parametreler ile aynı biçimde tanımlanmalıdır.

Bundan sonra **`DelegateMetod`** isimli metodumuzu çağırmak için:

```
Del_Metod event_handler = DelegateMetod;
event_handler("Mesaj","Delegate Örneği");
```

Biçiminde **`delegates`** yardımı ile metodumuzu çağırabiliriz.

Kodumuzun tamamı aşağıdaki biçimde olacak:

```
public void DelegateMetod(string mesaj, string baslık)
{
    MessageBox.Show(baslık,mesaj);
}
public delegate void Del_Metod(string mesaj, string baslık);
private void btnMesaj_Click(object sender, EventArgs e)
{
    Del_Metod event_handler = DelegateMetod;
    event_handler("Mesaj","Delegate Örneği");
}
```



Örneğimizi çalıştırdığımızda kendisine parametre olarak gönderdiğimiz mesaj penceresi başlığının ve mesajın mesaj penceresinde görüntülendiğini görürüz.

Bir delegates'i başka bir metot içerisinden çağırarak delegates ile tanımlanmış metodun işletilmesini sağlayabiliriz.

```
public void Mesaj(string mesaj, string baslık)
{
    MessageBox.Show(baslık, mesaj);
}
public void Topla(int sayi1, int sayi2, Del_Metod Mesaj)
{
    Mesaj("Sonuç", (sayi1 + sayi2).ToString());
}
public delegate void Del_Metod(string mesaj, string baslık);
private void btnMesaj_Click(object sender, EventArgs e)
{
    Del_Metod event_handler = new Del_Metod(Mesaj);
    Topla(5, 10, event_handler);
}
```



Örneğimizde **Topla** isimli metot içerisinden **Del_Metod** isimli delegates yardımı ile **Mesaj** metodunu çağırıyoruz.

```
Topla(5, 10, event_handler)
```

Biçiminde **Topla** metodu içerisinden **event_handler** isimli değişken yardımı ile **Mesaj** metodunun işletilmesini sağlıyoruz.

Delegates ile birden fazla metodu çağırarak sıra ile işletilmesini sağlayabiliriz. Çağıracağımız metotları bir **structure**, **modül** ya da **class** içerisinde tanımlayarak çağırabiliriz.

Örneğin Metotlar isminde bir **struct** tanımlayarak içerisinde **Mesaj**, **Topla** ve **Cıkar** adı ile 3 adet metot tanımlayalım.

```
struct Metodlar
{
    public void Mesaj(string mesaj, string baslık)
    {
        MessageBox.Show(baslık, mesaj);
    }
    public void Topla(int sayi1, int sayi2)
    {
        Mesaj("Sonuç", (sayi1 + sayi2).ToString());
    }
    public void Cıkar(int sayi1, int sayi2)
    {
        Mesaj("Sonuç", (sayi1 - sayi2).ToString());
    }
}
```

Bu metotları çağırmak için bir delagates hazırlayalım:

```
public delegate void Del_Metod(int sayi1, int sayi2);
```

Delegates ile yazdığımız tanımlamada parametrelerimiz çağıracağımız metoddaki parametreler ile aynı olacak biçimde **int sayi1**, **int sayi2** şeklinde tanımlanmalıdır.

Tanımladığımız **structure** içerisindeki elemanlara ulaşmak için:

```
Metodlar metod = new Metodlar();
```

"**metod**" isminde yeni bir değişken tanımlayarak değişkenin tipini oluşturduğumuz **Metodlar** adlı **struct** tipinden veriyoruz.

Son olarak delegemize metotlarımızı ekliyoruz:

```
Del_Metod ehTopla = metod.Topla;
Del_Metod ehCıkar = metod.Cıkar;
```

Kodumuzun tamamı aşağıdaki gibi olacak:

```
struct Metodlar
{
    public void Mesaj(string mesaj, string baslık)
    {
        MessageBox.Show(baslık, mesaj);
    }
    public void Topla(int sayi1, int sayi2)
    {
        Mesaj("Toplam", (sayi1 + sayi2).ToString());
    }
    public void Cıkar(int sayi1, int sayi2)
    {
        Mesaj("Fark", (sayi1 - sayi2).ToString());
    }
}
```

```
public delegate void Del_Metod(int sayi1, int sayi2);
Metodlar metod = new Metodlar();
private void btnMesaj_Click(object sender, EventArgs e)
{
    Del_Metod ehTopla = metod.Topla;
    Del_Metod ehCıkar = metod.Cıkar;
    ehTopla(5, 10);
    ehCıkar(5, 10);
}
```

Bir diğer yöntemde ise **+=** operatörünü kullanarak delegates içerisine metot ekleme işlemi yapabiliriz. **-=** operatörü ile de eklenmiş olan metotlar çıkartılabilir. Yukarıda yaptığımız delegate içerisine metot ekleme işlemini **+=** operatörü ile aşağıdaki gibi yapabiliriz.

```
private void btnMesaj_Click(object sender, EventArgs e)
{
    Del_Metod event_handler = new Del_Metod(metod.Topla);
    event_handler += metod.Cıkar;
    event_handler.Invoke(5, 10);
}
```

Invoke metodu ile metodumuzun çalıştırılmasını sağlıyoruz.

Her iki yönteme göre de kodumuzu çalıştırdığımızda **ekran çıktısı** aşağıda görüldüğü gibi olacak. Önce sayıların toplamı bir mesaj penceresi içerisinde görüntülenecek, daha sonra ise farkı.





Anonim (Anonymous) Metotlar

Delegates konusunda hatırlayacak olursak hem bir delegate tanımlayıp hem de delegate ile çağırılacak olan metodu tanımlıyorduk. Anonymous metotları kullanarak bu iki adımı bir tek adıma düşürebiliriz. Yani aynı anda hem **`delegate`** tanımlayıp hem de **`delegate`** ile çağıracağımız metodu tanımlayabiliriz. Anonymous metotları, tanımlanan metot projenin başka bir bölümünde kullanılmayıp sadece tanımlandığı kod bloğu içerisinde kullanılacaksa tercih etmeliyiz.

Örnek olarak daha önce **`delegate`** konusunda yaptığımız mesaj penceresi görüntüleyen metodu, anonim metot ile yeniden yazalım.

Metodun ilk hâli aşağıdaki gibiydi:

```
public void DelegateMetod(string mesaj, string baslık)
{
    MessageBox.Show(baslık,mesaj);
}
public delegate void Del_Metod(string mesaj, string baslık);
private void btnMesaj_Click(object sender, EventArgs e)
{
    Del_Metod event_handler = DelegateMetod;
    event_handler("Mesaj","Delegate Örneği");
}
```

Anonim metot hâlinde yazılmış biçimi ise aşağıdaki gibi olacak:

```
public delegate void Del_Metod(string mesaj, string baslık);
Del_Metod Mesaj = delegate(string mesaj, string baslık)
{
    MessageBox.Show(baslık, mesaj);
};
private void btnMesaj_Click(object sender, EventArgs e)
{
    Mesaj("Mesaj", "Anonymous Metod Örneği");
}
```

Kodumuzda **`Mesaj`** adlı metoda parametre olarak **`Del_Metod`** isimli **`delegate`**'i yolluyoruz. **`Del_Metod`** isimli metodu yazmak yerine, metodumuzu **anonim metot** olarak yazarak kod satırlarımızı kısaltıyoruz.

Lambda Operatörü

Anonymous metotların **delegate** kullanılarak yapılan işlemleri kısalttığından bahsetmiştik. Lambda operatörü de Anonymous metotlar ile yapılan işlemleri **basitleştirir** ve **kısaltır**. Lambda operatörünün kullanıldığı metotlar **static** olarak tanımlanmak zorundadır. Lambda operatörü **=>** şeklinde gösterilir. Geriye **bool** türünden sonuç döndürür. Örnek olarak **delegate** konusunda yaptığımız **toplam** ve **fark** örneğini önce anonymous metot şeklinde yazıp, daha sonra lambda operatörü kullanarak yeniden yazalım.Anonymous metot şeklinde yazımı aşağıdaki gibidir:

```
public delegate void Del_Metod(int sayi1, int sayi2);
Del_Metod Mesaj = delegate(int sayi1, int sayi2)
{
    MessageBox.Show((sayi1 + sayi2).ToString(),"Sonuç");
};
private void btnMesaj_Click(object sender, EventArgs e)
{
    Mesaj(5,10);
}
```

Kodumuzda **Mesaj** isimli metoda parametre olarak, **Del_Metod** isimli **delegate**'i anaonim metot olarak yazıp yolladık. Aynı kodu Lambda operatörü kullanarak yeniden yazalım:

```
delegate int Del_Metod(int sayi1, int sayi2);
private void btnMesaj_Click(object sender, EventArgs e)
{
    Del_Metod sonuc = (sayi1, sayi2) => sayi1 + sayi2;
    MessageBox.Show(sonuc.Invoke(5, 10).ToString());
}
```

Kodumuzda:

```
Del_Metod sonuc = (sayi1, sayi2) => sayi1 + sayi2;
```

Satırında **sonuc** isimli metoda parametre olarak **sayi1** ve **sayi2** değişkenlerini aktardık. Daha sonra bu parametrelere değer olarak gelen **sayi1** ve **sayi2** değişkenlerinin **+** operatörü kullanılarak toplanacağını belirttik.**Invoke** metodunu kullanarak da **sonuc** metodunun çalıştırılmasını sağlıyoruz. Görüldüğü gibi Lambda operatörünü kullandığımızda kodumuz daha **kısa** ve **anlaşılır** olmaktadır.

Action Delegate

Action, bir delegate türüdür. Geriye herhangi bir değer döndürmez. **Delegate tanımı**; delegate'den nesne oluşturma ve delegate çağırma adımlarını kısaltarak kod satırlarımızı azaltır.

Örnek olarak daha önce delegates konusunda yapmış olduğumuz mesaj görüntüleme örneğini **action** ile yeni baştan yazalım. Kodumuzun ilk hâli aşağıdaki gibidir:

```
public void DelegateMetod(string mesaj, string baslık)
{
    MessageBox.Show(baslık,mesaj);
}
public delegate void Del_Metod(string mesaj, string baslık);
private void btnMesaj_Click(object sender, EventArgs e)
{
    Del_Metod event_handler = DelegateMetod;
    event_handler("Mesaj","Delegate Örneği");
}
```

Bu kodun **action** kullanarak yazılmış yeni hâli aşağıdaki gibi olacaktır:

```
public void DelegateMetod(string mesaj, string baslık)
{
    MessageBox.Show(baslık,mesaj);
}
private void btnMesaj_Click(object sender, EventArgs e)
{
    Action<string, string> actionMesaj = delegate (string mesaj,
string baslik) { DelegateMetod("Mesaj", "Action Örneği"); };
    actionMesaj("Mesaj", "Action Örneği");
}
```

Görüldüğü gibi **delegate** tanımı yapmadan, **delegate** metodumuza değer göndererek çağırdık.

Klavyeden girilen operatöre göre işlem yapan bir uygulamayı **action delegate** ile yazalım.

```
public static void Topla(int s1, int s2)
{
    Console.WriteLine("Toplam=" + (s1 + s2));
}
public static void Cikar (int s1, int s2)
{
    Console.WriteLine("Fark=" + (s1 - s2));
}
public static Action<int, int> actionSonuc;
static void Main(string[] args)
{
    Console.WriteLine("Operatörü giriniz");
    string opt = Console.ReadLine();
    if (opt == "+")
        actionSonuc = delegate (int sayi1, int sayi2) { Topla(sayi1,
      sayi2); };
    else if (opt == "-")
        actionSonuc = delegate (int sayi1, int sayi2) { Cikar(sayi1,
sayi2); };
    actionSonuc(5, 3);
    Console.ReadLine();
}
```

Kodumuzda girilen operatör "**+**" ise **actionSonuc** isimli **action delegate Topla** metodunu, girilen operatör "**-**" ise **actionSonuc** isimli **action delegate Cikar** metodunu çağırmaktadır. Görüldüğü gibi **Topla** ve **Cikar** isimli iki farklı metodu **actionSonuc** isimli tek bir **action delegate** ile çalıştırdık.

Lambda operatörünü kullanarak **action delegate** metotlarını daha da kısaltabiliriz. Örneğin yukarıda yaptığımız operatöre göre işlem yapan kodumuzu lambda operatörünü kullanarak yeniden yazalım.

```
public static Action<int, int> actionSonuc;
static void Main(string[] args)
{
    Console.WriteLine("Operatörü giriniz");
    string opt = Console.ReadLine();
    if (opt == "+")
        actionSonuc = (sayi1, sayi2) => Console.WriteLine(sayi1 +
sayi2);
    else if (opt == "-")
        actionSonuc = (sayi1, sayi2) => Console.WriteLine(sayi1 -
sayi2);
    actionSonuc(5, 3);
    Console.ReadLine();
}
```

Kodumuzda **actionSonuc** isimli action delegate'e parametre olarak **sayi1** ve **sayi2** değişkenlerini aktardık. Daha sonra bu parametrelere değer olarak gelen **sayi1** ve **sayi2** değişkenlerinin girilen operatöre göre toplanacağını ya da çıkartılacağını belirttik.

Func Delegate

Func da Action'a benzer yapısı olan bir delegate türüdür. Ancak Action'dan farklı olarak **geriye** değer döndürür. İki farklı şekilde kullanılabilir.

```
public delegate TResult Func<out TResult>()
```

kullanımında parametre almadan sadece geriye değer döndürürken,

```
public delegate TResult Func<in T, out TResult>(T arg)
```

kullanımında ise hem parametre alıp hem de geriye değer döndürür.

```
public static string Ortalama(int s1, int s2)
{
    string sonuc = "";
    int ort = (s1 + s2)/2;
    Console.WriteLine("Ortalama=" + ort);
    if (ort >= 50)
        sonuc = "Geçti";
    else
        sonuc = "Kaldı";
    return sonuc;
}
static void Main(string[] args)
{
    Func<int, int,string> funcSonuc=Ortalama;
    Console.WriteLine(funcSonuc(80, 80));
    Console.ReadLine();
}
```

Örnek uygulamamızda Func delegate, **`int`** türde **`s1`** ve **`s2`** parametrelerini alıp, girilen sayıların ortalamasını hesaplayarak işlem sonucunu geriye **`string`** türde döndürür.

Predicate Delegate

Kullanımı **Func delegate**'e benzer. Parametre alır ve geriye bir değer döndürür. Ancak farklı olarak geriye yalnızca **boolean** türde bir değer döndürür.

```
public static bool Cift(int sayi)
{
    return (sayi%2==0);
}
static void Main(string[] args)
{
    Predicate<int> predicateSonuc=Cift;
    Console.WriteLine(predicateSonuc(4));
    Console.ReadLine();
}
```

Kodumuzda parametre olarak aldığı **int** türdeki değerin çift sayı olup olmadığı kontrol edilmektedir. Geriye **boolean** türde bir değer dönmektedir.

Local Function

Local Function, **C# 7** versiyonundan itibaren gelen yeniliklerden biridir. Haricî bir metot ya da sınıf içerisinden erişilmeyecek metotlarımızı herhangi bir metodun içerisinde tanımlayabiliriz. Bu şekildeki bir metot, ancak tanımlandığı metodun içerisinde kullanılacaktır.

```
static void Main(string[] args)
{
    bool Cift_LocalFunction(int sayi)
    {
        bool sonuc = false;
        if (sayi % 2 == 0)
            sonuc = true;
        else
            sonuc = false;
        return sonuc;
    }
    Console.WriteLine("Bir Sayı Giriniz");
    int okunandeger = Convert.ToInt32(Console.ReadLine());
    if (Cift_LocalFunction(okunandeger))
```

```
        Console.WriteLine("Sayı çift");
    else
        Console.WriteLine("Sayı tek");
    Console.ReadKey();
}
```

Örneğimizde **Cift_LocalFunction** isimli metodumuzu **Main** metodu içerisinde tanımladık. Dışarıdan metoda **int** türünden sayı gönderip sayının **çift** olup olmadığı bilgisini **boolean** türden geriye döndürdük.

Lambda operatörü kullanarak metodumuzu daha da kısaltabiliriz.

```
static void Main(string[] args)
{
    bool Cift_LocalFunction(int sayi) =>(sayi % 2 == 0);
    Console.WriteLine("Bir Sayı Giriniz");
    int okunandeger = Convert.ToInt32(Console.ReadLine());
    if (Cift_LocalFunction(okunandeger))
        Console.WriteLine("Sayı çift");
    else
        Console.WriteLine("Sayı tek");
    Console.ReadKey();
}
```

Yukarıdaki kodumuzda **Cift_LocalFunction** isimli metoda parametre olarak **sayi** değişkenini aktardık. Daha sonra lambda operatörüyle bu parametreye gelen değerin çift sayı olup olmadığını bularak geriye **boolean** türde değer döndürdük.

C# 8 versiyonundan itibaren local function'ları static erişim belirleyicisi ile kullanabiliriz. **`static`** olarak tanımlanan **yerel metotlar** bir üst metodun yerel değişkenlerine erişemez. Örnek olarak yukarıda tanımladığımız yerel metodu:

```
static void Main(string[] args)
{
    Console.WriteLine("Bir Sayı Giriniz");
    int okunandeger = Convert.ToInt32(Console.ReadLine());
    static bool Cift_LocalFunction() => (okunandeger % 2 == 0);
    if (Cift_LocalFunction())
      Console.WriteLine("Sayı çift");
    else
      Console.WriteLine("Sayı tek");
    Console.ReadKey();
}
```

`static` erişim belirleyicisi kullanarak yeniden yazalım. Bu şekildeki kullanımda "**`A static local function cannot contain a reference to 'okunandeger`**" şeklinde bir hata oluşarak **`Main`** metot içerisindeki **`okunandeger`** isimli değişkene, **`Cift_LocalFunction`** isimli yerel metot içerisinden erişim sağlanamayacaktır. Erişimi sağlamak için kodumuzu aşağıdaki gibi:

```
static void Main(string[] args)
{
    Console.WriteLine("Bir Sayı Giriniz");
    int okunandeger = Convert.ToInt32(Console.ReadLine());
    static bool Cift_LocalFunction(int okunandeger) =>
(okunandeger % 2 == 0);
    if (Cift_LocalFunction(okunandeger))
      Console.WriteLine("Sayı çift");
    else
      Console.WriteLine("Sayı tek");
    Console.ReadKey();
}
```

Şeklinde düzenlememiz gerekir.

Caller Info Attributes

Caller Info Attributes, optional parameters ile birlikte kullanılarak; uygulamamızın çalışma yolu, çağrıldığı satırın numarası ve çağıran üye adı bilgilerini almamızı sağlar. Bu bilgileri alarak uygulamalarımızda loglama yapabiliriz. **Caller Info Attributes**, System.Runtime.CompilerServices namespace'i altında yer almaktadır.

Caller Info Attributes ile 3 attribute (*nitelik*) bilgisine ulaşabilmekteyiz. Bunlar:

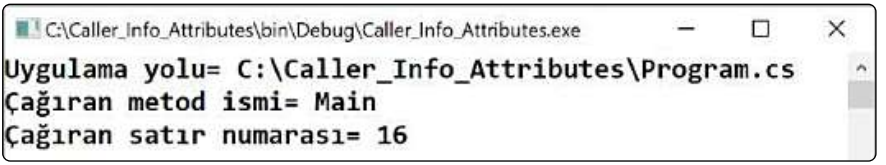
CallerMemberName: Çağıran üye adını belirtir.

CallerFilePath: Çağıran uygulamanın dosya yolunu belirtir.

CallerLineNumber: Çağrılan satır numarasını belirtir.

```
using System;
using System.Runtime.CompilerServices;
using System.Text;
namespace Caller_Info_Attributes
{
  class Program
  {
    static void bilgi([CallerMemberName] string metodismi = "",
  [CallerFilePath] string dosyayolu = "",
  [CallerLineNumber] int satırnumarası = 0)
    {
      Console.WriteLine("Uygulama yolu= " +dosyayolu);
      Console.WriteLine("Çağıran metod ismi= " +metodismi);
      Console.WriteLine("Çağıran satır numarası= "+satırnumarası);
    }
    static void Main(string[] args)
    {
      bilgi();
      Console.ReadLine();
    }
  }
}
```

Kodumuzda `bilgi` isimli metot içerisinden `CallerMemberName`, `CallerFilePath`, `CallerLineNumber` attribute bilgilerini elde ederek metodun çağrıldığı üye adı, dosya yolu ve satır numarası bilgilerini konsol ekranında görüntülüyoruz.



GENİŞLETME METODLARI (EXTENSION METHODS)

20

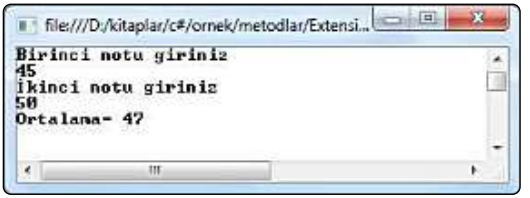
Genişletme metotları (*extension method*) kullanmakta olduğumuz metotlara yeni özellikler eklemek amacı ile kullanılırlar. Genişletme metodu tanımlarken aşağıdaki işlemleri yapmamız gerekmektedir:

- Genişletme metodu bir class içerisinde tanımlanırlar.
- Genişletme metotlarının içerisinde yer aldığı sınıf **`static`** olarak tanımlanmalıdır.
- Genişletme metodunun ilk parametresinin değişken tipi metodun bağlandığı değişken tipi ile aynı olmalıdır.
- Genişletme metodunun ilk parametresi **`this`** ifadesi ile tanımlanmalıdır.

Örnek olarak bir **`string`** değeri **`Int32`** türüne dönüştürmek için bir genişletme metodu yazalım.

```
using System;
using System.Text;
namespace Extension_Method_ToInt32
{
  static class Extensions
  {
    public static int ToInt32(this string deger)
    {
      return Convert.ToInt32(deger);
    }
  }
```

```
  class Program
  {
    static void Main(string[] args)
    {
      string sayi1,sayi2;
      Console.WriteLine("Birinci notu giriniz");
      sayi1 = Console.ReadLine();
      Console.WriteLine("İkinci notu giriniz");
      sayi2 = Console.ReadLine();
      Console.WriteLine("Ortalama= " + (sayi1.ToInt32() +
sayi2.ToInt32()) / 2);
      Console.ReadLine();
    }
  }
}
```



sayi1 ve **sayi2** değişkenlerinden sonra . (nokta) işaretini yazdığımızda genişletme metodunun **intellisense** menüsü içerisine eklendiğini görürüz.



Genişletme metodu çağırılırken genişletme metodu üzerine gelindiğinde (**extension**) yazılarak, çağırılan metodun genişletme metodu olarak belirtildiğini görürüz.

```
static class Extensions
{
  public static int ToInt32(this string deger)
  {
    return Convert.ToInt32(deger);
  }
}
```

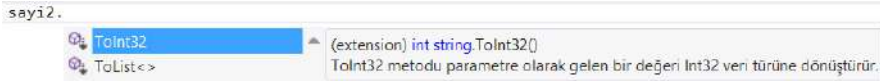
Satırlarında genişletme metodu içerisinde kullanacağımız fonksiyon ve parametreleri tanımlıyoruz. Burada tanımladığımız **deger** parametresinin tipi ile genişletme metodunun çağırıldığı **sayi1** ve **sayi2** değişkenlerinin aynı tipte olduğuna dikkat edelim.

Genişletme metodu tanımlanırken XML yorumlama satırları ile kullanıcılara bilgi verilebilir.

ToInt32 adlı genişletme metodu tanımlamasını aşağıdaki gibi düzenleyelim:

```
static class Extensions
{
  /// <summary>
  /// ToInt32 metodu parametre olarak gelen bir değeri
  /// Int32 veri türüne dönüştürür.
  /// </summary>
  public static int ToInt32(this string deger)
  {
    return Convert.ToInt32(deger);
  }
}
```

Genişletme metodu çağrılırken yazım esnasında, kullanıcıya bilgi verme amacı olarak yazdığımız açıklama satırları görüntülenecektir:



Genişletme metotlarını namespace içerisinde tanımlayıp **using** ifadesi ile namespace adı ve genişletme metodu isimlerini proje içerisine dâhil ederek kullanabiliriz.

Bir konsol projesi açarak içerisine **Extension_Class** adıyla bir **Class** ekleyelim. **Class** içerisine aşağıdaki kodları yazalım:

```
using System;
using System.Text;
namespace Extension
{
  public static class ExMetod_Int32
```

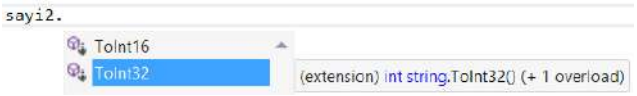
```
  {
    public static int ToInt32(this string deger)
    {
      return Convert.ToInt32(deger);
    }
  }
  public static class ExMetod_Int16
  {
    public static int ToInt16(this string deger)
    {
      return Convert.ToInt16(deger);
    }
  }
}
```

Class içerisinde **`Extension`** adlı bir namespace tanımlayarak **`ToInt32`** ve **`ToInt64`** adlı iki adet genişletme metodu tanımladık.

Genişletme metotlarını çağıracağımız kod satırlarını **`Program.cs`** içerisinde yazalım:

```
using System;
using System.Text;
using Extension;
class Program
{
  static void Main(string[] args)
  {
    string sayi1, sayi2;
    Console.WriteLine("Birinci notu giriniz");
    sayi1 = Console.ReadLine();
    Console.WriteLine("İkinci notu giriniz");
    sayi2 = Console.ReadLine();
    Console.WriteLine("Ortalama= " +
      (sayi1.ToInt32() + sayi2.ToInt32()) / 2);
    Console.ReadLine();
  }
}
```

using komutu ile **Extension** adlı namespace ismini yazarak **Extension_Class** isimli Class içerisinde yer alan **Extension** adlı namespace'e erişiyoruz. Bu şekilde **Extension** namespace'i içerisinde yer alan **ToInt16** ve **ToInt32** adlı genişletme metodlarına erişebiliriz.



Genişletme Metotlarının Aşırı Yüklenmesi (Overloading)

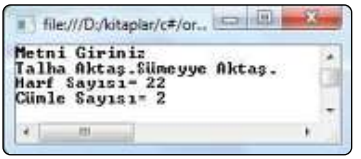
Genişletme metotları **aşırı yüklenme** (*overloading*) işlemi ile metotlar ek parametreler yazılarak birden fazla tanımlanabilirler. Framework 3.5 versiyonu ile birlikte kullanılmaya başlanmıştır.

Örnek olarak bir metnin uzunluğunu bulan **Length** metodunun işlevini gerçekleştiren bir genişletme metodu yazalım ve bu metot üzerinde overloading işlemi uygulayalım.

```
using System;
using System.Text;
namespace Uzunluk_Extension
{
  static class Extensions
  {
    /// <summary>
    /// Parametre olarak girilen içeriğin
    /// uzunluğunu geriye döndürür
    /// </summary>
    public static int uzunluk(this string metin)
    {
      int harfsayisi=0,noktasayisi, bosluksayisi;
      noktasayisi = metin.Split('.').Length-1;
      bosluksayisi = metin.Split(' ').Length-1;
      harfsayisi = metin.Length - (bosluksayisi + noktasayisi);
      return (harfsayisi);
    }
```

```
    /// <summary>
    /// Parametre olarak girilen içeriğin
    /// cümle sayısını geriye döndürür
    /// </summary>
    public static int uzunluk(this string metin,bool cumlesay)
    {
       return (metin.Split('.').Length-1);
    }
  }
}

namespace Uzunluk_Extension_Kullanım
{
  using Uzunluk_Extension;
  class Program
  {
    static void Main(string[] args)
    {
      string metin;
      Console.WriteLine("Metni Giriniz");
      metin = Console.ReadLine();
      Console.WriteLine("Harf Sayısı= " + metin.uzunluk());
      Console.WriteLine("Cümle Sayısı= " + metin.uzunluk(true));
      Console.ReadLine();
    }
  }
}
```



Uzunluk adlı genişletme metodumuz ilk tanımlamasında…

```
public static int uzunluk(this string metin)
```

… şeklinde yalnızca "*metin*" isimli tanımlayıcı parametresi ile tanımlanarak belirtilen metnin karakter uzunluğunu bulmaktadır. Metnin karakter uzunluğunu bulurken boşluk sayısını ve nokta sayısını çıkartarak yalnızca harf sayısının görüntülenmesini sağlıyoruz.

Uzunluk adlı genişletme metodumuzun ikinci tanımlama biçiminde ise…

```
public static int uzunluk(this string metin, bool cumlesay)
```

… "*cumlesay*" şeklinde farklı isim ve farklı türde bir parametre kullanarak bu parametre **true** değerini aldığında cümle sayısı geriye döndürülmektedir.

Extension metotlarımızı **Uzunluk_Extension** isimli namespace içerisinde tanımlıyoruz. **Extension** metodu çağırdığımız kod satırlarını ise **Uzunluk_Extension_Kullanım** isimli namespace içerisinde yazıyoruz.

```
using Uzunluk_Extension
```

… satırı ile **Uzunluk_Extension** isimli namespace'i **Uzunluk_Extension_Kullanım** isimli namespace içerisinden çağırarak **Extension** metotların kullanılabilmesini sağlıyoruz.

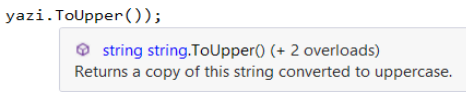
C#.Net içerisinde mevcut olan metotlardan birinin ismi ile aynı isme sahip bir genişletme metodu yazdığımızda, yazdığımız genişletme metodu yerine var olan metot görüntülenecektir.

Örneğin; bir string'i büyük harfe çeviren **ToUpper** metodu ile aynı isimde bir genişletme metodu yazalım.

```
using System;
using System.Text;
namespace Extension
{
  static class Extensions
  {
    /// <summary>
    /// ToUpper metodu parametre olarak gelen bir değeri
    /// büyük harfe dönüştürür.
    /// </summary>
    public static string ToUpper(this string icerik)
    {
      return icerik.ToUpper();
    }
  }
}
```

```
namespace Extension_Method_ToUpper
{
  using Extension;
  class Program
  {
    static void Main(string[] args)
    {
      string yazi;
      Console.WriteLine("Yazıyı giriniz");
      yazi = Console.ReadLine();
      Console.WriteLine("Büyük harfe çevrilmiş hali= " +
            yazi.ToUpper());
      Console.ReadLine();
    }
  }
}
```

Genişletme metodunu çağırırken metodun ismini yazdığımızda genişletme metodunun değil de C#.Net içerisinde zaten mevcut olan **`ToUpper`** metodunun çağrıldığını görürüz.



Ancak bu durum, C#.Net metotlarının isimlerini genişletme metodu olarak kullanamayacağımız anlamına gelmiyor. Overloading işlemi ile var olan C#.Net metotlarını genişletme metodu olarak kullanabiliriz.

`ToUpper` metodu üzerinde overloading işlemi uygulayalım.

```
using System;
using System.Text;
namespace Extension
{
  static class Extensions
  {
    /// <summary>
    /// ToUpper metodu parametre olarak gelen bir değeri
    /// büyük harfe dönüştürür.
    /// </summary>
    public static string ToUpper(this string icerik)
```

```
    {
      return icerik.ToUpper();
    }

    /// <summary>
    /// ToUpper metodu parametre olarak gelen bir değerin
    /// kelimelerinin ilk harflerini büyük harfe dönüştürür.
    /// </summary>
    public static string ToUpper(this string icerik, bool ilkharfbuyuk)
    {
      bool bosluk = true;
      StringBuilder metin = new StringBuilder(icerik.Length);
      foreach(Char karakter in icerik)
      {
        if (bosluk & Char.IsLetter(karakter))
          metin.Append(Char.ToUpper(karakter));
        else
          metin.Append(karakter);
        bosluk = Char.IsWhiteSpace(karakter);
      }
      if (ilkharfbuyuk == true)
        return (metin.ToString());
      else
        return icerik.ToUpper();
    }
  }
}
namespace Extension_Method_ToUpper
{
  using Extension;
  class Program
  {
    static void Main(string[] args)
    {
      string yazi;
      Console.WriteLine("Yazıyı giriniz");
      yazi = Console.ReadLine();
      Console.WriteLine("Büyük harfe çevrilmiş hali= " +
              yazi.ToUpper());
      Console.WriteLine("Kelimelerin ilk harfleri büyük hali= " +
              yazi.ToUpper(true));
      Console.ReadLine();
    }
  }
}
```



`ToUpper` metodunu genişletme metodu olarak tanımlayıp **`ilkharfbuyuk`** adlı **`boolean`** bir parametre ekleyerek overloading işlemi uyguladık.

Genişletme metodunu çağırdığımızda metodun ismini yazdığımızda bu defa overloading işlemi uyguladığımız genişletme metodunun çağrıldığını görürüz.

```
yazi.ToUpper(true));
```

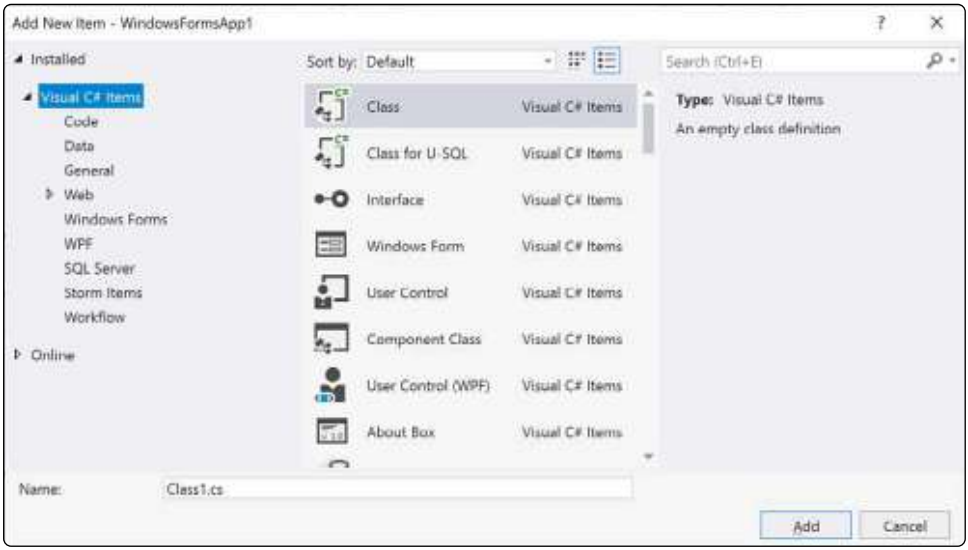
(extension) string string.ToUpper(bool ilkharfbuyuk) (+ 3 overloads)
ToUpper metodu parametre olarak gelen bir değerin kelimelerinin ilk harflerini büyük harfe dönüştürür.

CLASS (SINIF) YAPISI

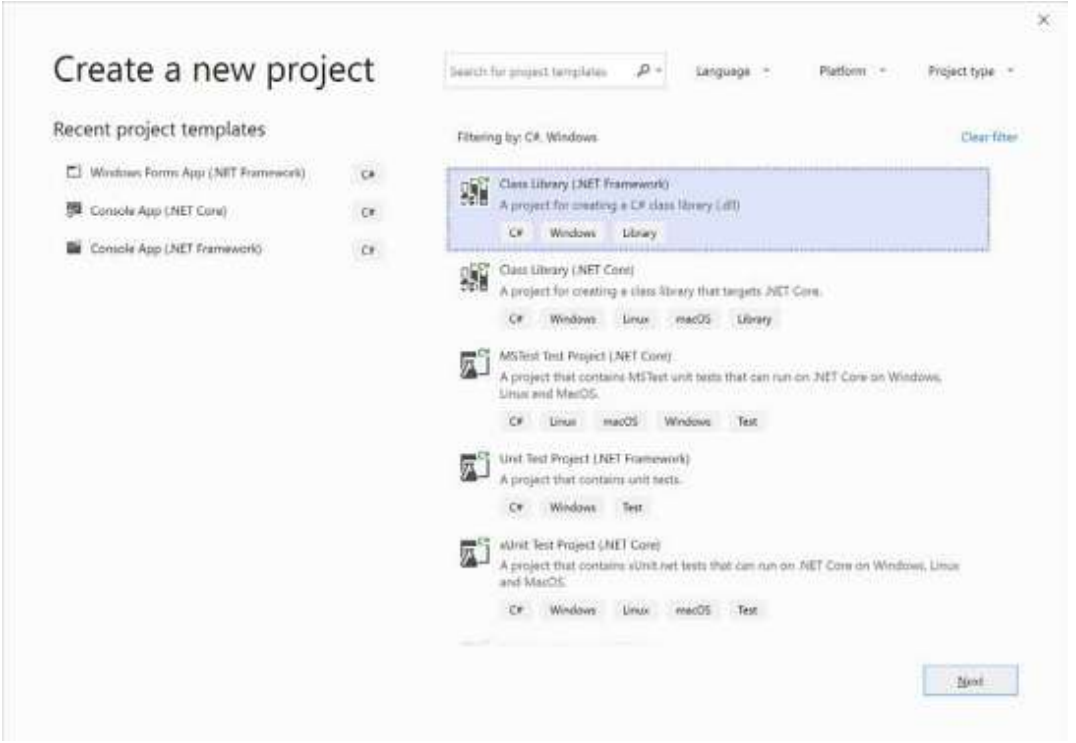
21

Programlarımız içerisinde sıklıkla kullandığımız metotları ya da değişkenleri bir çatı altında toplayarak projelerimiz içerisinde tekrar yazmaya gerek kalmadan kullanmak amacı ile bu tip tanımlamaları **Class** yapıları içerisinde tanımlayabiliriz. Class yapılarını, derlemeden tanımlandığı projeler içerisinde kullanabileceğimiz gibi derleyip **dll** dosyası hâline getirerek başka projeler içerisine referans olarak ekleyip kullanmamız da mümkündür.

Eğer bir proje içerisine yeni bir class eklenecekse **Project** menüsünden **Add Class** seçeneği ile ya da **Project** menüsünden **Add New Item** seçeneği tıklanıp gelen pencereden **Class** seçeneği tıklanarak ekleme işlemi yapılabilir.



Eğer Class projemizi ayrı bir dll dosyası şeklinde derleyeceksek "**File/ New Project**" seçeneği ile gelen "**Create a new Project**" penceresindeki proje şablonları içerisinden "**Class Library (.NET Framework)**" şablonunu seçmeliyiz.



Class yapısı içerisinde belli bir işlemi gerçekleştiren kod yapılarına **metot** adı verilir.

Örnek olarak kendisine parametre şeklinde gönderilecek olan formu kapatacak olan bir **`class`** tasarlayalım. Yeni bir windows form projesi açıp içerisine "**`snf_form_kapat`**" adı ile bir **`class`** ekliyoruz. Class içerisine şu kodları ekliyoruz;

```
using System;
using System.Text;
using System.Windows.Forms;
namespace form_kapat_class
{
    class snf_form_kapat
```

```
    {
     public void mtd_kapat(Form frm)
     {
        DialogResult cevap= MessageBox.Show("Çıkmak İster misiniz",
                "Çıkış", MessageBoxButtons.YesNo);
        if (cevap == DialogResult.Yes)
                frm.Close();
     }

    }
}
```

Class içerisinde **mtd_kapat** adı ile bir metot tanımlıyoruz. Bu metodumuz parametre olarak **frm** parametresini form tipinden alıyor.

Formumuzun içerisine ise şu kod satırlarını yazıyoruz:

```
private void btnKapat_Click(object sender, EventArgs e)
{
    snf_form_kapat nsn_kapat = new snf_form_kapat();
    nsn_kapat.mtd_kapat(this);
}
```

Hazırladığımız Class içerisindeki metodu çağırmak için, metodumuzu **Public** olarak tanımladığımızdan dolayı, "**nsn_kapat**" adlı bir nesneyi "**snf_form_kapat**" adlı Class türünden tanımlıyoruz. Daha sonra:

```
nsn_kapat.mtd_kapat(this)
```

Satırı ile **nsn_kapat** nesnesinin **mtd_kapat** adlı metodunu formumuzun adını göndererek çalıştırıyoruz.

Eğer metoda direkt olarak sınıfın ismini kullanarak erişmek istersek metodu **static** olarak tanımlamamız gerekir. static olarak tanımlanan metotlar sınıflardan türetilen nesnelerden **bağımsız** hareket ederler. Dolayısı ile static metotları içerisinde tanımlandıkları class'dan yeni bir nesne oluşturmadan direkt olarak "**sınıf ismi.metod ismi**" şeklinde kullanabiliriz. Buna göre sınıfımızı ve metodumuzu yeniden düzenleyelim.

"**snf_form_kapat**" isimli **Class** kodu:

```
using System;
using System.Text;
using System.Windows.Forms;
namespace form_kapat_class
{
    class snf_form_kapat
    {
     public static void mtd_kapat(Form frm)
     {
        DialogResult cevap= MessageBox.Show("Çıkmak İster misiniz",
                "Çıkış", MessageBoxButtons.YesNo);
        if (cevap == DialogResult.Yes)
                frm.Close();
      }

    }
}
```

Formumuzun kodu:

```
private void btnKapat_Click(object sender, EventArgs e)
{
    snf_form_kapat.mtd_kapat(this);
}
```

Sınıflardan nesne türetilirken **new** ifadesi ile çalıştırılan ve sınıf ile aynı ismi taşıyan metotlara **constructor method** (*yapıcı metod*) adı verilir.

```
snf_form_kapat nsn_kapat = new snf_form_kapat();
```

Örneğin yukarıda yazdığımız biçimde **snf_form_kapat** isimli sınıfımız ile aynı ismi taşıyan **snf_form_kapat( )** isimli yapıcı metodumuzu çalıştırdık. Eğer kullanıcı hazırladığı sınıf içerisinde yapıcı metodu kendisi oluşturmazsa **.Net** tarafından yapıcı metot **otomatik** olarak oluşturulur. Yapıcı metotlar varsayılan olarak parametresiz tanımlanırlar. Ancak eğer kullanıcı isterse parametreler tanımlayabilir. Parametrelere değer ataması yapılmazsa **.Net** tarafından varsayılan ilk değer atamaları yapılır. Sayısal değerlere **0** atanması gibi…

Snf_form_kapat isimli sınıfımızı, aşağıdaki gibi yapıcı metodu biz yazarak da tanımlayabiliriz.

```
using System;
using System.Text;
using System.Windows.Forms;
namespace form_kapat_class
{
    class snf_form_kapat
    {
     public snf_form_kapat()
     {
     }
     public void mtd_kapat(Form frm)
     {
        DialogResult cevap= MessageBox.Show("Çıkmak İster misiniz",
                "Çıkış", MessageBoxButtons.YesNo);
        if (cevap == DialogResult.Yes)
                frm.Close();
      }

    }
}
```

Yukarıdaki **class** kodumuzda:

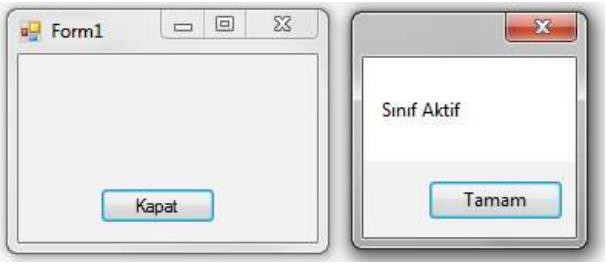
```
public snf_form_kapat()
{
}
```

snf_form_kapat isimli yapıcı metodumuzu yazarak:

Yapıcı metot içerisine eklediğimiz kod satırları sınıf çağrıldığında öncelikli olarak çalışacaktır.

```
using System;
using System.Text;
using System.Windows.Forms;
namespace form_kapat_class
```

```
{
    class snf_form_kapat
    {
     public snf_form_kapat()
     {
         MessageBox.Show("Sınıf Aktif");
     }
     public void mtd_kapat(Form frm)
     {
        DialogResult cevap= MessageBox.Show("Çıkmak İster misiniz",
                "Çıkış", MessageBoxButtons.YesNo);
        if (cevap == DialogResult.Yes)
                frm.Close();
      }
    }
}
```



Yukarıda görüldüğü gibi yapıcı metot içerisine yazdığımız mesaj penceresi görüntüleme kodu, sınıf çağrıldığında öncelikli olarak çalışmıştır.

Constructor metotların yanında sınıftan türettiğimiz nesne, hafızadan silinirken çalışan **destructor** (*yıkıcı*) metotlar da mevcuttur. Yıkıcı metotlar, metot isminin başına ~ karakteri eklenerek tanımlanırlar.

```
using System;
using System.Text;
using System.Windows.Forms;
namespace form_kapat_class
{
  class snf_form_kapat
  {
    public snf_form_kapat()
```

```
    {
       MessageBox.Show("Sınıf Aktif");
    }
    ~snf_form_kapat()
    {
       MessageBox.Show("Sınıf Pasif");
    }
    public void mtd_kapat(Form frm)
    {
       DialogResult cevap= MessageBox.Show("Çıkmak İster misiniz",
              "Çıkış", MessageBoxButtons.YesNo);
       if (cevap == DialogResult.Yes)
              frm.Close();
    }
  }
}
```



Yukarıdaki örneğimizde, yıkıcı metodumuz içerisine yazdığımız kod ile çağrılan sınıfın kullanımı bittikten sonra, ekrana "**Sınıf Pasif**" şeklinde bir mesaj penceresi görüntülenmektedir.

Yapıcı ve **yıkıcı** metotlarımızın çağırıldığı form kodu aşağıdaki gibi olacak:

```
private void btnKapat_Click(object sender, EventArgs e)
{
    snf_form_kapat nsn_kapat = new snf_form_kapat();
    nsn_kapat.mtd_kapat(this);
}
```

Diğer metotlarda olduğu gibi **yapıcı** (***constructor***) metotlarımızda da parametreleri kullanabiliriz.

```
class Meslekler
{
    private string meslek;
    public Meslekler(string meslek)
    {
        this.meslek = meslek;
    }
    public string Tanim()
    {
        return "Mesleği= " + Meslek;
    }

    public string Meslek
    {
        get { return meslek; }
        set { meslek = value; }
    }
}
class Program
{
    static void Main(string[] args)
    {
        Meslekler m;

        m = new Meslekler("Öğretmen");
        Console.WriteLine(m.Tanim());

        m = new Meslekler("Mühendis");
        Console.WriteLine(m.Tanim());
        Console.ReadLine();
    }
}
```

Örneğimizde **`Meslekler`** sınıfını, aynı isimli yapıcı metodu ile birlikte oluşturduk. Yapıcı metodumuzu **`meslek`** parametresini dışarıdan alacak biçimde tanımladık. **`Meslekler`** nesnesi yalnızca bir meslek ile oluşturulabilir. Tanim metodu ise seçili olan meslek değerini döndürür.

Main metot içerisinde ise yapıcı metodumuza sırasıyla "**Öğretmen**" ve "**Mühendis**" değerlerini göndererek konsol ekranına yazdırdık.

Primary Constructors

C# 6.0 ile birlikte kullanılmaya başlanan Primary Constructor sayesinde yapıcı metodumuzda tanımladığımız parametreleri, artık **class** tanımlaması sırasında yazabilmekteyiz.

Örneğin yukarıdaki örneğimizde:

```
class Meslekler
{
    public Meslekler(string meslek)
    {
        this.meslek = meslek;
    }
}
```

Şeklinde yazdığımız kod satırlarını Primary Constructor ile birlikte artık:

```
class Meslekler(string meslek)
```

şeklinde yazabilmekteyiz.

Buna göre örneğimizin son hâli aşağıdaki gibi olacaktır:

```
class Meslekler(string meslek)
{
    string _meslek = meslek;
    public string Tanim()
    {
        return "Mesleği= " + Meslek;
    }
    public string Meslek
    {
        get { return _meslek; }
        set { _meslek = value; }
    }
}
class Program
```

```
{
    static void Main(string[] args)
    {
        Meslekler m;

        m = new Meslekler("Öğretmen");
        Console.WriteLine(m.Tanim());

        m = new Meslekler("Mühendis");
        Console.WriteLine(m.Tanim());

        Console.ReadLine();
    }
}
```

Primary Constructor kullanırken sınıf tanımlaması sırasında yazılan parametrelere direkt olarak erişemiyoruz. Bunun için yeni bir değişken tanımlayarak yapıcı metot parametrelerine erişebilmekteyiz.

Örneğimizde:

```
string _meslek = meslek;
```

Satırında olduğu gibi meslek parametresini **`_meslek`** değişkeni şeklinde yeniden tanımlayarak yapıcı metot parametresi olan meslek parametresine erişim sağladık.

Parametre Dizisi

Class yapısı içerisindeki metotlarımızda da **`class`** dışarısında tanımladığımız metotlarda olduğu gibi sayısı belli olmayan parametre kullanma durumlarında, parametre dizisi tanımlamamız mümkündür. Parametre dizisini tanımlarken "**`params`**" anahtar sözcüğünü kullanarak tanımlama yaparız.

Örnek olarak form üzerindeki butonları **aktif** ve **pasif** yapacak bir **`class`** hazırlayalım.

Yeni bir **Windows Form** projesi açarak projemiz içerisine "**snf_aktif_pasif**" adı ile yeni bir **class** ekliyoruz. **Class**'ımızın kodu şu şekilde olacak:

```
using System;
using System.Text;
using System.Windows.Forms;
namespace Params_Class
{
    public class snf_aktif_pasif
    {
        byte i;
        public void mtd_aktif(params Button[] btn)
        {
            for (i = 0; i <= btn.Length - 1; i++)
            {
                btn[i].Enabled = true;
            }
        }
        public void mtd_pasif(params Button[] btn)
        {
            for (i = 0; i <= btn.Length - 1; i++)
            {
                btn[i].Enabled = false;
            }
        }
    }

}
```

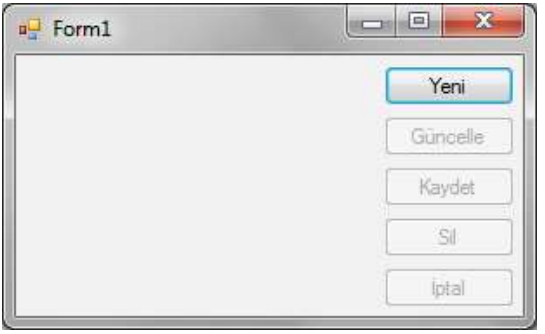
mtd_aktif ve **mtd_pasif** adlı iki metot tanımlayarak bu metotlara **btn** adlı parametre dizisi ile aktif ya da pasif yapılacak butonların isimlerini gönderiyoruz.

Formun kodu:

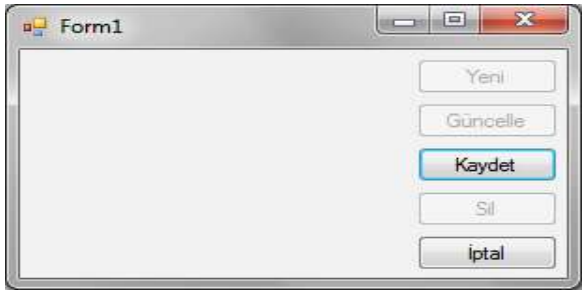
```
using System;
using System.Windows.Forms;
namespace Params_Class
{
  public partial class Form1 : Form
```

```
  {
      public Form1()
      {
          InitializeComponent();
      }
      snf_aktif_pasif nsn_aktif_pasif = new snf_aktif_pasif();
      private void Form1_Load(object sender, EventArgs e)
      {
          nsn_aktif_pasif.mtd_aktif(btnYeni);
          nsn_aktif_pasif.mtd_pasif(btnGuncelle, btniptal,
btnKaydet, btnSil);
      }
      private void btnYeni_Click(object sender, EventArgs e)
      {
          nsn_aktif_pasif.mtd_aktif(btnKaydet, btniptal);
          nsn_aktif_pasif.mtd_pasif(btnYeni, btnGuncelle,
btnSil);
      }
  }
}
```

Formun **`Load`** olayında **`mtd_aktif`** isimli metoda **`btnYeni`** isimli butonu parametre olarak, **`mtd_pasif`** isimli butona da diğer butonları parametre olarak göndererek **Yeni** butonunun **aktif**, diğer butonların ise **pasif** olmasını sağlıyoruz.



Yeni butonuna tıklandığında ise `mtd_aktif` isimli metoda `btnKaydet` ve `btniptal` isimli butonları parametre olarak, `mtd_pasif` isimli butona da diğer butonları parametre olarak göndererek **Kaydet** ve **İptal** butonlarının **aktif**, diğer butonların ise **pasif** olmasını sağlıyoruz.



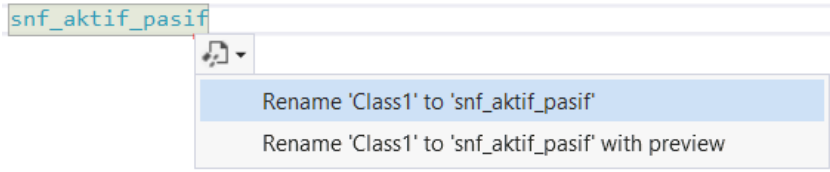
Hazırlamış olduğumuz bu `class`'ı **dll** dosyası şeklinde oluşturacak bir proje yazalım.

"`aktif_pasif_dll`" adı ile yeni bir **Class Library** proje oluşturuyoruz. Projemiz içerisinde butonları **aktif** ya da **pasif** yapacağımız için **System.Windows.Forms dll** dosyasını **referans** olarak ekliyoruz.

```
using System.Windows.Forms;
```

Satırı ile bu **dll** dosyasını projemize dâhil ediyoruz.

`Class` ismini daha anlaşılır olması için "`snf_aktif_pasif`" şeklinde değiştiriyoruz.

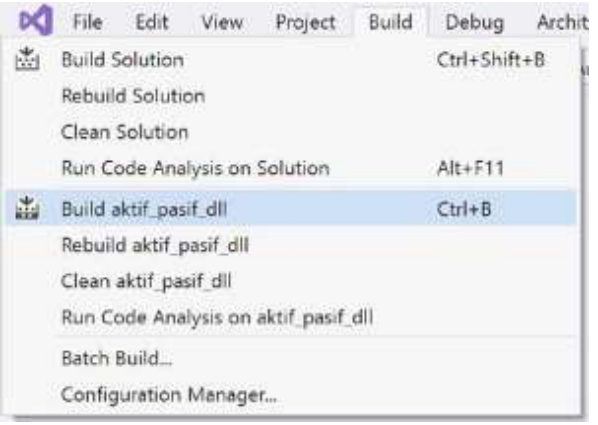


`Class` kodumuz şu şekilde olacak:

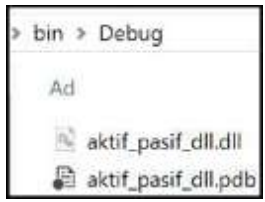
```
using System;
using System.Text;
using System.Windows.Forms;
namespace aktif_pasif_dll
```

```
{
    public class snf_aktif_pasif
    {
        byte i;
        public void mtd_aktif(params Button[] btn)
        {
            for (i = 0; i <= btn.Length - 1; i++)
                btn[i].Enabled = true;
        }
        public void mtd_pasif(params Button[] btn)
        {
            for (i = 0; i <= btn.Length - 1; i++)
                btn[i].Enabled = false;
        }
    }
}
```

Class'ımızı **Build** menüsünden **Build aktif_pasif_dll** seçeneği ile derleyip **dll** dosyası hâline getiriyoruz.

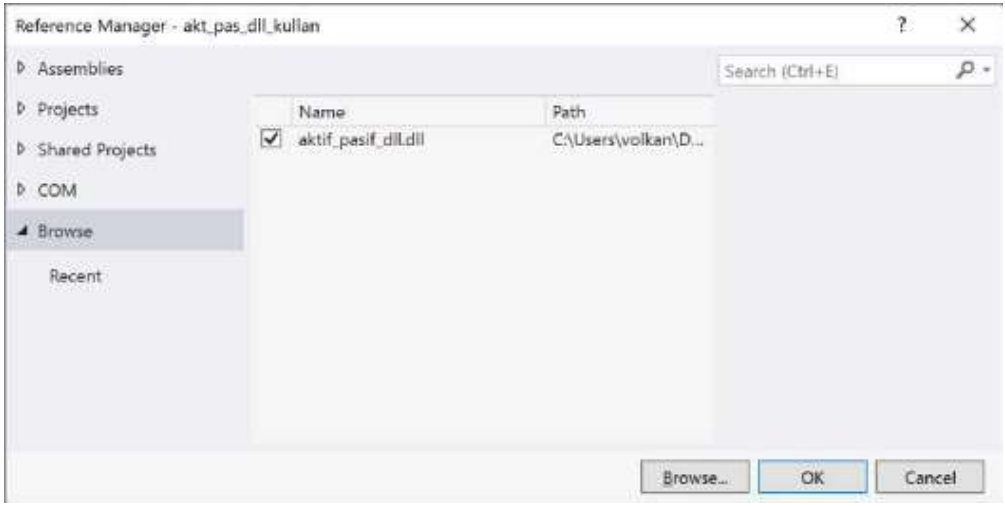


Dll dosyası hatasız olarak derlendiğinde **Visual Studio IDE** penceresinin sol alt köşesinde "**Build succeed**" şeklinde bir mesaj görüntülenir.

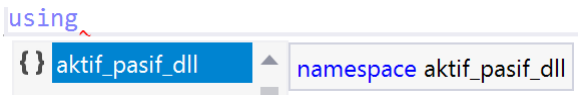


Dll dosyasına projemizin **Bin** klasörü içerisindeki **Debug** klasöründen erişebiliriz.

Bu dll dosyasını kullanabilmek için yeni bir **Windows Form** projesi oluşturalım. Bu projenin Bin/Debug klasörüne oluşturduğumuz dll dosyasını kopyalayarak, bunu **Project**/**Add Reference** menüsünden ya da Solution Explorer'da proje ismine **sağ** tıklayarak Add Reference seçeneği ile gelen penceredeki **Browse** sekmesinden **referans** olarak ekleyelim.



Bu referans dosyasını `using` ile projemize dâhil edelim.



`using` yazıp **tab** tuşuna bastığımızda dll dosyamızın kullanıma hazır olduğunu görürüz.

Formumuzun kodu aşağıdaki gibi olacak:

```
using System;
using System.Windows.Forms;
using aktif_pasif_dll;
namespace akt_pas_dll_kullan
{
    public partial class Form1 : Form
    {
        public Form1()
        {
          InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
          snf_aktif_pasif nsn_akt_pas = new snf_aktif_pasif();
          nsn_akt_pas.mtd_aktif(btnYeni);
          nsn_akt_pas.mtd_pasif(btnGuncelle, btniptal,
btnKaydet, btnSil);
        }
    }
}
```

"**nsn_akt_pas**" değişkenimizi "**snf_aktif_pasif**" sınıfından tanımlayarak bu sınıfın "**mtd_aktif**" ve "**mtd _pasif**" adlı metotlarına erişimi sağladık.

XML Yorum Satırı Eklemek

Oluşturduğumuz bir **dll** uzantılı class dosyamızın derlendiği klasöre baktığımızda, bu klasörde "**dll_dosya_adı.xml**" şeklinde bir dosyanın da mevcut olduğunu görürüz. Bu dosya, sınıfımız ve sınıf içerisinde kullandığımız metotlarla ilgili açıklama satırlarını içermektedir. Eğer bu dosya, otomatik olarak oluşturulmamış ise **Project** menüsünden **Project Properties** seçeneği ile gelen penceredeki **Build** sekmesinden **Output Path** bölümündeki **XML documentation file** seçeneğini işaretlemeliyiz. XML yorum dosyasının ismi, oluşturulacak olan **dll** dosyası ile aynı olmalıdır.

XML yorum satırlarında kullanılabilecek etiketler şunlardır:

c: Açıklama satırları içerisinde örnek bir kod tanımlamak için kullanılır. Yalnızca bir satırlık kod tanımlamalarında kullanılır.

code: Kod örnekleri tanımlamak için kullanılır. Uzun kod satırları bu etiket içerisinde yazılabilir.

example: Bir sınıfı ya da metodu örnek vererek açıklama amacı ile kullanılır.

exception: Metotlar için istisna yönetimini sağlar. Metodun kullanımı sırasında oluşabilecek istisnaların kontrolünde kullanılır. `Cref` etiketi ile birlikte kullanılır. Örneğin: `<exception cref= "System.OverflowException">`

include: Bu etiket, açıklamaların haricî bir XML dosyasından alınması yerine, kaynak kodla beraber saklanmasına izin verir.

list: Liste şeklinde yorum satırları oluşturmak amacı ile kullanılır.

para: Açıklama satırları içerisinde paragraf tanımlamak amacı ile kullanılır.

param: Bir metodun parametreleri hakkında bilgi vermek amacı ile kullanılır.

paramref: Açıklama satırları içerisinde metotlarda kullanılan parametrelerle ilgili farklı bilgi vermek amacı ile kullanılır.

summary: Uzun açıklama satırları yazmak amacı ile kullanılır.

remarks: Kısa açıklamalar yazmak amacı ile kullanılır.

returns: Belirtilen sınıf ya da metoda erişim şekilleri ile ilgili bilgi verir. İznin niteliği `cref` etiketi ile birlikte belirtilir.

see: Aynı proje içerisinde bulunan diğer elemanları referans vermek amacı ile kullanılır. Referans verilecek eleman `cref` etiketi ile belirtilir.

seealso: Belirtilen elemana benzer söz dizimi olan elemanlara referans vermek amacı ile kullanılır.

typeparam: Param etiketi ile aynı şekilde kullanılır. Bir sınıf ya da fonksiyonun parametreleri hakkında bilgi vermek amacı ile kullanılır.

Value: Sadece bir özelliği (*property*) açıklamak ve ona atanabilecek değeri tanımlamak amacı ile kullanılır.

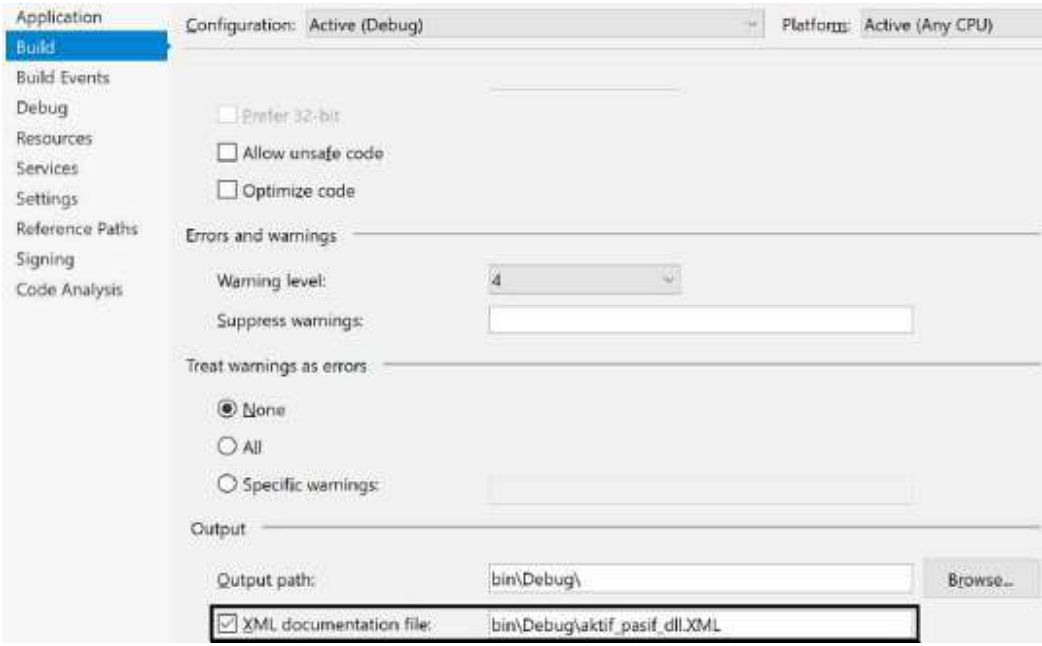
Örnek olarak "`snf_aktif_pasif`" adlı class projemiz içerisine gerekli XML açıklama satırlarını ekleyelim.

XML yorum satırını eklemek için (///) 3 adet bölü işareti eklemeliyiz. İşaretler eklendiği anda **C#** otomatik olarak gerekli standart XML etiketlerini ekleyecektir.

```
using System;
using System.Text;
using System.Windows.Forms;
namespace aktif_pasif_dll
{
    /// <summary>
    /// snf_aktif_pasif sınıfı form üzerindeki butonların
    /// aktif ve pasif yapılabilmesi için gerekli metodları sunar.
    /// </summary>
    public class snf_aktif_pasif
    {
        byte i;
        /// <summary>
        /// Parametre olarak gelen buton dizisini (btn) aktif
        /// yapmak için kullanılır.
        /// </summary>
        /// <param name="btn">Aktif yapılacak buton dizisi. Örneğin
        ///"Button1,Button2" şeklinde</param>
        /// <remarks></remarks>
```

```
        public void mtd_aktif(params Button[] btn)
        {
            for (i = 0; i <= btn.Length - 1; i++)
            {
                btn[i].Enabled = true;
            }
        }
        /// <summary>
        /// Parametre olarak gelen buton dizisini (btn) pasif
        /// yapmak için kullanılır.
        /// </summary>
        /// <param name="btn">Pasif yapılacak buton dizisi. Örneğin
        ///"Button1,Button2" şeklinde</param>
        /// <remarks></remarks>
        public void mtd_pasif(params Button[] btn)
        {
            for (i = 0; i <= btn.Length - 1; i++)
            {
                btn[i].Enabled = false;
            }
        }
    }
}
```

XML açıklama dosyasının otomatik olarak eklenmesi için **Project** / **Proje Adı Properties** menüsünden **Build** sekmesindeki "**XML documentation file**" seçeneğinin işaretlenmesi gereklidir.



"**dll_aktif_pasif.xml**" dosyasını projemizin **debug** klasörüne kopyalayıp kod görünümünde kodumuzu inceleyelim.

Sınıf isminin üzerine imleci getirelim.

```
snf_aktif_pasif nsn_akt_pas = new snf_aktif_pasif();
```

class aktif_pasif_dll.snf_aktif_pasif
snf_aktif_pasif sınıfı form üzerindeki butonların aktif ve pasif yapılabilmesi için gerekli metodları sunar.

Metotlarımızdan birinin üzerine imleci getirelim.

```
nsn_akt_pas.mtd_aktif(btnYeni);
```

void snf_aktif_pasif.mtd_aktif(params Button[] btn)
Parametre olarak gelen buton dizisini (btn) aktif yapmak için kullanılır.

Parametre yazmaya başlarken **xml** yorumuna bakalım.



Property Kullanimi

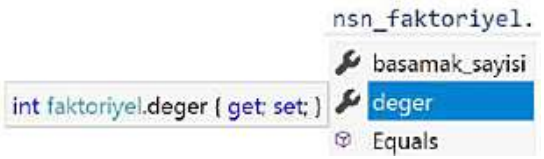
Hazırladığımız sınıflar içerisinde **property** (*özellik*) kullanarak sınıflar içerisine değer gönderip bu değerleri okuyabiliriz. **`set`** ve **`get`** olmak üzere iki tür property vardır;

`set` ile değer gönderirken; **`get`** ile değer alırız.

Faktöriyel işlemi yapan bir sınıf hazırlayalım. "**`faktoriyel`**" isimli **`class`** kodumuz:

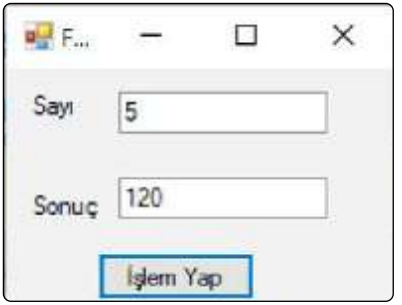
```
class faktoriyel
{
    int i,sayi,sonuc;
    public int deger
    {
        get
        {
            sonuc = 1;
            for (i = 1; i <= sayi; i++)
            {
                sonuc = sonuc * i;
            }
            return sonuc;
        }
        set
        { sayi = value; }
    }
}
```

Oluşturduğumuz nesnenin özelliklerine baktığımızda tanımlamış olduğumuz **`deger`** isimli property'nin eklenmiş olduğunu görürüz.



Sınıfı çağırdığımız form kodu:

```
private void btnislem_Click(object sender, EventArgs e)
{
  faktoriyel nsn_faktoriyel = new faktoriyel();
  nsn_faktoriyel.deger = Convert.ToInt32(txtSayi.Text);
  txtSonuc.Text = nsn_faktoriyel.deger.ToString();
}
```



Sınıfımıza faktöriyeli alınacak sayıyı **`set`** ile gönderirken hesaplanan işlem sonucunu **`get`** ile alıyoruz.

Property tanımlaması yapılırken **Read Only** ve **WriteOnly** olmak üzere iki biçimde tanımlanabilir.

Read Only property sadece değer gönderir. Read Only property içerisinde yalnızca **`get`** bloğu kullanılabilir.

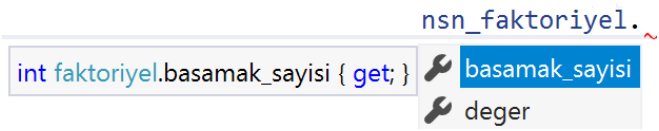
Write Only property sadece değer alır. Write Only property içerisinde yalnızca **`set`** bloğu kullanılabilir.

"**`faktoriyel`**" isimli faktöriyel hesabı yapan sınıfımızı geliştirerek içerisine basamak sayısını bulan bir **read only property** ekleyelim.

Bu property basamak sayısını bulup dışarıdan değer almadan sadece dışarıya değer göndereceğinden dolayı **read only property** olarak tanımlıyoruz.

```
class faktoriyel
{
    int i,sayi,sonuc;
    public int deger
    {
        get
        {
            sonuc = 1;
            for (i = 1; i <= sayi; i++)
            {
                sonuc = sonuc * i;
            }
            return sonuc;
        }
        set
        { sayi = value; }
    }
    public int basamak_sayisi
    {
        get
        {
            return (sonuc.ToString().Length);
        }
    }
}
```

Oluşturduğumuz nesnenin özelliklerine baktığımızda tanımlamış olduğumuz **`basamak_sayisi`** isimli read only property'nin eklenmiş olduğunu görürüz.



Sınıfı çağırdığımız form kodu:

```
private void btnislem_Click(object sender, EventArgs e)
{
  faktoriyel nsn_faktoriyel = new faktoriyel();
  nsn_faktoriyel.deger = Convert.ToInt32(txtSayi.Text);
  txtSonuc.Text = nsn_faktoriyel.deger.ToString();
  lblBasamak.Text = "Basamak Sayısı= " +
nsn_faktoriyel.basamak_sayisi;
}
```



Sınıfımıza faktöriyeli alınacak sayıyı **`set`** ile gönderirken hesaplanan işlem sonucunu **`get`** ile alıp faktöriyel sonucunu ve faktöriyel sonucunun basamak sayısını görüntülüyoruz.

Auto-Implemented Property sayesinde property tanımlama kodlarını daha da kısaltabiliriz. Ancak Auto-Implemented Property tanımlanırken hem **`get`** hem de **`set`** ifadelerini kullanmalıyız. Yalnızca birini kullanamıyoruz.

Örneğin aşağıdaki gibi yalnızca isim değeri alan ve geriye bu değeri döndüren basit bir **`property`** tanımlayalım. Uygulamamız, konsol uygulaması biçiminde olup property'nin içerisinde yer aldığı **`class`** ile **`property`**'nin çağrıldığı kod satırları aynı form içerisinde olacak.

```
using System;
using System.Text;
namespace Auto_Implemented_Property
{
    class bilgi
```

```
    {
        string ib;
        public string isim_bilgisi
        {
            get
            {
                return ib;
            }
            set
            {
                ib = value;
            }
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            bilgi ad = new bilgi();
            ad.isim_bilgisi = "Volkan AKTAŞ";
            Console.WriteLine(ad.isim_bilgisi.ToString());
            Console.ReadLine();
        }
    }
}
```

Yukarıda yazdığımız **bilgi** isimli property'yi **Auto-Implemented Property** biçiminde tanımladığımızda kod satırları aşağıdaki biçimde olacak. Burada **.NET**'in bize sağladığı bir kolaylıktan bahsetmek istiyorum.

Kod satırları içerisinde **prop** yazarak iki defa **Tab** tuşuna bastığımızda:

```
public int MyProperty { get; set; }
```

Biçiminde otomatik olarak **Auto-Implemented Property** varsayılan biçimi ile yazılacaktır.

Bizim yazacağımız property için bu kalıp üzerinde değişiklikleri yaparak property'e şu biçimi veriyoruz.

```
public string isim_bilgisi { get; set; }
```

Sonuç olarak kod satırlarımız aşağıdaki gibi olacak.

```
using System;
using System.Text;
namespace Auto_Implemented_Property
{
    class bilgi
    {
        public string isim_bilgisi { get; set; }
    }
    class Program
    {
        static void Main(string[] args)
        {
            bilgi ad = new bilgi();
            ad.isim_bilgisi = "Enver ARSLAN";
            Console.WriteLine(ad.isim_bilgisi.ToString());
            Console.ReadLine();
        }
    }
}
```

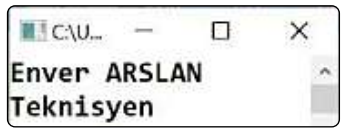
Örneğimiz, kendisine gönderilen "**Enver ARSLAN**" değerini alarak konsol ekranında görüntülemektedir. Görüldüğü gibi kod satırlarını Auto-Implemented Property sayesinde oldukça kısalttık.

Auto Property Initializers

Auto Property Initializers ile birlikte property tanımlanırken varsayılan değer ataması yapmak mümkün olmaktadır. Örneğin yukarıda Auto-Implemented Property konusu için yapmış olduğumuz örneği **Auto Property Initializers** ile varsayılan değer ataması yapacak şekilde değiştirelim. Kodumuzun yeni hâli aşağıdaki gibi olacak:

```
using System;
using System.Text;
namespace Auto_Property_Initializers
{
    class bilgi
    {
        public string isim_bilgisi { get; set; }
      public string meslek_bilgisi { get;} = "Teknisyen";
    }
    class Program
    {
        static void Main(string[] args)
        {
            bilgi bilgiler = new bilgi();
            bilgiler.isim_bilgisi = "Enver ARSLAN";
            Console.WriteLine(bilgiler.isim_bilgisi.ToString());
            Console.WriteLine(bilgiler.meslek_bilgisi.ToString());
            Console.ReadLine();
        }
    }
}
```

Uygulamamızı çalıştırdığımızda **ekran çıktısı** aşağıdaki gibi görüntülenecektir.



Kodumuzda **meslek_bilgisi** isimli property, **read only property** olarak tanımlanmış ve varsayılan olarak "**Teknisyen**" değeri atanmıştır.

```
Console.WriteLine(bilgiler.meslek_bilgisi.ToString());
```

Satırında ise varsayılan değer konsol ekranına yazdırılmıştır.

Ayrıca Auto Property Initializers ile birlikte artık **set** ve **get** niteliklerini birlikte yazma gerekliliği de ortadan kalkarak yalnızca **set** ya da yalnızca **get** kullanılarak varsayılan değer ataması yapılabilmektedir. Kodumuzda **meslek_bilgisi** isimli **property**'e yalnızca **get** niteliğini kullanarak varsayılan değer ataması yaptık.



Yukarıdaki kodumuzda **get** ve **set** niteliklerini kullanırken yaptığımız tanımlamaları C# 7 sürümünden itibaren gelen "Expression Bodied Members" özelliği ile daha kısa ve daha okunabilir hâle getirebiliriz. Örneğimizdeki;

```
public string meslek_bilgisi { get;} = "Teknisyen";
```

satırını "Expression Bodied Members" özelliği ile birlikte

```
public string meslek_bilgisi => "Teknisyen";
```

şeklinde yazabiliriz.



C# 7.3 sürümünden itibaren metotlarımızda kullandığımız parametrelerde sorgular kullanarak geliştirmeler yapabilmekteyiz. Örneğin klavyeden girilen bir değerin çift sayı olup olmadığını bulan bir uygulama yapalım. Örneğimizi öncelikle standart metotlar kullanarak yapacağız.

```
public class SayiKontrol
{
    public static object VeriOku() => Console.ReadLine();

    public bool mtd_CiftSayi()
    {
        var a = VeriOku();
        bool durum = false;
        if (Convert.ToInt32(a)%2 == 0) {durum=true;}
        return durum;
    }
```

```
}
class Program
{
    static void Main(string[] args)
    {
        SayiKontrol clsSayiKontrol = new SayiKontrol();
        Console.WriteLine(clsSayiKontrol.mtd_CiftSayi());
        Console.ReadLine();
    }
}
```

Kodumuzda **SayiKontrol** sınıfı içerisinde bulunan **mtd_CiftSayi()** isimli metot, **VeriOku()** metodundan aldığı değerin çift sayı olup olmadığını kontrol ederek eğer girilen değer çift sayı ise geriye **True** değerini döndürmektedir. **mtd_CiftSayi()** isimli metodumuzu **C# 7.3** versiyonundan itibaren aşağıdaki gibi yazabiliriz.

```
public bool mtd_CiftSayi { get; } = VeriOku() is string veriler
                                 && int.TryParse(veriler, out var v)
                                    && v %2 == 0;
```

Yukarıdaki kodda **VeriOku()** metodundan aldığımız değeri **veriler** isimli **string** türdeki değişkene aktarıyoruz. **veriler** değişkenini **int.TryParse()** metodu ile **int** türüne dönüştürerek sonucu **v** değişkenine aktarıyoruz. **v** değişkeninin değerinin de çift sayı olup olmadığını kontrol ederek eğer **v** değişkeninin değeri çift sayı ise geriye **True** değerini döndürüyoruz. Bu şekildeki kullanımda daha işlevsel bir tanımlama gerçekleştirmiş olduk. İlk metot tanımlamamızda konsol ekranında boş bir değer girildiğinde hata durumu oluşup programın çalışması yarıda kesilirken bu yazım biçimindeki kodumuzda ise boş değer girilirse hata durumu oluşmayıp geriye **False** değeri dönmektedir.

Metodumuzu çağırırken de artık:

```
Console.WriteLine(clsSayiKontrol.mtd_CiftSayi);
```

Biçiminde, bir metot olarak değil de **property** (özellik) olarak çağırıyoruz.

INotifyPropertyChanged Interface'i

INotifyPropertyChanged Interface'i property değerlerinde yapılan değişiklikleri yakalamak için kullanılır. Property değerinde değişiklik meydana geldiğinde **PropertyChanged** olayı tetiklenmektedir. Bu olay yakalanarak property adı gibi nitelikler hakkında bilgi verilebilir. **INotifyPropertyChanged Interface**'ini kullanabilmek için **System.ComponentModel** referansını proje kodları içerisine eklememiz gerekmektedir.

```
using System;
using System.ComponentModel;
namespace PropertyChanged
{
  class Bilgi : INotifyPropertyChanged
  {
    public event PropertyChangedEventHandler PropertyChanged;
    string ad = string.Empty;
    string soyad = string.Empty;

    public string Ad
    {
      get { return ad; }
      set
      {
        if (value != ad)
        {
          ad = value;
          if(PropertyChanged !=null)
          {
            PropertyChanged(this,
new PropertyChangedEventArgs("Ad"));
          }
        }
      }
    }
    public string Soyad
    {
      get { return soyad; }
      set
      {
```

```
        if (value != soyad)
        {
          soyad = value;
          if (PropertyChanged != null)
          {
            PropertyChanged(this,
new PropertyChangedEventArgs("Soyad"));
          }
        }
      }
    }
  }
  class Program
  {
    static void Main(string[] args)
    {
      var bilgi = new Bilgi { Ad = "Volkan", Soyad = "Aktaş" };
      bilgi.PropertyChanged += (sender, e) =>
Console.WriteLine(string.Format("{0} değişti.",
e.PropertyName));
      bilgi.Ad = "Talha Osman";
      bilgi.Soyad = "Aktaş";
      Console.ReadLine();
    }
  }
}
```

Örneğimizde **Bilgi** isimli sınıfı **INotifyPropertyChanged** interface'ini uygulayarak tanımlıyoruz. Sınıf içerisinde **Ad** ve **Soyad** isimli metotları tanımlıyoruz.

```
if (PropertyChanged != null)
{
  PropertyChanged(this,
PropertyChangedEventArgs("Soyad"));
}
```

Yukarıdaki kod satırlarında görüldüğü gibi bu metotların property'lerinde herhangi bir değişiklik olduğunda **PropertyChanged** olayına parametre olarak ilgili property isminin gönderilmesini sağlıyoruz.

```
var bilgi = new Bilgi { Ad = "Volkan", Soyad = "Aktaş" };
```

Satırında **Bilgi** sınıfı türünden **bilgi** değişkenini tanımlayarak **Ad** ve **Soyad** parametrelerine "**Volkan**" ve "**Aktaş**" değerlerini aktarıyoruz.

```
bilgi.PropertyChanged += (sender, e) =>
Console.WriteLine(string.Format("{0} değişti.",
e.PropertyName));
```

Satırında ise **bilgi** değişkeninin property'lerinde herhangi bir değişiklik olduğunda, içeriğinde değişiklik yapılan property isminin konsol ekranına yazdırılmasını sağlıyoruz.

Uygulamamızı çalıştırdığımızda konsol ekranında "**Ad değişti.**" mesajı görüntülenecektir. Çünkü kodumuzda:

```
bilgi.Ad = "Talha Osman";
bilgi.Soyad = "Aktaş";
```

Kod satırlarında görüldüğü gibi "**Soyad**" property'si içeriğinde herhangi bir değişiklik yapılmayıp "**Aktaş**" değeri aynen kalmıştır. "**Ad**" property'sinin ilk değeri olan "**Volkan**" değeri ise değiştirilip "**Talha Osman**" olmuştur.

nameof İfadesi

C# 7 sürümünden itibaren kullanılmaya başlanan **`nameof`** ifadesi bir metodun ismini **`string`** olarak elde etmek amacıyla kullanılmaktadır. **INotifyPropertyChanged Interface** örneğindeki:

```
PropertyChanged(this,
new PropertyChangedEventArgs("Ad"));
PropertyChanged(this,
new PropertyChangedEventArgs("Soyad"));
```

kod satırlarını **`nameof`** ifadesini kullanarak

```
PropertyChanged(this,
new PropertyChangedEventArgs(nameof(Ad)));
PropertyChanged(this,
new PropertyChangedEventArgs(nameof(Soyad)));
```

Şeklinde yazabiliriz. Bu şekilde "**`Ad`**" ve "**`Soyad`**" isimli property'lerin isimleri derleyici tarafından belirtilmektedir.

Inheritence (Miras, Kalıtım)

OOP (*Object Oriented Programming, Nesne Yönelimli Programlama*)'nin temel unsurlarından biri olan **inheritence** ile bir sınıftan başka bir sınıf türeterek bu sınıfa ait metotları kullanabiliriz. Miras alınan sınıfın özellik ve metotları üzerinde değişiklikler yapılarak geliştirilebilir, yeni özellik ve metotlar elde edilebilir. Bir sınıfa mirasçı olarak bir başka sınıfı bağlamak için : işaretini kullanırız.

```
Mirasçı sınıf : Miras Alınan Sınıf
```

şeklinde kullanılır.

Projemize **`Snf_Mesaj`** adıyla yeni bir class ekleyelim. Class kodumuz aşağıdaki gibi olacak:

```
using System;
using System.Windows.Forms;
using System.Text;
namespace Miras
```

```
{
    class Snf_Mesaj
    {
        public Snf_Mesaj()
        {
            MessageBox.Show("Ana Sınıf");
        }

        public void mtd_mesaj(string icerik)
        {
            MessageBox.Show(icerik);
        }
    }
    class snf_mirascı : Snf_Mesaj
    {
        public snf_mirascı()
        {
            MessageBox.Show("Mirasçı Sınıf");
        }

        public void mtd_gelismis_mesaj(string icerik, bool
buyukharf)
        {
            if (buyukharf == true)
            {
                mtd_mesaj(icerik.ToUpper());
            }
        }
    }
}
```

Kodumuzda:

```
class snf_mirascı : Snf_Mesaj
```

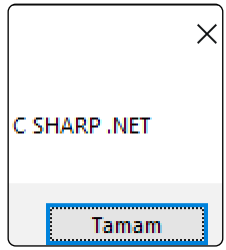
Biçiminde görüldüğü gibi **`Snf_Mesaj`** adlı sınıfa mirasçı olarak **`Snf_mirascı`** adlı sınıfımızı bağlamak için : işaretini kullanırız.

Mirasçı sınıfımıza ait olan metodu çağıran kodu yazmaya başladığımızda bu metoda ait parametrelerin **intellisense** menüsüne eklendiğini görürüz.



Sınıfı çağırdığımız formun kodu ise aşağıdaki gibi olacak:

```
private void btnMesaj_Click(object sender, EventArgs e)
{
    Miras.snf_mirascı nsn_mesaj = new Miras.snf_mirascı();
    nsn_mesaj.mtd_gelismis_mesaj("c sharp .net", true);
}
```



Görüldüğü gibi temel sınıftan miras aldığımız mesaj metodunu geliştirerek büyük harfe çevirme parametresi de ekledik. Bu arada temel sınıf ile miras alınan sınıfların aktifleştiğini görebilmek için birer mesaj penceresi ile "**Ana Sınıf**""**Mirasçı Sınıf**" biçiminde görüntüledik.

Yeni bir örnek kod ile rastgele sayı üreten bir temel sınıf ile bu sınıfa parametre gönderen mirasçı bir sınıf hazırlayalım.

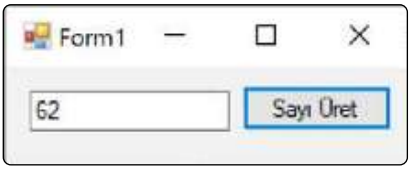
Sınıfımızın kodu:

```
using System;
using System.Text;
namespace Miras_SayiTuretme
{
  class Temel_Sinif
  {
      public int mtd_sayi_turet(int x, int y)
      {
```

```
            int rtrn = 0;
            System.Random rnd = new Random();
            rtrn = rnd.Next(x, y);
            return rtrn;
        }
    }
    public class snf_mirascı : Temel_Sinif
    {
        public object mtd_turet()
        {
            int sayi = 0;
            sayi = mtd_sayi_turet(0, 100);
            return sayi;
        }
    }
}
```

Mirasçı sınıfımızı çağıran kod:

```
private void Button1_Click(object sender, EventArgs e)
{
   Rastgele_Sayi.snf_mirascı nsn_sayi = new Rastgele_Sayi.snf_
mirascı();
   textBox1.Text = nsn_sayi.mtd_turet().ToString(); ;
}
```



Kodumuzda **`Temel_Sinif`** isimli sınıfı miras alan **`snf_mirasci`** isimli sınıf içerisinde **`mtd_turet`** metodu içerisinde belirtilen sayı aralığında rastgele sayı üretiyoruz. Form içerisinde de üretilen rastgele sayıyı **`textBox`** içerisine aktarıyoruz.

Access Modifiers (Erişim Belirleyiciler)

Hazırladığımız class yapıları içerisindeki değişkenlere ve metotlara başka formlardan erişilmesini sağlamak ya da yasaklamak ya da bir metodun bazı parametrelerine erişiminin serbest bırakılıp bazılarına yasaklanması gibi işlemler için **Access Modifiers** (*Erişim Belirleyiciler*) kullanılır.

C# .NET içerisinde kullanabileceğimiz erişim belirleyiciler şunlardır:

public: public belirleyicisinin kullanıldığı metot ya da parametreye erişim tamamen serbesttir. Gerek tanımlandığı sınıf içerisinden gerekse tanımlandığı sınıf dışından erişmek mümkündür.

internal: internal belirleyicisinin kullanıldığı metot ya da parametreye sadece tanımlandığı sınıf içerisinden erişilebilir. Hazırladığınız **dll** dosyasını aynı proje ya da program içerisinde kullanabilir, ancak başka proje ya da programdan erişemezsiniz.

protected: protected belirleyicisinin kullanıldığı metot ya da parametreye sadece tanımlandığı sınıf içerisinden veya bu sınıftan türetilmiş sınıflardan erişilebilir.

static: Eğer metoda direkt olarak sınıfın ismini kullanarak erişmek istersek metodu `static` olarak tanımlamamız gerekir. `static` olarak tanımlanan metotlar, sınıflardan türetilen nesnelerden bağımsız hareket ederler. Dolayısı ile `static` metotları içerisinde tanımlandıkları class'dan yeni bir nesne oluşturmadan direkt olarak "**sınıf ismi.metod ismi**" şeklinde kullanabiliriz.

protected internal: protected internal belirleyicisinin kullanıldığı metot ya da parametreye tanımlandığı sınıf içerisinden, bu sınıftan türetilmiş başka bir sınıf içerisinden ya da aynı assembly içerisinden erişilebilir.

private: private belirleyicisinin kullanıldığı metot ya da parametreye sadece tanımlandığı sınıf içerisinden erişilebilir. Sınıflar, varsayılan olarak private tanımlanırlar.

private protected: private protected belirleyicisinin kullanıldığı metot ya da parametreye sadece tanımlandığı sınıf içerisinden veya bu sınıftan türetilmiş sınıflardan erişilebilirken aynı zamanda private olduklarından yalnızca aynı namespace içerisinden erişilebilirler.

Miras konusunda yaptığımız mesaj örneğini inceleyelim.

```
using System;
using System.Windows.Forms;
using System.Text;
namespace Miras
{
    class Snf_Mesaj
    {
        public Snf_Mesaj()
        {
            MessageBox.Show("Ana Sınıf");
        }
        public void mtd_mesaj(string icerik)
        {
            MessageBox.Show(icerik);
        }
    }
    class snf_mirascı : Snf_Mesaj
    {
        public snf_mirascı()
        {
            MessageBox.Show("Mirasçı Sınıf");
        }
        public void mtd_gelismis_mesaj(string icerik, bool
buyukharf)
        {
            if (buyukharf == true)
            {
                mtd_mesaj(icerik.ToUpper());
            }
        }
    }
}
```

Örneğin; yukarıdaki kod satırlarımızda olduğu gibi "**Snf_Mesaj**" isimli ana sınıfımızdaki metodu **public** olarak tanımladığımızdan dolayı, form içerisinde "**snf_mirascı**" adlı mirasçı sınıfımızdan türettiğimiz nesneye ait metotlar arasında listelenmektedir.



Bu metoda, mirasçı sınıfımızdan türetilmiş nesnemiz içerisinden erişilmesini engellemek için bu metodu **Protected** olarak tanımlayalım.

```
using System;
using System.Windows.Forms;
using System.Text;
namespace Miras
{
    class Snf_Mesaj
    {
        public Snf_Mesaj()
        {
            MessageBox.Show("Ana Sınıf");
        }
        protected void mtd_mesaj(string icerik)
        {
            MessageBox.Show(icerik);
        }
    }
    class snf_mirascı : Snf_Mesaj
    {
        public snf_mirascı()
        {
            MessageBox.Show("Mirasçı Sınıf");
        }
        public void mtd_gelismis_mesaj(string icerik, bool
buyukharf)
```

```
        {
            if (buyukharf == true)
            {
                mtd_mesaj(icerik.ToUpper());
            }
        }
    }
}
```

Yukarıdaki kod satırlarımızda "**Snf_Mesaj**" isimli ana sınıfımızdaki metodu **Protected** olarak tanımladığımızdan dolayı, form içerisinde "**snf_mirasci**" adlı mirasçı sınıfımızdan türettiğimiz nesneye ait metotlar arasında listelenmeyip yalnızca mirasçı sınıfımıza ait "**mtd_gelismis_mesaj**" isimli metodumuz listelenmektedir.



Encapsulation (Kapsülleme)

Encapsulation işlemi, bir sınıf içerisindeki değişkenlere dışarıdan erişimi sağlamak; ancak değişken içeriğinde değişiklik yapılmasını engellemek amacı ile kullanılır. Bunun için öncelikle değişkeni **private** olarak tanımlayıp daha sonra **read only property** ile değişken üzerinde değişiklik yapılmasını engelleriz. Bu işleme **encapsulation** adı verilir.

Yeni bir **Windows Form** projesi oluşturarak içerisine **snf_encaps** isimli yeni bir sınıf ekleyelim. Sınıfımızın kodu aşağıdaki gibi olacak:

```
using System;
using System.Windows.Forms;
using System.Text;
namespace Encapsulation
```

```
{
    class snf_encaps
    {
        public readonly double PI = 22 / 7;
        private int x = 1;
        public int x_property {
            get { return x; }
        }
        public void A()
        {
           MessageBox.Show("Ben şu an A metodundayım");
        }
        protected void B()
        {
            MessageBox.Show("Ben şu an B metodundayım");
        }
        public void mtd_B_ye_KendiSinifindanErisim()
       {
            B();
        }
    }
    class snf_mtd_Bye_TuretilmisSiniftanErisim : snf_encaps
    {
        private void mtd_Bye_TuretilmisSiniftanErisim()
        {
            B();
        }
    }
    class snf_mtd_Bye_GecersizErisim
    {
        private void ByeErisim()
        {
            //B(); Hata veriyor
       }
    }
}
```

Encapsulation işlemini test etmek için kullandığımız formun kodu:

```
private void btnTest_Click(object sender, EventArgs e)
{
    Encapsulation.snf_encaps nsn_encaps = new Encapsulation.snf_
encaps();
    MessageBox.Show(nsn_encaps.x_property.ToString());
    nsn_encaps.A();
    //B ye erişim olamaz.
    //nsn_encaps.B()
    nsn_encaps.mtd_B_ye_KendiSinifindanErisim();
    //PI 'ye erişilebiliniyor ama üzerinde değişiklik yapılamıyor.
    MessageBox.Show(nsn_encaps.PI.ToString());
    //nsn_encaps.PI = 5
}
```

Programımızı çalıştırdığımızda çalışma sırası şu biçimdedir;

- x isimli property okunarak içeriğindeki değer olan **1** değeri mesaj penceresi içerisinde görüntülenir. **Encapsulation** işlemi bu değişken üzerinde uygulanmıştır. **x** değeri önce **private** olarak tanımlanmış daha sonra **read only property** tanımlanarak bu değişkene erişilip üzerinde değişiklik yapılmaması sağlanmıştır.
- "**Ben şu an A metodundayım**" şeklinde mesaj penceresi görüntülenerek **A** metoduna erişim sağlandığı belirtilir. Çünkü **A** metodu **public** olarak tanımlanmıştır.
- "**Ben şu an B metodundayım**" şeklinde mesaj penceresi görüntülenerek **B** metoduna erişim sağlandığı belirtilir.

  Ancak **B** metodu **protected** olarak tanımlandığından dolayı erişim hatası meydana gelmektedir. Bu nedenle **public** olarak tanımladığımız "**mtd_B_ye_KendiSinifindanErisim**" isimli metot ile **B** metodunu çağırmaktayız.

  Pi değeri mesaj penceresi içerisinde görüntülenir. **Pi Read Only** tanımlandığından dolayı değeri görüntülenebilir ama değiştirilemez.

Init-Only Properties

C# 9 sürümünden itibaren erişim belirleyici olarak **init** ifadesini de kullanabilirsiniz. Böylece değişkenlerin yalnızca nesne oluşturulurken tanımlanıp daha sonradan değer atanması engellenmiş olur. Yeni bir **.Net Core Console Application** projesi oluşturup aşağıdaki kod satırlarını yazalım:

```
class Personel
{
    public string Ad { get; set; }
    public string Dal { get; set; }
    public int Yas { get; set; }
    public bool Ogretmen { get; set; }
}
class Program
{
    static void Main(string[] args)
    {
        var p1 = new Personel
        {
            Ad = "Volkan AKTAŞ",
            Dal = "Veri Tabanı Programcılığı",
            Yas = 45,
            Ogretmen = true
        };
        p1.Ad = "Bünyamin KARAHAN";
        Console.WriteLine(p1.Ad);
        Console.ReadLine();
    }
}
```

Kodumuzda **Ad** özelliğini set ifadesi ile tanımladığımız için sonradan değer aktarımı mümkün olmakta ve ekran çıktısı olarak **p1.Ad** değişkeninin yeni değeri olan "**Bünyamin KARAHAN**" değeri görüntülenmektedir.

Kodumuzda

```
public string Ad { get; set; }
```

satırını **init** ifadesini kullanarak

```
public string Ad { get; init; }
```

şeklinde değiştirdiğimizde ise kodumuz hata vermektedir. Çünkü artık **p1.Ad** değişkenine yalnızca ilk tanımlama esnasında değer aktarılır ve daha sonra **değiştirilemez**.

Overriding (Metotları Ezme)

Overriding ile miras alınan sınıfın özellik ve metotları üzerinde değişiklikler yapılarak mirasçı sınıf içerisinde kullanabiliriz. Eğer miras aldığımız sınıfa ait metot ya da özellikler bizim için yetersiz geliyorsa **overriding** ile bu özellik ya da metotları geliştirebiliriz. Bu sayede hem temel sınıfa ait özellik ve metotları hem de mirasçı sınıf içerisindeki özellik ve metotları kullanabiliriz.

Örnek olarak girilen yazılı, sözlü ve ödev notlarına göre ortalamayı hesaplayacak bir uygulama yapalım. Yeni bir Windows form projesi açarak içerisine yeni bir **class** ekleyelim. Class kodumuz aşağıdaki gibi olacak:

```
using System;
namespace Override
{
  class Hesapla
  {
    public int yazılı1 { get; set; }
    public int yazılı2 { get; set; }
    public int sozlu { get; set; }
    public double islem()
    {
      double sonuc = (yazılı1 + yazılı2 + sozlu) / 3;
      return (sonuc);
    }
  }
  class Turemis_Sınıf : Hesapla
  {
    public int odev { get; set; }
    public double islem()
    {
      double sonuc = (yazılı1 + yazılı2 + sozlu + odev) / 4;
```

```
      return (sonuc);
    }
  }
}
```

Formumuzun kodu ise aşağıdaki gibi olacak:

```
using System;
using System.Windows.Forms;
namespace Override
{
  public partial class Form1 : Form
  {
      public Form1()
      {
        InitializeComponent();
      }
      private void btnMesaj_Click(object sender, EventArgs e)
      {
        Turemis_Sınıf ortalama = new Turemis_Sınıf();
        ortalama.yazılı1 = 60;
        ortalama.yazılı2 = 60;
        ortalama.sozlu = 60;
        ortalama.odev = 80;
        MessageBox.Show("Ortalama = " +  ortalama.islem());
      }
   }
}
```

Kodumuzda:

`Hesapla` adlı temel sınıfta, işlem adında bir metot tanımlayarak ortalamayı buluyoruz. Ayrıca **`Turemis_Sınıf`** adlı mirasçı sınıfımız içerisinde yine **işlem** adlı metodu tanımlayarak temel sınıftaki metoddan farklı olarak ödev notunu da ortalamaya dâhil ediyoruz. Burada **`Turemis_Sınıf`** adlı mirasçı sınıftaki **`Hesapla`** metodunu **`override`** etmiş oluyoruz.

Form içerisinde **`ortalama`** değişkenini **`Turemis_Sınıf`** adlı mirsaçı sınıf türünden tanımladık. **`yazılı1`**, **`yazılı2`** ve **`sozlu`** değişkenleri **`Hesapla`** adlı temel sınıfımıza ait özellikler olmasına rağmen bu özelliklere değer

aktarabiliyoruz. **odev** ozelliği ise mirasçı sınıfımıza ait bir özelliktir.

Kodumuz bu hâli yazdığımızda "**Override.Turemis_Sınıf.islem() hides inherited member 'Override.Hesapla.islem()'. Use the new keyword if hiding was intended.**" şeklinde uyarı verecektir.

Yani, **Override.Turemis_Sınıf.islem()** metodu, **'Override.Hesapla.islem()** metodunu ezmektedir. Bu ezme işleminin programcı tarafından kasten yapıldığından emin olunması için **new** ifadesini kullanmamız önerilmektedir. Bu uyarıyı gidermek için **new** ifadesini kullanabilir ya da metodumuzu **override**, **abstract** veya **virtual** niteleyicilerinden birini kullanarak tanımlayabiliriz.

new ifadesini kullanarak temel sınıftaki metodumuzu:

```
new public double islem()
{
        double sonuc = (yazılı1 + yazılı2 + sozlu + odev)/4;
        return (sonuc);
}
```

Şeklinde yazıyoruz. Bu şekilde temel sınıftaki **islem** metodunu gizlemiş oluyoruz. **new** ifadesini kullanmadan **override** işlemini gerçekleştirmek için **Turemis_Sınıf** adlı mirasçı sınıftaki **islem** adlı metodumuzu:

```
public override double islem()
{
    double sonuc = (yazılı1 + yazılı2 + sozlu + odev)/4;
    return (sonuc);
}
```

Şeklinde başına **override** ifadesini ekleyerek yeniden tanımlıyoruz. Ancak bu defa da şöyle bir hata alıyoruz:

"**Override.Turemis_Sınıf.islem(): cannot override inherited member 'Override.Hesapla.islem()' because it is not marked virtual, abstract, or override**"

Bu hatayı gidermek için **Hesapla** adlı temel sınıftaki **islem** adlı metodumuzu **virtual**, **abstract** veya **override** niteleyicilerinden birini kullanarak tanımlamamız gerekiyor. Niteleyicileri belirlerken şunlara dikkat etmeliyiz:

- Metodumuz temel sınıftaki bir metot ise tekrar **override** niteleyicisini kullanmıyoruz.
- Temel sınıftaki metot, virtual niteleyicisi ile tanımlanabilir.
- Temel sınıftaki metot **abstract** niteleyicisi ile belirlenebilir. Ancak **static** olamaz.
- Temel sınıf ve türemiş sınıftaki metotların her ikisi de aynı erişim belirleyiciye sahip olmalıdır. Örneğin her ikisi de **public** ya da **private** olabilir.

Hesapla adlı temel sınıfta **islem** adlı metodumuzu:

```
public virtual double islem()
{
    double sonuc = (yazılı1 + yazılı2 + sozlu) / 3;
    return (sonuc);
}
```

Şeklinde **virtual** niteleyicisini tanımlayarak bu hatayı da gidermiş ve **override** işlemini tamamlamış oluyoruz. **Virtual** (*sanal*) metot tanımlayarak temel sınıfta tanımlanan metodu mirasçı sınıfta istediğimiz biçimde değiştirip kullanabiliriz.

Override işlemi sırasında **abstract** (*soyut*) niteleyicisini de kullanabileceğimizden bahsetmiştik. Örneğimizi **abstract** niteleyicisini kullanacak şekilde düzenleyelim. Öncelikle şunu belirtelim; **Abstract** metodu tek başına kullanamayız. **Abstract** metodu tanımlayabilmek için metodun içerisinde olduğu sınıfı **abstract** olarak yazmak gerekmektedir. **Abstract** olarak tanımlanmış bir sınıf içerisindeki **abstract** metodu içi boş şekilde tanımlarız. Bu sayede, **abstract** sınıftan türetilen mirasçı sınıf içerisinde tanımlanan aynı metodu, istediğimiz biçimde tanımlayarak kullanabiliriz.

Sınıfımızı **abstract** niteleyicisini kullanacak şekilde düzenleyelim:

```
using System;
namespace Override
{
  abstract class Hesapla
  {
    public int yazılı1 { get; set; }
```

```
    public int yazılı2 { get; set; }
    public int sozlu { get; set; }

    abstract public double islem();
  }
  class Turemis_Sınıf : Hesapla
  {
    public int odev { get; set; }
     public override double islem()
    {
        double sonuc = (yazılı1 + yazılı2 + sozlu + odev)/4;
        return (sonuc);
    }
  }
}
```

Hesapla isimli temel sınıfımızı başına **abstract** niteleyicisini ekleyerek ve **islem** metodumuzu **abstract** niteleyicisi ile:

```
abstract public double islem();
```

Şeklinde tanımladık. Bu şekilde temel sınıf içerisindeki **islem** metodunun değil de **Turemis_Sınıf** adlı mirasçı sınıf içerisindeki **islem** metodunun kullanılmasını sağladık.

Kodumuzdan da anlaşıldığı üzere bir **abstract** metot ancak bir **abstract** sınıf içerisinde tanımlanır.

Abstract metot **public** olarak tanımlanmalıdır. Aksi takdirde mirasçı sınıf içerisinden temel sınıftaki metoda erişim sağlanamaz.

Ayrıca **abstract** sınıflar ile **new** anahtar ifadesi kullanılarak yeni bir nesne tanımlanamaz.

Örneğin:

```
Hesapla sonuc = new Hesapla();
```

Şeklinde yapacağımız bir tanımlama **hata** verecektir.

sealed (*mühürlemek*) ifadesini kullanarak bir sınıfın tümünün türetilmesini engelleyebilirsiniz. Ya da bir sınıfın tümü yerine yalnızca sınıf içerisindeki bir metodun **override** edilmesini engellemek için metodun başına **sealed** ifadesini ekleyebilirsiniz.

Örneğin **Hesapla** isimli temel sınıfımızın türetilmesini engellemek için:

```
sealed class Hesapla
```

Şeklinde tanımlarız. Tanımlamamızı bu şekilde yaptığımızda **Turemis_Sınıf** adlı mirasçı sınıf içerisindeki **islem** metodumuzda kullanılan **yazılı1**, **yazılı2** ve **sozlu** değişkenlerinin kullanımında derleme zamanı hatası oluşacaktır. Çünkü bu değişkenler, temel sınıfımız içerisinde tanımlanmıştı.

Hesapla adlı temel sınıfımızın tümünü engellemek yerine **islem** adlı metodumuzu:

```
sealed public virtual double islem()
```

Şeklinde tanımladığımızda ise uygulamamızı çalıştırdığımız zaman bu metodumuzun **override** edilmesini engellediğimiz için **hata** meydana gelecektir.



C# 9 sürümünden itibaren .NET Core projelerinizde, temel sınıftan override edilen bir metodun farklı türde bir değer döndürmesi mümkündür.

```
public class bolum
{
    public string okul_bolum { get; set; }
}
public class dal : bolum
{
    public string okul_dal { get; set; }
}
public class Personeller
{
    public string Ad { get; set; }
    public virtual bolum BilgiAl()
    {
        return new bolum
```

```
        {
            okul_bolum = null
        };
    }
}
public class Ogretmen : Personeller
{
    public override dal BilgiAl()
    {
        return new dal
        {
            okul_bolum = null,
            okul_dal = null
        };
    }
}
```

C# 9 öncesinde yukarıdaki kod satırlarını çalıştırdığımızda kodumuz "**`Ogretmen`**" sınıfı içerisinde hata verecekti. Çünkü dönüş türü "**`Personeller.BilgiAl()`**" metodu ile uyumlu olması için "**`bolum`**" olmalıdır. Ancak C# 9 ve sonrası sürümlerde override edilen metot farklı türlerden veriler döndürebildiği için kodumuz **hata vermeden** çalışmaktadır.

Polymorphism (Çok Biçimlilik)

Bir temel sınıfın özellik ve metotlarını, temel sınıftan türetilen bir sınıfa aktardığımızda temel sınıfa ait metotları türemiş sınıf içerisinde de kullanabiliriz. Ancak her iki sınıfta da aynı metot bulunduğunda **temel sınıftaki** metot kullanılacak, türemiş sınıftaki metot ise kullanılmayacaktır. Temel sınıfa ait metotları türemiş sınıfta farklı biçimlerde tanımlayarak kullanmak için **`override`** ve **`virtual`** niteleyicilerini kullanmak gereklidir. Bu şekilde türemiş sınıfta metotların farklı biçimlerde tanımlanarak kullanılabilmesine **`polymorphism`** (*çok biçimlilik*) adı verilir.

Örnek olarak yeni bir **Windows form** projesi oluşturarak içerisinde **`Snf_Mesaj`** adıyla yeni bir **`class`** ekleyelim. **`Class`** kodumuz aşağıdaki gibi olacak:

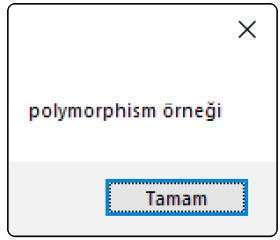
```
using System;
using System.Windows.Forms;
```

```
using System.Text;
namespace Polymorphism
{
  public class snf_Temel
  {
      public void mtd_mesaj(string icerik)
      {
          MessageBox.Show(icerik);
      }
  }
  public class snf_Turemis : Snf_Temel
  {
      public void mtd_mesaj(string icerik)
      {
           MessageBox.Show(icerik.ToUpper());
      }
  }
}
```

Form içerisindeki kodumuz ise aşağıdaki gibi olacak:

```
private void btnMesaj_Click(object sender, EventArgs e)
{
     snf_Temel mesaj = new snf_Turemis();
     mesaj.mtd_mesaj("Polymorphism Örneği");
}
```

Kodumuzda **`mesaj`** isimli değişkenimizi **`snf_Turemis`** isimli türemiş sınıfımız, **`snf_Temel`** isimli temel sınıfı belirtecek şekilde tanımlıyoruz.



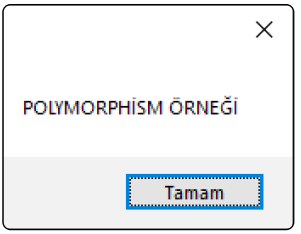
Kodumuzu çalıştırdığımızda temel sınıf içerisindeki metodumuz çalışarak içeriğimiz **küçük harfli** olarak görüntülenecektir. Çünkü temel sınıf, türemiş

sınıf içerisindeki metotlara erişemeyecektir.

İşte burada **polymorphism** devreye girecek. Bunun için temel sınıf içerisindeki metodu **virtual** niteleyicisi ile sanallaştıracak ve türemiş sınıf içerisindeki metodu ise **override** niteleyicisi ile ezeceğiz.

Sınıfımızın kodunu aşağıdaki gibi düzenliyoruz:

```
using System;
using System.Windows.Forms;
using System.Text;
namespace Polymorphism
{
  public class snf_Temel
  {
      public virtual void mtd_mesaj(string icerik)
      {
          MessageBox.Show(icerik);
      }
  }
  public class snf_Turemis : Snf_Temel
  {
      public override void mtd_mesaj(string icerik)
      {
           MessageBox.Show(icerik.ToUpper());
      }
  }
}
```



Uygulamamızı çalıştırdığımızda artık temel sınıf içerisindeki metot değil de türemiş sınıf içerisindeki metot çalıştırıldığı için uygulamamız içeriği **büyük harfli** olarak görüntüleyecektir.

Interface (Arayüz)

C# içerisinde bir sınıf birden fazla sınıftan türetilemez. Eğer birden fazla sınıftan türeyecek bir sınıf tanımlamak istiyorsak bunun için interface kullanmanız gereklidir. **interface** tanımlaması şu şekilde yapılır:

interface sınıf ismi: temel sınıf 1, temel sınıf 2 ... temel sınıf n

Bir interface içerisindeki tüm metotlar **`abstract`** olarak kabul edilir. Ayrıca metotların başına **`abstract`** niteleyicisini yazmaya gerek yoktur.

interface ve interface içerisindeki tüm metotlar **`public`** olarak kabul edilir. Ayrıca başlarına **`public`** erişim belirleyicisini yazmaya gerek yoktur.

interface içerisindeki metotlar interface'in uygulandığı sınıf tarafından ezilmelidir. Ezme işlemi, metotların başına **`override`** niteleyicisi eklenmese de gerçekleştirilir.

Örnek olarak yeni bir **Windows form** projesi oluşturarak içerisinde **`Hesapla`** adıyla yeni bir **`class`** ekleyelim. **`Class`** kodumuz aşağıdaki gibi olacak:

```
using System;
using System.Text;
namespace Interface
{
  interface KDVliSonuc
  {
    double kdvoran { get; set; }
    int adet { get; set; }
    int birimfiyat { get; set; }
    double kdvliislem();
  }
  interface KDVsizSonuc
  {
    int adet { get; set; }
    int birimfiyat { get; set; }
    double kdvsizislem();
  }
  public class Turemis_Sınıf :KDVliSonuc,KDVsizSonuc
  {
```

```
    public double kdvoran { get; set; }
    public int adet { get; set; }
    public int birimfiyat { get; set; }
    public double kdvliislem()
    {
        double sonuc = (adet*birimfiyat)+
((adet*birimfiyat)*kdvoran);
        return (sonuc);
    }
    public double kdvsizislem()
    {
        double sonuc = adet * birimfiyat ;
        return (sonuc);
    }
  }
}
```

Kodumuzda **KDVliSonuc** ve **KDVsizSonuc** adlı iki ayrı **interface** tanımladık, **interface** içerisinde tüm özellikleri ve metotları içleri boş olacak şekilde aynen **abstract class** yapısında olduğu gibi içleri boş bir biçimde tanımladık. Tanımladığımız **interface**'leri kullanacak olan sınıfımızı ise:

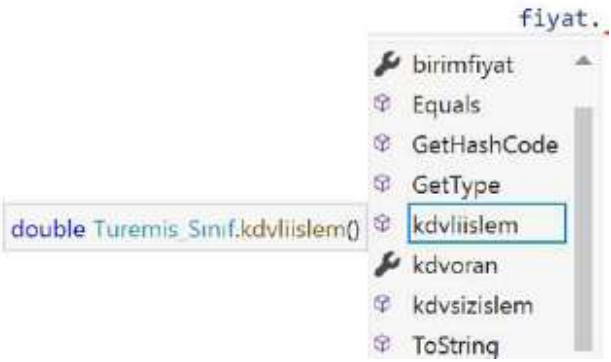
```
public class Turemis_Sınıf :KDVliSonuc,KDVsizSonuc
```

Şeklinde tanımladık. Bu sınıf içerisinde daha önce **interface** içerisinde içleri boş olarak tanımladığımız **kdvliislem** ve **kdvsizislem** adlı metotları yeniden tanımlıyoruz. Bu şekilde bu metotlar, **interface** içerisindeki aynı isimli metotları eziyor.

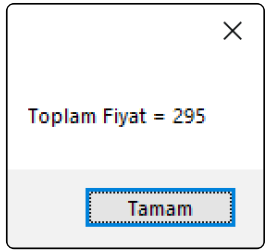
Form kodumuz ise aşağıdaki gibi olacak:

```
private void btnMesaj_Click(object sender, EventArgs e)
{
    Turemis_Sınıf fiyat = new Turemis_Sınıf();
    fiyat.adet = 10;
    fiyat.birimfiyat = 25;
    fiyat.kdvoran = 0.18;
    MessageBox.Show("Toplam Fiyat = " + fiyat.kdvliislem());
}
```

Kodumuzda **`Turemis_Sınıf`** adlı sınıfımız **`KDVliSonuc`** ve **`KDVsizSonuc`** adlı her iki **`interface`**'i de kullanabildiği için bu **`interface`**'ler içerisinde tanımlanan **`kdvliislem`** ve **`kdvsizislem`** adlı metotların her ikisini de birden kullanabiliriz.



Uygulamamızı çalıştırdığımızda belirtilen KDV oranı ve birim fiyata göre toplam fiyatı hesaplayıp mesaj penceresi içerisinde görüntüleyecektir.



Record

Bir nesnenin tümüyle değişmez olmasını ve bir *değer gibi* davranmasını isterseniz bunu bir **`record`** olarak tanımlamalısınız. C# 9 sürümünden itibaren .NET Core projelerinizde **`record`** tanımlaması yapabilirsiniz. Bir **`record`** hâlen bir sınıftır ama ek olarak bazı **özellikler kazanır**.

```
record [Name]
{
    ErişimBelirleyici DeğişkenTürü İsim {get; init;}
}
```

Şeklinde tanımlanırlar.

Örneğin

```
public class Personel
{
    public string Ad { get; set; }
    public string Dal { get; set; }
    public int Yas { get; set; }
    public bool Ogretmen { get; set; }
}
```

Şeklinde yaptığımız sınıf tanımlamasını **`record`** olarak

```
public record Personel
{
    public string Ad { get; init; }
    public string Dal { get; init; }
    public int Yas { get; init; }
    public bool Ogretmen { get; init; }
}
```

Şeklinde tanımlayabiliriz. Aşağıdaki gibi **`p1`** değişkenini tanımlayıp, **değer ataması yaparak** içeriğini yazdırabiliriz.

```
var p1 = new Personel {
    Ad = "Volkan AKTAŞ",
    Dal = "Veri Tabanı Programcılığı",
    Yas = 45,
    Ogretmen = true
};
Console.WriteLine($"Personel: {p1.Ad}");
```

C# 9 sürümünden itibaren tanımlamayı

```
Personel p1 = new() {
    Ad = "Volkan AKTAŞ",
    Dal = "Veri Tabanı Programcılığı",
    Yas = 45,
    Ogretmen = true
```

```
};
```

Şeklinde de yapabileceğimizi daha önce kitabımızın **6. Bölümü** olan "Kontrol Yapıları" bölümü "**Pattern Matching**" başlığı altında da belirtmiştik.

Record içerisinde değişkenleri tanımlarken **init** kullanımı nedeniyle

```
p1.Ad = "Bünyamin KARAHAN";
```

şeklinde **p1.Ad** değişkeninin içeriğini ilk tanımlama sonrasında değiştirmeye kalktığımızda hata verecektir. Bu konuyu kitabımızın bu bölümünde "**Init-only Properties**" başlığı altında da belirtmiştik. **record** yapısı içerisinde bu tip değişiklikleri C# 9 sürümünden itibaren "**with**" ifadesini kullanarak gerçekleştirebiliriz.

```
var p2 = p1 with { Ad = "Bünyamin KARAHAN" };
Console.WriteLine($"Personel: {p2.Ad}");
```

Yukarıdaki kod satırlarında "**with**" ifadesini kullanarak **p1** nesnesinden **p2** isimli yeni bir nesneyi, **Ad** değişkeninin değerine yeni bir değer atayarak tanımlıyoruz. Dolayısıyla ekrana **p2.Ad** değişkeninin yeni değeri olan "**Bünyamin KARAHAN**" yazdırılacaktır. Diğer yandan örneğin

```
Console.WriteLine($"Personel: {p2.Dal}");
```

p2.Dal değişkeninin değerini yazdırdığımızda bu değişkenin değeri üzerinde değişiklik yapılmadığı için "**Veri Tabanı Programcılığı**" değeri ekrana yazdırılır.

Şimdiye kadar yaptığımız **record** tanımlamalarına **norminal records** ismi verilmektedir. Bunun dışında **record** tanımlaması **positional records** şeklinde de yapılabilir. **Positional records** şeklindeki tanımlamayla **constructor**, **deconstructor** ve **property** tanımlamadan tek satırda **record** oluşturabilirsiniz.

```
public record Personel
{
    public string Ad { get; set; }
    public string Dal { get; set; }
    public int Yas { get; set; }
```

```
    public bool Ogretmen { get; set; }
    public Personel(string ad, string dal,
int yas, bool ogretmen)
    => (Ad, Dal, Yas, Ogretmen) = (ad, dal, yas, ogretmen);

    public void Deconstruct(out string ad, out string dal,
out int yas, out bool ogretmen)
    => (ad, dal, yas, ogretmen) = (Ad, Dal, Yas, Ogretmen);
}
```

Örneğin yukarıdaki **`record`** tanımlamasını

```
public record Personel(string Ad, string Dal, int Yas, bool
Ogretmen);
```

şeklinde **tek bir satırda** yazabiliriz.

Positional records ile property'leri daha sonradan da tanımlamak mümkündür. Örneğin yukarıda yaptığımız **`record`** tanımlamasını aşağıdaki gibi düzenleyelim:

```
public record Personel
{
    public string Ad { get; set; }
    public string Dal { get; set; }
    public int Yas { get; set; }
    public bool Ogretmen { get; set; }
    public Personel(string ad, string dal)
    => (Ad, Dal) = (ad, dal);

    public void Deconstruct(out string ad, out string dal)
     => (ad, dal) = (Ad, Dal);
}
```

Bu tanımlamayı **positional records** ile aşağıdaki gibi yazabiliriz. Böylece **`record`** tanımlamasını yaparken aynı anda özellik tanımlamasını da yapabilirsiniz. Bu özellikler **`Ad`** ve **`Dal`** örneğinde olduğu gibi varsayılan olarak "**`init`**" olarak tanımlanmaktadır.

```
public record Personel(string Ad, string Dal)
```

```
{
    public int Yas { get; init; }
    public bool Ogretmen { get; init; }
}
```

Değişkenlere **değer aktarımını** da aşağıdaki gibi gerçekleştirebiliriz:

```
Personel p1 = new Personel("Volkan AKTAŞ", "Veri Tabanı
Programcılığı")
{
    Yas = 45,
    Ogretmen = true
};
```

`Yas` ve **`Ogretmen`** özellikleri init olarak tanımlandığı için **hata** verecektir. Eğer tanımlanma esnasında aktarılan değerlerin sonradan değiştirilmesini isterseniz ilgili özellikleri "**`init`**" yerine

```
public record Personel(string Ad, string Dal)
{
    public int Yas { get; set; }
    public bool Ogretmen { get; set; }
}
```

şeklinde "**`set`**" ile **tanımlamalısınız**.

`record` olarak tanımladığınız değişkenlerin değer türünden değişkenler gibi davrandığını belirtmiştik. Bir **`sınıf`** ve **`record`** tanımlaması yaparak bunu inceleyelim. Öncelikle bir sınıf tanımlaması yapalım.

```
public class Personel
{
   public string Ad { get; init; }
   public string Dal { get; init; }
   public int Yas { get; init; }
   public bool Ogretmen { get; init; }
}
class Program
{
   static void Main(string[] args)
```

```
    {
        Personel p1 = new()
        {
            Ad = "Volkan AKTAŞ",
            Dal = "Veri Tabanı Programcılığı",
            Yas = 45,
            Ogretmen = true
        };
        Personel p2 = new()
        {
            Ad = "Volkan AKTAŞ",
            Dal = "Veri Tabanı Programcılığı",
            Yas = 45,
            Ogretmen = true
        };
        Console.WriteLine(p1 == p2);
        Console.ReadLine();
    }
}
```

Yukarıdaki kod satırlarını çalıştırdığımızda **`False`** sonucunu üretecektir. Çünkü karşılaştırma işlemi yapılırken nesnelerin **referansları karşılaştırılmaktadır**. Her ikisi de farklı nesneler olduğu için karşılaştırma işlemi sonucunda **`False`** değeri dönmektedir. Aynı karşılaştırmayı **`record`** tanımlayarak yapalım.

```
public record Personel
{
    public string Ad { get; init; }
    public string Dal { get; init; }
    public int Yas { get; init; }
    public bool Ogretmen { get; init; }
}
class Program
{
    static void Main(string[] args)
    {
        Personel p1 = new()
        {
            Ad = "Volkan AKTAŞ",
            Dal = "Veri Tabanı Programcılığı",
```

```
            Yas = 45,
            Ogretmen = true
        };
        Personel p2 = new()
        {
            Ad = "Volkan AKTAŞ",
            Dal = "Veri Tabanı Programcılığı",
            Yas = 45,
            Ogretmen = true
        };
        Console.WriteLine(p1 == p2);
        Console.ReadLine();
    }
}
```

Bu şekilde yaptığımız karşılaştırma ise **`True`** sonucunu üretecektir. Çünkü karşılaştırma işlemi yapılırken **`record`** ile tanımlanan değişkenlerin aldıkları **değerler karşılaştırılmaktadır**. Her iki değişkenin aldığı değerler de eşit olduğu için geriye **`True`** değeri dönmektedir.



Bir record bir sınıftan miras alamaz yalnızca bir record'dan miras alabilir.

```
public record Personeller
{
   public string Ad { get; init; }
   public int Yas { get; init; }
}
public record Personel:Personeller
{
   public bool Ogretmen { get; init; }

}
Personel p1 = new Personel { Ad = "Volkan Aktaş",
Yas = 45,Ogretmen=true };
```

Örneğin yukarıda görüldüğü gibi **`Personeller`** isimli temel **`record`** içerisinden miras alan **`Personel`** adında **mirasçı** bir **`record`** tanımladık.

Yukarıdaki kullanım şekli norminal **`record`** için geçerlidir. Aynı **`record`** tanımlamasını **positional record** olarak tanımlayalım.

```
public record Personeller(string Ad);
public record Personel(string Ad,bool Ogretmen) : Personeller(Ad);
```

Mirasçı sınıftan bir önceki kodumuzda olduğu gibi

```
Personel p1 = new Personel { Ad = "Volkan Aktaş", Ogretmen=true };
```

Şeklinde değer aktararak yeni bir nesne oluşturmaya kalktığımızda **hata** verecektir. Bunun yerine

```
var p1 = new Personel("Volkan Aktaş", true);
```

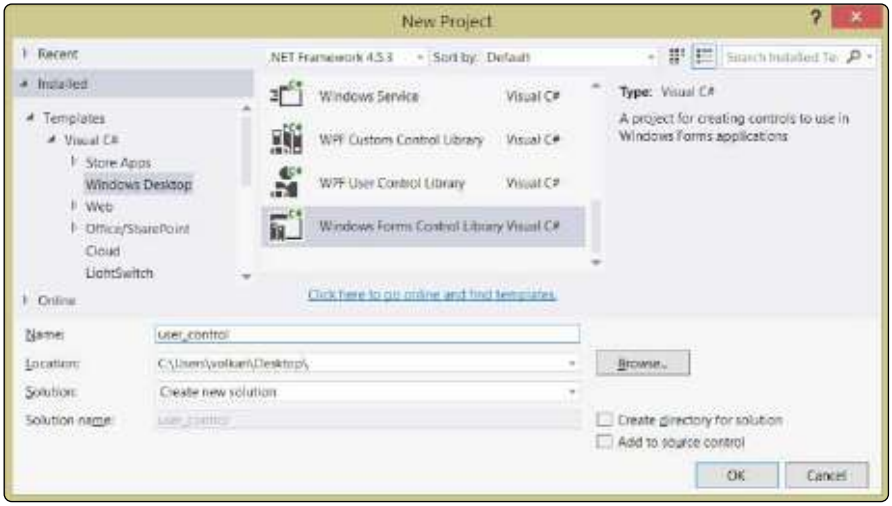
satırında olduğu gibi tanımlama yapabiliriz.

USER CONTROL HAZIRLAMA

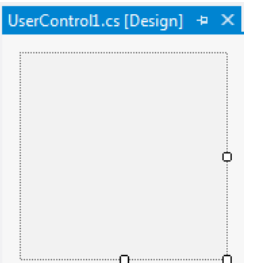
22

C#, kullanıcıların hemen hemen bütün ihtiyaçlarını karşılayabilecek kontrollere sahiptir. Ancak bazen kontrollerin özellik ya da metotları bize yeterli gelmediğinde ya da bir den fazla kontrole ait özellikleri bir araya getirmek istediğimizde, yeni kontroller tasarlayabilir ya da var olan kontrolleri istediğimiz biçimde geliştirerek yeni kontroller elde edebiliriz.

Yeni bir **User Control** hazırlamak için **File \ New Project \ Windows \ Windows Forms Control Library** seçenekleri ile yeni bir **User Control** projesi açılır.

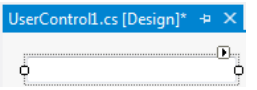


Karşımıza boş bir tasarım ekranı gelecektir. Bunun içerisine, **ToolBox** içerisindeki kontrollerden istediğimiz kontrolleri ekleyebiliriz.



Biz örnek olarak bir metin kutusu ekleyerek özelliklerini biraz geliştireceğiz. Text kutusu ile formu, aynı boyutlara gelecek biçimde ayarlıyoruz. **TextBox** nesnesinin ismini **`metin_kutusu`** biçiminde belirliyoruz.

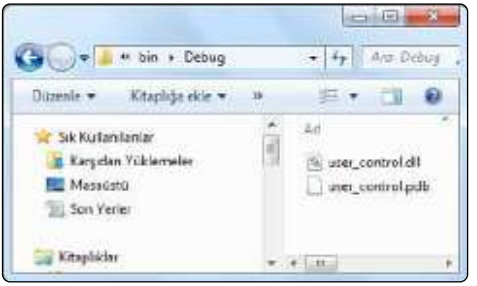
TextBox'ımızı ekledikten sonra **User Control**'un kod görünümüne geçip aşağıdaki kod satırlarını yazıyoruz:




```
public int uzunluk
{
  get
  {
    int harfsayisi=0,noktasayisi, bosluksayisi;
    noktasayisi = metin_kutusu.Text.Split('.').Length-1;
    bosluksayisi = metin_kutusu.Text.Split(' ').Length-1;
    harfsayisi = metin_kutusu.Text.Length - (bosluksayisi + noktasayisi);
    return (harfsayisi);
  }
}
public int cumle_sayısı
{
  get
  {
    int noktasayisi;
    noktasayisi = metin_kutusu.Text.Split('.').Length - 1;
    return (noktasayisi);
  }
}
public string yazan
{
  get { return "Volkan AKTAŞ"; }
}
```

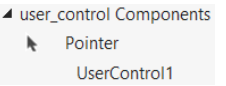
User Control'a 3 adet yalnız okunabilir özellik ekledik. Yalnız okunabilir özellik olduğundan dolayı **`Get`** bloğunu kullandık. Eğer dışarıya değer göndermek isteseydik, **`Set`** bloğunu da kullanmamız gerekirdi.

`uzunluk` isimli property, **TextBox** içerisine girilen harf sayısını bulacak. `cumle_sayısı` adlı property, **TextBox** içerisindeki cümle sayısını görüntüleyecek. `yazan` isimli property ise, yalnızca **User Control**'u yazan kişi hakkında bilgi vermek amaçlı kullanılacak.

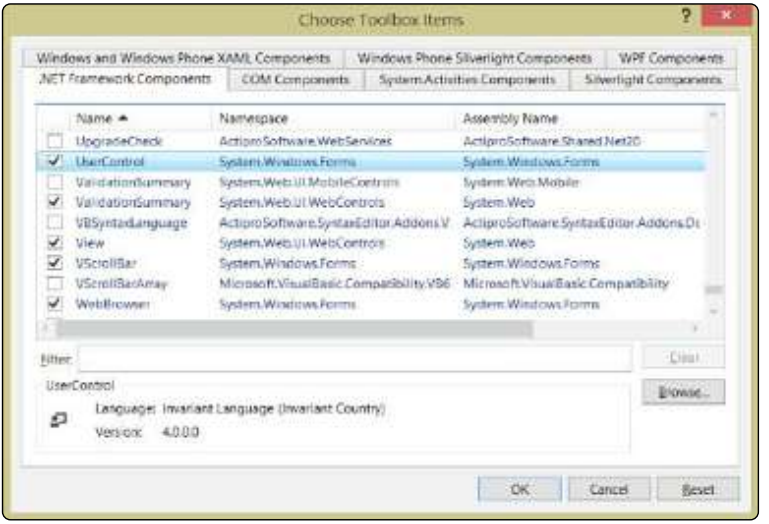


Hazırladığımız **User Control**'u, **Build \ Build user_control** seçenekleri ile derliyoruz. Derleme işlemi sonucunda oluşan **User Control** için bir dll dosyası oluşturulacaktır.

Derleme işlemi sonrasında, oluşturmuş olduğumuz **User Control**, **ToolBox**'ın en üst tarafında **user_control Components** başlığı altında eklenecektir.



Eğer diğer projelerinizin içerisinde **User Control**'u ekleyerek kullanmak isterseniz **ToolBox**'a farenin sağ tuşu ile tıklayıp **Choose Items…** seçeneğini tıklayarak, oluşturmuş olduğumuz **User Control**'u **ToolBox** içerisine ekleyebilirsiniz.

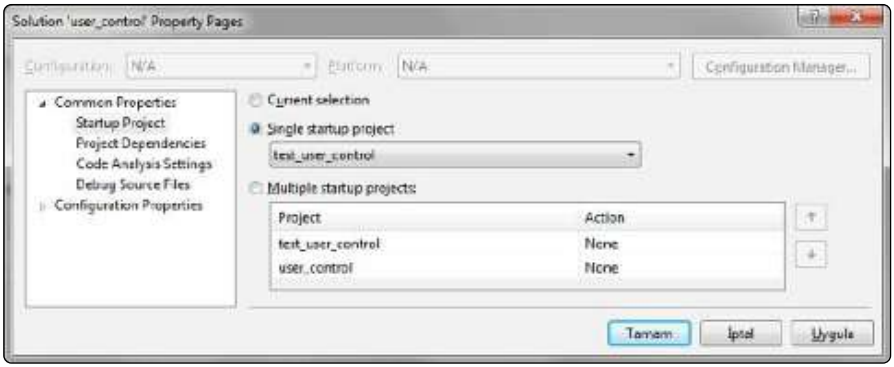


User Control'u test etmek için **Add New Project** seçeneği ile **Solution** içerisine yeni bir **Windows Forms Application** ekleyelim.

ToolBox içerisine eklenen **User Control**'u form üzerine ekleyerek aşağıdaki kod satırlarını yazıyoruz.

```
private void btnBilgi_Click(object sender, EventArgs e)
{
  label1.Text = "Harf Sayısı= " + txt_metin.uzunluk.ToString();
  label2.Text = "Cümle Sayısı= " + txt_metin.cumle_sayısı.ToString();
}
```

Başlangıçta çalışacak projemizi belirlemek için **Solution** ismine sağ tıklayarak **Properties** seçeneğini tıklıyoruz. Gelen pencerede **Startup Project** bölümünden `test_user_control` isimli **Windows Forms** projemizi seçiyoruz.





Uygulamamızı çalıştırıp, örnek bir metin girip test edelim. Görüldüğü gibi **TextBox** içerisindeki yazının harf sayısı "*uzunluk*" özelliği ile, cümle sayısı ise "*cümle_sayısı*" özelliği ile hesaplandı.

User Control'u yazan kişi özelliğini test etmek için ise **User Control** seçiliyken **Properties** penceresinden `yazan` isimli özelliğe bakıyoruz.

uzunluk	0
Visible	True
yazan	Volkan AKTAŞ

REGISTRY İŞLEMLERİ

23

Registry, işletim sistemi ve bu işletim sistemi üzerinde yüklü olan programlar, kullanıcı bilgileri gibi bilgilerin tutulduğu kayıt defteridir. Biz de programlarımızla ilgili ya da kullanıcılarla ilgili saklamak istediğimiz kısa bilgileri registry üzerinde saklayabiliriz.

Registry üzerinde şu ana anahtarlar bulunur.

- **`HKEY_CLASSES_ROOT`:** Windows işletim sistemi ve çalışmakta olan programlar hakkında bilgiler saklanır.
- **`HKEY_CURRENT_USER`:** Geçerli kullanıcı hakkındaki bilgiler ve ayarlar saklanır.
- **`HKEY_LOCAL_MACHINE`:** Bilgisayarın konfigürasyon bilgileri saklanır.
- **`HKEY_USERS`:** Tüm kullanıcılar hakkındaki bilgiler ve ayarlar saklanır.
- **`HKEY_CURRENT_CONFIG`:** Donanım bilgileri saklanır.

Registry üzerinde işlem yapabilmek için **Microsoft.Win32** sınıfı kullanılmaktadır. Bu sınıfı import ettikten sonra, registry işlemlerinde kullanacağımız nesneleri **`RegistryKey`** sınıfından tanımlayarak, registry üzerinde işlemler gerçekleştirebiliriz.

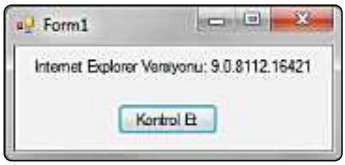
Registry Anahtar ve Değerlerine Erişim

Registry'deki anahtarlara erişmek için **`OpenSubKey`** metodu, değerlere erişmek için ise **`GetValue`** metodu kullanılır.

Örnek olarak bilgisayarda yüklü olan **Internet Explorer** versiyonunu bulan bir kod yazalım. Internet Explorer'ın versiyon bilgisi *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Internet Explorer* anahtarı altında **`Version`** adlı bir değer içerisinde saklanmaktadır. Biz bu değere erişerek versiyon bilgisini elde edeceğiz.

```
using System;
using System.Windows.Forms;
using Microsoft.Win32;
namespace Registry_GetValue
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
  }
  private void btnKontrol_Click(object sender, EventArgs e)
  {
   try
   {
    RegistryKey anahtar = Registry.LocalMachine.OpenSubKey
        ("Software\\Microsoft\\Internet Explorer", false);
    string deger = anahtar.GetValue("Version").ToString();
    label1.Text = "Internet Explorer Versiyonu: " + deger;
   }
   catch
   {
    label1.Text = "Bilgisayarda Internet Explorer yüklü değil";
   }
  }
 }
}
```

`OpenSubKey` metodunun ikinci parametresi yazma izni verilip verilmeyeceğini belirtmektedir. **`true`** ve **`false`** şeklinde iki **`boolean`** değer alır. Registry'den yalnızca okuma yapacağımız için bu parametreye **`false`** değerini verdik.



Kodumuzu çalıştırdığımızda **Internet Explorer** versiyonu tespit edilerek, **Label** içerisinde görüntülenmektedir.

Registry'e Anahtar ve Değer Ekleme

Yeni bir anahtar oluşturmak için **`CreateSubKey`** metodu, yeni bir değer eklemek için ise **`SetValue`** metodu kullanılır.

`SetValue` metodu; **`SetValue(DeğerAdı, Atanacak Değer, Değer Türü)`** şeklinde kullanılır.

Değer türü (**`RegistryValueKind`**) parametresi, şu değerleri alabilir: Bu değerler, registry içerisinde yeni değer oluştururken karşımıza çıkan değer türü çeşitleridir.

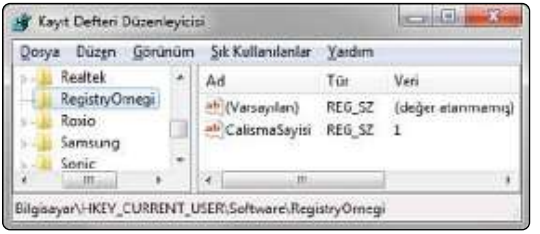
- **`String`**: Dize değeri.
- **`Binary`**: İkilik değer.
- **`DWord`**: 32 bit binary değer.
- **`MultiString`**: Çok dizeli değer.
- **`QWord`**: 64 bit binary değer
- **`Unknown`**: Genişletilebilir dize değerine karşılık gelir.

Örneğin; *HKEY_CURRENT_USER\Software* anahtarı altında **`RegistryOrnegi`** adı ile yeni bir anahtar oluşturalım. Bu anahtarın içerisinde de **`CalismaSayisi`** adı ile bir değer oluşturarak, ilk değer olarak **`1`** değerini verelim.

Anahtarımızı *HKEY_LOCAL_MACHINE* yerine *HKEY_CURRENT_USER* altında tanımlamamızın nedeni; **`current_user`** içerisine anahtar eklemek için yönetici hakkına ihtiyaç olmamasındandır.

```
using System;
using System.Windows.Forms;
using Microsoft.Win32;
namespace Registry_SetValue
{
 public partial class Form1 : Form
 {
```

```
  public Form1()
  {
    InitializeComponent();
  }
  private void btnDegerOlustur_Click(object sender, EventArgs e)
  {
   RegistryKey anahtar = Registry.CurrentUser.OpenSubKey("Software",true);
   RegistryKey yenianahtar = anahtar.CreateSubKey("RegistryOrnegi");
   yenianahtar.SetValue("CalismaSayisi", "1");
   MessageBox.Show("Anahtar oluşturuldu");
  }
 }
}
```



Görüldüğü gibi anahtarımızı ve değerimizi oluşturduk.

Örneğimizi biraz daha geliştirip değer mevcutsa değeri 1 arttıran, mevcut değilse oluşturan bir örnek hazırlayalım.

```
private void btnDegerOlustur_Click(object sender, EventArgs e)
{
  RegistryKey anahtar = Registry.CurrentUser.OpenSubKey ("Software", true);
  if (anahtar.OpenSubKey("RegistryOrnegi") != null)
  {
    byte tekrar=Convert.ToByte(anahtar.OpenSubKey("RegistryOrnegi",
true).GetValue
("CalismaSayisi").ToString());
    anahtar.OpenSubKey("RegistryOrnegi", true).SetValue
   ("CalismaSayisi", tekrar+1, RegistryValueKind.String);
  }
  else
  {
    anahtar.CreateSubKey("RegistryOrnegi");
```

```
    anahtar.OpenSubKey("RegistryOrnegi", true).SetValue
        ("CalismaSayisi", 1, RegistryValueKind.String);
    MessageBox.Show("Program Bu Bilgisayarda İlk Defa Çalışıyor");
  }
  anahtar.Close();
}
```

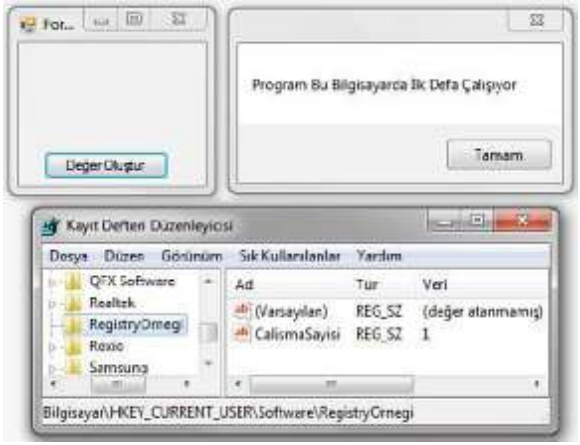
Kodumuzu ilk çalıştırdığımızda…

```
RegistryKey anahtar = Registry.CurrentUser.OpenSubKey ("Software", true);
if (anahtar.OpenSubKey("RegistryOrnegi") != null)
```

… satırı ile **`RegistryOrnegi`** adlı değer kontrol edilerek, eğer oluşturulmamışsa…

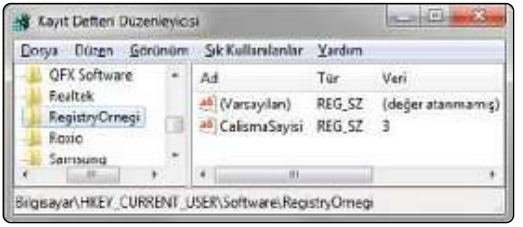
```
anahtar.CreateSubKey("RegistryOrnegi");
anahtar.OpenSubKey("RegistryOrnegi", true).SetValue
   ("CalismaSayisi", 1,RegistryValueKind.String);
MessageBox.Show("Program Bu Bilgisayarda İlk Defa Çalışıyor");
```

… satırları ile **`CreateSubKey`** metodu kullanılarak aşağıda görüldüğü gibi **`RegistryOrnegi`** adlı anahtar oluşturulur ve **`SetValue`** metodu ile bu anahtar altında ilk değer olarak `1` değeri girilmekte, "*Program Bu Bilgisayarda İlk Defa Çalışıyor*" mesajı görüntülenmektedir.



```
byte tekrar=Convert.ToByte(anahtar.OpenSubKey("RegistryOrnegi",
     true).GetValue("CalismaSayisi").ToString());
anahtar.OpenSubKey("RegistryOrnegi", true).SetValue("CalismaSayisi",
     tekrar+1,RegistryValueKind.String);
```

Bundan sonraki çalıştırmalarımızda ise; **CalismaSayisi** adlı değer mevcut olduğundan butonumuza her tıklandığında, bu değerin içeriği **tekrar** isimli değişkene aktarılarak bir arttırılıp **SetValue** metodu ile yeniden yazılacaktır.



anahtar.close() metodu ile de registry içerisinde açmış olduğumuz anahtarları kapatıyoruz.

Anahtar ve Değer Sayısını Bulma

Bir anahtarın altında bulunan anahtarların sayısını bulmak için **SubKeyCount** özelliği kullanılırken, bir anahtar içerisinde mevcut olan değerlerin sayısını bulmak için ise, **ValueCount** özelliği kullanılır.

Örneğin; bir önceki uygulamamızda yaptığımız, bir değerin var olup olmadığını **GetValue** metodu ile kontrol etmek yerine, **ValueCount** özelliğinden faydalanarak **RegistryOrnegi** adlı anahtarın altındaki değerlerin sayısını da kontrol edebiliriz. Örneğimizi **ValueCount** özelliğini kullanacağımız şekilde değiştirelim.

```
private void btnDegerOlustur_Click(object sender, EventArgs e)
{
  RegistryKey anahtar = Registry.CurrentUser.OpenSubKey ("Software", true);
  int ds = Registry.CurrentUser.OpenSubKey("Software").
         OpenSubKey("RegistryOrnegi").ValueCount;
  if (ds >= 1)
  {
```

```
    byte tekrar=Convert.ToByte(anahtar.
         OpenSubKey("RegistryOrnegi", true).GetValue
         ("CalismaSayisi").ToString());
    anahtar.OpenSubKey("RegistryOrnegi", true).SetValue
         ("CalismaSayisi", tekrar+1, RegistryValueKind.String);
  }
  else
  {
    anahtar.CreateSubKey("RegistryOrnegi");
    anahtar.OpenSubKey("RegistryOrnegi", true).SetValue
          ("CalismaSayisi", 1, RegistryValueKind.String);
    MessageBox.Show("Program Bu Bilgisayarda İlk Defa Çalışıyor");
  }
  anahtar.Close();
}
Kodumuzda…
int ds = Registry.CurrentUser.OpenSubKey("Software").
         OpenSubKey("RegistryOrnegi").ValueCount;
if (ds >= 1)
```

… satırlarında **`Software`** anahtarı altında bulunan değer sayısını **`ValueCount`** özelliği ile elde ederek, eğer değer sayısı bir ya da birden fazla ise sonraki kod satırlarında **`CalismaSayisi`** adlı değerin arttırılmasını sağlıyoruz.

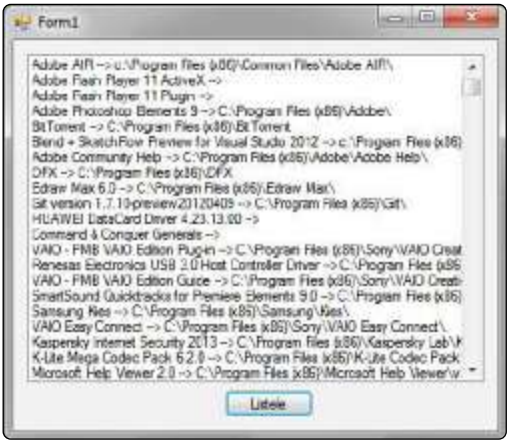
Anahtar ve Değer Listesini Alma

`GetSubKeyNames` metodu ile alt anahtarların listesini dizi şeklinde elde ederken, **`GetValueNames`** metodu ile de bir anahtarın altındaki değerlerin listesini bir dizi şeklinde elde edebiliriz.

Örnek olarak bilgisayarda yüklü olan programların adlarını ve bulundukları klasörleri listeleyen bir örnek yapalım. Bilgisayarda yüklü olan programlar *HKey_Local_Machine\Software\Microsoft\Windows\CurrentVersion\Uninstall* anahtarı altında listelenmektedir. Biz bu anahtarın altındaki alt anahtarları listeleyerek bu anahtarların özelliklerinden, isimlerini listelemek için **`DisplayName`** özelliğini, yüklü oldukları klasörleri listelemek için ise **`InstallLocation`** özelliklerini kullanacağız.

```
private void btnListele_Click(object sender, EventArgs e)
{
  RegistryKey anahtar = Registry.LocalMachine.OpenSubKey
    ("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall");
  foreach(string alt_anahtar_adı in anahtar.GetSubKeyNames())
  {
    RegistryKey alt_anahtar = anahtar.OpenSubKey(alt_anahtar_adı);
    if (alt_anahtar.GetValue("DisplayName") != null)
   listBox1.Items.Add(alt_anahtar.GetValue("DisplayName")
  + " --> " + alt_anahtar.GetValue("InstallLocation"));
  }
}
```

Uygulamamızı çalıştırdığımızda bilgisayarda yüklü olan uygulamaların listelendiğini göreceğiz. **`InstallLocation`** bilgisine erişemediğimiz uygulamaların yüklü oldukları klasör bilgileri görüntülenmeyecektir.



Yüklü olan programlarla ilgili registry içerisinden ulaşabileceğimiz tüm bilgiler şunlardır.

- **`Contact`**: Destek için iletişime geçilecek kişi.
- **`Display Name`**: Programın görünen ismi.
- **`Display Version`**: Görünen versiyon bilgisi.
- **`Estimated Size`**: Tahmini boyutu.

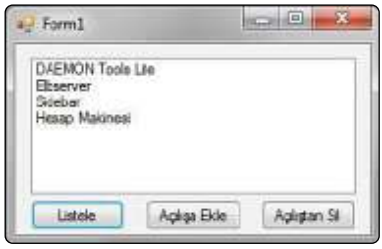
- **`Help Link`**: Yardım linki.
- **`Help Telephone`**: Yardım numarası.
- **`Installed Date`**: Yüklenme tarihi.
- **`Install Location`**: Yüklendiği yer.
- **`Install Source`**: Yüklendiği klasör.
- **`Language`**: Uygulamanın dili.
- **`Local Package`**: MSI File lokasyon bilgisi.
- **`ModifyPath`**: Şunun gibi bir bilgi döndürür. *MsiExec.exe/X{8FB53850-246A-3507-8ADE-0060093FFEA6}*
- **`Publisher`**: Uygulamanın yayıncısı.
- **`Readme`**: Uygulama için hazırlanmış olan Readme dosyasının yeri.
- **`UninstallString`**: ModifyPath'deki veriye benzer bir veriyi geriye döndürür.
- **`URLInfoAbout`**: Ürün hakkında daha fazla bilgi için ziyaret edilecek web adresi.
- **`URLUpdateInfo`**: Güncelleştirmeler için ziyaret edilecek web adresi.
- **`Version`**: Yazılımın versiyon bilgisi.

Anahtar ve Değerleri Silme

Registry içerisindeki bir alt anahtarı silmek için **`DeleteSubKey`** metodu kullanılırken, bir alt anahtarı içerisindeki tüm alt anahtar ve özellikleri ile beraber silmek için ise **`DeleteSubKeyTree`** metodu kullanılır. Bir değeri silmek için ise **`DeleteValue`** metodu kullanılır.

Örnek olarak bilgisayarın açılışında otomatik olarak çalışan programların listesini elde ederek, buraya istediğimiz bir programı ekleyen ya da silebilen bir örnek uygulama yapalım. Örneğimizde açılışa Windows **Hesap Makinası**nı ekleyip kaldıracağız. Bilgisayarın açılışında otomatik olarak çalışan programlar registry içerisinde *HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run* anahtarı altında bulunmaktadır.

```
using System;
using Microsoft.Win32;
using System.Security.AccessControl;
using System.Windows.Forms;
namespace Registry_Delete_Value
{
 public partial class Form1 : Form
 {
  public Form1()
  {
    InitializeComponent();
  }
  private void btnListele_Click(object sender, EventArgs e)
  {
   RegistryKey anahtar = Registry.CurrentUser.OpenSubKey
     ("Software\\Microsoft\\Windows\\CurrentVersion\\Run");
   listBox1.Items.Clear();
   foreach (string deger_adı in anahtar.GetValueNames())
     listBox1.Items.Add(deger_adı);
  }
  private void btnEkle_Click(object sender, EventArgs e)
  {
   RegistryKey anahtar = Registry.CurrentUser.OpenSubKey
     ("Software\\Microsoft\\Windows\\CurrentVersion\\Run",
     RegistryKeyPermissionCheck.ReadWriteSubTree,
     RegistryRights.SetValue);
   anahtar.SetValue("Hesap Makinesi",
     "C:\\Windows\\System32\\calc.exe",
     RegistryValueKind.String);
   anahtar.Close();
  }
  private void btnSil_Click(object sender, EventArgs e)
  {
   RegistryKey anahtar = Registry.CurrentUser.OpenSubKey
     ("Software\\Microsoft\\Windows\\CurrentVersion\\Run",
     RegistryKeyPermissionCheck.ReadWriteSubTree,
  RegistryRights.SetValue);
   anahtar.DeleteValue("Hesap Makinesi", false);
   anahtar.Close();
  }
 }
}
```



Kodumuzu çalıştırıp **Açılışa Ekle** düğmesine basarak, bilgisayarın açılışına **Hesap Makinası** uygulamasını ekleyelim. Daha sonra **Listele** düğmesine basarak açılış listesine **Hesap Makinası** uygulamasının eklendiğini görelim.

Örneğimizde görüldüğü gibi `OpenSubKey` metodunda bazı parametreler kullandık. Bu parametrelerden biri, `RegistryKeyPermissionCheck` parametresidir. Bu parametre anahtara verilen okuma yazma izinleri ile ilgilidir. Şu değerleri alabilir:

- `ReadSubTree`: Geçerli anahtar okuma modunda açılır.
- `ReadWriteSubTree`: Geçerli anahtar okuma ve yazma modunda açılır.

Diğer parametre ise; `Security.AccessControl.RegistryRights` parametresidir. Registry nesnelerinin erişim haklarını belirler. Bu parametreyi kullanabilmek için `System.Security.AccessControl` sınıfını projemize dahil etmememiz gerekmektedir. Alabileceği bazı değerler şunlardır:

- `SetValue`: Bir değer oluşturma, düzenleme ya da silme işlemlerinin gerçekleştirilebilmesini sağlar.
- `CreateSubKey`: Bir alt anahtar oluşturulabilmesini sağlar.
- `EnumerateSubKeys`: Bir alt anahtarın altındaki anahtarların listesine erişimi sağlar.
- `Notify`: Bir anahtar üzerinde değişiklik yapılabilmesini sağlar.
- `ReadKey`: Anahtar ve değerlerin okunabilmesini, listelenilebilmesini sağlar.
- `WriteKey`: Anahtar ve değer oluşturup, silme, düzenleme işlemlerinin gerçekleştirilebilmesini sağlar.
- `Delete`: Bir anahtarın silinebilmesini sağlar.
- `FullControl`: Registry üzerinde tam kontrol elde edilerek tüm erişim haklarını sağlar.

Örneğimizde DeleteValue parametresini…

```
anahtar.DeleteValue("Hesap Makinesi", False)
```

… şeklinde **Boolean** (*mantıksal*) bir parametre vererek kullandık. **DeleteValue**, **DeleteSubKey** ya da **DeleteSubKeyTree** metotlarının üçünde de **true** ya da **false** şeklinde mantıksal parametre kullanabiliriz. Bu parametreye **true** değeri verildiğinde alt anahtar ya da değer bulunamadığında bir hata durumu oluşacak ve bir istisna oluşturulacaktır. **false** verildiğinde ise herhangi bir istisna oluşturulmayacaktır.

```
anahtar.Close();
```

Bu satırında ise, **Close** parametresi yardımı ile kullandığımız registry anahtarını kapattık.

WORD VE EXCEL PROGRAMLARINI KULLANMA

24

C# .Net veya diğer .NET ailesi ile ofis uygulamaları geliştirmenin başlıca iki yöntemi vardır. Birincisi **VBA** (*Visual Basic For Application*) yapısını kullanmak, ikinci yöntem ise **VSTO** (*Visual Studio Tools For Office*) isimli ofis yazılımı geliştirme ortamını kullanmaktır.

VBA, ofis uygulamalarını geliştirebilmek ve özelleştirebilmek için kullanılan bir ortak dildir. Bu sayede kullanıcılar VBA'ya destek veren ofis ürünleri (Word, Excel, Powerpoint,Access gibi), Microsoft Messenger, AutoCAD gibi birçok programı özelleştirebilir ya da geliştirebilir.

Word Programini Kullanma

Belge Açma, Kapama ve Kaydetme Olayları

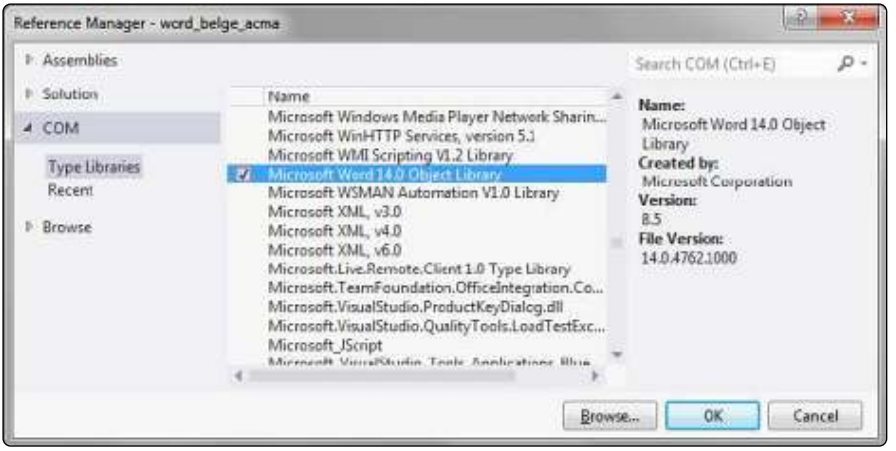
Yeni bir Word belgesi oluşturmak için `Documents.Add` metodu kullanılır.

Eğer daha önceden mevcut olan bir word belgesini açacaksak `Documents.Open(dosya adı)` metodu kullanılır.

Belgeyi kapatmak için `Close`, Word uygulamasını kapatmak için ise `Quit` metodu kullanılır. Ayrıca bu işlemler bittikten sonra `null` ataması ile de Word belgemizi ve Word uygulamamızı bellekten temizlememiz gerekmektedir.

Örneğimiz için yeni bir proje açalım ve iki adet buton ekleyelim. Birinci buton, Word belgemizi açmak, ikincisi ise kapatmak için olacak.

Project menüsünden **Add Reference** seçeneği ile gelen pencereden **COM** başlığı altından, **Microsoft Word 14.0 Object Library** seçeneğini seçip **OK** tuşuna basıyoruz. Elbette kullandığınız Word programının sürümüne göre buradaki 14 rakamı değişecektir.



Daha sonra referansımızı projemize dâhil ediyoruz.

```
using Word = Microsoft.Office.Interop.Word;
```

Kodumuz aşağıdaki gibi olacak:

```
using System;
using System.Windows.Forms;
using Word = Microsoft.Office.Interop.Word;
namespace word_belge_acma
{
 public partial class Form1 : Form
 {
  public Form1()
  {
     InitializeComponent();
  }
  Word.Application WordApp = new Word.Application();
  Word.Document WordDoc = new Word.Document();
  private void btnYeniBelge_Click(object sender, EventArgs e)
  {
```

```
    WordDoc = WordApp.Documents.Add();
    WordApp.Visible = true;
  }
  private void btnKapat_Click(object sender, EventArgs e)
  {
    WordDoc.Close();
    WordApp.Quit() ;

    WordDoc = null ;
    WordApp = null;
  }
 }
}
```

Programımızı çalıştırdığımızda birinci butona tıkladığımızda, yeni bir Word belgesi açılacak, ikinci butona tıkladığımızda ise kapatılacaktır.

Formumuzun tanımlama kısmında **WordApp** ve **WordDoc** nesnelerimizi tanımlıyoruz. **WordApp**, Word uygulaması, **WordDoc** ise Word belgesidir.

Kodumuzda…

```
Word.Application WordApp = new Word.Application();
```

… satırıyla, Word nesnesi oluşturuyoruz.

```
Word.Document WordDoc = new Word.Document();
```

… satırıyla Word uygulamasını açıyoruz.

```
WordDoc = WordApp.Documents.Add();
```

… satırıyla yeni bir Word belgesi oluşturuyoruz.

```
WordApp.Visible = true;
```

… satırıyla oluşturduğumuz Word belgesini görüntülüyoruz.

```
WordDoc.Close();
```

… satırı ile Word belgesini kapatıyoruz.

```
WordApp.Quit();
```

… satırı ile Word uygulamasını kapatıyoruz.

```
WordDoc = null ;
```

… satırı ile Word belgesini bellekten siliyoruz.

```
WordApp = null;
```

… satırı ile ise Word uygulamasını bellekten siliyoruz.

Kaydetme olayı için **`Save`**, farklı biçimde kaydetmek için **`SaveAs`** metotları kullanılır. Ayrıca **`Close`** metodunun parametreleri kullanılarak, kayıt işleminin yapılması için farklı kaydet penceresi görüntülenerek kapatma işlemi yapılır ya da bu pencere görüntülenmeyip değişiklikler kaydedilmeden kapatma işlemi gerçekleştirilir .

`Close` parametresi isteğe bağlı olarak iki değer alır.

- **`wdDoNotSaveChanges`**: Yapılan değişiklikler kaydedilmeden kapatma işlemi yapılır.
- **`wdSaveChanges`**: Yapılan değişikliklerin kaydedilebilmesi için farklı kaydet penceresi görüntülenerek kapatma işlemi yapılır.

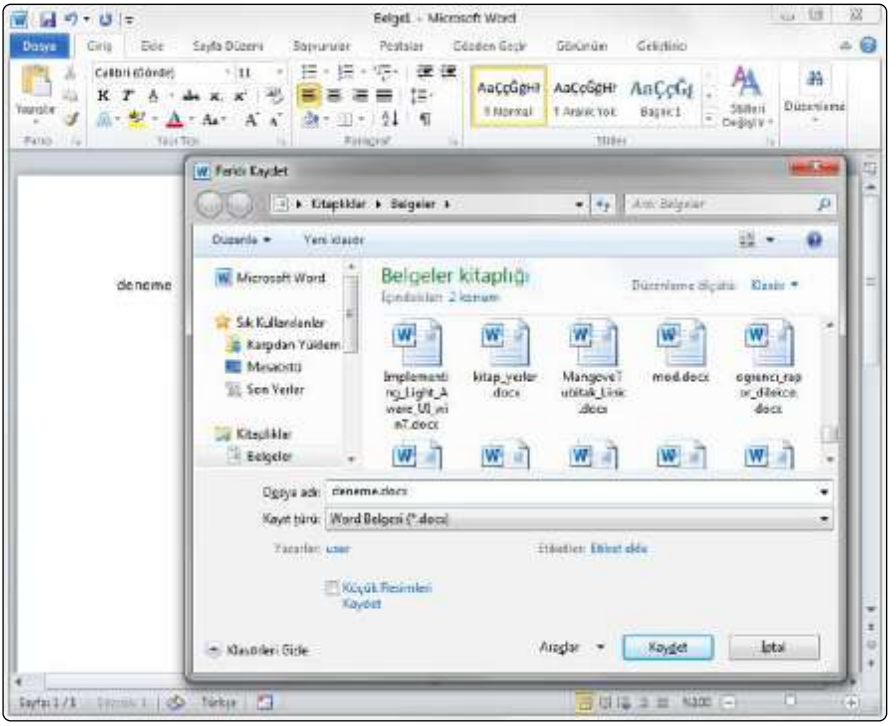
`Save` metodunu kullanarak kaydetme işlemi ile ilgili bir örnek yapalım. Bu örneğimize bir de hata kontrolünü de ekleyelim. Çünkü eğer bir belge açılamadı ise kayıt ya da kapatma olayını yapmaya kalktığımızda hata verecektir.

Form üzerine **`btnYeniBelge`**, **`btnKaydet`**, **`btnKapat`** adı ile 3 adet buton ekliyoruz. Kodumuz aşağıdaki gibi olacak:

```
using System;
using System.Windows.Forms;
using Word = Microsoft.Office.Interop.Word;
namespace word_belge_acma
{
 public partial class Form1 : Form
 {
```

```
  public Form1()
  {
   InitializeComponent();
  }
  Word.Application WordApp = new Word.Application();
  Word.Document WordDoc = new Word.Document();
  private void btnYeniBelge_Click(object sender, EventArgs e)
  {
   WordDoc = WordApp.Documents.Add();
   WordApp.Visible = true;
  }
  private void btnKapat_Click(object sender, EventArgs e)
  {
   try
   {
     WordDoc.Close();
     WordApp.Quit();
     WordDoc = null;
     WordApp = null;
   }
   catch
   {
     MessageBox.Show("Önce belge açmanız gerekiyor");
     return;
   }
  }
  private void btnKaydet_Click(object sender, EventArgs e)
  {
   try
   {
     WordDoc.Save();
   }
   catch
   {
     MessageBox.Show("Önce belge açmanız gerekiyor");
     return;
   }
  }
 }
}
```

Projemizi çalıştırdığımızda yeni bir belge açılacak ve açılan belgede değişiklik yaparak, onu kapat butonu ile kapatmaya çalıştığımızda karşımıza farklı kaydet penceresi gelecektir.



`WordDoc.Save();` metodu yerine `WordDoc.SaveAs();` kullanırsak, farklı kaydet işlemi gerçekleştirilir.

Uygulamamızı bir adım ilerleterek belgede değişiklik yapılsa bile, bize sorulmadan otomatik olarak kayıt işlemi yapılarak, kapatma işlemi yapılmasını sağlayalım. Bunun için `SaveAs` metodunun `FileName` isimli argümanı isteğe bağlı olarak kullanılabilir.

Yukarıda yaptığımız örneğimizin kodundaki `SaveAs` satırını değiştireceğiz.

Buna göre btnkaydet isimli butonumuzun yeni kodu şu şekilde olacak:

```
private void btnKaydet_Click(object sender, EventArgs e)
{
 try
 {
  WordDoc.SaveAs(Application.StartupPath + "\\deneme.doc");
 }
 catch
 {
  MessageBox.Show("Önce belge açmanız gerekiyor");
  return;
 }
}
```

Uygulamamızı çalıştırıp yeni bir belge oluşturup üzerinde değişiklik yapıp kapat dediğimizde, karşımıza farklı kaydet penceresi çıkmadan kayıt işleminin yapılıp belgenin kapatıldığı görülecektir. Belge, **`Application.StartupPath`** komutu ile proje klasörünüz içerisinde, ***Bin/Debug*** alt klasörleri içerisinde **deneme.doc** adı ile kaydedilecektir.

Daha önceden kaydedilmiş olan belgeleri açma işlemi ile ilgili bir örnek yapalım. Öncelikle belge açma penceresini görüntüleyebilmemiz için **`OpenFileDialog`** nesnesini eklememiz gerekiyor. Daha sonra formumuza bir buton ekleyelim ve ismini **`btnBelgeAc`** verelim. Butonumuza şu kodları yazıyoruz:

```
using System;
using System.Windows.Forms;
using Word = Microsoft.Office.Interop.Word;
namespace word_kayıtlı_belge_acma
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
  }
  private void btnBelgeAc_Click(object sender, EventArgs e)
  {
   openFileDialog1.Title = "Belge seçiniz...";
```

```
    openFileDialog1.Filter =
      "Ofis 97-2003 Belgeleri (*.doc)|*.doc|Ofis 2007-2010 Belgeleri
      (*.docx)|*.docx";
    openFileDialog1.FilterIndex = 1;
    if (openFileDialog1.ShowDialog() == DialogResult.Cancel)
       MessageBox.Show("Dosya seçilmedi");
    else
    {
     Word.Application WordApp = new Word.Application();
     Word.Document WordDoc = new Word.Document();
     WordDoc = WordApp.Documents.Open(openFileDialog1.FileName);
     WordApp.Visible = true;
    }
   }
  }
}
```

Kodumuzda **Documents.Open** metodunu kullanarak, **openFileDialog** penceresi içerisinde seçilmiş olan Word belgesini görüntülüyoruz.

Yazı Ekleme ve Düzenleme İşlemleri

Word belgesi içerisine yazı ekleme işlemi **Range**, **Sentences**, **Paragraph** ya da **Selection** nesnesi kullanılarak birden fazla şekilde yapılabilir.

Range Nesnesi Kullanılarak

Range nesnesi, belirtilen aralıklarda işlem yapmak amacı ile kullanılır. **Range** nesnesi ile 2 metot kullanılabilir. Bunlar; **InsertBefore**, **InsertAfter** metotlarıdır.

InsertAfter metodunda, ekleme işlemi belgede mevcut olan yazının sonuna eklenir.

InsertBefore metodunda ise; ekleme işlemi belgede mevcut olan yazının başına eklenir.

Bu metoda göre bir örnek yapalım. Form üzerine **btnyaziekle** adıyla bir adet buton, bir adet de metin kutusu ekleyelim.

Kodumuz ise aşağıdaki gibi olacak:

```
using System;
using System.Windows.Forms;
using Word = Microsoft.Office.Interop.Word;
namespace word_yazi_ekleme_range
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
  }
  private void btnyaziekle_Click(object sender, EventArgs e)
  {
   Word.Application WordApp = new Word.Application();
   Word.Document WordDoc = new Word.Document();
   WordDoc = WordApp.Documents.Add();
   WordDoc.Range().InsertAfter(textBox1.Text);
   WordApp.Visible = true;
  }
 }
}
```

Programımızı çalıştırdığımızda görürüz ki, ekleme işlemi belgedeki yazının sonuna yapılacaktır. Eğer ekleme işleminin belgedeki yazının başına yapılmasını istiyorsak;

```
WordDoc.Range().InsertAfter(textBox1.Text);
```

… satırını…

```
WordDoc.Range().InsertBefore(textBox1.Text);
```

… şeklinde değiştirmemiz gerekir.

Sentences Nesnesi Kullanılarak

Sentences nesnesi, belge içerisindeki cümleleri ifade etmektedir. Ancak dikkat edilmesi gereken husus şudur ki; Eğer belge içerisinde hiçbir yazı yok ise, **sentences[1]** şeklinde ekleme yapmalıyız. Eğer birden fazla cümle mevcut ise; **sentences[cümle numarası]** şeklinde ekleme yapabiliriz.

Sentences nesnesini kullanarak bir örnek yapalım: Form üzerinde yine **btnyaziekle** adıyla bir adet buton, bir adet de text kutusu olacak.

Kodumuz şu şekilde olacak:

```
private void btnyaziekle_Click(object sender, EventArgs e)
{
 Word.Application WordApp = new Word.Application();
 Word.Document WordDoc = new Word.Document();
 WordDoc = WordApp.Documents.Add();
 WordApp.Visible = true;

 int cumlesayisi = WordDoc.Sentences.Count;
 if (cumlesayisi > 0)
   WordDoc.Sentences[cumlesayisi].Text = textBox1.Text;
 else
   WordDoc.Sentences[1].Text = textBox1.Text;
}
```

Örneğimizde aşağıdaki komut satırı ile cümle sayısını tespit ederek, bunu **cumlesayisi** adlı değişkene aktardık.

```
int cumlesayisi = WordDoc.Sentences.Count;
```

Eğer cümle sayısı 0'dan büyük ise, yani belge içerisinde mevcut cümle varsa; aşağıdaki komut satırı çalıştırılarak belge içerisindeki son cümle **textBox1** içerisine yazılan cümle ile değiştirilmektedir.

```
WordDoc.Sentences[cumlesayisi].Text = textBox1.Text;
```

Eğer belge içerisinde mevcut cümle yoksa; aşağıdaki komut satırı ile belge içerisine ilk cümle olarak **textBox1** içerisindeki cümleyi aktardık.

```
WordDoc.Sentences[1].Text = textBox1.Text;
```

Paragraph Nesnesi Kullanılarak

Paragraph nesnesi, belge içerisindeki paragrafları temsil etmektedir. **Range** metodu yardımı ile kullanılabilir.

Paragraph nesnesini kullanarak kayıtlı bir Word belgesini açarak ekleme yapan bir örnek yapalım. Form üzerinde yine **btnyaziekle** adıyla bir adet buton, bir adet de text kutusu olacak. Ayrıca kayıtlı bir Word belgesini açmak için **openFileDialog** nesnesi de ekleyeceğiz.

Kodumuz şu şekilde olacak:

```
private void btnyaziekle_Click(object sender, EventArgs e)
{
  openFileDialog1.Title = "Belge seçiniz...";
  openFileDialog1.Filter =
    "Ofis 97-2003 Belgeleri (*.doc)|*.doc|Ofis 2007-2010 Belgeleri
    (*.docx)|*.docx";
  openFileDialog1.FilterIndex = 1;
  if (openFileDialog1.ShowDialog() == DialogResult.Cancel)
    MessageBox.Show("Dosya seçilmedi");
  else
  {
    Word.Application WordApp = new Word.Application();
    Word.Document WordDoc = new Word.Document();
    WordDoc = WordApp.Documents.Open(openFileDialog1.FileName);
    WordApp.Visible = true;

    int paragrafsayisi = WordDoc.Paragraphs.Count;
    if (paragrafsayisi > 0)
      WordDoc.Paragraphs[paragrafsayisi].Range.Text = textBox1.Text;
    else
      WordDoc.Paragraphs[1].Range.Text = textBox1.Text;
  }
}
```

Örneğimizde aşağıdaki komut satırı ile paragraf sayısını tespit ederek, bunu **paragrafsayisi** adlı değişkene aktardık.

```
int paragrafsayisi = WordDoc.Paragraphs.Count;
```

Eğer paragraf sayısı 0'dan ise, yani belge içerisinde mevcut paragraf varsa…

```
WordDoc.Paragraphs[paragrafsayisi].Range.Text = textBox1.Text;
```

…. komut satırı çalıştırılarak belge içerisindeki son paragraf, **`textBox1`** içerisine yazılan cümle ile değiştirilmektedir.

Eğer belge içerisinde mevcut paragraf yoksa; aşağıdaki komut satırı ile belge içerisine ilk paragraf olarak **`textbox1`** içerisindeki cümleyi aktardık.

```
WordDoc.Paragraphs[1].Range.Text = textBox1.Text;
```

Selection Nesnesi Kullanılarak

`Selection` nesnesi, belgedeki yazının seçili olan kısmını temsil eder. **`TypeText`** metodu ile birlikte kullanılarak yazı ekleme işlemi yapılabilir.

`Selection` nesnesini kullanarak kayıtlı bir Word belgesini açıp, **TextBox** içerisindeki yazıyı belgeye ekleyelim. Form üzerinde yine **`btnyaziekle`** adıyla bir adet buton, bir adet de text kutusu olacak.

Kodumuz şu şekilde olacak:

```
private void btnyaziekle_Click(object sender, EventArgs e)
{
  openFileDialog1.Title = "Belge seçiniz...";
  openFileDialog1.Filter =
    "Ofis 97-2003 Belgeleri (*.doc)|*.doc|Ofis 2007-2010 Belgeleri
    (*.docx)|*.docx";
  openFileDialog1.FilterIndex = 1;
  if (openFileDialog1.ShowDialog() == DialogResult.Cancel)
    MessageBox.Show("Dosya seçilmedi");
  else
  {
    Word.Application WordApp = new Word.Application();
    Word.Document WordDoc = new Word.Document();
    WordDoc = WordApp.Documents.Open(openFileDialog1.FileName);
    WordApp.Visible = true;

  if (textBox1.Text != "")
    WordApp.Selection.TypeText(textBox1.Text);
  }
}
```

Kodumuzda, açtığımız belge içerisinde seçili kısım varsa; seçili kısım **TextBox** içerisindeki metinle değiştirilmekte, seçili kısım yoksa belgenin en başına **TextBox** içerisindeki metin eklenmektedir.

`Selection` nesnesinin **`TypeText`** özelliği ile metin üzerinde işlem yapabilirken, **`TypeParagraph`** nesnesi ile paragraf üzerinde işlem yapabiliriz.

Önizleme Yapma ve Yazdırma İşlemleri

Baskı önizleme işlemi için **`PrintPreview()`** metodunu kullanırken, yazdırma işlemi için **`PrintOut()`** metodunu kullanırız. Örneğimizde bir Word belgesini yazdırma ve baskı ön izleme işlemlerini yapalım.

Bir önceki örneğimizdeki form üzerine **`btn_baski_onizleme`** ve **`btn_yazdir`** adı ile 2 adet buton ekleyelim ve kodumuzu aşağıdaki şekilde yazalım:

```
Word.Application WordApp = new Word.Application();
Word.Document WordDoc = new Word.Document();
private void btnyaziekle_Click(object sender, EventArgs e)
{
 WordDoc = WordApp.Documents.Add();
 WordApp.Visible = true;

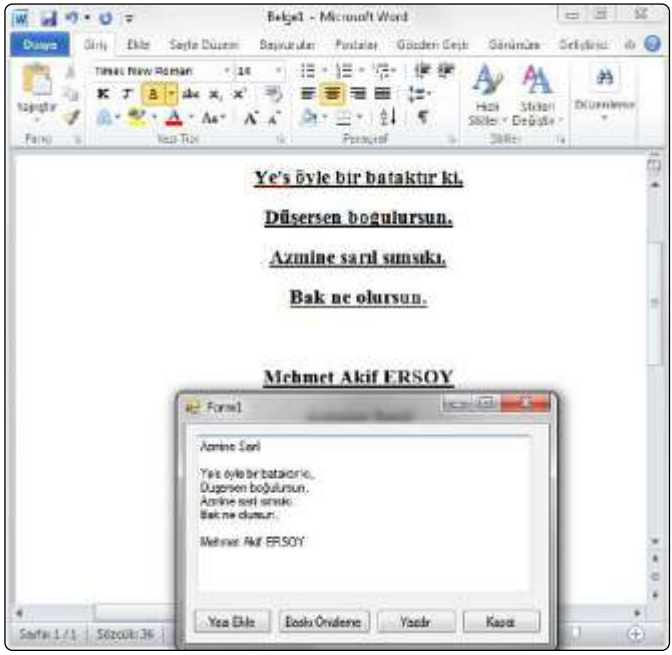
 if (!string.IsNullOrEmpty(textBox1.Text))
 {
  WordDoc.Paragraphs[1].Alignment =
    Word.WdParagraphAlignment.wdAlignParagraphCenter;

  WordApp.Selection.Font.Name = "Times New Roman";
  WordApp.Selection.Font.Size = 14;
  WordApp.Selection.Font.Underline =
    Word.WdUnderline.wdUnderlineSingle;
  WordApp.Selection.BoldRun();

  //Textbox içerisindeki yazıyı ekliyoruz.
  WordApp.Selection.TypeText(textBox1.Text);
  //Paragrafı seçiyoruz
  WordApp.Selection.TypeParagraph();
  //Yazının kalınlık özelliğini kaldırıyoruz
  WordApp.Selection.BoldRun();
  //Textbox içerisindeki yazıyı yeniden ekliyoruz.
  WordApp.Selection.TypeText(textBox1.Text);
```

```
        }
       }
       private void btn_baski_onizleme_Click(object sender, EventArgs e)
       {
        try
        {
         WordDoc.PrintPreview();
        }
        catch
        {
         MessageBox.Show("Önce belge açmanız gerekiyor");
         return;
        }
       }
       private void btn_yazdir_Click(object sender, EventArgs e)
       {
        try
        {
         WordDoc.PrintOut();
        }
        catch
        {
         MessageBox.Show("Önce belge açmanız gerekiyor");
         return;
        }
       }
       private void btnkapat_Click(object sender, EventArgs e)
       {
        try
        {
         WordDoc.Close();
         WordApp.Quit();
         WordDoc = null;
         WordApp = null;
        }
        catch
        {
         MessageBox.Show("Önce belge açmanız gerekiyor");
         return;
        }
       }
```

Programımızı çalıştırıp text kutusu içerisine herhangi bir yazı yazalım. **Yazı Ekle** düğmesine basıp, baskı önizleme düğmesine basalım. Sonuç aşağıdaki gibidir:




```
WordDoc.Paragraphs[1].Alignment =
Word.WdParagraphAlignment.wdAlignParagraphCenter;
```

Bu satırda **`Paragraphs`** nesnesinin **`Alignment`** özelliği ile Word belgesi içerisindeki paragrafın hizalanma şeklini belirtiyoruz. Bu özelliğe **`Word.WdParagraphAlignment`** nesnesinin özellikleri ile atama yapılabilir. Atanabilecek değerler şunlardır:

Özellik	Tanımı
`wdAlignParagraphCenter`	Ortaya hizalama.
`wdAlignParagraphDistribute`	Paragrafı oluşturan karakterler paragrafın her tarafını dolduracak şekilde dağıtılır.
`wdAlignParagraphJustify`	Tam hizalama.
`wdAlignParagraphJustifyHi`	Büyük harfler, aralarında daha az boşluk bırakılarak yazılır.

Özellik	Tanımı
wdAlignParagraphJustifyLow	Küçük harfler, aralarında daha az boşluk bırakılarak yazılır.
wdAlignParagraphJustifyMed	Bütün harfler, aralarında ortalama bir boşluk bırakılarak yazılır.
wdAlignParagraphLeft	Sola dayalı hizalama.
wdAlignParagraphRight	Sağa dayalı hizalama.

```
WordApp.Selection.Font.Name = "Times New Roman";
```

… satırı ile seçili olan yazının yazı tipini **Times New Roman** olarak değiştiriyoruz.

```
WordApp.Selection.Font.Size = 14;
```

… satırı ile seçili olan yazının boyutunu **14** olarak belirliyoruz.

```
WordApp.Selection.Font.Underline = Word.WdUnderline.wdUnderlineSingle;
```

… satırı ile seçili olan yazının altını çizili yapıyoruz. Yazının altını çizili olarak belirlerken değişik biçimler kullanmamız mümkündür. Kullanabileceğimiz biçimler şunlardır:

Özellik	Tanımı
wdUnderlineDashHeavy	Kalın kesik çizgiler.
wdUnderlineDashLong	Uzun kesik çizgiler.
wdUnderlineDashLongHeavy	Kalın ve uzun kesik çizgiler.
wdUnderlineDotDash	Nokta ve çizgi.
wdUnderlineDotDashHeavy	Kalın nokta ve çizgi.
wdUnderlineDotDotDash	Nokta nokta çizgi .
wdUnderlineDotDotDashHeavy	Kalın nokta nokta çizgi.
wdUnderlineDotted	Noktalar.
wdUnderlineDottedHeavy	Kalın noktalar.
wdUnderlineDouble	Çift çizgi.
wdUnderlineNone	Altı çizili değil.
wdUnderlineSingle	Tek çizgi. Varsayılan değerdir.
wdUnderlineThick	Tek ince çizgi.

Özellik	Tanımı
wdUnderlineWavy	Dalgalı tek çizgi.
wdUnderlineWavyDouble	Dalgalı çift çizgi.
wdUnderlineWavyHeavy	Dalgalı kalın çizgi.
wdUnderlineWords	Yalnızca kelimelerin altı çizilir. Arada kalan boşlukların altı çizilmez.

```
WordApp.Selection.BoldRun();
```

… satırı ile seçili olan yazıyı kalın yapıyoruz. Aynı işlemi seçili olan aynı kısım üzerinde gerçekleştirirsek, yazının kalınlık özelliği kaldırılır.

Tablo İşlemleri

Tables nesnesi kullanılarak bir Word belgesine tablo eklenebilir ya da eklenmiş olan tablolar üzerinde düzenleme yapılabilir.

Wordnesnesi.Tables.Add şeklinde bir kullanım ile bir Word belgesi içerisine tablo eklemesi yapılabilir.

Wordnesnesi.Range.Tables(tablo numarası) şeklinde bir kullanım ile de eklenmiş olan bir tablo içerisine erişilir.

Dokümannesnesi.Range.Tables(tablo numarası).Columns(sütun numarası) ile tablo içerisindeki sütunlara erişilir.

Dokümannesnesi.Range.Tables(tablo numarası).Rows(satır numarası) ile tablo içerisindeki satırlara erişilir.

Dokümannesnesi.Range.Tables(tablo numarası).Cell(satır numarası,sütun numarası) ile tablo içerisindeki hücrelere erişilir.

Belgemize tablo eklemek için aşağıdaki örneği inceleyelim. Formumuza tablo ekle ve kapat isimli iki adet komut butonu ekleyelim ve şu kodları yazalım:

```
Word.Application WordApp = new Word.Application();
Word.Document WordDoc = new Word.Document();
private void btntabloekle_Click(object sender, EventArgs e)
{
 WordApp.Visible = true;
```

```
        // 5 satır 2 sütun genişliğinde tablo oluşturuyoruz
        WordDoc.Tables.Add(WordApp.Selection.Range, 5, 2);

        //Belgeyi sonuna kadar seçiyoruz
        WordApp.Selection.EndKey(Word.WdUnits.wdStory);
        // Seçim tipini paragraf olarak belirliyoruz
        WordApp.Selection.TypeParagraph();

        WordDoc.Range().Tables[1].Columns[1].SetWidth(120, 0);
        WordDoc.Range().Tables[1].Columns[2].SetWidth(170, 0);

        // Tablonun ilk satırını parlak gri yapıyoruz
        WordDoc.Range().Tables[1].Rows[1].Cells.Shading.
            BackgroundPatternColorIndex = Word.WdColorIndex.wdGray50;

        // İlk satırı koyu yapıyoruz.
        WordDoc.Range().Tables[1].Rows[1].Range.Bold = 1;

        // (1,1) hücresini ortalıyoruz
        WordDoc.Range().Tables[1].Cell(1, 1).Range.Paragraphs.Alignment =
            Word.WdParagraphAlignment.wdAlignParagraphCenter;

        // 1. satır 1. sütun
        WordDoc.Range().Tables[1].Cell(1, 1).Range.InsertAfter("Sınıf");
        WordDoc.Range().Tables[1].Cell(1, 2).Range.InsertAfter("Dal");
        WordDoc.Range().Tables[1].Cell(2, 1).Range.InsertAfter("T11A");
        WordDoc.Range().Tables[1].Cell(2, 2).Range.
            InsertAfter("Veri Tabanı Programcılığı");
        WordDoc.Range().Tables[1].Cell(3, 1).Range.InsertAfter("A11A");
        WordDoc.Range().Tables[1].Cell(3, 2).Range.
            InsertAfter("Web Programcılığı");
        WordDoc.Range().Tables[1].Cell(4, 1).Range.InsertAfter("E11A");
        WordDoc.Range().Tables[1].Cell(4, 2).Range.
            InsertAfter("Donanım Teknik Servis");
        WordDoc.Range().Tables[1].Cell(5, 1).Range.InsertAfter("E11B");
        WordDoc.Range().Tables[1].Cell(5, 2).Range.
            InsertAfter("Ağ İşletmenliği");
    }
    private void btnkapat_Click(object sender, EventArgs e)
    {
      try
      {
        WordDoc.Close();
```

```
    WordApp.Quit();
    WordDoc = null;
    WordApp = null;
   }
   catch
   {
    MessageBox.Show("Önce belge açmanız gerekiyor");
    return;
   }
}
```

Program çalıştırılıp **tablo ekle** düğmesine basıldığında aşağıdaki gibi bir tablo oluşturulacaktır:

Sınıf	Dal
T11A	Veri Tabanı Programcılığı
A11A	Web Programcılığı
E11A	Donanım Teknik Servis
E11B	Ağ İşletmenliği

SetWidth özelliği, bir tablodaki sütun genişliklerini ayarlamak için kullanılan metottur ve kullanımı şu şekildedir:

```
ifade.SetWidth(Sütun genişliği, sitil)
```

… ifadesi ile belirtilen yerde **Cell**, **Cells**, **Column** veya **Columns** nesneleri kullanılabilir

- **ColumnWidth**: Bir ya da birden fazla sütunun genişliğini nokta cinsinden ayarlar.

Sitil olarak ise şu tanımlamalar kullanılabilir:

- **wdAdjustFirstColumn**: Yalnızca ilk sütunun genişliği belirlenen değere göre ayarlanır. Eğer birden fazla sütun varsa, bu sütun genişlikleri sağ kenara göre yeniden ayarlanır.
- **wdAdjustNone**: Seçili olan bütün sütun veya hücrelerin genişlikleri belirlenen değere göre ayarlanır. Seçili olmayan sütunların genişlikleri sağa veya sola gerekli miktarda kaydırılarak otomatik olarak ayarlanır. Varsayılan değer budur.

- **wdAdjustProportional:** Yalnızca ilk sütunun genişliği belirlenen değere göre ayarlanır. Eğer seçili olan birden fazla sütun varsa; tablonun sağ kenarına göre ve seçili olmayan sütunlarla seçili olan sütunların genişliklerinin orantılı olarak ölçülendirilmesiyle genişlikleri yeniden düzenlenir. Eğer yalnızca bir hücre ya da sütun seçili ise; tablonun sağ kenarına göre diğer hücre ya da sütunlar orantılı olarak ölçülendirilir.
- **wdAdjustSameWidth:** Bütün sütunlar eşit olacak şekilde ayarlanır.

```
WordDoc.Range().Tables[1].Rows[1].Cells.Shading.BackgroundPatternColorIndex
```

Satırı ise, hücremizin zemin rengini değiştirmek için kullanılır.

WdColorIndex nesnesi aşağıdaki renk değerlerini alabilir:

wdAuto
wdBlack
wdBlue
wdBrightGreen
wdByAuthor
wdDarkBlue
wdDarkRed
wdDarkYellow
wdGray25
wdGray50
wdGreen
wdNoHighlight
wdPink
wdRed
wdTeal
wdTurquoise
wdViolet
wdWhite
wdYellow

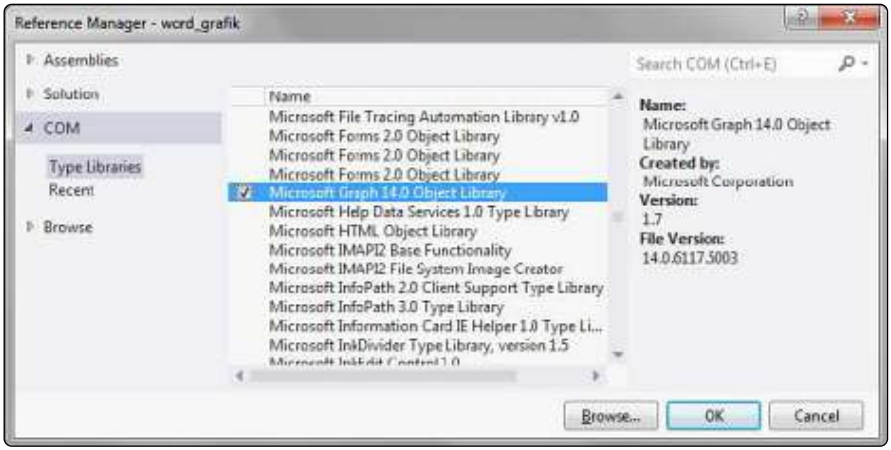
```
.Range().Tables[1].Cell(1, 1).Range.Paragraphs.Alignment
```

… satırı ile tablomuzdaki hücrelerin hizalama şeklini belirledik. Herhangi bir yazı ya da paragrafı hizalamak için de **Alignment** ifadesi kullanılabilir. Hizalama biçimi olarak şu seçenekler kullanılabilir:

wdAlignParagraphCenter
wdAlignParagraphDistribute
wdAlignParagraphJustify
wdAlignParagraphJustifyHi
wdAlignParagraphJustifyLow
wdAlignParagraphJustifyMed
wdAlignParagraphLeft
wdAlignParagraphRight

Grafik İşlemleri

Genellikle veri tabanı projelerimizde elde ettiğimiz verilerin daha iyi anlaşılması için verileri grafiklere dökmek isteriz. Bu işlem için **Chart** adı verilen grafik öğeleri kullanılır. Grafik öğelerini kullanabilmek için **MSGraph** nesnesini projemize ekleyeceğiz. Graph nesnesi için, daha önce eklediğimiz **Microsoft Word 14.0 Object Library** referansına ek olarak **Microsoft Graph 14.0 Object Library** referansını ekliyoruz.



Örnek olarak form üzerine iki adet **ListBox** ekleyelim. `lstisim` adlı **ListBox**'a isimler, `lstyas` isimli **ListBox**'a da bu isimlere karşılık olarak yaşlarını ekleyelim. `btnAktar` adı ile de bir adet **Button** ekleyelim.

Kodumuz aşağıdaki gibi olacak:

```
using System;
using System.Windows.Forms;
using System.Reflection;
using Word = Microsoft.Office.Interop.Word;
using Microsoft.Office.Interop.Graph;
using System.Drawing;
namespace word_grafik
{
  public partial class Form1 : Form
  {
    public Form1()
```

```
    {
      InitializeComponent();
    }
    private void btnAktar_Click(object sender, EventArgs e)
    {
      Word.Application WordApp = new Word.Application();
      Word.Document WordDoc = new Word.Document();
      WordApp.Visible = true;
      Word.InlineShape Shape;
      Word.Range WordRng = WordDoc.Bookmarks.
           get_Item("\\endofdoc").Range;
      Shape = WordRng.InlineShapes.AddOLEObject("MSGraph.Chart.8");

      object oChart;
      object oChartApp;
      oChart = Shape.OLEFormat.Object;
      oChartApp = oChart.GetType().InvokeMember("Application",
        BindingFlags.GetProperty, null, oChart, null);

      Chart objChart = (Chart)Shape.OLEFormat.Object;
      objChart.ChartType = XlChartType.xl3DColumnStacked;

      DataSheet ds;
      ds = objChart.Application.DataSheet;
      ds.Columns.Clear();
      ds.Rows.Clear();
      for (int i = 0; i < lstisim.Items.Count; i++)
      {
        ds.Cells[1, 2 + i] = lstisim.Items[i].ToString();
        ds.Cells[2, 2 + i] = lstyas.Items[i].ToString();
      }

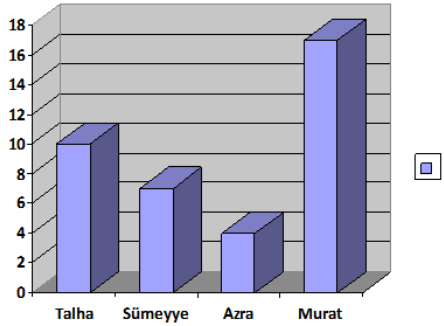
      Shape.Width = WordApp.InchesToPoints(4.25f);
      Shape.Height = WordApp.InchesToPoints(3.25f);

      objChart.Application.Update();
      oChartApp.GetType().InvokeMember("Update",
        BindingFlags.InvokeMethod, null, oChartApp, null);
      oChartApp.GetType().InvokeMember("Quit",
        BindingFlags.InvokeMethod, null, oChartApp, null);
    }
  }
}
```

Programımızın ekran görüntüleri aşağıdaki gibi olacaktır:



Formumuzu çalıştırıp **Aktar** butonuna bastığımızda, **ListBox** içerisinde bulunan değerler yeni bir Word belgesi açılıp, 3 boyutlu sütun grafiği şeklinde gösteriliyor.



Kodumuzu açıklayalım…

```
Word.InlineShape Shape;
      Word.Range WordRng = WordDoc.Bookmarks.
              get_Item("\\endofdoc").Range;
      Shape = WordRng.InlineShapes.AddOLEObject("MSGraph.Chart.8");
```

Bu komut satırı ile grafik nesnemizi oluşturuyoruz. Bu şekilde grafik nesnesi ekleyebileceğimiz gibi ayrıca…

```
Shape = WordRng.InlineShapes.AddChart(Microsoft.Office.
Core.XlChartType.xl3DColumn).Chart;
```

… biçiminde de grafik nesnesi ekleyebiliriz. Ancak bu ikinci yöntem yalnızca Ofis 2007 ve üzeri versiyonlarda geçerlidir. Grafik nesnesine ait özellikleri kullanırken **`InvokeMember`** metodundan faydalanıyoruz. **`InvokeMember`** metodu,

Reflection kullanarak türü bilinmeyen nesnelerin metotlarını çalıştırmak amacı ile kullanılır.

```
oChartApp = oChart.GetType().InvokeMember("Application",
        BindingFlags.GetProperty, null, oChart, null);
```

BindingFlags.GetProperty ile metoda ait özelliğin kullanılacağı belirtilir.

```
oChartApp.GetType().InvokeMember("Update",
        BindingFlags.InvokeMethod, null, oChartApp, null);
```

Eğer bir metot kullanılacaksa **BindingFlags.InvokeMethod** biçiminde kullanılmalıdır.

```
Shape.Width = WordApp.InchesToPoints(4.25f);
Shape.Height = WordApp.InchesToPoints(3.25f);
```

Width ve **Height** özellikleri ile grafik nesnemizin genişlik ve yüksekliğini ayarlıyoruz...

```
objChart.ChartType = XlChartType.xl3DColumnStacked;
```

Bu komut satırı ile grafiğimizin tipini 3 boyutlu sütun grafiği olarak belirliyoruz.

```
 chart.application.datasheet.Cells.Clear()
```

Komut satırı ile öncelikle grafik nesnemizin içeriğini temizliyoruz.

```
for (int i = 0; i < lstisim.Items.Count; i++)
{
 ds.Cells[1, 2 + i] = lstisim.Items[i].ToString();
 ds.Cells[2, 2 + i] = lstyas.Items[i].ToString();
}
```

Komut satırlarında grafik nesnesinin **x** ve **y** koordinatlarında grafik çizimi için kullanılacak değerleri belirliyoruz. Burada isim ve yaş için kullanılan iki adet **ListBox** olmasına rağmen, ikisinin de eleman sayıları aynı olduğu için bir adet döngü kullanmamız yeterlidir.

ListBox'ların eleman sayıları eşit olmadığı takdirde 2 adet döngü kullanmamız ve döngü satırlarını aşağıdaki gibi değiştirmemiz gerekliydi.

```
for (int i = 0; i < lstisim.Items.Count; i++)
{
  ds.Cells[1, 2 + i] = lstisim.Items[i].ToString();
}

for (int j = 0; j < lstyas.Items.Count; i++)
{
  ds.Cells[1, 2 + j] = lstisim.Items[j].ToString();
}
```

Excel Programını Kullanma

Belge Açma, Kapama ve Kaydetme Olayları

Bir Excel uygulaması oluştururken…

```
Excel.Application ExcelApp = new Excel.Application();
```

… şeklinde bir Excel uygulaması oluşturulur.

Excel'de çalışma kitapları **`Workbook`** nesnesi içerisinde yer alırken, çalışma sayfaları da **`WorkSheet`** nesnesi içerisinde yer alır. Hücreler ise **`Sheets`** nesnesi içerisinde bulunmaktadır.

Yeni bir Excel çalışma sayfası oluşturmak için **`Workbooks.Add`** metodu kullanılır.

Eğer daha önceden mevcut olan bir Excel çalışma sayfasını açacaksak **`Workbooks.Open(dosya adı)`** metodu kullanılır.

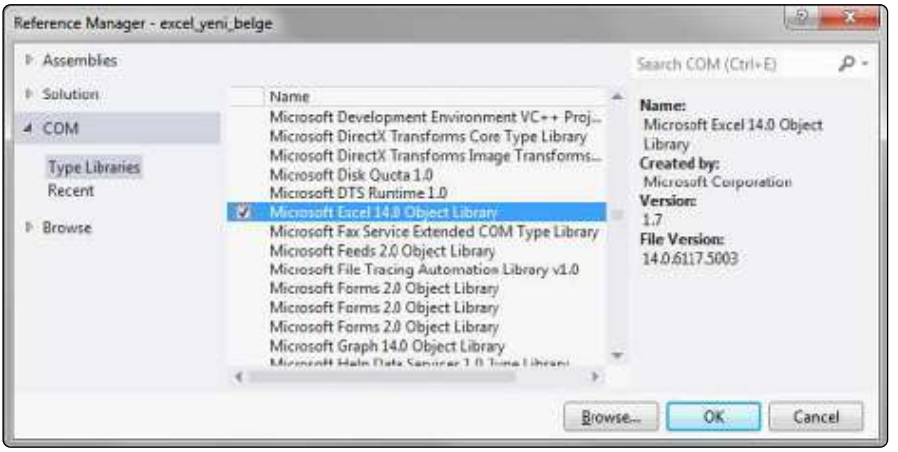
Çalışma sayfasını kapatmak için **`Close`**, Excel uygulamasını kapatmak için ise **`Quit`** metodu kullanılır.

Ayrıca bu işlemler bittikten sonra **`null`** ataması ile de çalışma sayfamızı ve Excel uygulamamızı bellekten temizlememiz gerekmektedir.

Eğer istenirse **`System.GC.Collect()`** komutu ile **Garbage Collector**'u çalıştırıp kullanılmayan nesnelerin hafızadan kaldırılmasını sağlayabiliriz.

Excel ile ilk uygulamamızda yeni bir Excel uygulaması oluşturma uygulaması yapalım. Öncelikle form üzerine bir adet komut butonu ekleyelim.

Daha sonra **Project** menüsünden **Add Reference** seçeneği ile gelen pencereden **COM** başlığı altından **Microsoft Excel 14.0 Object Library** seçeneği seçilip **OK** tuşuna basıyoruz. Elbette kullandığınız Excel programının sürümüne göre buradaki **14** rakamı değişecektir.




```
using Excel = Microsoft.Office.Interop.Excel;
```

… satırı ile referansımızı projemize dâhil ediyoruz. Formumuzun tanımlama kısmında **`ExcelApp`** nesnemizi tanımlıyoruz. **`ExcelApp`**, Excel uygulamasını temsil etmektedir.

Kod olarak da şunları yazalım:

```
using System;
using System.Windows.Forms;
using Excel = Microsoft.Office.Interop.Excel;
namespace excel_yeni_belge
{
 public partial class Form1 : Form
 {
  public Form1()
  {
```

```
    InitializeComponent();
  }
  private void btnYeni_Click(object sender, EventArgs e)
  {
   Excel.Application ExcelApp = new Excel.Application();
   ExcelApp.Workbooks.Add();
   ExcelApp.Visible = true;
   ExcelApp.Worksheets[1].Activate();
   ExcelApp.Cells[1, 1].value = "Deneme";
  }
 }
}
```

Programımızı çalıştırdığımızda **A1** hücresine **Deneme** yazıldığını görürüz.

```
Excel.Application ExcelApp = new Excel.Application();
```

… satırı ile yeni bir Excel nesnesi oluşturuyoruz.

```
ExcelApp.Workbooks.Add();
```

Workbooks nesnesinin **Add()** metodu ile yeni bir çalışma kitabı ekliyoruz.

```
ExcelApp.Visible = true;
```

Visible özelliğini **true** yaparak, oluşturmuş olduğumuz Excel sayfasını görüntülüyoruz...

```
ExcelApp.Worksheets[1].Activate();
```

1 nolu çalışma sayfasını aktif ediyoruz.

```
ExcelApp.Cells[1, 1].value = "Deneme";
```

A1 hücresine **Deneme** değerimizi yazıyoruz.

ExcelApp.Cells[1, 1] ifadesinde ilk değer satırı, ikinci değer ise sütunu belirtmektedir.

Daha önceden var olan bir Excel çalışma sayfasını açan bir örnek yapalım. Form üzerine bir adet **btnAc** adı ile buton ve dosya açmak için de bir adet de **openFileDialog** nesnesi ekleyelim. Kodumuz aşağıdaki şekilde olacak:

```
using System;
using System.Windows.Forms;
using Excel = Microsoft.Office.Interop.Excel;
namespace excel_ac
{
 public partial class Form1 : Form
 {
  public Form1()
  {
    InitializeComponent();
  }
  private void btnAc_Click(object sender, EventArgs e)
  {
    openFileDialog1.Title = "Belge seçiniz...";
    openFileDialog1.Filter =
     "Excel 2003 dosyaları (*.xls)|*.xls|Excel 2007-2010
                dosyaları (*.xlsx)|*.xlsx";
    openFileDialog1.FilterIndex = 1;
    if (openFileDialog1.ShowDialog() == DialogResult.Cancel)
      MessageBox.Show("Dosya seçilmedi");
    else
    {
     Excel.Application ExcelApp = new Excel.Application();
     ExcelApp.Workbooks.Open(openFileDialog1.FileName);
     ExcelApp.Visible = true;
    }
  }
 }
}
```

Örneğimizde aşağıdaki komut satırında **Workbooks** nesnesinin **Open** metodu ile dosya açma diyalog penceresinde seçilen Excel dosyası açılıyor.

```
ExcelApp.Workbooks.Open(openFileDialog1.FileName);
```

Kaydetme olayı için **Save**, farklı biçimde kaydetmek için **GetSaveAsFilename** metotları kullanılır.

Save, **GetSaveAsFilename** metotları ile belge kaydetme ve farklı kaydetme işlemleri ve çalışma sayfasını ve Excel uygulamasını kapatma ile ilgili örnek yapalım. Bu olay ve metotlar Word konusunda detaylıca anlatıldığı için ayrıntılara girmiyoruz. Form üzerinde **btnYeni**, **btnKapat**, **btnKaydet**, **btnFarkliKaydet** adları ile 4 adet buton olacak.

Kodumuz ise aşağıdaki şekilde olacak:

```
Excel.Application ExcelApp = new Excel.Application();
private void btnYeni_Click(object sender, EventArgs e)
{
  ExcelApp.Workbooks.Add();
  ExcelApp.Visible = true;
  ExcelApp.Worksheets[1].Activate();
  ExcelApp.Cells[1, 1].value = "Deneme";
}
private void btnKaydet_Click(object sender, EventArgs e)
{
  try
  {
    ExcelApp.Save();
  }
  catch
  {
    MessageBox.Show("Önce belge açmanız gerekiyor");
    return;
  }
}
private void btnFarkliKaydet_Click(object sender, EventArgs e)
{
  try
  {
    ExcelApp.GetSaveAsFilename();
  }
  catch
  {
    MessageBox.Show("Önce belge açmanız gerekiyor");
```

```
    return;
  }
}
private void btnKapat_Click(object sender, EventArgs e)
{
  try
  {
    ExcelApp.Workbooks.Close();
    ExcelApp.Quit();
    ExcelApp = null;
    System.GC.Collect();
  }
  catch
  {
    MessageBox.Show("Önce belge açmanız gerekiyor");
    return;
  }
}
```

Örneğimizde…

```
ExcelApp.Save();
```

… satırında **Save** metodu ile Excel uygulamamızı kaydederken,

```
ExcelApp.GetSaveAsFilename();
```

… satırında ise **GetSaveAsFilename** metodu ile Excel uygulamamızı farklı kaydediyoruz.

```
ExcelApp.Workbooks.Close();
```

Workbooks.Close metodu ile açık olan Excel çalışma kitaplarını kapatıyoruz.

```
ExcelApp.Quit();
```

Quit metodu ile Excel uygulamasını kapatıyoruz.

```
ExcelApp = null;
System.GC.Collect();
```

Excel uygulamasını bellekten silerek, belleği temizliyoruz.

Hücreye Erişim, Yazı Ekleme ve Biçimlendirme İşlemleri

Excel'de bir hücre içeriğine erişirken temel olarak iki nesne kullanılır. Bunlar **`Range`** ve **`Cell`** nesneleridir. **`Range`**, bir ya da daha fazla hücre aralığını ifade ederken, **`Cell`** ise bir ya da daha fazla hücreyi temsil etmektedir.

Buna göre bir hücreye erişim için aşağıdaki yöntemler kullanılabilir:

`Range` nesnesi ile

```
Excelnesnesi.Worksheets[Çalışma Sayfası Numarası].Range("sütun satır").Value
Excelnesnesi.Worksheets["Çalışma Sayfası Adı"].Range("sütun satır").Value)
```

Örneğin; ilk çalışma sayfasına **`Worksheets[1]`** şeklinde erişmek mümkün olduğu gibi,

`Worksheets["Sayfa1"]` şeklinde de erişmek mümkündür.

```
Excelnesnesi.Cells.Range["$Sütunadı $Satıradı"].Value)
```

`$` karakterini kullandığımızda sütun ve satır numaralarını yazmadan isimlerini yazarak hücreye erişmek mümkündür.

```
Excelnesnesi.ActiveSheet.Range("sütun satır").Value
```

`ActiveSheet` nesnesi bir çalışma kitabında aktif olan çalışma sayfasını belirtmektedir.

`Cells` nesnesi ile

```
Excelnesnesi.Worksheets[Çalışma Sayfası Numarası].Cells.Item(sütun,
satır).Value
Excelnesnesi.Worksheets[Çalışma Sayfası Numarası].Cells(sütun,
satır).Value
```

```
Excelnesnesi.Worksheets[Çalışma Sayfası Numarası].Rows(satır
numarası).Cells(hucre sıra numarası).Value
Excelnesnesi.Worksheets[Çalışma Sayfası Numarası].Columns(sütun
numarası). Cells(hücre sıra numarası).Value
```

Buna göre bir Excel çalışma sayfasındaki **A2**, yani 1. sütun, 2. satırdaki veriye erişen bir program yazalım. Form üzerine **btnAc** adı ile bir adet buton nesnesi, dosya açmak için de bir adet **openFileDialog** nesnesi ekleyelim.

Kodumuz aşağıdaki şekilde olacak:

```
private void btnAc_Click(object sender, EventArgs e)
{
  openFileDialog1.Title = "Belge seçiniz...";
  openFileDialog1.Filter =
  "Excel 2003 dosyaları (*.xls)|*.xls|Excel 2007-2010
              dosyaları (*.xlsx)|*.xlsx";
  openFileDialog1.FilterIndex = 1;
  if (openFileDialog1.ShowDialog() == DialogResult.Cancel)
    MessageBox.Show("Dosya seçilmedi");
  else
  {
    Excel.Application ExcelApp = new Excel.Application();
    ExcelApp.Workbooks.Open(openFileDialog1.FileName);
    ExcelApp.Visible = true;

    MessageBox.Show(ExcelApp.Worksheets[1].Range("A2").Value);
    // MessageBox.Show(ExcelApp.Worksheets["Sayfa1"].Range("A2").Value);
    // MessageBox.Show(ExcelApp.Cells.Range["$A$2"].Value);
    // MessageBox.Show(ExcelApp.ActiveSheet.Range("A2").Value);
    // MessageBox.Show(ExcelApp.Worksheets[1].Cells.Item(2, 1).Value);
    // MessageBox.Show(ExcelApp.Worksheets[1].Cells(2, 1).Value);
    // MessageBox.Show(ExcelApp.Worksheets[1].Rows(2).Cells(1).Value);
    // MessageBox.Show(ExcelApp.Worksheets[1].Columns(1).Cells(2).Value);
  }
}
```

Örneğimizde hücreye erişim için kullanabileceğiniz diğer yöntemleri pasif yaparak gösterdim. Siz aktifleştirerek deneyebilirsiniz.

Şimdiki örneğimizde ise, Excel hücrelerine değer aktarma işlemi ile birlikte hücre içerisindeki değerleri biçimlendirme işlemlerini de yapalım. Yine form üzerine bir adet buton ekleyip kod olarak şu satırları ekliyoruz:

```
private void btnYaz_Click(object sender, EventArgs e)
{
  Excel.Application ExcelApp = new Excel.Application();
  ExcelApp.Workbooks.Add();
  ExcelApp.Visible = true;
  ExcelApp.Worksheets[1].Activate();
  ExcelApp.Cells[1, 1].value = "Deneme1";
  ExcelApp.Cells[1, 1].Font.Name = "Verdana";
  // A1 hücresindeki yazının yazı boyutunu 8 olarak değiştiriyoruz
  ExcelApp.Cells[1, 1].Font.Size = 8;
  ExcelApp.Cells[1, 1].Font.color =
   System.Drawing.ColorTranslator.ToOle(System.Drawing.Color.Green);
  ExcelApp.Cells[1, 2].value = "Deneme2";
  ExcelApp.Cells[1, 3].value = "Deneme3";
  ExcelApp.Cells[1, 2].font.bold = true;
  ExcelApp.Cells[1, 2].Interior.ColorIndex = 3;
  ExcelApp.Cells[1, 3].Interior.Color =
   System.Drawing.Color.FromArgb(0, 0, 255);
}
```

Örneğimizde…

```
ExcelApp.Cells[1, 1].Font.Name = "Verdana";
```

… satırında **A1** hücresindeki yazının yazı tipini **Verdana** olarak değiştirdik.

```
ExcelApp.Cells[1, 1].Font.color =
  System.Drawing.ColorTranslator.ToOle(System.Drawing.Color.Green)
```

Color.Translator.ToOle(System.Drawing.Color.Green) metodu ile **A1** hücresindeki yazının yazı rengini **yeşil** olarak değiştiriyoruz.

```
ExcelApp.Cells[1, 2].Interior.ColorIndex = 3;
```

… satırında geçen **Interior.ColorIndex** ifadesi ile ise; **B1** hücresinin zemin

rengini **kırmızı** renk olarak değiştiriyoruz. Bunu ister o renge karşılık gelen rakam ile istersek **RGB** değerleri ile yapabiliriz.

Rakam ile **`ExcelApp.Cells[1, 2].Interior.ColorIndex = 3`** şeklinde yaparken aynı işlemi RGB değerleri ile…

```
ExcelApp.Cells[1, 3].Interior.Color =
System.Drawing.Color.FromArgb(0, 0, 255);
```

… şeklinde yaparak **C1** hücresinin zemin rengini **mavi** olarak değiştirdik.

Programımızı çalıştırdığımızda ise; bir Excel sayfası açılacak ve karşımıza şu şekilde biçimlendirilmiş yazılar çıkacaktır:



Seçim İşlemleri

Seçim işlemleri için **`Select`** metodu kullanılır.

Satır seçmek için...

```
Excelnesnesi.Rows["Sütun adı "].Select( )
```

… şeklinde bir yöntem kullanılabilir.

Sütun seçmek için…

```
Excelnesnesi.Columns["Sütun adı "].Select( )
```

… şeklinde bir yöntem kullanılabilir.

Hücre seçmek için…

```
Excelnesnesi.Cells.Range["$Sütun adı $Satır numarası "].Select( )
```

… şeklinde bir yöntem kullanılabilir.

Bunların dışında Excel hücresine erişim konusunda belirttiğimiz erişim metotlarını kullanmamız da mümkündür.

Hücre, satır ve sütun seçme işlemlerine dair bir örnek uygulama yapalım. Form üzerine 3 adet buton ekliyoruz.

```
Excel.Application ExcelApp = new Excel.Application();
private void Form1_Load(object sender, EventArgs e)
{
  ExcelApp.Workbooks.Add();
  ExcelApp.Visible = true;
  ExcelApp.Worksheets[1].Activate();
  // A1 ve B1 hücrelerine değer yazıyoruz
  ExcelApp.Cells[1, 1].value = "1.değer";
  ExcelApp.Cells[1, 2].value = "2.değer";
  // Hücre isimlendirmede diğer bir yöntemde range nesnesidir
  ExcelApp.Cells.Range["$C$1"].Value = "Deneme";
}
private void btnSatirSec_Click(object sender, EventArgs e)
{
  ExcelApp.Worksheets[1].Activate();
  ExcelApp.Rows["1:1"].Select();
}
private void btnSutunSec_Click(object sender, EventArgs e)
{
  ExcelApp.Worksheets[1].Activate();
  ExcelApp.Columns["C"].Select();
}
private void btnHucreSec_Click(object sender, EventArgs e)
{
  ExcelApp.Worksheets[1].Activate();
  ExcelApp.Cells.Range["$C$1"].Select();
}
```

Örneğimizde…

```
ExcelApp.Rows["1:1"].Select();
```

… satırı ile 1 nolu satırı seçtik.

```
ExcelApp.Columns["C"].Select();
```

… satırı ile **C** sütununu seçtik.

```
ExcelApp.Cells.Range["$C$1"].Select();
```

… satırında ise **C1** hücresini seçtik. Bunun yerine…

```
ExcelApp.Cells[1, 3].Select();
```

… şeklinde **`Cells`** nesnesi kullanılarak da **C1** hücresine erişebiliriz.

Önizleme Yapma ve Yazdırma İşlemleri

Excel dosyalarını önizleme yapmak için **`PrintPreview`** metodu kullanılırken, yazdırma işlemi için **`PrintOut`** metodu kullanılır.

Bir çalışma kitabı içerisindeki çalışma sayfalarını yazdırmak ve baskı ön izleme yapmak için **`PrintPreview`** ve **`PrintOut`** metotlarını aşağıdaki biçimlerde kullanırız.

```
Excelnesnesi. Worksheets["Çalışma Sayfası Adı"].PrintPreview()
Excelnesnesi.ActiveSheet.PrintOut()
```

Bu metotlar ile baskı önizleme ve yazdırma işlemleri yapabileceğimiz gibi, sayfa içerisinde belli bir bölümü…

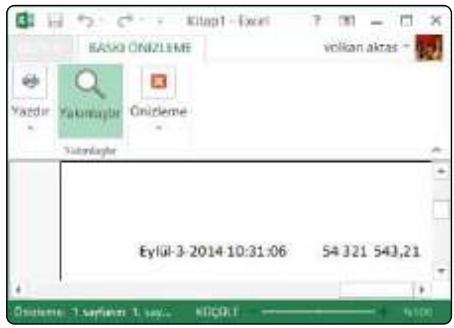
```
Excelnesnesi.Worksheets[Çalışma Sayfası Numarası].Range["sütun
satır"].PrintPreview
Excelnesnesi.Worksheets[Çalışma Sayfası Numarası].Range["sütun
satır"].PrintOut
```

… Metotları ile de baskı önizleme ve yazdırma yapabiliriz.

Bir Excel tablosunu yazdırma, baskı önizleme yapma gibi yazıcı ve yazdırma işlemlerine dair bir örneği inceleyelim.

Form üzerine **btn_yaziaktar**, **btn_baski_onizleme** ve **btn_yazdir** adı ile 3 adet buton ekleyelim ve kodumuzu aşağıdaki şekilde yazalım:

```
Excel.Application ExcelApp = new Excel.Application();
private void btn_yaziaktar_Click(object sender, EventArgs e)
{
  ExcelApp.Workbooks.Add();
  ExcelApp.Visible = true;
  ExcelApp.Worksheets[1].Activate();
  ExcelApp.Cells[1, 1].value = (DateTime.Now).
           ToString("MMMM-d-yyyy hh:mm:ss");
  ExcelApp.Cells[1, 1].NumberFormat = "MMMM d yyyy";
  ExcelApp.Cells[1, 2].value = "54321";
  ExcelApp.Cells[1, 2].NumberFormat = "### ###";
  ExcelApp.Cells[1, 3].value = "543,21";
  ExcelApp.Cells[1, 3].NumberFormat = "(#,##0.00)";
}
private void btn_baskı_onizleme_Click(object sender, EventArgs e)
{
  ExcelApp.ActiveSheet.PrintPreview();
}
private void btn_yazdır_Click(object sender, EventArgs e)
{
  ExcelApp.ActiveSheet.PageSetup.PrintArea = "$A$1:$B$3";
  ExcelApp.ActiveSheet.PageSetup.PrintTitleRows = "";
  ExcelApp.ActiveSheet.PageSetup.PrintTitleColumns = "";
  ExcelApp.ActiveSheet.PageSetup.PrintHeadings = false;
  ExcelApp.ActiveSheet.PageSetup.PrintGridLines = false;
  ExcelApp.ActiveSheet.PageSetup.PrintQuality = 600;
  ExcelApp.ActiveSheet.PrintOut();
}
```



Kodumuzda…

```
ExcelApp.Cells[1, 1].value = (DateTime.Now).
ToString("MMMM-d-yyyy hh:mm:ss");
```

… satırında **A1** hücresine o anki tarih ve zamanı ekliyoruz.

Daha sonra…

```
ExcelApp.Cells[1, 1].NumberFormat = "MMMM d yyyy";
```

… bu hücreye tekrar bir tarih biçimlendirmesi uyguluyoruz.

```
ExcelApp.Cells[1, 2].value = "54321";
ExcelApp.Cells[1, 2].NumberFormat = "### ###";
```

B1 hücresine "*54321*" değerini ekleyerek, hücredeki değeri basamaklara ayırarak tam sayı biçimlendirmesi yapıyoruz.

```
ExcelApp.Cells[1, 3].value = "543,21";
ExcelApp.Cells[1, 3].NumberFormat = "(#,##0.00)";
```

C1 hücresine "*543,21*" değerini ekleyerek, ondalıklı sayı biçimlendirmesi yapıyoruz.

Ayrıca kodumuzda yazdırma ile ilgili bazı ayarları da yaptık.

PrintArea = "$A$1:$B$3"	Yazdırma alanını belirler.
PrintTitleRows = ""	Satır başlıklarının yazdırılmamasını sağlar.
PrintTitleColumns = ""	Sütun başlıklarının yazdırılmamasını sağlar.
PrintHeadings = false	Başlıkların yazdırılmamasını sağlar.
PrintGridLines = false	Hücre çizgilerinin yazdırılmamasını sağlar.
PrintQuality = 600	Baskı kalitesini ayarlar.

Bu ayarlara ek olarak **PrintOut** fonksiyonu şu parametrelerle birlikte kullanılabilir:

PrintOut [From], [To], [Copies], [Preview], [ActivePrinter], [PrintToFile], [Collate]

- **Copies**: Yazdırılacak kopya sayısını belirler:
- **From**, **To**: Başlangıç ve bitiş aralığı belirlenerek bu aralıktaki sayfalar yazdırılır.

Örneğin;

```
ExcelApp.Sheets.PrintOut(1, 2, 3);
```

Satırı ile 1 ve 3 nolu çalışma sayfaları ve bu aralıktaki çalışma sayfaları 3 kopya olarak yazdırılır.

```
ExcelApp.Sheets.PrintOut(1, 2, 3, Collate:true);
```

Satırı ile ise 1 ve 3 nolu çalışma sayfaları ve bu aralıktaki çalışma sayfaları, 3 kopya olarak yazdırılır. **Collate** parametresine **true** değeri verildiğinden, kopyalar sıralanarak yazdırılır.

- **PrintToFile**: Dosyaya yazdırmak için kullanılır.

Örneğin…

```
ExcelApp.ActiveSheet.PrintOut(PrintToFile: true, PrToFileName: "D:\\excel.prn");
```

… şeklindeki satır ile aktif çalışma sayfasını **D** sürücüsüne **excel.prn** adıyla yazıcı dosyası olarak kaydeder.

Arama İşlemleri

Arama işlemleri için **Find** metodu kullanılır. Kullanımı şu şekildedir:

```
Arama sonucunun aktarılacağı değişken = Excelnesnesi.Cells.find(aranacak metin)
```

Ya da…

```
find(what:=aranacak metin)
```

… şeklinde de kullanılabilir. Arama sonucunun aktarıldığı değişkende, aranan metnin bulunduğu hücrenin sütun ve satır bilgileri yer alır. Bu bilgiler kullanılarak arama sonucu üzerinde işlem yapılabilir.

Örneğimizde bir Excel tablosu üzerinde bulma işlemi yapacağız. Örnek bir Excel tablosu hazırlayalım. Bu tablo içerisinde öğrencilerin isim numara ve 3

adet notları yer alacak. Bir giriş kutusu yardımı ile önce öğrencinin sınıfı daha sonra numarası giriliyor. Daha sonra kaçıncı yazılı notunu öğrenmek istiyorsak, bir mesaj penceresinde öğrencinin notu görüntüleniyor. Programı çalıştırmak için form üzerinde bir adet buton olması yeterli.

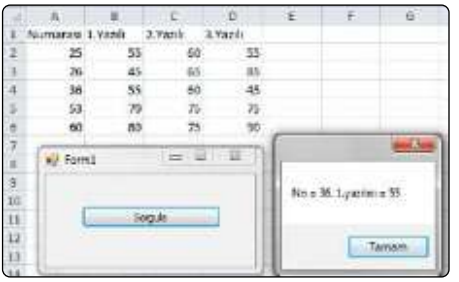
```
using System;
using System.Windows.Forms;
using Microsoft.VisualBasic;
using System.IO;
using Excel = Microsoft.Office.Interop.Excel;
namespace Excel_bulma
{
 public partial class Form1 : Form
 {
  public Form1()
  {
    InitializeComponent();
  }
  private void btnSorgula_Click(object sender, EventArgs e)
  {
    Excel.Range bulunan = null;
    int no, yazilino;
    string sınıf,dosya;
    Boolean aranan_dosya;

    sınıf = Interaction.InputBox("Lütfen sınıfı giriniz");
    aranan_dosya = File.Exists(Application.StartupPath +
       "\\" + sınıf + ".xlsx");

    if (aranan_dosya == false)
      MessageBox.Show("Sınıf Bulunamadı");
    else
    {
     dosya = Application.StartupPath + "\\" + sınıf + ".xls";
     Excel.Application ExcelApp = new Excel.Application();
     ExcelApp.Workbooks.Open(dosya);
     ExcelApp.Visible = false;

     ExcelApp.Worksheets[1].Activate();
     no = Convert.ToInt32(Interaction.InputBox
       ("Öğrencinin numarasını girin"));
     bulunan = ExcelApp.Cells.Find(What:no);
```

```
    if (bulunan==null)
      MessageBox.Show("Bulunamadı!");
    else
    {
     bulunan.Activate();
     yazilino = Convert.ToInt32(Interaction.InputBox
         ("Kaçıncı yazılıyı öğrenmek istiyorsunuz?"));
     switch (yazilino)
     {
      case 1:
      {
       MessageBox.Show("No = " +no +". 1.yazılısı = " +
        ExcelApp.Cells[bulunan.Row, bulunan.Column + 1].value);
       break;
      }
      case 2:
      {
       MessageBox.Show("No = " +no +". 1.yazılısı = " +
        ExcelApp.Cells[bulunan.Row, bulunan.Column + 2].value);
       break;
      }
      case 3:
      {
       MessageBox.Show("No = " +no +". 1.yazılısı = " +
        ExcelApp.Cells[bulunan.Row, bulunan.Column + 3].value);
       break;
      }
     }
    }
    ExcelApp.Application.Quit();
   }
  }
 }
}
```



Kodumuzda öncelikle **InputBox** yardımı ile sınıf ismi istenerek, sınıf ismi ile aynı isimli Excel dosyasının disk üzerinde mevcut olup olmadığı `File.Exists` metodu ile kontrol edilmekte, eğer dönen değer true ise dosyanın mevcut olduğu anlaşılmaktadır. Dosya mevcut ise bir **InputBox** yardımı ile öğrencinin numarası girilerek…

```
bulunan = ExcelApp.Cells.Find(What:no);
```

… satırında Excel çalışma sayfası içerisinde belirtilen no değeri aratılmaktadır. Eğer dönen değer `null` değilse, arama işlemi başarılı olmuştur ve…

```
bulunan.Activate();
```

… satırında bulunan değerin ait olduğu hücre aktifleştirilir. Daha sonra yine bir **InputBox** yardımı ile kaçıncı yazılının öğrenilmek istendiği sorularak…

```
ExcelApp.Cells[bulunan.Row, bulunan.Column + 1].value);
```

… satırında olduğu gibi ilgili satırdaki hücre değeri, **InputBox** içerisine girilen yazılı sırasına göre elde edilerek mesaj penceresi içerisine görüntülenir.

Grafik İşlemleri

Grafik işlemleri için Word da olduğu gibi Excel'de de **Chart** adı verilen grafik öğeleri kullanılır.

Örnek olarak form üzerine 2 adet **ListBox** ekleyelim. `lstisim` adlı **ListBox**'a isimler, `lstyas` isimli **ListBox**'a da bu isimlere karşılık olarak yaşlarını ekleyelim. `btnAktar` adı ile de bir adet buton ekleyelim.

Kodumuz aşağıdaki gibi olacak:

```
private void btnAktar_Click(object sender, EventArgs e)
{
 string[] isimler = new string[6];
 byte[] yaslar=new byte[6];

 Excel.Application ExcelApp = new Excel.Application();
 ExcelApp.Workbooks.Add();
 ExcelApp.Visible = true;

 ExcelApp.Worksheets[1].Activate();

 ExcelApp.Cells[1, 1].value = "İsimler";
 ExcelApp.Cells[1, 2].value = "Yaşlar";
 for (int i =2; i <= lstisim.Items.Count+1; i++)
   ExcelApp.Cells[i, 1].value = lstisim.Items[i-2];
 for (int j = 2; j <= lstyas.Items.Count+1; j++)
   ExcelApp.Cells[j, 2].value = lstyas.Items[j-2];

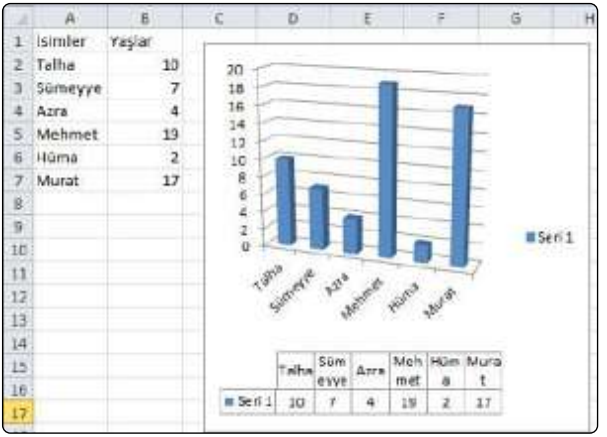
 Excel.Worksheet xlWorkSheet=
   (Excel.Worksheet)ExcelApp.ActiveSheet;
 Excel.ChartObjects xlCharts =
   (Excel.ChartObjects)xlWorkSheet.ChartObjects();
 Excel.ChartObject Chart =
   (Excel.ChartObject)xlCharts.Add(110, 10, 250, 250);
 Excel.Chart chartPage =Chart.Chart;
 Excel.Range chartRange;
 chartRange = xlWorkSheet.get_Range("A2", "B7");
 chartPage.SetSourceData(chartRange);

 chartPage.ChartType = Excel.XlChartType.xl3DColumnClustered;
 chartPage.HasDataTable = true;
}
```

Programımızın ekran görüntüleri aşağıdaki gibidir:



Formumuzu çalıştırıp **Aktar** butonuna bastığımızda, **ListBox** içerisinde bulunan değerler bir yeni bir Excel çalışma sayfası açılıp 3 boyutlu sütun grafiği şeklinde gösteriliyor.



Kodumuzu açıklayalım:

```
Excel.ChartObject Chart = (Excel.ChartObject)xlCharts.Add(110, 10, 250, 250);
```

… ifadesi ile grafik nesnesini Excel belgesine ekliyoruz.

`Add` metodu…

```
Add(yatay konum, dikey konum, genişlik, yükseklik)
```

… biçiminde kullanılır.

```
Excel.Range chartRange;
chartRange = xlWorkSheet.get_Range("A2", "B7");
chartPage.SetSourceData(chartRange);
```

Bu kod satırlarında grafiğimizde gösterilecek olan verilerin hangi hücre aralıklarından alınacağını belirtiyoruz. **`get_Range`** metodu ile bu aralığı belirtirken, **`SetSourceData`** ile belirtilen aralığı grafiğe bağlıyoruz.

```
chartPage.ChartType = Excel.XlChartType.xl3DColumnClustered;
```

Bu komut satırı ile grafik tipini 3 boyutlu sütun grafiği şeklinde ayarlıyoruz.

```
for (int i =2; i <= lstisim.Items.Count+1; i++)
    ExcelApp.Cells[i, 1].value = lstisim.Items[i-2];
for (int j = 2; j <= lstyas.Items.Count+1; j++)
    ExcelApp.Cells[j, 2].value = lstyas.Items[j-2];
```

Komut satırlarında, grafiğimizde kullanmak üzere **`lstisim`** ve **`lstyas`** adlı **ListBox**'larda mevcut olan değerleri Excel çalışma sayfası içerisindeki 1 ve 2 numaralı sütunlara sırası ile yazdırıyoruz.

Ayrıca…

```
ExcelApp.Cells[1, 1].value = "İsimler";
ExcelApp.Cells[1, 2].value = "Yaşlar";
```

… satırlarında 1 ve 2 numaralı sütun başlıklarını "*İsimler*" ve "*Yaşlar*" biçiminde değiştiriyoruz.

```
chartPage.HasDataTable = true;
```

Komut satırlarında ise; **`HasDataTable`** özelliğine **`true`** değerini vererek veri tablosunun, grafiğin alt tarafında görüntülenmesini sağlıyoruz.

İŞLETİM SİSTEMİ KAYNAKLARINA ERİŞİM

25

C# içerisinde işletim sistemi kaynaklarına erişim için Windows Api'leri ve WMI sınıfından faydalanabiliriz. Bu sayede işletim sistemi dâhilinde kullanılmakta olan metotlara ve özelliklere erişebilir ve projelerimiz içerisinde kullanabiliriz.

Windows API'lerini Kullanma

Windows api'leri Windows işletim sisteminin parçası olan **Dll** kütüphaneleridir. .NET yardımı ile yapmak istediğimiz işlemlere ait api'leri kullanabiliriz.

.NET altındaki api'lerin tüm listesini elde edebilmek için herhangi bir program yok. Ancak, *http://www.pinvoke.net* adresinde hemen hemen bütün api'ler ve tanımlamaları hatta kullanımları bile mevcut.

Örneğin ses çalma ile ilgili api'yi kullanmak için *http://www.pinvoke.net/winmm.PlaySound* adresinden **winmm.dll** dosyasını seçip, onunla ilgili tanımlamaları kullanabiliriz.

Kodlama aşamasında ise C# .NET içerisinde bir api'yi kullanmak için **`DllImport`** metodundan yararlanırız.

Bu metodu kullanmak için…

```
using System.Runtime.InteropServices;
```

… şeklinde **`System.Runtime.InteropServices`** sınıfını projemize dahil etmeliyiz.

Disk Toplam Alanını ve Boş Alanını Bulma

Disk sürücüleri hakkında bilgi almak için **kernel32.dll** dosyasını kullanabiliriz. Disk boyutunu ve disk üzerindeki boş alan miktarını bulmak için bu dll dosyasının **GetDiskFreeSpaceEx** metodundan faydalanacağız.

GetDiskFreeSpaceEx metodunun kullanımı şu şekildedir:

```
GetDiskFreeSpaceEx(Disk sürücüsü adı, Ulaşılan boş alan, Toplam alan, Boş alan)
```

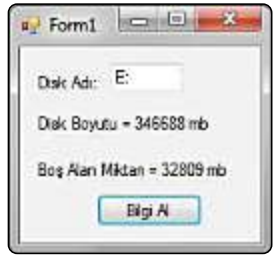
```
using System;
using System.Runtime.InteropServices;
using System.Windows.Forms;
namespace Api_GetDiskFreeSpaceExA
{
  public partial class Form1 : Form
  {
   public Form1()
   {
    InitializeComponent();
   }
   [DllImport("kernel32.dll", SetLastError = true, CharSet
= CharSet.Auto)]
   [return: MarshalAs(UnmanagedType.Bool)]
   static extern bool GetDiskFreeSpaceEx(string lpDirectoryName,
   out ulong lpFreeBytesAvailable,
   out ulong lpTotalNumberOfBytes,
   out ulong lpTotalNumberOfFreeBytes);

   private void btnBilgiAl_Click(object sender, EventArgs e)
   {
    ulong toplamalan;
    ulong ulasilanbosalan;
    ulong bosalan;
    double alan;

    bool sonuc = GetDiskFreeSpaceEx(txtDiskAdi.Text, out ulasilanbosalan,
out toplamalan, out bosalan);
    if (!sonuc)
      throw new System.ComponentModel.Win32Exception();
```

```
    alan = bosalan / (1024 * 1024);
    lblBosAlan.Text = lblBosAlan.Text + " = " + alan.ToString() + " mb";

    alan = toplamalan / (1024 * 1024);
    lblToplamAlan.Text = lblToplamAlan.Text + " = " +
alan.ToString() + " mb";
   }
  }
}
```




```
bool sonuc = GetDiskFreeSpaceEx(txtDiskAdi.Text,
out ulasilanbosalan, out toplamalan, out bosalan);
```

… satırında **`out`** ifadesi ile geriye dönen değerlerin hangi değişkenlere aktarılacağını belirtiyoruz. Eğer sonuç değişkeni **`false`** değerinden farklı bir değer içeriyorsa, işlem başarı ile gerçekleştirilmiş demektir.

Mikroişlemci Hakkında Bilgi Alma

Mikroişlemci hakkında bilgi almak için **kernel32.dll** dosyasını kullanabiliriz. Bu işlem için bu dll dosyasının **`GetSystemInfo`** metodunu kullanacağız.

`GetSystemInfo` metodunun kullanımı şu şekildedir:

```
public static extern void GetSystemInfo([MarshalAs(UnmanagedType.
Struct)] ref SYSTEM_INFO lpSystemInfo);
```

Bu metot ile birlikte kullanılan **SYSTEM_INFO** yapısı şu şekildedir:

```
public struct SYSTEM_INFO
{
 internal _PROCESSOR_INFO_UNION uProcessorInfo;
 public uint dwPageSize;  // Cache bellek boyutunu belirtir.
 public IntPtr lpMinimumApplicationAddress;
 public IntPtr lpMaximumApplicationAddress;
 public IntPtr dwActiveProcessorMask;
 public uint dwNumberOfProcessors; // İşlemci sayısını belirtir.
 public uint dwProcessorType; // İşlemci tipini belirtir.
 public uint dwAllocationGranularity;
 public ushort dwProcessorLevel;
 public ushort dwProcessorRevision;
}
```

Yine bu metot ile birlikte kullanılan **PROCESSOR_INFO_UNION** yapısı ise şu şekildedir:

```
public struct _PROCESSOR_INFO_UNION
{
 [FieldOffset(0)]
 internal uint dwOemId;
 [FieldOffset(0)]
 internal ushort wProcessorArchitecture; // İşlemci türünü belirtir.
 [FieldOffset(2)]
 internal ushort wReserved;
}
```

Kodumuz aşağıdaki gibi olacak:

```
[DllImport("kernel32.dll")]
public static extern void GetSystemInfo
  ([MarshalAs(UnmanagedType.Struct)]
  ref SYSTEM_INFO lpSystemInfo);

[StructLayout(LayoutKind.Sequential)]
public struct SYSTEM_INFO
{
  internal _PROCESSOR_INFO_UNION uProcessorInfo;
  public uint dwPageSize;
```

```
  public IntPtr lpMinimumApplicationAddress;
  public IntPtr lpMaximumApplicationAddress;
  public IntPtr dwActiveProcessorMask;
  public uint dwNumberOfProcessors;
  public uint dwProcessorType;
  public uint dwAllocationGranularity;
  public ushort dwProcessorLevel;
  public ushort dwProcessorRevision;
}

[StructLayout(LayoutKind.Explicit)]
public struct _PROCESSOR_INFO_UNION
{
  [FieldOffset(0)]
  internal uint dwOemId;
  [FieldOffset(0)]
  internal ushort wProcessorArchitecture;
  [FieldOffset(2)]
  internal ushort wReserved;
}
static void Main(string[] args)
{
  SYSTEM_INFO sysinfo=new SYSTEM_INFO();
  GetSystemInfo(ref sysinfo);

  switch (sysinfo.uProcessorInfo.wProcessorArchitecture)
  {
    case 9:
      Console.WriteLine("İşlemci Tipi = AMDx64");
      break;
    case 6:
      Console.WriteLine("İşlemci Tipi = Itaniumx64");
      break;
    case 0:
      Console.WriteLine("İşlemci Tipi = Intelx86");
      break;
    default:
      Console.WriteLine("İşlemci Tipi bilinmiyor");
      break;
  }
  Console.WriteLine("İşlemci Sayısı = " +
```

```
        sysinfo.dwNumberOfProcessors.ToString());
    Console.WriteLine("Cache Bellek Miktarı = " +
        sysinfo.dwPageSize.ToString());
    Console.ReadKey();
}
```

Kodumuzda **wProcessorArchitecture** özelliği ile işlemci tipini, **dwNumberOfProcessors** özelliği ile işlemci çekirdeği sayısını, **dwPageSize** özelliği ile de cache bellek miktarını öğreniyoruz.



Ram Bellek Hakkında Bilgi Alma

Ram bellek hakkında bilgi almak için **kernel32.dll** dosyasını kullanabiliriz. Bu işlem için bu dll dosyasının **GlobalMemoryStatusEx** metodunu kullanacağız.

GlobalMemoryStatusEx metodunun kullanımı şu şekildedir:

```
[return: MarshalAs(UnmanagedType.Bool)]
[DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]
static extern bool GlobalMemoryStatusEx( [In,Out] MEMORYSTATUSEX lpBuffer);
```

- **lbBuffer**: Bellek hakkında hangi tür bilgi alınacağını belirler. Aşağıdaki değerleri alabilir:

  dwLength: Structure boyutunu belirtir. **GlobalMemoryStatusEx** metodu çağırılmadan önce bu özellik ayarlanmalıdır.

  dwMemoryLoad: 0 ile 100 arasında bir değer alarak kullanılan fiziksel bellek yüzdesini belirtir.

  ullTotalPhys: Kullanılan toplam fiziksel bellek miktarını byte cinsinden belirtir.

  ullAvailPhys: Boş fiziksel bellek miktarını byte cinsinden belirtir.

  ullTotalPageFile: Toplam **Page File** (*Takas bellek dosyası*) miktarını byte cinsinden belirtir.

  ullAvailPageFile: Boş **Page File** miktarını belirtir.

  ullTotalVirtual: Kullanılan toplam sanal bellek miktarını byte cinsinden belirtir.

ullAvailVirtual: Boş sanal bellek miktarını byte cinsinden belirtir.

ullAvailExtendedVirtual: Daima 0 değerini alır.

Api dosyasını çağırma satırı aşağıdaki gibi olacak:

```
[DllImport("kernel32.dll")]
public static extern void GlobalMemoryStatusEx(ref MEMORYSTATUSEX hafıza);
```

Tanımlama satırları ise aşağıdaki gibi olacak:

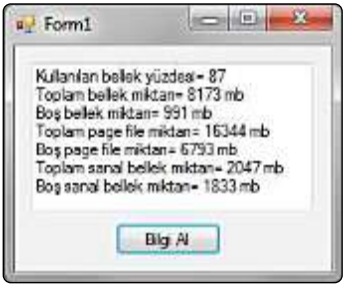
```
MEMORYSTATUSEX hafıza = new MEMORYSTATUSEX();
hafıza.dwLength = 64;
GlobalMemoryStatusEx(ref hafıza);
```

Sonuç olarak kodumuz aşağıdaki gibi olacak:

```
[DllImport("kernel32.dll")]
public static extern void GlobalMemoryStatusEx
     (ref MEMORYSTATUSEX hafıza);
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto)]
public struct MEMORYSTATUSEX
{
  public uint dwLength;
  public uint dwMemoryLoad;
  public ulong ullTotalPhys;
  public ulong ullAvailPhys;
  public ulong ullTotalPageFile;
  public ulong ullAvailPageFile;
  public ulong ullTotalVirtual;
  public ulong ullAvailVirtual;
  public ulong ullAvailExtendedVirtual;
}
private void btnBilgiAl_Click(object sender, EventArgs e)
{
  MEMORYSTATUSEX hafıza = new MEMORYSTATUSEX();
  hafıza.dwLength = 64;
  GlobalMemoryStatusEx(ref hafıza);

  textBox1.Text = textBox1.Text + "Kullanılan bellek yüzdesi= " +
    (hafıza.dwMemoryLoad) + "\r\n";
```

```
  textBox1.Text = textBox1.Text + "Toplam bellek miktarı= " +
    (hafıza.ullTotalPhys / (1024 * 1024)) + " mb \r\n";
  textBox1.Text = textBox1.Text + "Boş bellek miktarı= " +
    (hafıza.ullAvailPhys / (1024 * 1024)) + " mb \r\n";
  textBox1.Text = textBox1.Text + "Toplam page file miktarı= " +
    (hafıza.ullTotalPageFile / (1024 * 1024)) + " mb \r\n";
  textBox1.Text = textBox1.Text + "Boş page file miktarı= " +
    (hafıza.ullAvailPageFile / (1024 * 1024)) + " mb \r\n";
  textBox1.Text = textBox1.Text + "Toplam sanal bellek miktarı= " +
    (hafıza.ullTotalVirtual / (1024 * 1024)) + " mb \r\n";
  textBox1.Text = textBox1.Text + "Boş sanal bellek miktarı= " +
    (hafıza.ullAvailVirtual / (1024 * 1024)) + " mb";
}
```



Masaüstü Resmini Değiştirme

Masaüstü resmini değiştirmek için **user32.dll** dosyasını kullanabiliriz. Bu dll dosyasının **`SystemParametersInfo`** metodundan faydalanacağız. Kullanımı şu şekildedir:

```
[DllImport("user32.dll", CharSet = CharSet.Auto)]
    private static extern Int32 SystemParametersInfo(UInt32
uiAction, UInt32 uiParam, String pvParam, UInt32 fWinIni);
```

Kodumuz aşağıdaki gibi olacak:

```
[DllImport("user32.dll", CharSet = CharSet.Auto)]
private static extern Int32 SystemParametersInfo(UInt32 uiAction,
  UInt32 uiParam, String pvParam, UInt32 fWinIni);
private static UInt32 SPI_SETDESKWALLPAPER = 20;
private static UInt32 SPIF_UPDATEINIFILE = 0x1;
```

```
private String dosyaadi = "d:\\arka_plan.jpg";
private void btnMasaustunuDegistir_Click(object sender, EventArgs e)
{
  SystemParametersInfo(SPI_SETDESKWALLPAPER, 0,
    dosyaadi, SPIF_UPDATEINIFILE);
}
```

Kodumuzda **D** sürücüsü içerisindeki **arka_plan.jpg** isimli resim dosyası masaüstü resmi olarak ayarlanmıştır.

Ekran Çözünürlüğünü Bulma ve Değiştirme

Ekran çözünürlüğü ile ilgili işlemler için **user32.dll** dosyasını kullanabiliriz. Desteklenen ekran çözünürlüğü modlarını ve frekansları bulmak için bu dll dosyasının **EnumDisplaySettings** metodundan, ekran çözünürlüğünü değiştirmek için ise **ChangeDisplaySettings** metodundan faydalanacağız. Kullanımı şu şekildedir:

```
EnumDisplaySettings(string lpszDeviceName, int iModeNum, ref DEVMODE lpDevMode)
```

- **lpszDeviceName:** Görüntüleme cihazını belirtir.
- **iModeNum:** Çözünürlük modları değerlerini alır. Ya da aşağıdaki değerleri alabilir:

  ENUM_CURRENT_SETTINGS: Monitörün geçerli ayar bilgilerini döndürür.

  ENUM_REGISTRY_SETTINGS: Monitörün registry içerisinde saklanan ayar bilgilerini döndürür.

- **lpDevMode:** Grafik modları ile ilgili bilgileri içerir. Şu değerleri alır:

  dmBitsPerPel: Bit başına desteklenen renk sayısını belirtir. Örneğin 16 bit 65,536 renk gibi.

  dmPelsWidth: Çözünürlük değerlerinden genişlik değerini belirtir.

  dmPelsHeight: Çözünürlük değerlerinden yükseklik değerini belirtir.

  dmDisplayFlags: Monitörün görüntüleme biçimini belirtir. Şu değerleri alabilir:

  DM_GRAYSCALE: Monitörün renksiz olduğunu belirtir. Bu seçenek ayarlanmazsa cihazın renkli olduğu varsayılır.

DM_INTERLACED: Monitörün titreşimli olduğunu belirtir. Bu seçenek ayarlanmazsa cihazın titreşimsiz olduğu varsayılır.

dmDisplayFrequency: Monitörün tazeleme hızını (frekansını) hertz türünden belirtir.

ChangeDisplaySettings metodunun kullanımı şu şekildedir:

```
ChangeDisplaySettings(ref DEVMODE devMode, int flags)
```

- **DEVMODE**: Grafik modları ile ilgili bilgileri içerir. Alacağı değerler yukarıda anlatılmıştır.
- **flags**: Grafik modunun nasıl değiştirileceğini belirtir. Şu değerleri alır:

  0: Seçilen grafik modu dinamik olarak değiştirilir.

  CDS_FULLSCREEN: Bu seçenek zaten kullanılmaktadır. Ancak başka bir masaüstü kullanılması durumunda değiştirilemez.

  CDS_GLOBAL: Bu seçenek bilgisayarda bulunan bütün kullanıcılar için genel ayarları saklar. Aksi takdirde yalnızca işlem yapan kullanıcı için ayarlar saklanır. Bu seçenek yalnızca **CDS_UPDATEREGISTRY** seçeneği ile birlikte kullanılabilir.

  CDS_NORESET: Ayarlar registry içerisine kaydedilir, ama uygulanmaz. Bu seçenek yalnızca **CDS_UPDATEREGISTRY** seçeneği ile birlikte kullanılabilir.

  CDS_RESET: Eğer ayarlar geçerli ayarlar ile aynı ise, değişiklikler uygulanır.

  CDS_SET_PRIMARY: Belirtilen cihaz birincil cihaz önceliğini alacaktır.

  CDS_TEST: Belirtilen grafik modunu test eder.

  CDS_UPDATEREGISTRY: Seçilen grafik modu dinamik olarak değiştirilir ve registry içerisinde saklanan grafik bilgileri güncellenir.

```
[StructLayout(LayoutKind.Explicit, CharSet = CharSet.Ansi)]
public struct DEVMODE
{
  public const int CCHDEVICENAME = 32;
  public const int CCHFORMNAME = 32;

  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = CCHDEVICENAME)]
  [System.Runtime.InteropServices.FieldOffset(0)]
  public string dmDeviceName;
  [System.Runtime.InteropServices.FieldOffset(32)]
```

```
public Int16 dmSpecVersion;
[System.Runtime.InteropServices.FieldOffset(34)]
public Int16 dmDriverVersion;
[System.Runtime.InteropServices.FieldOffset(36)]
public Int16 dmSize;
[System.Runtime.InteropServices.FieldOffset(38)]
public Int16 dmDriverExtra;
[System.Runtime.InteropServices.FieldOffset(40)]
public DM dmFields;
[System.Runtime.InteropServices.FieldOffset(40)]
Int16 dmOrientation;
[System.Runtime.InteropServices.FieldOffset(46)]
Int16 dmPaperSize;
[System.Runtime.InteropServices.FieldOffset(48)]
Int16 dmPaperLength;
[System.Runtime.InteropServices.FieldOffset(50)]
Int16 dmPaperWidth;
[System.Runtime.InteropServices.FieldOffset(52)]
Int16 dmScale;
[System.Runtime.InteropServices.FieldOffset(54)]
Int16 dmCopies;
[System.Runtime.InteropServices.FieldOffset(56)]
Int16 dmDefaultSource;
[System.Runtime.InteropServices.FieldOffset(58)]
Int16 dmPrintQuality;
[System.Runtime.InteropServices.FieldOffset(52)]
public Int32 dmDisplayOrientation;
[System.Runtime.InteropServices.FieldOffset(56)]
public Int32 dmDisplayFixedOutput;
[System.Runtime.InteropServices.FieldOffset(60)]
public short dmColor;
[System.Runtime.InteropServices.FieldOffset(62)]
public short dmDuplex;
[System.Runtime.InteropServices.FieldOffset(64)]
public short dmYResolution;
[System.Runtime.InteropServices.FieldOffset(66)]
public short dmTTOption;
[System.Runtime.InteropServices.FieldOffset(68)]
public short dmCollate;
[System.Runtime.InteropServices.FieldOffset(72)]
```

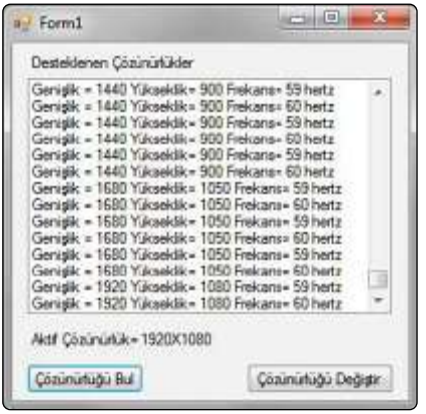
```
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = CCHFORMNAME)]
    public string dmFormName;
    [System.Runtime.InteropServices.FieldOffset(102)]
    public Int16 dmLogPixels;
    [System.Runtime.InteropServices.FieldOffset(104)]
    public Int32 dmBitsPerPel;
    [System.Runtime.InteropServices.FieldOffset(108)]
    public Int32 dmPelsWidth;
    [System.Runtime.InteropServices.FieldOffset(112)]
    public Int32 dmPelsHeight;
    [System.Runtime.InteropServices.FieldOffset(116)]
    public Int32 dmDisplayFlags;
    [System.Runtime.InteropServices.FieldOffset(116)]
    public Int32 dmNup;
    [System.Runtime.InteropServices.FieldOffset(120)]
    public Int32 dmDisplayFrequency;
  }
  [Flags()]
  public enum DM : int
  {
    Orientation = 0x1,
    PaperSize = 0x2,
    PaperLength = 0x4,
    PaperWidth = 0x8,
    Scale = 0x10,
    Position = 0x20,
    NUP = 0x40,
    DisplayOrientation = 0x80,
    Copies = 0x100,
    DefaultSource = 0x200,
    PrintQuality = 0x400,
    Color = 0x800,
    Duplex = 0x1000,
    YResolution = 0x2000,
    TTOption = 0x4000,
    Collate = 0x8000,
    FormName = 0x10000,
    LogPixels = 0x20000,
    BitsPerPixel = 0x40000,
    PelsWidth = 0x80000,
```

```
  PelsHeight = 0x100000,
  DisplayFlags = 0x200000,
  DisplayFrequency = 0x400000,
  ICMMethod = 0x800000,
  ICMIntent = 0x1000000,
  MediaType = 0x2000000,
  DitherType = 0x4000000,
  PanningWidth = 0x8000000,
  PanningHeight = 0x10000000,
  DisplayFixedOutput = 0x20000000
}
public enum DISP_CHANGE : int
{
  Successful = 0,
  Restart = 1,
  Failed = -1,
  BadMode = -2,
  NotUpdated = -3,
  BadFlags = -4,
  BadParam = -5,
  BadDualView = -1
}
[DllImport("user32.dll")]
public static extern bool EnumDisplaySettings(string lpszDeviceName,
   int iModeNum, ref DEVMODE lpDevMode);
[DllImport("user32.dll")]
public static extern DISP_CHANGE ChangeDisplaySettings(ref DEVMODE
   devMode, int flags);
DEVMODE vDevMode = new DEVMODE();
private void btnCozunurlukBul_Click(object sender, EventArgs e)
{
  listBox1.Items.Clear();
  int i=0;
  while ((EnumDisplaySettings(null, i, ref vDevMode)))
  {
    listBox1.Items.Add("Genişlik = " + vDevMode.dmPelsWidth + " " +
      "Yükseklik= " + vDevMode.dmPelsHeight + " " + "Frekans= " +
      vDevMode.dmDisplayFrequency + " hertz");
    i += 1;
  }
```

```
    Screen ekran = Screen.PrimaryScreen;
    lblAktifCozunurluk.Text = "Aktif Çözünürlük= " + ekran.Bounds.Width +
      "X" + ekran.Bounds.Height;
}
private void btnCozunurlukDegistir_Click(object sender, EventArgs e)
{
    DISP_CHANGE changeResult;
    const int ENUM_CURRENT_SETTINGS= -1;
    bool enumResult;

    enumResult = EnumDisplaySettings(vDevMode.dmDeviceName,
      ENUM_CURRENT_SETTINGS, ref vDevMode);
    vDevMode.dmSize = (short)Marshal.SizeOf(typeof(DEVMODE));
    vDevMode.dmPelsWidth = 800;
    vDevMode.dmPelsHeight = 600;
    vDevMode.dmDisplayFrequency = 60;

    changeResult = ChangeDisplaySettings(ref vDevMode, 0);
    if (changeResult != DISP_CHANGE.Successful)
      throw new Exception("Hata oluştu!" + changeResult.ToString());
    else
      MessageBox.Show("Çözünürlük ayarları değiştirildi");
}
```



Programımız içerisinde **`EnumDisplaySettings`** metodu ile desteklenen çözünürlük modlarını **`dmPelsWidth`** ve **`dmPelsHeight`** değerleriyle, frekans değerlerini ise **`dmDisplayFrequency`** değeri ile elde edip **ListBox** içerisine ekliyoruz.

Aktif ekran çözünürlük değerlerini **Screen** nesnesinin **Bounds.Width** ve **Bounds.Height** metotları ile elde edip görüntülüyoruz.

Ekran çözünürlüğünü değiştirmek için ise **ChangeDisplaySettings** metodunu aşağıdaki gibi kullanarak seçilen grafik modu dinamik olarak değiştirilir.

```
ChangeDisplaySettings(ref vDevMode, 0)
```

DisplayChangeResultCode metodunu kullanarak çözünürlük değiştirme işleminin sonucu hakkında bilgi sahibi olabiliriz. Yapılan işlem sonucunda geriye dönen değerler şunlardır:

- **Succesfull:** Değişiklik başarı ile uygulandı.
- **BadDualView:** Değişiklik uygulanamadı. Çünkü sistemde çift ekran etkin.
- **BadFlags:** Geçersiz çözünürlük değiştirme biçimi.
- **BadMode:** Seçilen grafik modu desteklenmiyor.
- **BadParam:** Geçersiz parametre kullanıldı.
- **Failed:** Seçilen grafik modunu ekran kartı sürücüsü desteklemiyor.
- **NotUpdated:** Yapılan değişiklikler registry içerisine yazılamadı.
- **Restart:** Grafik modu değiştirme işleminin ardından bilgisayarın yeniden başlatılması gerekiyor.

Geri Dönüşüm Kutusunu Kullanma

Geri dönüşüm kutusu üzerinde işlemler gerçekleştirmek için **shell32.dll** dosyasını kullanabiliriz. Geri dönüşüm kutusu içerisinde mevcut olan dosyalar hakkında bilgi almak için bu dll dosyasının **SHQueryRecycleBin** metodunu, geri dönüşüm kutusunu boşaltmak için ise **SHEmptyRecycleBinA** metodunu kullanacağız.

SHQueryRecycleBin metodunun kullanımı şu şekildedir:

```
SHQueryRecycleBin(string pszRootPath, ref SHQUERYRBINFO pSHQueryRBInfo)
```

- **pszRootPath:** Geri dönüşüm kutusunun konumlandığı sürücü yolunu belirtir.
- **pSHQueryRBInfo: SHQUERYRBINFO** yapısı yardımı ile geri dönüşüm kutusu bilgisini gönderir.

Bu metot ile birlikte kullanılan **SHQUERYRBINFO** yapısı şu şekildedir:

```
public struct SHQUERYRBINFO
{
  public int cbSize;
  public long i64Size;
  public long i64NumItems;
}
```

- **cbSize**: Structure boyutunu belirtir. Bu özellik çağıran fonksiyon tarafından belirlenir.
- **i64Size**: Geri dönüşüm kutusu içerisindeki öğelerin toplam boyutunu byte türünden belirtir.
- **i64NumItems**: Geri dönüşüm kutusu içerisindeki öğelerin toplam sayısını belirtir.

SHEmptyRecycleBin metodunun kullanımı şu şekildedir:

```
SHEmptyRecycleBin(IntPtr hWnd, string pszRootPath, uint dwFlags)
```

- **hWnd**: İşlem yapılırken görüntülenecek olan herhangi bir diyalog penceresinin handle kodu. Bu değer boş bırakılabilir.
- **pszRootPath**: Geri dönüşüm kutusunun konumlandığı sürücü yolunu belirtir. Eğer bu değer boş bırakılırsa, bütün sürücülere konumlanmış olan geri dönüşüm kutusu içerikleri boşaltılır.
- **dwFlags**: Aşağıdaki bir ya da daha fazla değeri içerebilir.

  SHERB_NOCONFIRMATION: Geri dönüşüm kutusunun boşaltılması sırasında herhangi bir diyalog penceresi görüntülenmez.

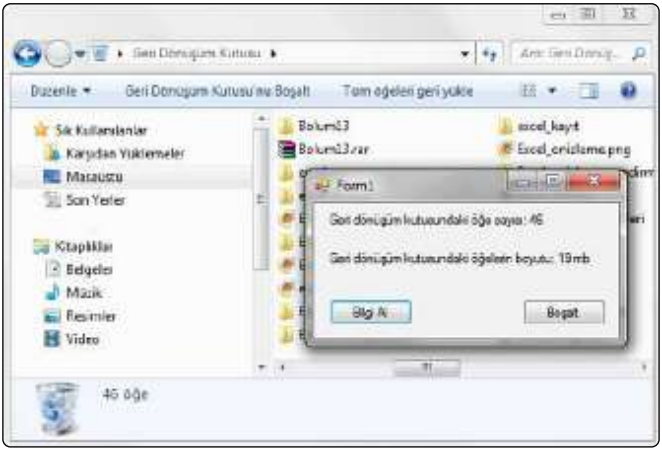
  SHERB_NOPROGRESSUI: Geri dönüşüm kutusunun boşaltılması sırasında ilerleme çubuğu (*progress bar*) görüntülenmez.

  SHERB_NOSOUND: İşlem tamamlandığında herhangi bir ses dosyası çalınmaz.

```
[DllImport("shell32.dll")]
static extern int SHQueryRecycleBin(string pszRootPath,
  ref SHQUERYRBINFO pSHQueryRBInfo);
[StructLayout(LayoutKind.Sequential, Pack = 4)]
public struct SHQUERYRBINFO
{
  public int cbSize;
  public long i64Size;
  public long i64NumItems;
}

[DllImport("shell32.dll")]
static extern int SHEmptyRecycleBin(IntPtr hWnd, string pszRootPath,
  uint dwFlags);
const int SHERB_NOCONFIRMATION = 0x00000001;
const int SHERB_NOPROGRESSUI = 0x00000002;
const int SHERB_NOSOUND = 0x00000004;
private void btnBilgiAl_Click(object sender, EventArgs e)
{
  SHQUERYRBINFO shqrbinf = new SHQUERYRBINFO();
  shqrbinf.cbSize = Marshal.SizeOf(typeof(SHQUERYRBINFO));
  int hresult = SHQueryRecycleBin(string.Empty, ref shqrbinf);
  lblSayi.Text = lblSayi.Text + " " +
    (Convert.ToInt32(shqrbinf.i64NumItems).ToString());
  lblBoyut.Text = lblBoyut.Text + " " +
    (Convert.ToInt32(shqrbinf.i64Size /
    (1024 * 1024)).ToString()) + " mb";
}
private void btnBosalt_Click(object sender, EventArgs e)
{
  SHEmptyRecycleBin(IntPtr.Zero, "C:",
    SHERB_NOCONFIRMATION | SHERB_NOPROGRESSUI | SHERB_NOSOUND);
  MessageBox.Show("Geri dönüşüm kutusu boşaltıldı");
}
```

Programımız ile aldığımız bilgilerle Windows gezgini yardımı ile aldığımız bilgileri karşılaştırarak programımızın doğru çalıştığını kontrol ettik.



İNTERNET BAĞLANTISINI TEST ETME

İnternet bağlantısını test etmek için **wininet.dll** dosyasını kullanabiliriz. İnternete bağlı olup olmadığımızı, bağlı isek hangi tip bağlantı ile bağlandığımızı bulmak için `InternetGetConnectedState` metodunu kullanacağız.

`InternetGetConnectedState` metodunun kullanımı şu şekildedir:

```
[DllImport("wininet.dll", CharSet = CharSet.Auto)]
extern static bool InternetGetConnectedState(ref
InternetGetConnectedStateFlags Description, int ReservedValue);
```

- `InternetGetConnectedStateFlags`: Bağlantı tipini belirtir ve şu değerleri alır:

```
INTERNET_CONNECTION_MODEM = 0x01,
INTERNET_CONNECTION_LAN = 0x02,
INTERNET_CONNECTION_PROXY = 0x04,
INTERNET_CONNECTION_RAS_INSTALLED = 0x10,
INTERNET_CONNECTION_OFFLINE = 0x20,
INTERNET_CONNECTION_CONFIGURED = 0x40
```

- **ReservedValue**: Bu parametre rezerve edilmiştir ve 0 değerini alır.

```
[DllImport("wininet.dll", CharSet = CharSet.Auto)]
extern static bool InternetGetConnectedState(ref
  InternetGetConnectedStateFlags Description, int ReservedValue);
[Flags]
public enum InternetGetConnectedStateFlags
{
  INTERNET_CONNECTION_MODEM = 0x01,
  INTERNET_CONNECTION_LAN = 0x02,
  INTERNET_CONNECTION_PROXY = 0x04,
  INTERNET_CONNECTION_RAS_INSTALLED = 0x10,
  INTERNET_CONNECTION_OFFLINE = 0x20,
  INTERNET_CONNECTION_CONFIGURED = 0x40
}
private void btnKontrolEt_Click(object sender, EventArgs e)
{
  InternetGetConnectedStateFlags flags=0;
  bool baglantı = InternetGetConnectedState(ref flags, 0);
  if (baglantı)
    MessageBox.Show("İnternet bağlantısı var.");
  else
    MessageBox.Show("İnternet bağlantısı yok.");
}
```

Eğer istenirse aşağıdaki gibi **InternetGetConnectedStateFlags** parametresine dönen değer okunarak, buna göre bağlantı türü belirlenebilir.

```
if (flags== InternetGetConnectedStateFlags.INTERNET_CONNECTION_LAN)
  MessageBox.Show("Bağlantı türü LAN.");
```

Ses Dosyalarını Çalma

Ses işlemleri için **winmm.dll** dosyasını kullanabiliriz. Bilgisayarda yüklü olan ses donanımı bilgisi için **waveOutGetNumDevs** metodunu kullanırken, ses dosyalarını çalmak için **PlaySound** metodunu kullanacağız.

waveOutGetNumDevs metodu geriye **integer** bir değer döndürür, dönen değer mevcut ses kartı ya da sürücüsünün sayısını verecektir. Eğer geriye dönen değer 0 ise, bilgisayarda ses kartı mevcut değildir ya da sürücüsü yüklenmemiştir.

PlaySound metodunun kullanımı ise şu şekildedir:

```
PlaySound(string pszSound, UIntPtr hmod, uint fdwSound)
```

- **pszSound:** Çalınacak olan ses dosyası adı ve yolunu belirtir.
- **hMod:** Resource dosyası kullanılıp kullanılmadığını belirtilir. fdwSound parametresinde **SND_RESOURCE** değeri kullanılmamışsa bu değer boş bırakılmalıdır.
- **fdwSound:** Ses dosyasının yürütülme şeklini belirtir. Aşağıdaki değerleri alabilir:

  SND_ALIAS: **WIN.INI** dosyası veya registry dosyası içerisinde **PszSound** parametresi için belirtilen takma isimdir. **SND_FILENAME** veya **SND_RESOURCE** parametreleri ile birlikte kullanılmaz.

  SND_ASYNC: Ses dosyası arka planda çalınır. Ses dosyası çalınmaya başladığında program kontrolü devralınır. Çalma işlemi, **pszSound** parametresi boş bir değer alana kadar devam eder.

  SND_FILENAME: **pszSound** parametresi için bir dosya adını belirtir.

  SND_LOOP: Çalma işlemi, sürekli olarak **pszSound** parametresi boş bir değer alana kadar devam eder. Bu parametre **SND_ASYNC** parametresi ile birlikte kullanılmalıdır.

  SND_MEMORY: Ses dosyası RAM belleğe yüklenir.

  SND_NODEFAULT: Ses dosyası bulunamadığında varsayılan ses dosyası çalınmadan çalma işlemi sonlandırılır.

  SND_NOSTOP: O an çalınmakta olan bir ses dosyası varsa yeni bir çalma işlemi başlatılmaz.

  SND_PURGE: Eğer ses kartı sürücüsü meşgul ise ses dosyası çalınmaz.

  SND_NOWAIT: Eğer ses kartı sürücüsü meşgul ise beklenilmez. **SND_FILENAME** veya **SND_RESOURCE** parametreleri ile birlikte kullanılmaz.

  SND_SYNC: Ses dosyası çalınma işlemi tamamlandığında program kontrolü devralınır.

  SND_RESOURCE: Resource dosyası adını belirtir.

Örnek olarak bir klasördeki **wav** uzantılı ses dosyalarını, bir **ListBox** içerisinde listeleyerek tek tek ya da tümünü sıra ile çalan bir uygulama yapalım.

```
using System;
using System.Runtime.InteropServices;
using System.Windows.Forms;
using System.IO;
namespace Api_PlaySound
{
 public partial class Form1 : Form
 {
   public Form1()
   {
     InitializeComponent();
   }
   [DllImport("winmm.dll", SetLastError = true)]
   static extern bool PlaySound(string pszSound,
     UIntPtr hmod, uint fdwSound);
   [DllImport("winmm.dll", SetLastError = true)]
   static extern uint waveOutGetNumDevs();

   [Flags]
   public enum SoundFlags
   {
     SND_SYNC = 0x0000,
     SND_ASYNC = 0x0001,
     SND_NODEFAULT = 0x0002,
     SND_MEMORY = 0x0004,
     SND_LOOP = 0x0008,
     SND_NOSTOP = 0x0010,
     SND_PURGE = 0x40,
     SND_NOWAIT = 0x00002000,
     SND_ALIAS = 0x00010000,
     SND_ALIAS_ID = 0x00110000,
     SND_FILENAME = 0x00020000,
     SND_RESOURCE = 0x00040004
   }
   private void btnListele_Click(object sender, EventArgs e)
   {
    listBox1.Items.Clear();
    if (txtYol.Enabled)
    {
     foreach (string bulunandosya in Directory.GetFiles
```

```
        (txtYol.Text, "*.wav", SearchOption.TopDirectoryOnly))
       listBox1.Items.Add(bulunandosya);
      }
     }
     private void btnCal_Click(object sender, EventArgs e)
     {
      int i = 0;
      if (waveOutGetNumDevs() == 0)
       MessageBox.Show("Ses kartı yok ya da sürücüsü yüklü değil");
      else
      {
       if (checkBox1.Checked)
       {
        while (i < listBox1.Items.Count)
        {
         listBox1.SetSelected(i, true);
         PlaySound(listBox1.Items[i].ToString(),
           UIntPtr.Zero, (uint)(SoundFlags.SND_FILENAME |
           SoundFlags.SND_NOSTOP));
         i++;
        }
       }
       else
        PlaySound(listBox1.SelectedItem.ToString(),
          UIntPtr.Zero, (uint)(SoundFlags.SND_FILENAME |
          SoundFlags.SND_ASYNC));
      }
     }
   }
}
```



Örneğimizde **My.Computer.FileSystem.GetFiles** metodu ile **TextBox** nesnesi içerisine yazılan klasör yolu içerisinde mevcut olan **wav** uzantılı dosyalar bulunarak **ListBox** içerisine aktarılmıştır.

waveOutGetNumDevs metodu ile bilgisayarda yüklü ses kartı ya da sürücüsü olup olmadığının kontrolünü yaptırıyoruz. Eğer dönen değer **0** ise; "*Ses kartı yok ya da sürücüsü yüklü değil*" şeklindeki bir mesaj ile kullanıcıyı uyarıyoruz. Dönen değer **0**'dan farklı ise; çalma işlemi ile ilgili kodları yürütüyoruz.

Eğer **checkBox1** işaretli değilse **PlaySound** metodunu…

```
(SoundFlags.SND_FILENAME | SoundFlags.SND_ASYNC)
```

… parametreleri ile birlikte kullanarak **ListBox** içerisinde seçili olan **wav** dosyasının normal bir şekilde yürütülmesini sağlıyoruz.

Eğer **checkBox1** işaretli ise **PlaySound** metodunu…

```
(SoundFlags.SND_FILENAME | SoundFlags.SND_NOSTOP)
```

… parametreleri ile birlikte kullanarak **ListBox** içerisinde mevcut olan **wav** dosyaları sıra ile çalınırken, bir dosyanın çalınma işlemi bittikten sonra diğer parçanın çalınması işlemine geçilmesini sağlıyoruz. Aksi takdirde sesler birbirine karışacaktır.

Ses Ayarlarını Değiştirme

Ses ayarlarını değiştirmek için **winmm.dll** dosyasını kullanabiliriz. Bilgisayarın ses seviyesini öğrenmek için **waveOutGetVolume** metodunu kullanırken, ses seviyesini değiştirmek için **waveOutSetVolume** metodunu kullanacağız.

waveOutGetVolume metodunun kullanımı şu şekildedir:

```
waveOutGetVolume(IntPtr hwo, uint dwVolume)
```

waveOutSetVolume metodunun kullanımı ise şu şekildedir:

```
waveOutSetVolume(IntPtr uDeviceID, uint dwVolume)
```

- **hwo**: Ses çıkış aygıtının handle numarasını belirtir.

- **`dwVolume`**: Ses seviyesini belirtir. **Hexadesimal** değer alır. **`0xFFFF`** değeri en üst ses seviyesini belirtirken, **`0x0000`** değeri en alt ses seviyesini belirtir.

```
[DllImport("winmm.dll", SetLastError = true, CharSet = CharSet.Auto)]
public static extern uint waveOutGetVolume(IntPtr hwo, uint dwVolume);
[DllImport("winmm.dll", SetLastError = true, CallingConvention
= CallingConvention.Winapi)]
public static extern int waveOutSetVolume(IntPtr uDeviceID, uint dwVolume);
private void Form1_Load(object sender, EventArgs e)
{
  uint ses = 0;
  waveOutGetVolume(IntPtr.Zero, ses);
  ushort sesduzeyi = Convert.ToUInt16((ses & 0xffff));
  trackBar1.Value = sesduzeyi / (ushort.MaxValue / 10);
}
private void TrackBar1_Scroll(object sender, EventArgs e)
{
  int yenises = ((ushort.MaxValue / 10) * trackBar1.Value);
  uint sonucses = (((uint)yenises & 0xffff) | ((uint)yenises << 16));
  waveOutSetVolume(IntPtr.Zero, sonucses);
}
```

Örneğimiz için form üzerine bir adet **TrackBar** nesnesi ekleyerek, **`Minimum`** değerini **`0`**, **`Maximum`** değerini **`10`**, **`SmallChange`** değerini **`1`** olarak ayarlıyoruz.



Video Dosyalarını Yürütme

Video dosyalarını yürütmek için **winmm.dll** dosyasını kullanabiliriz. Bu işlem için **`mciSendString`** metodunu kullanacağız.

`mciSendString` metodunun kullanımı şu şekildedir:

```
mciSendString(string command, StringBuilder buffer, int bufferSize,
IntPtr hwndCallback)
```

- **`command`**: Video işlemlerinin yürütülmesinde kullanılacak komutları belirtir. Alabileceği değerlerden bazıları şunlardır:

```
break           info        quality
capability      load        record
capture         open        resume
close           paste       save
copy            pause       seek
cut             play        setaudio
delete          put         spin
stop            sysinfo     undo
where
```

```
using System;
using System.Runtime.InteropServices;
using System.Windows.Forms;
using System.Text;
using System.IO;
namespace Api_MciSendString
{
 public partial class Form1 : Form
 {
   public Form1()
   {
     InitializeComponent();
   }
   [DllImport("winmm.dll")]
   static extern Int32 mciSendString(string command,
     StringBuilder buffer, int bufferSize, IntPtr hwndCallback);
   const int MM_MCINOTIFY = 953;
```

```
    private void btnListele_Click(object sender, EventArgs e)
    {
     listBox1.Items.Clear();
     if (txtYol.Enabled)
     {
       foreach (string bulunandosya in Directory.GetFiles(txtYol.Text,
             "*.wmv", SearchOption.TopDirectoryOnly))
         listBox1.Items.Add(bulunandosya);
     }
    }
    string komut_string;
    private void btnOynat_Click(object sender, EventArgs e)
    {
     komut_string = "Open " + listBox1.SelectedItem +
       " type mpegvideo alias Video";
     mciSendString(komut_string, null, 0, IntPtr.Zero);
     if (btnOynat.Text == "Oynat")
     {
       btnOynat.Text = "Durdur";
       komut_string = "Play Video";
       mciSendString(komut_string, null, 0, IntPtr.Zero);
     }
     else
     {
       btnOynat.Text = "Oynat";
       komut_string = "Pause Video";
       mciSendString(komut_string, null, 0, IntPtr.Zero);
     }
    }
    private void btnKapat_Click(object sender, EventArgs e)
    {
     komut_string = "Close Video";
     mciSendString(komut_string, null, 0, IntPtr.Zero);
    }
  }
}
```

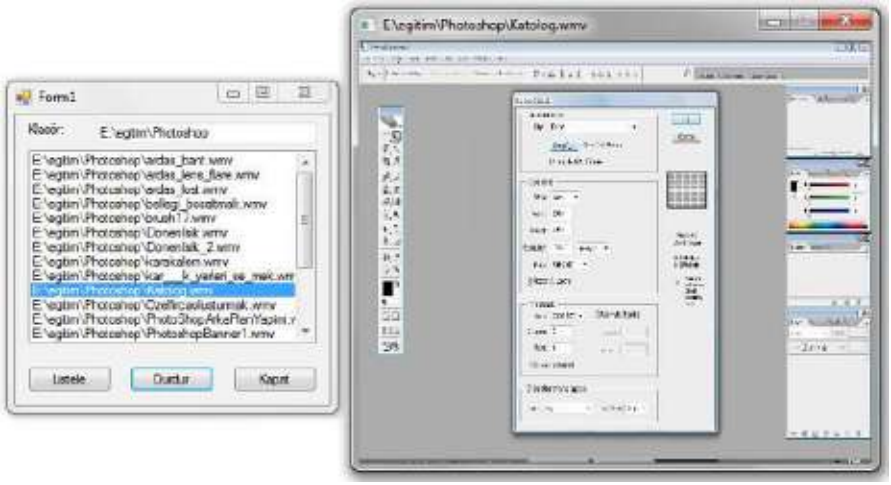
Kodumuzda yer alan…

```
komut_string = "Open " + listBox1.SelectedItem + " type mpegvideo alias Video";
mciSendString(komut_string, null, 0, IntPtr.Zero);
```

… satırları ile **`mciSendString`** metoduna **`Open`** komutunu göndererek video dosyasını yüklüyoruz. Yüklenen video dosyasının türünü **`mpegvideo`** olarak belirtiyoruz ki, diğer video türlerini de desteklesin. Örneğin; yalnızca **avi** uzantılı dosyaları yürütmesini isteseydik, dosya türü olarak **`AVIVideo`** yazmalıydık. **`alias Video`** yazarak **`mciSendString`** metoduna bundan sonra göndereceğimiz komutların hangi isimle gönderileceği belirlenir. "*Play Video*" gibi.

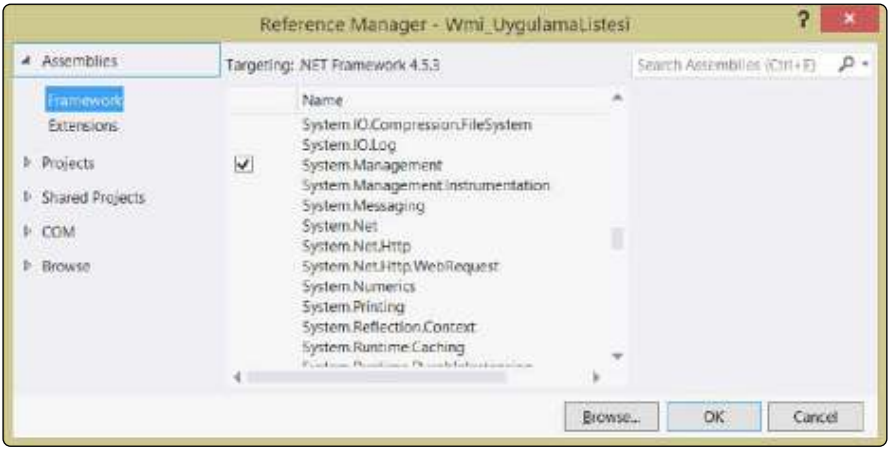
`Play` komutu ile video dosyasını oynatmaya başlarız. **`Pause`** ile duraklatıp, **`Close`** komutu ile de açık olan video penceresini kapatırız.

Örneğimizde **`Directory.GetFiles`** metodu ile **TextBox** nesnesi içerisine yazılan klasör yolu içerisinde mevcut olan **wmv** uzantılı dosyalar bulunarak, **ListBox** içerisine aktarılmıştır.



WMI (Windows Management Instrumentation) Sınıfı

WMI genellikle sistem özelliklerini sorgulamak amacıyla kullanılan bir sınıftır. Ayrıca uzak masaüstü bağlantıları, sistem servislerini durdurmak ya da başlatmak gibi işlemler için de kullanılmaktadır. **`WMI`** sınıfını kullanabilmek için uygulamamıza **`System.Management`** referansını dahil etmeniz gerekmektedir.



`WMI` sınıfını kullanırken **SQL** sorgularına benzer bir şekilde sorgu cümleleri yazarız. Bu sorgu cümlelerine **WQL** (*Windows Query Language*) adı verilmektedir. Sorgularımızı **`ManagementObjectSearcher`** sınıfını kullanarak yazarız.

Örneğin; hard disk ile ilgili bilgileri listeleyen bir uygulama yapalım. Yeni bir **Windows Form** uygulaması açarak içerisine 1 adet **ListBox**, 1 adet de **Button** ekleyelim. Kodumuz aşağıdaki gibi olacak:

```
using System;
using System.Windows.Forms;
using System.Management;
namespace WMI_DiskBilgi
{
 public partial class Form1 : Form
 {
  public Form1()
```

```
  {
   InitializeComponent();
  }
  private void btnListele_Click(object sender, EventArgs e)
  {
   ManagementObjectSearcher disk = new ManagementObjectSearcher
    ("Select FreeSpace,Size,Name from Win32_LogicalDisk where
      DriveType=3");
   foreach (ManagementObject disk_bilgi in disk.Get())
   {
    listBox1.Items.Add("Sürücü İsmi: " + disk_bilgi["Name"].ToString());
    listBox1.Items.Add("Boyutu (gb) : " + string.Format("{0:00.0}",
      Convert.ToDouble(disk_bilgi["Size"])
      / (1024 * 1024 * 1024)).ToString());
    listBox1.Items.Add("Boş Alan (gb) : " + string.Format("{0:00.0}",
      Convert.ToDouble(disk_bilgi["FreeSpace"])
      / (1024 * 1024 * 1024)).ToString());
   }
  }
 }
}
```

Kodumuzda yer alan…

```
Select FreeSpace,Size,Name from Win32_LogicalDisk where DriveType=3
```

… sorgusu ile **`Win32_LogicalDisk`** namespace'i altında yer alan sabit disk sürücülerinin isim, boyut ve boş alan bilgilerini elde ediyoruz. Bu sorguyu **`ManagementObjectSearcher`** sınıfını kullanarak tanımlıyor ve sorgumuzu **`Get`** metodu ile çalıştırıyoruz.

Kodumuzu çalıştırdığımızda bilgisayarımızda mevcut olan sabit disk sürücülerinin isim, boyut ve boş alan bilgilerini bir **ListBox** içerisinde listeleyecektir.



Mikroişlemci özelliklerini listeleyen bir uygulama yapalım. Kodumuz aşağıdaki gibi olacak:

```
private void btnListele_Click(object sender, EventArgs e)
{
 ManagementObjectSearcher cpu = new ManagementObjectSearcher
   ("Select Name, NumberOfLogicalProcessors,L2CacheSize From Win32_Processor");
 foreach (ManagementObject cpu_bilgi in cpu.Get())
 {
  listBox1.Items.Add("İsmi : " +
    (cpu_bilgi["Name"]).ToString());
  listBox1.Items.Add("İşlemci Çekirdeği Sayısı : " +
    (cpu_bilgi["NumberOfLogicalProcessors"]).ToString());
  listBox1.Items.Add("L2 Cache Boyutu : " +
    (cpu_bilgi["L2CacheSize"]).ToString());
 }
}
```

Kodumuzda yer alan…

```
Select Name, NumberOfLogicalProcessors,L2CacheSize From Win32_Processor
```

… sorgusu ile **`Win32_Processor`** namespace'i altında yer alan mikroişlemcinin isim, L2Cache bellek miktarı ve işlemci çekirdeği sayısı bilgilerini elde ediyoruz.



`ManagementObjectCollection` sınıfını kullanarak sorgu sonucunu koleksiyon şeklinde elde edebilirsiniz. Yukarıda yaptığımız mikroişlemci özelliklerini görüntüleyen örneği, **`ManagementObjectCollection`** kullanarak yeniden yazalım.

```
private void btnListele_Click(object sender, EventArgs e)
{
 ManagementClass cpu = new ManagementClass("Win32_Processor");
 ManagementObjectCollection cpu_bilgi = cpu.GetInstances();
 foreach (ManagementObject m in cpu_bilgi)
 {
  listBox1.Items.Add("İsmi : " + m.Properties["Name"].Value.ToString());
  listBox1.Items.Add("İşlemci Çekirdeği Sayısı : " +
    m.Properties["NumberOfLogicalProcessors"].Value.ToString());
  listBox1.Items.Add("L2 Cache Boyutu : " +
    m.Properties["L2CacheSize"].Value.ToString());
 }
}
```

Kodumuzda **`ManagementClass`** sınıfını kullanarak **`WMI`** sorgumuzu yazdık.

```
ManagementObjectCollection cpu_bilgi = cpu.GetInstances();
```

Bu komut satırında **`cpu_bilgi`** adında **`ManagementObjectCollection`** türünden bir koleksiyon tanımlayarak, **`GetInstances`** metodu ile bu koleksiyon içerisine tüm mikroişlemci özelliklerini aktardık. Bu özellikleri daha sonra **`Properties["Özellik İsmi"]`** şeklinde elde ederek **ListBox** içerisine aktardık.

`InvokeMethod` metodunu kullanarak belirtilen bir metodu çalıştırabilirsiniz. Örneğin; bilgisayarda yüklü uygulamaları listeleyip, seçtiğimiz uygulamayı bilgisayardan kaldıran bir uygulama yapalım. Kodumuz aşağıdaki gibi olacak:

```
private void btnListele_Click(object sender, EventArgs e)
{
 listBox1.Items.Clear();
 ManagementObjectSearcher programlar = new ManagementObjectSearcher
   ("SELECT * FROM Win32_Product");

 foreach (ManagementObject m in programlar.Get())
   if (m.Properties["Name"].Value != null)
     listBox1.Items.Add(m["Name"].ToString());
 label1.Text = listBox1.Items.Count + " adet program yüklü";
}
private void btnKaldır_Click(object sender, EventArgs e)
```

```
{
 ManagementObjectSearcher programlar = new ManagementObjectSearcher
   ("SELECT * FROM Win32_Product WHERE Name = '" +
   listBox1.SelectedItem.ToString() + "'");

 foreach (ManagementObject m in programlar.Get())
   if (m["Name"].ToString() == listBox1.SelectedItem.ToString())
   {
    try
    {
      m.InvokeMethod("Uninstall", null);
      MessageBox.Show(listBox1.SelectedItem.ToString() +
        " uygulaması başarıyla kaldırıldı");
    }
    catch
    {
      MessageBox.Show(listBox1.SelectedItem.ToString() +
        " uygulaması kaldırılamadı");
    }
  }
}
```

Kodumuzda yer alan…

```
SELECT * FROM Win32_Product
```

… sorgusunu kullanarak, yüklü olan uygulamaları listeledik. **ListBox** içerisinde seçili olan uygulamayı kaldırmak için ise öncelikle sorgumuzu…

```
SELECT * FROM Win32_Product WHERE Name = '" +
   listBox1.SelectedItem.ToString() + "'
```

… şeklinde değiştiriyoruz.

```
foreach (ManagementObject m in programlar.Get())
  if (m["Name"].ToString() == listBox1.SelectedItem.ToString())
```

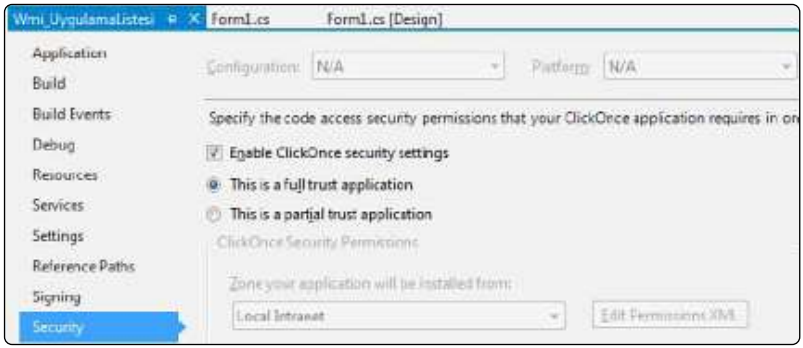
… satırlarında program listesini `Get` metodu ile elde ederek, **ListBox** içerisinde seçili olan program ismi ile döngü içerisinde listelenmekte olan sıradaki program ismini karşılaştırıyoruz.

Eğer aynı ise…

```
m.InvokeMethod("Uninstall", null)
```

… satırında **`InvokeMethod`** metodu ile **`Uninstall`** metodunu çalıştırıyoruz.

Seçili olan uygulamayı bilgisayardan kaldırabilmek için uygulamamıza **Administrator** yetkisi vermemiz gerekmektedir. Bunun için **Project / Proje Adı Properties** menüsünü seçiyoruz. Gelen pencerede **Security** sekmesinde **Enable ClickOnce security settings** seçeneği işaretliyoruz.



Bu işlemin ardından **Properties** klasörü içerisinde **app.manifest** dosyası oluşturulacaktır. Yeniden **Security** sekmesini açarak **Enable ClickOnce security settings** seçeneğinin işaretini kaldırıyoruz.



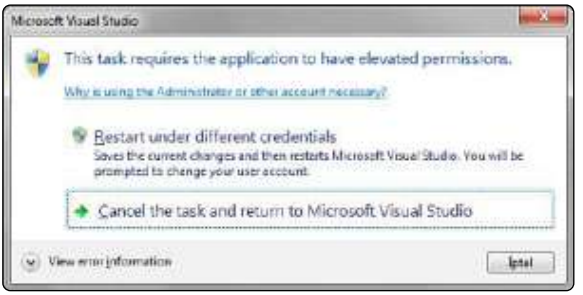
app.manifest dosyasını açarak içerisindeki…

```
<requestedExecutionLevel level="asInvoker" uiAccess="false" />
```

… satırını…

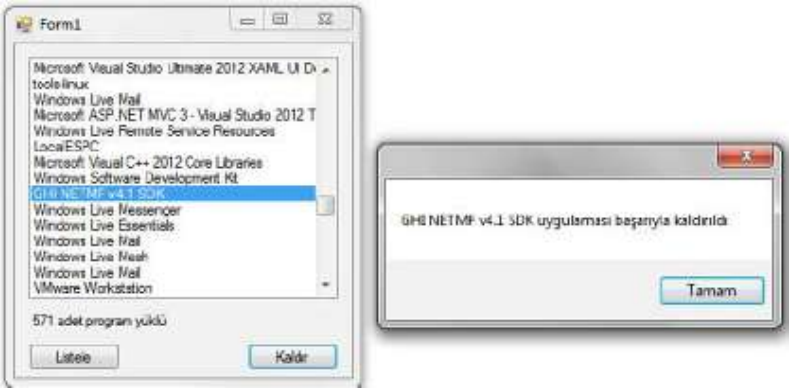
```
<requestedExecutionLevel level="requireAdministrator" uiAccess="false"/>
```

… şeklinde değiştirip uygulamamızı kaydedip çalıştırıyoruz. Karşımıza aşağıdaki gibi bir uyarı ekranı gelecektir. Yapılan değişikliklerin etkili olması için **Restart under different credentials** seçeneğini tıklıyoruz.



Bu işlemin ardından yapılan değişiklikler uygulanacak ve projemiz yeniden başlatılacaktır.

Uygulamamızı çalıştırarak test edelim.



Listele butonuna tıkladığımızda bilgisayarımızda yüklü olan uygulamalar listelendi. Herhangi bir uygulamayı seçerek **Kaldır** butonunu tıklayalım. Görüldüğü gibi uygulamamız başarı ile kaldırıldı.



Yeniden **Listele** butonuna tıkladığımızda, kaldırdığımız uygulamanın artık uygulama listesinde mevcut olmadığını görürüz.

Bilgisayarda o an çalışmakta olan uygulamaları listeleyerek, seçtiğimiz uygulamayı sonlandıran bir örnek yapalım. Kodumuz aşağıdaki gibi olacak:

```
private void btnListele_Click(object sender, EventArgs e)
{
 listBox1.Items.Clear();
 ManagementObjectSearcher programlar = new ManagementObjectSearcher
   ("SELECT * FROM Win32_Process");

 foreach (ManagementObject m in programlar.Get())
   if (m.Properties["Name"].Value != null)
     listBox1.Items.Add(m["Name"].ToString());

 label1.Text = listBox1.Items.Count + " adet uygulama çalışıyor";
}
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
 ManagementObjectSearcher programlar = new ManagementObjectSearcher
   ("SELECT * FROM Win32_Process WHERE Name = '" +
   listBox1.SelectedItem.ToString() + "'");

 foreach (ManagementObject m in programlar.Get())
   if (m["Name"].ToString() == listBox1.SelectedItem.ToString())
   {
     if (m.Properties["ExecutablePath"].Value != null)
       label2.Text=(m["ExecutablePath"].ToString());
   }
}
private void btnSonlandır_Click(object sender, EventArgs e)
{
 ManagementObjectSearcher programlar = new ManagementObjectSearcher
   ("SELECT * FROM Win32_Process WHERE Name = '" +
   listBox1.SelectedItem.ToString() + "'");

 foreach (ManagementObject m in programlar.Get())
   if (m["Name"].ToString() == listBox1.SelectedItem.ToString())
       m.InvokeMethod("Terminate", null);
}
```

Kodumuzda yer alan…

```
SELECT * FROM Win32_Process
```

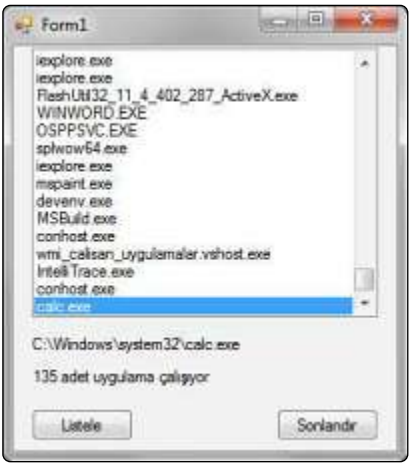
… sorgusunu kullanarak çalışmakta olan uygulamaları listeledik. **ListBox** içerisinde seçilen uygulamanın klasör yolunu bulmak için sorgumuzu…

```
SELECT * FROM Win32_Process WHERE Name = `" +
   listBox1.SelectedItem.ToString() + "'
```

… şeklinde değiştiriyoruz. **`Properties["ExecutablePath"]`** şeklinde çalışmakta olan uygulamanın klasör yolunu tespit ederek, **Label** içerisinde görüntülüyoruz. Aynı sorguyu kullanarak…

```
m.InvokeMethod("Terminate", null)
```

… satırında **`InvokeMethod`** metodu ile **`Terminate`** metodunu çalıştırarak uygulamamızı sonlandırıyoruz.



DOSYA VE KLASÖR İŞLEMLERİ

26

Sınırlı sayıdaki bilgilerin disk ortamında saklanması, bu bilgilere erişilmesi, üzerinde değişiklik yapılması gibi işlemler için veri tabanı oluşturmak yerine dosyalama işlemleri kullanılabilir. .Net; klasörler ve dosyalar üzerinde işlemler yapılması için yeterli sayıda metoda sahiptir. Bu bölümde kullanılacak olan yöntemler şunlardır:

- **`System.IO`** sınıfı kullanılarak
- **`IsolatedStorage`** sınıfı kullanılarak
- **`System.Text.Json`** sınıfı kullanılarak dosya ve klasör işlemlerini gerçekleştirmek

System.IO Sınıfı Kullanılarak Dosya ve Klasör İşlemlerini Gerçekleştirme

Dosyalar ve klasörler üzerinde işlem yapmak için **`System.IO`** sınıfı pek çok metoda sahiptir.

Dosyalar Üzerinde Yapılan İşlemler

`File` nesnesi ile dosyalar üzerinde birçok temel işlem gerçekleştirilebilir. **`File`** nesnesine ek olarak ayrıca **`FileStream`** nesnesi yardımı ile dosya oluşturabilir, **`StreamWriter`** nesnesi ile dosyalar içerisine yazı yazabilir ve **`StreamReader`** nesnesi yardımı ile de dosya içerisindeki bilgiler okunabilir.

File Nesnesi ve Metotları

Dosyalar üzerinde işlem yapmak için `File` nesnesi kullanılabilir. Aşağıdaki metotlara sahiptir.

- `AppendAllText`: Belirtilen bir metni dosya içerisine eklemek amacı ile kullanılır. Eğer dosya mevcut değilse oluşturulur.
- `AppendText`: Bir `StreamWriter` nesnesi oluşturarak var olan bir dosya içerisine **UTF-8** olarak kodlanmış metni ekler.
- `Copy`: Bir dosyayı yeni bir kopya olarak kopyalar.
- `Create`: Yeni bir dosya oluşturur.
- `CreateText`: **UTF-8** olarak kodlanmış olan yazıyı açar ya da oluşturur.
- `Decrypt`: Encrypt metodunu kullanarak şifrelenmiş olan bir dosyayı çözümler.
- `Encrypt`: Bir dosyayı şifrelemek için kullanılır.
- `Delete`: Belirtilen dosyayı siler. Eğer dosya mevcut değilse, geriye hata döndürülmez.
- `Exists`: Belirtilen dosyanın mevcut olup olmadığını bulmak amacı ile kullanılır.
- `GetAccessControl`: Belirtilen dosyanın **erişim kontrol listesi bilgisini** (*access control list (ACL)*) geriye döndürür.
- `GetAttributes`: Belirtilen dosyanın dosya özelliklerini bulur.
- `GetCreationTime`: Belirtilen dosya ya da klasörün oluşturulma tarihini geriye döndürür.
- `GetCreationTimeUtc`: Belirtilen dosya ya da klasörün oluşturulma tarihini **UTC** (*Coordinated Universal Time*) formatında geriye döndürür

NOT

UTC Vikipedi®'de şu şekilde tarif edilmiştir:

UTC, yani eşgüdümlü evrensel zaman anlamına gelir. Teknik olarak aynı olmasa da, GMT ile aynı anlamda da kullanıldığı olur. İngilizce kısaltma CUT ve Fransızca kısaltma TUC arasında seçim yapılamadığından orta yol olarak UTC kısaltması seçilmiştir. İngiltere'de Greenwich'ten geçen meridyen referans kabul edilerek hesaplanır. Türkiye Doğu Avrupa Zaman Dilimi'ndedir; bu yüzden Türkiye saati UTC+2, yani UTC'den 2 saat ileridedir.

- `GetLastAccessTime`: Belirtilen dosya ya da klasöre son erişim zamanını geriye döndürür.

- **`GetLastAccessTimeUtc`:** Belirtilen dosya ya da klasöre son erişim zamanını UTC formatında geriye döndürür.
- **`GetLastWriteTime`:** Belirtilen dosya ya da klasörün son yazım zamanını geriye döndürür.
- **`GetLastWriteTimeUtc`:** Belirtilen dosya ya da klasörün son yazım zamanını UTC formatında geriye döndürür.
- **`Move`:** Belirtilen dosyayı eğer belirtilirse yeni bir adla taşımak amacı ile kullanılır.
- **`Open`:** Belirtilen yol ile bir FileStream nesnesi açar.
- **`OpenRead`:** Belirtilen dosyayı okumak üzere açar.
- **`OpenText`:** UTF-8 olarak kodlanmış bir dosyayı okunmak üzere açar.
- **`OpenWrite`:** Belirtilen dosyayı yazmak üzere açar.
- **`ReadAllBytes`:** Binary bir dosyayı açarak içeriğini bir byte dizisine aktarır ve dosyayı kapatır.
- **`ReadAllLines`:** Bir text dosyayı açarak, bütün satırlarını okur ve bir string dizisine aktarır ve dosyayı kapatır.
- **`ReadAllText`:** Bir text dosyayı açarak, bütün satırlarını okur ve bir string değişkene aktarır ve dosyayı kapatır.
- **`Replace`:** Bir dosyanın içeriğini belirtilen başka bir dosya ile değiştirir. Orijinal dosyayı siler ve değiştirilen dosyayı yedekler.
- **`SetAccessControl`:** Erişim kontrol listesi bilgisini (*access control list (ACL)*) değiştirmek amacı ile kullanılır.
- **`SetAttributes`:** Dosya niteliklerini değiştirir.
- **`SetCreationTime`:** Belirtilen dosya ya da klasörün oluşturulma tarihini ve zamanını değiştirir.
- **`SetCreationTimeUtc`:** Belirtilen dosya ya da klasörün oluşturulma tarihini ve zamanını UTC formatında değiştirir.
- **`SetLastAccessTime`:** Belirtilen dosya ya da klasöre son erişim tarihi ve zamanını formatında değiştirir.
- **`SetLastAccessTimeUtc`:** Belirtilen dosya ya da klasöre son erişim tarihi ve zamanını UTC formatında değiştirir.
- **`SetLastWriteTime`:** Belirtilen dosya ya da klasöre son yazım tarihi ve zamanını değiştirir.

- **SetLastWriteTimeUtc:** Belirtilen dosya ya da klasöre son yazım tarihi ve zamanını UTC formatında değiştirir.
- **WriteAllBytes:** Yeni bir dosya oluşturur, içerisine bir byte dizisi içerisindeki verileri aktarır ve dosyayı kapatır. Eğer dosya mevcutsa üzerine yazar.
- **WriteAllLines:** Yeni bir dosya oluşturur, içerisine bir string değişken içerisindeki verileri aktarır ve dosyayı kapatır. Eğer dosya mevcutsa üzerine yazar.
- **WriteAllText:** Yeni bir dosya oluşturur, içerisine bir dosyanın içerisindeki verileri aktarır ve dosyayı kapatır. Eğer dosya mevcutsa üzerine yazar.

Şimdi bu işlemlerin ayrıntılarına girerek bu metotların bazılarına dair örnekler yapalım:

Örnek olarak **File.Create** metodu ile bir text dosyası oluşturarak içerisine yazı yazalım.

File.Create Metodu

Create metodunun kullanımı şu şekildedir:

```
Create(Dosya ismi, Int32, FileOptions, FileSecurity)
```

- **Int32:** Buffer boyutunu belirtir.
- **FileOptions:** Dosya seçeneklerini belirtir. Şu değerleri alır:

  None: Herhangi bir özellik içermez.

  WriteThrough: Sistemin herhangi bir ara bellek vasıtası ile ve doğrudan disk üzerine yazacağını gösterir.

  Asynchronous: Bir dosyanın asenkron olarak okuma ve yazma için kullanılabileceğini gösterir.

  RandomAccess: Dosyaya rastgele erişilebileceğini gösterir. Örneğin; bir önbellek dosyasını optimize etmek için kullanılabilir.

  DeleteOnClose: Dosyanın kullanımda olmadığı zaman otomatik olarak silinen olduğunu göster.

  SequentialScan: Dosyanın sıralı erişimli dosya olduğunu gösterir.

  Encrypted: Dosyanın şifrelendiğini ve şifrelemede kullanılan aynı kullanıcı hesabı ile çözülebileceğini gösterir.
- **FileSecurity:** Dosyaya erişim ve güvenlik ayarlarını belirler.

Disk üzerinde bir text dosyası oluşturan örnek bir uygulama yapalım.

```
using System;
using System.IO;
using System.Text;
using System.Windows.Forms;
namespace FileStream_CreateFile
{
 public partial class Form1 : Form
 {
  public Form1()
  {
    InitializeComponent();
  }
  private void btnDosyayaYaz_Click(object sender, EventArgs e)
  {
   FileStream fs = File.Create("d:\\ornek.txt");
   byte[] text = new UTF8Encoding(true).GetBytes("www.kodlab.com");
   fs.Write(text, 0, text.Length);
   fs.Close();
   MessageBox.Show("Dosya oluşturuldu ve yazılar eklendi");
  }
 }
}
```

Örneğimizde…

```
FileStream fs = File.Create("d:\\ornek.txt");
```

… satırında **fs** adlı bir **FileStream** nesnesi türünden bir değişken tanımlayarak, bu değişkene **File.Create** metodu ile oluşturmuş olduğumuz dosyayı atadık.

```
byte[] text = new UTF8Encoding(true).GetBytes("www.kodlab.com");
```

text adı ile bir byte dizisi oluşturarak **UTF-8** ile kodlanmış string bilgimizi aktarıyoruz.

```
fs.Write(text, 0, text.Length);
```

Write metodu ile text adlı byte dizi içerisinde 0 nolu index değerinden başlayarak, dizinin son elemanına kadar olan karakterleri dosya içerisine yazdırıyoruz.

Write metodunun kullanımı ise şu şekildedir:

```
FileStream.Write(değişken adı, ilk elemanının index değeri, uzunluk)
```

File.WriteAll Metodu

WriteAll metodu ile tek bir komut ile yeni bir dosya oluşturup, içerisine bilgi yazmak ve dosyayı kapatmak mümkündür.

WriteAll metodunu kullanarak bir **TextBox** içerisindeki yazıları, bir dosya oluşturarak içerisine kaydeden bir uygulama yapalım

```
private void btnDosyayaYaz_Click(object sender, EventArgs e)
{
  File.WriteAllText("D:\\ornek.txt", textBox1.Text);
  MessageBox.Show("Dosya oluşturuldu ve yazılar eklendi");
}
```

Örneğimizde **WriteAllText** metodu ile **textBox1** içerisine girilmiş olan yazıyı, bir dosya oluşturarak içerisine ekliyor ve dosyayı kapatıyoruz.

File.ReadAllLines Metodu

Bir dosya içerisindeki bütün satırları okumak amacı ile kullanılan metottur.

Örnek olarak formun yüklenişi sırasında bir dosya içerisindeki değerleri okuyarak **ListBox**'a aktaran ve **ListBox** içerisine eklenen değerleri dosyaya yazan bir uygulama yapalım. Uygulamamıza **Microsoft.VisualBasic.dll** dosyasını referans olarak eklemeyi unutmayalım.

```
using System;
using Microsoft.VisualBasic;
using System.IO;
using System.Windows.Forms;
namespace FileStream_ReadAllLines
{
```

```
public partial class Form1 : Form
{
  public Form1()
  {
    InitializeComponent();
  }
  private void btnEkle_Click(object sender, EventArgs e)
  {
    string eklenecek;
    eklenecek = Interaction.InputBox("Eklenecek değeri girin");
    listBox1.Items.Add(eklenecek);
  }
  private void Form1_Load(object sender, EventArgs e)
  {
    try
    {
      String[] degerler = new String[] { };
      degerler = (File.ReadAllLines("D:\\ornek.txt"));
      for (int i = 0; i <= degerler.Length; i++)
        listBox1.Items.Add(degerler.GetValue(i));
    }
    catch
    { }
  }
  private void btnDosyayaYaz_Click(object sender, EventArgs e)
  {
    FileStream fs = new FileStream("D:\\ornek.txt",
      FileMode.Create, FileAccess.Write, FileShare.Write);
    StreamWriter sw = new StreamWriter(fs);
    for (int i = 0; i <= listBox1.Items.Count - 1; i++)
        sw.WriteLine(listBox1.Items[i]);
    sw.Close();
  }
}
}
```

Kodumuzda, **Form**'un **`Load`** olayı içerisinde disk üzerinde oluşturulmuş olan text dosya içeriğini okumak için **`degerler`** adında string bir dizi oluşturarak, text dosyası içerisinden **`ReadAllLines`** metodu ile okuduğumuz satırları içerisine aktarıyoruz. **`degerler`** dizisinin boyutu kadar bir döngü kurarak, bu döngü içerisinde **ListBox** içerisine **`degerler`** dizisinin elemanlarını ekliyoruz.

ListBox içerisindeki değerleri dosyaya yazdırmak için **ListBox** içerisindeki eleman sayısı kadar bir döngü kurarak, bu döngü içerisinde **`WriteLine`** metodu ile **ListBox** elemanlarını dosyaya yazıyoruz.

Dosya Özelliklerini Elde Etme ve Değiştirme

Dosyaların oluşturulma zamanı, son yazım tarihi, son erişim tarihi gibi bilgilere erişip bu bilgileri değiştirmek mümkündür. Bunun için aşağıdaki metotlar kullanılır.

- **`GetCreationTime`**: Belirtilen dosya ya da klasörün oluşturulma tarihini geriye döndürür.
- **`GetLastAccessTime`**: Belirtilen dosya ya da klasöre son erişim zamanını geriye döndürür.
- **`GetLastWriteTime`**: Belirtilen dosya ya da klasörün son yazım zamanını geriye döndürür.

`GetLastWriteTime` metodunun kullanımı şu şekildedir:

```
File.GetLastWriteTime(Dosya yolu)
```

Diğer iki metodun da kullanımı aynı şekildedir.

- **`SetCreationTime`**: Belirtilen dosya ya da klasörün oluşturulma tarihini ve zamanını değiştirir.
- **`SetLastAccessTime`**: Belirtilen dosya ya da klasöre son erişim tarihi ve zamanını değiştirir.
- **`SetLastWriteTime`**: Belirtilen dosya ya da klasöre son yazım tarihi ve zamanını değiştirir.

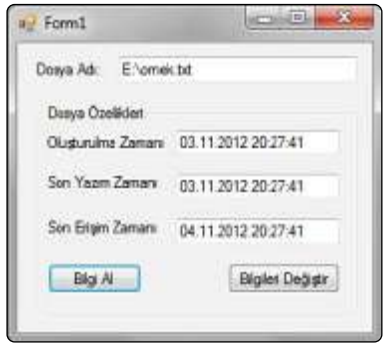
SetLastWriteTime metodunun kullanımı şu şekildedir:

```
File.SetLastWriteTime(Dosya yolu, tarih ve zaman)
```

Diğer iki metodun da kullanımı aynı şekildedir.

```
private void btnBilgiAl_Click(object sender, EventArgs e)
{
 txtOlusturulmaZamanı.Text =
```

```
    File.GetCreationTime(textBox1.Text).ToString();
  txtYazımZamanı.Text =
    File.GetLastWriteTime(textBox1.Text).ToString();
  txtErisimZamanı.Text =
    File.GetLastAccessTime(textBox1.Text).ToString();
}
private void btnBilgiDegistir_Click(object sender, EventArgs e)
{
  if (txtOlusturulmaZamanı.Text != "")
    File.SetCreationTime(textBox1.Text,
      Convert.ToDateTime(txtOlusturulmaZamanı.Text));
   if (txtYazımZamanı.Text != "")
    File.SetLastWriteTime(textBox1.Text,
      Convert.ToDateTime(txtYazımZamanı.Text));
   if (txtErisimZamanı.Text != "")
    File.SetLastAccessTime(textBox1.Text,
      Convert.ToDateTime(txtErisimZamanı.Text));
}
```



Bilgi Al butonuna tıklayarak belirtilmiş olan dosyanın erişim ve oluşturulma bilgileri ilgili **TextBox** nesneleri içerisinde listelenmekte ve **Bilgileri Değiştir** butonuna tıklayarak da ilgili **TextBox**'lar içerisine girilen tarih ve zaman bilgilerine göre dosya erişim ve oluşturulma bilgileri değiştirilmektedir.

Dosyaları Kopyalama, Taşıma ve Silme

Dosyaları kopyalamak için `Copy` metodu, taşımak için `Move` metodu silmek için de `Delete` metodu kullanılır.

Copy ve **Move** metotlarının kullanımı şu şekildedir:

File.Copy veya **Move(Kaynak dosya adı, Hedef dosya adı)**

Delete metodunun kullanımı ise şu şekildedir:

```
File.Delete(Kaynak dosya adı)
```

```
private void btnKopyala_Click(object sender, EventArgs e)
{
  File.Copy(txtKaynak.Text, txtHedef.Text);
}
private void btnTası_Click(object sender, EventArgs e)
{
  File.Move(txtKaynak.Text, txtHedef.Text);
}
private void btnSil_Click(object sender, EventArgs e)
{
  if (File.Exists(txtHedef.Text))
      File.Delete(txtHedef.Text);
}
```



Örneğimizde **File.Delete** metodu ile dosyayı silmeden önce, **File.Exist** metodu ile öncelikle dosyanın var olup olmadığını kontrol ediyoruz. **Exist** metodu dosya mevcutsa, geriye **true** değeri döndürür, mevcut değilse **false** değerini döndürür.

FileStream Nesnesi

Yeni bir dosya oluşturmak için kullanılır. Kullanımı şu şekildedir:

```
FileStream(String, FileMode, FileAccess, FileShare, Int32,
FileOptions, FileSecurity)
```

- **`FileMode`**: Dosya açılış şekillerini belirler.

  `Append`: Dosyayı açarak dosyanın sonuna ekleme yapmak için kullanılır. Eğer dosya bulunmazsa oluşturulur.

  `Create`: Yeni dosya oluşturmak için kullanılır. Eğer dosya mevcutsa üzerine yazılır.

  `CreateNew`: Yeni dosya oluşturmak için kullanılır, Eğer dosya mevcutsa hata meydana gelir.

  `Open`: Dosyayı açmak için kullanılır.

  `OpenOrCreate`: Belirtilen dosya mevcutsa açılır, yoksa yeni dosya oluşturulur.

  `Truncate`: Belirtilen dosya açılarak içi silinir.

- **`FileAccess`**: Dosya erişim şekillerini belirler.

  `Read`: Dosyayı okumak üzere açar.

  `Write`: Dosya sadece yazılmak için açılır.

  `ReadWrite`: Dosyayı okunmak ve yazılmak üzere açar.

- **`FileShare`**: Dosya paylaşım şekillerini belirler.

  `None`: Dosya kapatılıncaya kadar başka prosesler tarafından açılmasını engeller.

  `Read`: Dosya açık iken başka proseslerce de dosyanın açılıp okunabilmesini sağlar.

  `Write`: Dosya açık iken başka proseslerce de dosyaya yazılabilmesini sağlar.

  `ReadWrite`: Dosya açık iken aynı anda başka proseslerce de dosyanın açılıp, okunup, yazılabilmesini sağlar.

  `Inheritable`: Dosyanın child (yavru) prosesler tarafından türetilebilmesini sağlar. Win32 tarafından doğrudan desteklenmez.

- **`Int32`**: Buffer boyutunu belirtir.
- **`FileOptions`**: Dosya seçeneklerini belirtir.

  `None`: Herhangi bir özellik içermez.

  `WriteThrough`: Sistemin herhangi bir ara bellek vasıtası ile ve doğrudan disk üzerine yazacağını gösterir.

Asynchronous: Bir dosyanın asenkron olarak okuma ve yazma için kullanılabileceğini gösterir.

RandomAccess: Dosyaya rastgele erişilebileceğini gösterir. Örneğin bir önbellek dosyasını optimize etmek için kullanılabilir.

DeleteOnClose: Dosyanın, kullanımda olmadığı zaman otomatik olarak silinen olduğunu göster.

SequentialScan: Dosyanın sıralı erişimli dosya olduğunu gösterir.

Encrypted: Dosyanın şifrelendiğini ve şifrelemede kullanılan aynı kullanıcı hesabı ile çözülebileceğini gösterir.

- **FileSecurity:** Dosyaya erişim ve güvenlik ayarlarını belirler.

FileStream Nesnesinin Özellikleri

- **CanRead:** Aktif stream nesnesinin okumayı destekleyip desteklemediğini belirtir.
- **CanSeek:** Aktif stream nesnesinin erişimi destekleyip desteklemediğini belirtir.
- **CanTimeout:** Aktif stream nesnesinin **TimeOut** (*zaman aşımı*) olayını destekleyip desteklemediğini belirtir.
- **CanWrite:** Aktif stream nesnesinin yazmayı destekleyip desteklemediğini belirtir.
- **Handle:** Aktif stream nesnesi için işletim sistemi tarafından üretilen handle numarasını belirtir.
- **IsAsync:** Aktif stream nesnesinin senkron (*eşzamanlı*) ya da asenkron (*eş zamansız*) açılıp açılmadığını belirtir.
- **Length:** Aktif stream nesnesinin boyutunu verir.
- **Name:** Constructor'a geçilen aktif stream nesnesinin ismini verir.
- **Position:** Aktif stream nesnesinin pozisyonunu verir ya da değiştirir.
- **ReadTimeout:** Zaman aşımına uğramadan önce stream nesnesinin ne kadar süre ile okunabileceğini belirler ya da bu değeri verir. Milisaniye cinsinden değer alır.
- **SafeFileHandle:** Aktif stream nesnesi için işletimi sistemi tarafından üretilen **SafeFileHandle** numarasını belirtir.
- **WriteTimeout:** Zaman aşımına uğramadan önce stream nesnesinin ne kadar

süre ile yazılabileceğini belirler ya da bu değeri verir. Milisaniye cinsinden değer alır.

FileStream Nesnesinin Metodları

- **`BeginRead`**: Okuma işlemini başlatır.
- **`BeginWrite`**: Yazma işlemini başlatır.
- **`Close`**: Açık olan stream nesnesini kapatır.
- **`CreateWaitHandle`**: WaitHandle nesnesi ataması yapar.
- **`Dispose`**: Stream nesnesini siler.
- **`EndRead`**: Okuma işlemini sonlandırır.
- **`EndWrite`**: Yazma işlemini sonlandırır.
- **`Equals`**: Belirtilen nesnenin geçerli nesneye eşit olup olmadığı bilgisini döndürür.
- **`Finalize`**: Stream işlemi için ayrılmış kaynakları serbest bırakır.
- **`Flush`**: Tampondaki bilgilerin boşaltılmasını ve stream dosyanın güncellenmesini sağlar
- **`GetAccessControl`**: Belirtilen dosyanın Erişim kontrol listesi bilgisini (*access control list (ACL)*) geriye döndürür.
- **`GetType`**: Geçerli olan stream nesnesinin tipini bulur.
- **`Lock`**: FileStream nesnesinin diğer işlemler tarafından değiştirilmesini engeller.
- **`Read`**: Stream nesnesinden byte bloğunu okur ve bir tampon içerisine yazar.
- **`ReadByte`**: Stream nesnesinden bir byte okur ve bir byte sonraki pozisyona ilerler.
- **`Seek`**: Verilen değerin stream nesnesi içerisindeki konumunu belirler.
- **`SetAccessControl`**: Belirtilen dosyanın Erişim kontrol listesi bilgisini (*access control list (ACL)*) belirler.
- **`SetLength`**: Stream nesnesinin uzunluğunu belirler.
- **`ToString`**: Geçerli nesneyi string türünden döndürür.
- **`Unlock`**: FileStream nesnesine diğer işlemler tarafından erişilebilmesini sağlar.
- **`Write`**: Tampon içerisinden byte bloğunu okuyarak yazar.
- **`WriteByte`**: Stream nesnesi içerisindeki geçerli pozisyona bir byte yazar.

FileStream nesnesi ve **StreamWriter** metodunu kullanarak bir **TextBox** içerisindeki bilgileri bir dosya içerisine yazalım.

```
private void btnYaz_Click(object sender, EventArgs e)
{
  FileStream fs = new FileStream("D:\\ornek.txt",
    FileMode.Create, FileAccess.Write, FileShare.Write);
  StreamWriter sw = new StreamWriter(fs);
  sw.Write(textBox1.Text);
  sw.Close();
  MessageBox.Show("Dosya oluşturuldu ve yazılar eklendi");
}
```

Kodumuzda **fs** adlı bir **FileStream** nesnesi türünden bir değişken tanımlayarak, bu değişkene **File.Create** metodu ile oluşturmuş olduğumuz dosyayı atadık. Text dosyamız içerisine yazı ekleyeceğimiz için **Write** modunda, yani yazma modunda oluşturuyoruz. **Write** metodu ile de **TextBox** içerisindeki yazıyı text dosyası içerisine ekliyoruz.

StreamWriter nesnesi kullanılırken yazma işleminde **Write** ve **WriteLine** metotları kullanılabilir.

Write metodu ile bir ya da birden fazla karakter yazılabilirken,

WriteLine Metodu ile bir ya da birden fazla satır yazdırılabilir.

StreamReader nesnesini kullanarak, bir dosya içerisindeki bilgileri okuyarak text kutusu içerisine aktaralım.

```
private void btnOku_Click(object sender, EventArgs e)
{
  StreamReader sr = new StreamReader("d:\\ornek.txt");
  textBox1.Text = sr.ReadToEnd();
}
```

StreamReader nesnesi kullanılırken yazma işleminde **Read**, **ReadBlock**, **ReadLine** ve **ReadToEnd** metotları kullanılabilir.

Read metodu ile bir ya da birden fazla karakter okunabilir.

ReadBlock metodu ile kaç karakter okunabileceği, okunmaya başlayacak

karakterin index numarası gibi parametreler kullanılarak okuma işlemi yapılabilir.

ReadLine metodu ile bir ya da birden fazla satır okunabilir.

ReadToEnd metodu ile dosya sonuna kadar okuma yapılır.

Bir önceki örneğimizde **ReadToEnd** metodu ile yaptığımız işlemi, **ReadLine** metodu ile tekrar yapalım.

```
private void btnOku_Click(object sender, EventArgs e)
{
  StreamReader sr = new StreamReader("d:\\ornek.txt");
  string satır = "";
  do
  {
    satır = sr.ReadLine();
    textBox1.Text = textBox1.Text + satır + Environment.NewLine;
  }
  while (!(satır == null));
}
```

satır değişkeni içerisine dosya içerisinden satırları okuyarak aktarıyoruz. Bu işlem **satır** değişkeni boş değer alana kadar, yani dosya sonuna gelinene kadar devam ediyor.

Environment.NewLine ile **TextBox** içerisine her eklediğimiz satırın ardından, satır sonu karakteri ekliyoruz.

Klasörler Üzerinde Yapılan İşlemler

Directory nesnesi, klasörler üzerinde işlem yapmak için kullanılır. **Directory** nesnesinin başlıca özellik ve metotları şunlardır:

- **CreateDirectory**: Yeni bir klasör ya da alt klasör oluşturur.
- **Delete**: Bir klasörü içeriği ile beraber siler.
- **Exists**: Belirtilen yolda bir klasörün mevcut olup olmadığı bilgisini geriye döndürür.
- **GetCreationTime**: Belirtilen klasörün oluşturulma tarihini geriye döndürür.
- **GetCreationTimeUtc**: Belirtilen klasörün oluşturulma tarihini UTC formatında

geriye döndürür

- **`GetCurrentDirectory`**: Aktif çalışma dizinini geriye döndürür.
- **`GetDirectories`**: Belirtilen klasörün altındaki alt klasörlerin listesini geriye döndürür.
- **`GetDirectoryRoot`**: Belirtilen yol için disk bilgisini, kök dizin bilgisini ya da her ikisini birden geriye döndürür.
- **`GetFiles`**: Belirtilen klasör altındaki dosya isimlerini geriye döndürür.
- **`GetFileSystemEntries`**: Belirtilen klasör altındaki tüm alt klasör ve dosyaların isimlerini geriye döndürür.
- **`GetLastAccessTime`**: Belirtilen klasöre son erişim zamanını geriye döndürür.
- **`GetLastAccessTimeUtc`**: Belirtilen klasöre son erişim zamanını UTC formatında geriye döndürür.
- **`GetLastWriteTime`**: Belirtilen klasörün son yazım zamanını geriye döndürür.
- **`GetLastWriteTimeUtc`**: Belirtilen klasörün son yazım zamanını UTC formatında geriye döndürür.
- **`GetLogicalDrives`**: Bilgisayarda mevcut bulunan disk sürücü isimlerinin listesini verir.
- **`GetParent`**: Belirtilen klasörün bağlı olduğu ana klasör ismini geriye döndürür.
- **`Move`**: Belirtilen bir dosya ya da klasörü içeriği ile birlikte yeni bir konuma taşımak için kullanılır.
- **`SetCreationTime`**: Belirtilen dosya ya da klasörün oluşturulma tarihini değiştirir.
- **`SetCreationTimeUtc`**: Belirtilen dosya ya da klasörün oluşturulma tarihini ve zamanını UTC formatında değiştirir.
- **`SetCurrentDirectory`**: Aktif çalışma dizinini değiştirir.
- **`SetLastAccessTime`**: Belirtilen klasörün son erişim zamanı tarihini değiştirir.
- **`SetLastAccessTimeUtc`**: Belirtilen dosya ya da klasöre son erişim tarihi ve zamanını UTC formatında değiştirir.
- **`SetLastWriteTime`**: Bir klasöre son yazım tarihini değiştirir.
- **`SetLastWriteTimeUtc`**: Bir klasöre son yazım tarihini UTC formatında değiştirir.

Klasör Oluşturma, Silme

Klasör oluşturmak için **Directory.CreateDirectory** metodu kullanılırken, silmek için **Directory.Delete** metodu kullanılır.

CreateDirectory metodunun kullanımı şu şekildedir:

```
Directory.CreateDirectory(Klasör yolu ve adı)
```

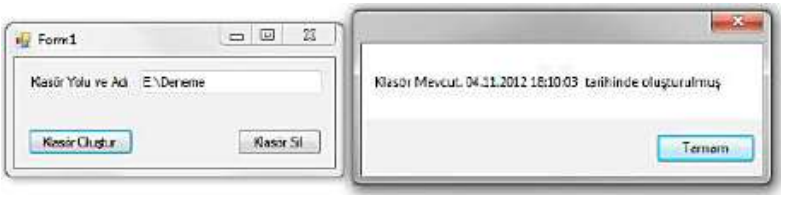
Delete metodunun kullanımı ise şu şekildedir:

```
Directory.Delete(Klasör yolu ve adı, boolean)
```

Boolean değer olarak **true** değeri verildiğinde, klasör içerisinde mevcut olan alt klasörler de silinir, **false** değeri verildiğinde ise alt klasörler silinmez.

```
private void btnKlasorOlustur_Click(object sender, EventArgs e)
{
  if (Directory.Exists(textBox1.Text))
  {
    MessageBox.Show("Klasör Mevcut. " +
      Directory.GetCreationTime(textBox1.Text) +
      " tarihinde oluşturulmuş");
  }
  else
  {
    Directory.CreateDirectory(textBox1.Text);
    MessageBox.Show("Klasör oluşturuldu");
  }
}
private void btnKlasorSil_Click(object sender, EventArgs e)
{
  if (Directory.Exists(textBox1.Text))
  {
    Directory.Delete(textBox1.Text, true);
    MessageBox.Show("Klasör silindi");
  }
  else
    MessageBox.Show("Klasör mevcut değil");
}
```

Exist metodu ile klasörün mevcut olup olmadığı bilgisini alıp, eğer klasör mevcutsa **GetCreationTime** metodu ile klasörün oluşturulma zamanı bilgisini alıp bir mesaj penceresi içerisinde görüntülüyoruz.



Eğer klasör mevcut değilse, **CreateDirectory** metodu ile klasörü oluşturuyoruz. Aynı şekilde **Exist** metodu ile klasörün mevcut olup olmadığı bilgisini alıp eğer klasör mevcut ise, **Delete** metodu ile klasörü siliyoruz.

Dosya İsimlerini Listeleme

Bir klasör içerisindeki dosyaları listelemek için **Directory.GetFiles** metodu kullanılabilir.

GetFiles metodunun kullanımı şu şekildedir:

```
Directory.GetFiles(Dosya yolu, dosya türü)
```

Örnek olarak bilgisayarın cookies klasöründe bulunan dosyaları temizleyen bir uygulama yapalım.

```
private void btnTemizle_Click(object sender, EventArgs e)
{
   string yol = Environment.GetEnvironmentVariable("HOMEDRIVE") +
    Environment.GetEnvironmentVariable("HOMEPATH");

  string[] cookies = Directory.GetFiles(yol +
    @"\AppData\Roaming\Microsoft\Windows\cookies\", "*.*");
  label1.Text = cookies.Length + " adet dosya silindi";

  foreach (string dosya in cookies)
  {
    try
    {
      File.Delete(dosya);
    }
```

```
        catch (Exception ex)
        { }
    }
}
```

Kodumuzda `GetEnvironmentVariable("HOMEDRIVE")` metodu ile işletim sisteminin kurulu olduğu sürücü ismini, `Environment.GetEnvironmentVariable ("HOMEPATH")` metodu ile ise işletim sisteminin kurulu olduğu klasör yolunu elde ederiz. Daha sonra belirtilen işletim sistemi klasörü altındaki **cookies** klasörü içerisinde bulunan dosya sayısını elde ederek, **Label** nesnesi içerisinde görüntüleriz. Son olarak `Delete` metodu ile `foreach` döngüsü içerisinde **cookies** klasörü içerisinde dosyaları tek tek sileriz.



Klasör Özelliklerini Elde Etme ve Değiştirme

Klasörlerin oluşturulma zamanı, son yazım tarihi, son erişim tarihi, alt klasör listesi, içerisindeki dosyaların listesi, ana dizin adı gibi bilgilere erişip bu bilgilerin bazılarını değiştirmek mümkündür.

- **`GetCreationTime`:** Belirtilen klasörün oluşturulma tarihini geriye döndürür.
- **`GetLastAccessTime`:** Belirtilen klasöre son erişim zamanını geriye döndürür.
- **`GetLastWriteTime`:** Belirtilen klasörün son yazım zamanını geriye döndürür.

Bu metodun kullanımı şu şekildedir:

```
Directory.GetLastWriteTime(Klasör yolu)
```

Diğer iki metodun da kullanımı bu şekildedir.

- **`SetCreationTime`:** Belirtilen klasörün oluşturulma tarihini ve zamanını değiştirir.
- **`SetLastAccessTime`:** Belirtilen klasöre son erişim tarihi ve zamanını değiştirir.
- **`SetLastWriteTime`:** Belirtilen klasöre son yazım tarihi ve zamanını değiştirir.

Bu metodun kullanımı şu şekildedir:

```
Directory.SetLastWriteTime(Klasör yolu, tarih ve zaman)
```

Diğer iki metodun da kullanımı bu şekildedir.

- **`GetDirectories`**: Belirtilen klasörün alt klasörlerinin listesini elde etmek için kullanılır. Elde edilen bu değer string dizi tipinden bir değerdir.
- **`GetFiles`**: Belirtilen klasörün içerisinde bulunan dosyaların listesini elde etmek için kullanılır. Elde edilen bu değer string dizi tipinden bir değerdir.

Kullanımı şu şekildedir:

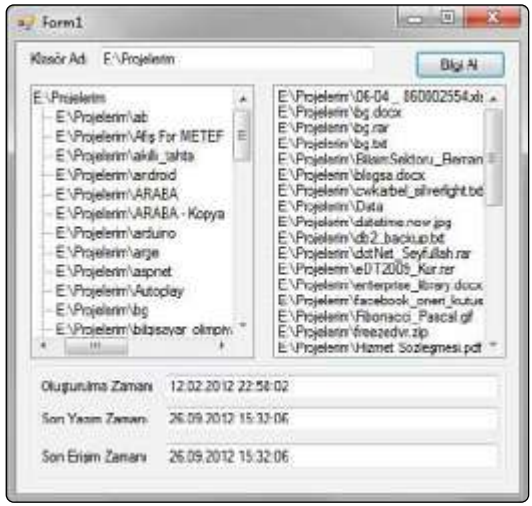
```
Directory.GetDirectories(Klasör Adı)
```

`GetDirectories` metodunun da kullanımı aynı şekildedir.

Örnek olarak bir klasör içerisindeki dosyaları listeleyerek, seçili olan dosyanın özelliklerini görüntüleyen bir uygulama yapalım. Örneğimiz için formumuza 1 adet **TreeView** nesnesi, 1 adet **ListBox**, bir adet **Button**, 4 adet de **TextBox** nesnesi ekliyoruz.

```
private void Form1_Load(object sender, EventArgs e)
{
  treeView1.Nodes.Add("Klasör Listesi");
}
private void btnBilgiAl_Click(object sender, EventArgs e)
{
  string[] klasorler = Directory.GetDirectories(txtKlasorAdı.Text);
  treeView1.Nodes[0].Text = txtKlasorAdı.Text;
  TreeNode anadugum = treeView1.Nodes[0];
  if (klasorler.Length != 0)
  {
       foreach (string aktifklasor in klasorler)
       {
        TreeNode dugum = new TreeNode(aktifklasor);
         anadugum.Nodes.Add(dugum);
       }
  }
  treeView1.ExpandAll();
```

```
  string[] dosyalar = Directory.GetFiles(txtKlasorAdı.Text);
  foreach (string aktifdosya in dosyalar)
       listBox1.Items.Add(aktifdosya);
  txtErisimZamanı.Text = Directory.
    GetLastAccessTime(txtKlasorAdı.Text).ToString();
  txtOlusturulmaZamanı.Text = Directory.
    GetCreationTime(txtKlasorAdı.Text).ToString();
  txtYazımZamanı.Text = Directory.
    GetLastWriteTime(txtKlasorAdı.Text).ToString();
}
```



Uygulamamız ile **TextBox** içerisine girdiğimiz yol bilgisi alınarak, bu yol içerisindeki klasörler **TreeView** nesnesi içerisinde, dosyalar ise **ListBox** içerisinde listelenmektedir. Ayrıca klasörün oluşturulma, son yazım ve son erişim zamanları bilgisi de alınarak **TextBox** nesneleri içerisinde gösterilmektedir.

Form'un `Load` olayı içerisinde…

```
treeView1.Nodes.Add("Klasör Listesi");
```

… satırı ile **TreeView** içerisine **Klasör Listesi** adında bir adet **parent node** ekliyoruz.

Klasörler üzerinde işlem yapmak için **System.IO** sınıfını projemize dahil ettik.

```
Directory.GetDirectories(txtKlasorAdı.Text)
```

… metodu ile **txtKlasorAdı** içerisinde yazılan yol bilgisini alıp, bu yol içerisinde bulunan klasörlerin listesini elde ediyoruz.

```
treeView1.Nodes[0].Text = txtKlasorAdı.Text;
```

… satırı ile eklemiş olduğumuz **parent node** ismini **TextBox** içerisinde yazan yol bilgisi ile değiştiriyoruz.

```
TreeNode anadugum = TreeView1.Nodes[0];
```

anadugum adında bir **TreeNode** nesnesini tanımlayarak, buna Form'un **Load** olayı esnasında oluşturmuş olduğumuz **parent node** nesnesini aktarıyoruz.

```
foreach (string aktifklasor in klasorler)
```

… satırı ile **klasorler** isimli değişkenimizin içerisinde bulunan, klasör listesi elemanları üzerinde işlem yapmak için bir döngü kuruyoruz.

```
TreeNode dugum = new TreeNode(aktifklasor);
```

… satırı ile **dugum** adında yeni bir **TreeNode** nesnesi oluşturarak **aktifklasor** değişkeni içerisinde saklanmakta olan klasör adı bilgisini aktarıyoruz.

```
anadugum.Nodes.Add(dugum);
```

Oluşturmuş olduğumuz yeni düğümü **anadugum** adlı parent node altına ekliyoruz.

```
treeView1.ExpandAll();
```

… satırı ile de ana düğüm ve tüm alt düğümlerin açık vaziyette listelenmesini sağlıyoruz.

```
Directory.GetFiles(txtKlasorAdı.Text)
```

… metodu ile **txtKlasorAdı** içerisinde yazılan yol bilgisini alıp, bu yol içerisinde bulunan dosyaların listesini elde ediyoruz.

```
foreach (string aktifdosya in dosyalar)
```

… satırı ile dosyalar isimli değişkenimizi içerisinde bulunan, dosya listesi elemanları üzerinde işlem yapmak için bir döngü kuruyoruz.

```
listBox1.Items.Add(aktifdosya);
```

Dosya isimlerini sıra ile **ListBox** içerisine ekliyoruz.

```
txtErisimZamanı.Text = Directory.
  GetLastAccessTime(txtKlasorAdı.Text).ToString();
txtOlusturulmaZamanı.Text = Directory.
  GetCreationTime(txtKlasorAdı.Text).ToString();
txtYazımZamanı.Text = Directory.
    GetLastWriteTime(txtKlasorAdı.Text).ToString();
```

… satırları ile de sıra ile klasöre son erişim zamanı, oluşturulma zamanı ve son yazım zamanı bilgilerini ilgili **TexBox** nesneleri içerisinde görüntülüyoruz.

Klasörleri Kopyalama ve Taşıma

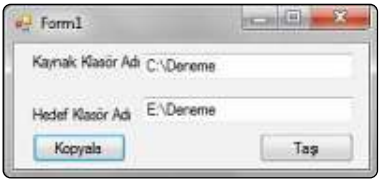
Klasörleri kopyalamak ve taşımak için **DirectoryInfo** ve **FileInfo** sınıflarından faydalanabiliriz. Dosyaları taşımak için **Directory** sınıfının **Move** metodu kullanılabilirken, **Directory** sınıfının **Copy** metodu olmadığı için **FileInfo** sınıfının **CopyTo** metodunu kullanmamız gerekmektedir.

FileInfo sınıfının **CopyTo** metodunun kullanımı şu şekildedir:

```
FileInfo.CopyTo (Kaynak dosya adı, Hedef dosya adı, Üzerine Yazma)
```

```
private void btnKopyala_Click(object sender, EventArgs e)
{
 DirectoryInfo kaynak_klasor = new DirectoryInfo(txtKaynak.Text);
 DirectoryInfo hedef_klasor = new DirectoryInfo(txtHedef.Text);

 if (!hedef_klasor.Exists)
    Directory.CreateDirectory(txtHedef.Text);
 foreach (FileInfo fi in kaynak_klasor.GetFiles())
    fi.CopyTo(Path.Combine(hedef_klasor.ToString(), fi.Name), true);
}
private void btnTası_Click(object sender, EventArgs e)
{
 Directory.Move(txtKaynak.Text, txtHedef.Text);
}
```



Örneğimizde öncelikle…

```
if (!hedef_klasor.Exists)
  Directory.CreateDirectory(txtHedef.Text);
```

… satırında, eğer hedef klasör mevcut değilse; **`Directroy`** sınıfının **`CreateDirectory`** metodu ile yeni bir klasör oluşturuyoruz.

```
foreach (FileInfo fi in kaynak_dosya.GetFiles())
  fi.CopyTo(Path.Combine(hedef_dosya.ToString(), fi.Name), true);
```

Daha sonra kaynak klasör içerisindeki dosyaları okuyarak oluşturmuş olduğumuz hedef klasör içerisine kopyalıyoruz.

Isolated Storage File Sınıfı ile Dosya ve Klasör İşlemlerini Gerçekleştirme

Yalnızca program içerisinde kod yardımı ile erişilebilecek, diğer kullanıcılara kapalı olacak küçük çaptaki verilerin saklanmasında **Isolated Storage File** (*İzole edilmiş Depolama Dosyası*) yapısı kullanılabilir.

Isolated Storage File Sınıfı Özellikleri

- **`AvailableFreeSpace`**: İzole edilmiş depolama alanı için mevcut boş alan miktarını verir.
- **`CurrentSize`**: İzole edilmiş depolama alanının geçerli boyutunu verir.
- **`IsEnabled`**: İzole edilmiş depolama alanının aktif olduğunu belirtir.
- **`MaximumSize`**: Quota ile belirlenen limitler dâhilinde kullanılacak izole edilmiş depolama alanının maksimum alan miktarı bilgisini döndürür.
- **`Quota`**: İnternet uygulamaları için kullanılacak izole edilmiş depolama alanının maksimum alan miktarı bilgisini döndürür.
- **`UsedSize`**: İzole edilmiş depolama alanı için kullanılan alan miktarı bilgisini döndürür.

Isolated Storage File Sınıfı Metotları

- **`Close`**: Daha önceden **`GetStore`**, **`GetUserStoreForAssembly`** veya **`GetUserStoreForDomain`** metotları ile açılmış olan depolama alanını kapatır.
- **`CopyFile`**: Geçerli dosyayı yeni bir kopya olarak kopyalar.
- **`CreateDirectory`**: İzole edilmiş depolama alanında bir klasör oluşturur.
- **`CreateFile`**: İzole edilmiş depolama alanında bir dosya oluşturur.
- **`DeleteDirectory`**: İzole edilmiş depolama alanındaki bir klasörü siler.
- **`DeleteFile`**: İzole edilmiş depolama alanındaki bir dosyayı siler.
- **`DirectoryExists`**: İzole edilmiş depolama alanında yolu verilen klasörün mevcut olup olmadığı bilgisini geriye döndürür.
- **`Dispose`**: İzole edilmiş depolama alanı tarafından kullanılan bütün kaynakları serbest bırakır.

- **`Equals`**: Belirtilen nesnenin geçerli nesneye eşit olup olmadığı bilgisini döndürür.
- **`FileExists`**: İzole edilmiş depolama alanında yolu verilen dosyanın mevcut olup olmadığı bilgisini geriye döndürür.
- **`Finalize`**: İşlemler için ayrılmış kaynakları serbest bırakır.
- **`GetCreationTime`**: Belirtilen dosya ya da klasörün oluşturulma tarihini geriye döndürür.
- **`GetDirectoryNames`**: İzole edilmiş depolama alanında kök dizin altındaki klasör listesini geriye döndürür.
- **`GetFileNames`**: İzole edilmiş depolama alanında kök dizin altındaki dosya listesini geriye döndürür.
- **`GetLastAccessTime`**: Belirtilen dosya ya da klasöre son erişim zamanını geriye döndürür.
- **`GetLastWriteTime`**: Belirtilen dosya ya da klasörün son yazım zamanını geriye döndürür.
- **`GetStore`**: Geçerli **application domain** (*etki alanı*) ve assembly için izole edilmiş depolama alanı alır.
- **`GetType`**: Geçerli olan nesnenin tipini bulur.
- **`GetUserStoreForApplication`**: Çağıran kodun **uygulama kimliği** (*application identity*) değerine uygun izole edilmiş depolama alanı alır.
- **`GetUserStoreForAssembly`**: Çağıran kodun **assembly identity** değerine uygun izole edilmiş depolama alanı alır. Masaüstü tabanlı **Windows** uygulamalar için izole edilmiş depolama alanı içerisinde alan ayırma işlemi için genellikle bu metot tercih edilir.
- **`GetUserStoreForDomain`**: Çağıran kodun **application domain identity** değerine uygun izole edilmiş depolama alanı alır.
- **`GetUserStoreForSite`**: Sanal host domaini için izole edilmiş depolama alanı alır. Web tabanlı (*Silverlight*) gibi uygulamalar için izole edilmiş depolama alanı içerisinde alan ayırma işlemi için genellikle bu metot tercih edilir.
- **`IncreaseQuotaTo`**: Bir kullanıcının izole edilmiş depolama alanı için daha büyük bir alan kullanmak istemesi durumu için kullanılır. Yeni alan daha izole edilmiş depolama alanının **`quota`** değerinden daha büyük olmalıdır.
- **`MoveDirectory`**: Belirtilen bir klasörü içeriği ile birlikte yeni bir konuma taşır.

- **MoveFile**: Belirtilen bir dosyayı yeni bir konuma taşır. İstenirse dosya ismi de değiştirilebilir.
- **OpenFile**: Belirtilen dosyayı seçilmiş olan paylaşım ve erişim seçenekleri ile açar.
- **Remove**: İzole edilmiş depolama alanını ve tüm içeriklerini siler.
- **ToString**: Geçerli olan nesneyi string türüne dönüştürür.

Klasör Oluşturma

IsolatedStorageFile işlemlerini gerçekleştirebilmek için öncelikle projelerimize **System.IO.IsolatedStorage** sınıfını dahil etmeliyiz.

İzole edilmiş depolama alanı içerisinde yeni bir dosya ya da klasör oluşturmak için öncelikle yeni bir **IsolatedStorageFile** nesnesi oluşturulur. Daha sonra **GetUserStoreForAssembly** metodu ile uygulamamız için izole edilmiş depolama alanı alıyoruz.

CreateDirectory metodu ile de yeni bir klasör nesnesi oluşturuyoruz.

Örneğin **IsolatedStorageFile** alanı içerisinde bir klasör oluşturalım.

```
using System;
using System.IO.IsolatedStorage;
using System.Windows.Forms;
namespace IsolatedStorageFile_KlasorOlusturma
{
 public partial class Form1 : Form
 {
  public Form1()
  {
    InitializeComponent();
  }
  IsolatedStorageFile isf = IsolatedStorageFile.
    GetUserStoreForAssembly();
  private void btnIslemYap_Click(object sender, EventArgs e)
  {
    if (txtKlasor.Text != "")
      isf.CreateDirectory(txtKlasor.Text);
  }
 }
}
```



Eğer oluşturacağımız klasör içerisinde bir alt klasör daha oluşturmak istiyorsak; / işareti ile ayırarak bu klasörü yazmalıyız. Örneğimizde *volkan_dosyalar/CSharp* şeklindeki kullanım ile *volkan_dosyalar* klasörü içerisinde *CSharp* adında bir alt klasör oluşturuyoruz.

Oluşturacağımız klasör ve dosyaları *C:\Users\kullanıcı_adı\AppData\Local\IsolatedStorage* klasörü içerisinde otomatik olarak oluşturulan *1qvpoxss.zz2\prpt4n5g.lub\Url.ygcemf2h2e5qeubawmpe1mf1502qifhm* şeklindeki klasör yapısı içerisinde AssemFiles klasörü altında oluştururuz.



Dosya Oluşturma

İzole edilmiş depolama alanı içerisinde yeni bir dosya oluşturabilmek için projemize…

```
System.IO.IsolatedStorage
System.IO
```

… sınıflarını dahil etmeliyiz.

`IsolatedStorageFileStream` nesnesi ile yeni bir dosya oluştururuz.

`IsolatedStorageFileStream` nesnesinin kullanımı şu şekildedir:

```
IsolatedStorageFileStream(Dosya adı ve yolu, FileMode, FileAccess,
Isolated Storage File değişkeni adı)
```

- **FileMode**: Dosya açma biçimini belirler. Aşağıdaki değerleri alır:

  Append: Dosyayı mevcutsa dosyayı açarak dosya sonuna konumlanır. Mevcut değilse, yeni dosya oluşturur. **FileMode.Append** yalnızca **FileAccess.Write** ile birlikte kullanılabilir.

  Create: Yeni bir dosya oluşturur. Eğer dosya mevcutsa üzerine yazar.

  CreateNew: Yeni bir dosya oluşturur.

  Open: Belirtilen dosyayı açar.

  OpenOrCreate: Belirtilen dosyayı açar. Eğer dosya mevcut değilse, yeni bir dosya oluşturur.

  Truncate: Yeni bir dosya açılacağını belirtir. Dosya boyutu 0 byte boyutundadır.

- **FileAccess**: Dosya erişim tipini belirler. Aşağıdaki değerleri alır:

  Read: Dosyayı okunmak üzere açar.

  Write: Dosyayı yazılmak üzere açar.

  ReadWrite: Dosyayı hem okunmak hem de yazılmak üzere açar.

Örneğin **IsolatedStorageFile** alanı içerisinde bir klasör oluşturarak içerisinde de yeni bir dosya oluşturalım.

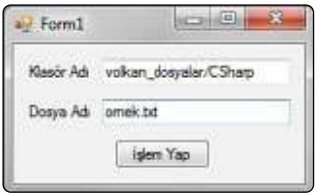
```
using System;
using System.Windows.Forms;
using System.IO;
using System.IO.IsolatedStorage;
namespace IsolatedStorageFile_DosyaOlusturma
{
 public partial class Form1 : Form
 {
  public Form1()
  {
    InitializeComponent();
  }
  IsolatedStorageFile isf = IsolatedStorageFile.GetUserStoreForAssembly();
```

```
  private void btnIslemYap_Click(object sender, EventArgs e)
  {
    if (txtKlasor.Text != "")
      isf.CreateDirectory(txtKlasor.Text);
    if (txtDosya.Text != "")
    {
     IsolatedStorageFileStream isf_fst = new
       IsolatedStorageFileStream(txtKlasor.Text + "\\" +
       txtDosya.Text, FileMode.Create, isf);
    }
  }
 }
}
```

Örneğimizde…

```
IsolatedStorageFileStream isf_fst = new
  IsolatedStorageFileStream(txtKlasor.Text + "\\" +
  txtDosya.Text, FileMode.Create, isf);
```

… komut satırları ile **`Create`** modunu kullanarak yolu ve dosya adı, **TextBox** nesnelerine girilen yeni bir **`IsolatedStorageFileStream`** nesnesi oluşturuyoruz.



Dosyaya Yazma

İzole edilmiş depolama alanı içerisinde oluşturulan dosyaya yazmak için **`StreamWriter`** nesnesini kullanabiliriz.

Örneğin **`IsolatedStorageFile`** alanı içerisinde bir klasör oluşturarak, içerisinde de yeni bir dosya oluşturalım ve içerisine yazılar ekleyelim.

```
IsolatedStorageFile isf = IsolatedStorageFile.
GetUserStoreForAssembly();
```

```
private void btnIslemYap_Click(object sender, EventArgs e)
{
  if (txtKlasor.Text != "")
    isf.CreateDirectory(txtKlasor.Text);

  if (txtDosya.Text != "")
  {
    IsolatedStorageFileStream isf_fst = new
      IsolatedStorageFileStream(txtKlasor.Text + "\\" +
      txtDosya.Text, FileMode.OpenOrCreate, isf);
    StreamWriter str_wrt = new StreamWriter(isf_fst);
    str_wrt.Write(txticerik.Text);
    str_wrt.Flush();
    str_wrt.Close();
  }
}
```

Örneğimizde…

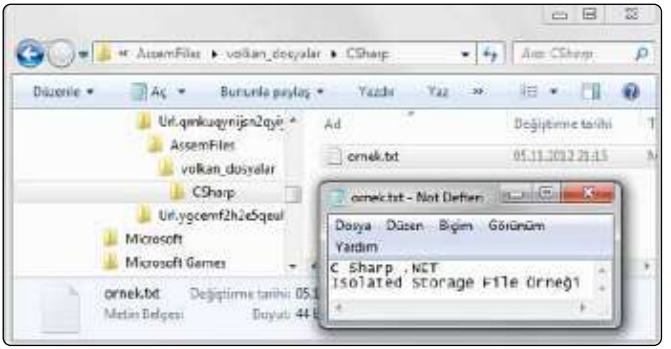
```
IsolatedStorageFileStream isf_fst = new
      IsolatedStorageFileStream(txtKlasor.Text + "\\" +
      txtDosya.Text, FileMode.OpenOrCreate, isf);
```

… komut satırları ile **`OpenOrCreate`** modunu kullanarak yolu ve dosya adı **TextBox** nesnelerine girilen **`IsolatedStorageFileStream`** nesnesini eğer mevcutsa açıyoruz. Mevcut değilse, yeni bir tane oluşturuyoruz. Oluşturulan nesneyi **`FileAccess.Write`** parametresi ile dosyaya yazma modunda açıyoruz.



Yeni bir **`StreamWriter`** nesnesi oluşturarak **`Write`** metodu ile **TextBox** nesnesi içerisine girilen yazıyı dosya içerisine ekliyoruz. Eğer satır satır eklemek istersek, **`WriteLine`** metodunu kullanabiliriz.

Flush metodu ile stream dosyanın güncellenmesi sağlanarak, **Close** metodu ile dosya kapatılıyor.



Dosyadan Okuma Yapma

İzole edilmiş depolama alanı içerisinde oluşturulmuş olan dosyaların içeriğini okumak için **StreamReader** nesnesini kullanabiliriz.

Dosya ve Klasör Silme

İzole edilmiş depolama alanı içerisinde oluşturulmuş olan klasörleri silmek için **IsolatedStorageFile** nesnesinin **DeleteDirectory** metodu kullanılırken, dosyaları silmek için ise **DeleteFile** metodu kullanılır.

Örnek olarak izole edilmiş depolama alanı içerisinde mevcut olan klasör ve dosyaları listeleyerek, listelenen dosya isminin içeriğini görüntüleyebilen ve listelenen klasör ve dosyaları silebilen bir uygulama yapalım.

```
IsolatedStorageFile isf = IsolatedStorageFile.GetStore
  (IsolatedStorageScope.User | IsolatedStorageScope.Assembly |
  IsolatedStorageScope.Domain, typeof(System.Security.Policy.Url),
  typeof(System.Security.Policy.Url));
string klasoradi, dosyaadi;
string[] klasorler, dosyalar;
private void btnListele_Click(object sender, EventArgs e)
{
  BilgiAl();
  if (klasorler.Length > 0)
    foreach (string isim in klasorler)
```

```
        lstKlasor.Items.Add(isim);
  if (dosyalar.Length > 0)
      foreach (string isim in dosyalar)
        lstDosya.Items.Add(isim);

  isf.Close();
}
private void BilgiAl()
{
  lstKlasor.Items.Clear();
  lstDosya.Items.Clear();
  klasorler = isf.GetDirectoryNames("*");
  dosyalar = isf.GetFileNames("*");
}
private void lstKlasor_SelectedIndexChanged(object sender, EventArgs
e)
{
  klasoradi = lstKlasor.SelectedItem.ToString();
  lstDosya.Items.Clear();

  IsolatedStorageFile isf = IsolatedStorageFile.GetStore
    (IsolatedStorageScope.User | IsolatedStorageScope.Assembly |
    IsolatedStorageScope.Domain, typeof(System.Security.Policy.Url),
    typeof(System.Security.Policy.Url));

  lstDosya.Items.AddRange(isf.GetFileNames(Path.Combine(klasoradi, "*")));
}
private void lstDosya_SelectedIndexChanged(object sender, EventArgs e)
{
  dosyaadi = lstDosya.SelectedItem.ToString();
  IsolatedStorageFileStream isf_fst = new IsolatedStorageFileStream
    (Path.Combine(klasoradi, dosyaadi), FileMode.Open, isf);
  StreamReader str_rdr = new StreamReader(isf_fst);
  textBox1.Text = str_rdr.ReadToEnd();
  str_rdr.Close();
  isf_fst.Close();
}
private void btnKlasorSil_Click(object sender, EventArgs e)
{
  IsolatedStorageFile isf = IsolatedStorageFile.GetStore
    (IsolatedStorageScope.User | IsolatedStorageScope.Assembly |
    IsolatedStorageScope.Domain, typeof(System.Security.Policy.Url),
```

```
    typeof(System.Security.Policy.Url));
  isf.DeleteDirectory(lstKlasor.SelectedItem.ToString());
  klasoradi = string.Empty;
  lstKlasor.Refresh();
  MessageBox.Show("Seçili olan klasör silindi");
}
private void btnDosyaSil_Click(object sender, EventArgs e)
{
  IsolatedStorageFile isf = IsolatedStorageFile.GetStore
    (IsolatedStorageScope.User | IsolatedStorageScope.Assembly |
    IsolatedStorageScope.Domain, typeof(System.Security.Policy.Url),
    typeof(System.Security.Policy.Url));

  isf.DeleteFile(Path.Combine(klasoradi, lstDosya.SelectedItem.ToString()));
  dosyaadi = string.Empty;
  lstDosya.Refresh();
  MessageBox.Show("Seçili olan dosya silindi");
}
private void btnDosyaOlustur_Click(object sender, EventArgs e)
{
  if (txtKlasor.Text != "")
   isf.CreateDirectory(txtKlasor.Text);
  if (txtDosya.Text != "")
  {
   IsolatedStorageFileStream isf_fst = new IsolatedStorageFileStream
     (txtKlasor.Text + "\\" + txtDosya.Text, FileMode.OpenOrCreate,
     FileAccess.Write, isf);

   StreamWriter str_wrt = new StreamWriter(isf_fst);
   str_wrt.Write(txticerik.Text);
   str_wrt.Flush();
   str_wrt.Close();
  }
}
```

Öncelikle uygulamamızın genel tanımlamalar bölümünde **`isf`** adlı bir **`IsolatedStorageFile`** nesnesi oluşturduk. Daha sonra **`IsolatedStorageFile`** nesnesinin **`GetStore`** metodu ile uygulamamız için izole edilmiş depolama alanı alıyoruz.

`GetStore` metodunun kullanımı şu şekildedir:

```
GetStore(IsolatedStorageScope, Type, Type) As IsolatedStorageFile
```

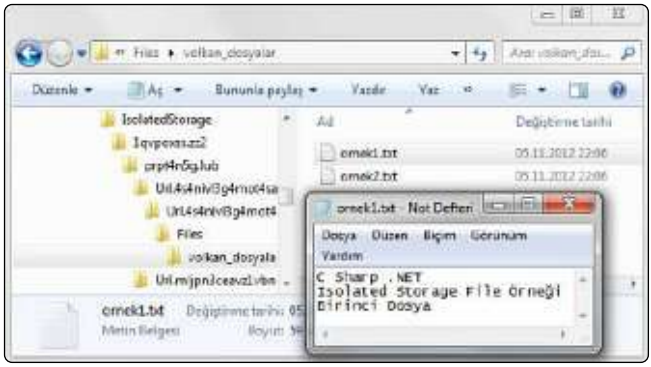
- **IsolatedStorageScope**: **IsolatedStorage** tarafından desteklenen izole depolama alanının düzeylerini belirtir. Şu değerleri alır: **Assembly**, **Domain**, **Roaming**, **User**, **None**.

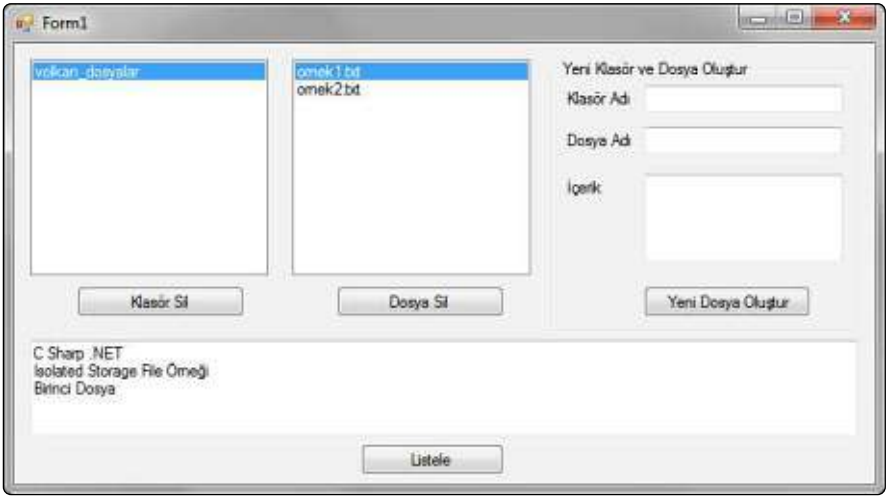
Dosya ve klasör isimlerini listelemek için **BilgiAl** adlı bir metot oluşturduk. Bu metot içerisinde **GetDirectoryNames** ve **GetFileNames** metotları ile klasör ve dosya isimlerini elde ettik. Bu metotlar kullanılırken, örneğimizde olduğu gibi **GetDirectoryNames("*")** ve **GetFileNames("*")** şeklinde parantez içerisinde arama kapsamını yazıyoruz.

Klasör ve dosya isimleri listelendikten sonra dosya isimlerinin listelendiği **ListBox** içerisindeki dosya isimlerinden biri seçildiğinde…

```
StreamReader str_rdr = new StreamReader(isf_fst);
textBox1.Text = str_rdr.ReadToEnd();
```

… komut satırları ile bir **StreamReader** nesnesi oluşturarak, **textBox1** nesnesi içerisine seçili olan dosya içerisindeki yazıları başından sonuna kadar aktarıyoruz.





`DeleteDirectory` ve `DeleteFile` metotları ile de listelenen klasör ve dosya isimlerinden seçili olanların izole edilmiş depolama alanı içerisinden silinmesini sağlıyoruz.

System.Text.Json Sınıfı Kullanılarak Dosya ve Klasör İşlemlerini Gerçekleştirme

JSON (**JavaScript Object Notation**), XML gibi düşük boyuttaki verileri saklamak, bu verilere erişimi ve üzerlerinde düzenlemeler yapmayı kolaylaştıran bir metin formatıdır. Veriler anahtar, değer ikilisi şeklinde saklanır. Değerler `Boolean`, `String`, `Number`, `Object` ve `Array` olmak üzere 5 türde saklanabilir.

Bir JSON nesnesin başlangıcı ve bitişi {} küme parantezleriyle olur. Anahtar / değer çiftleri, diğer çiftlerden **virgül** ile ayrılır. Anahtar ve değerler arasında **iki nokta üst üste** yazılır.

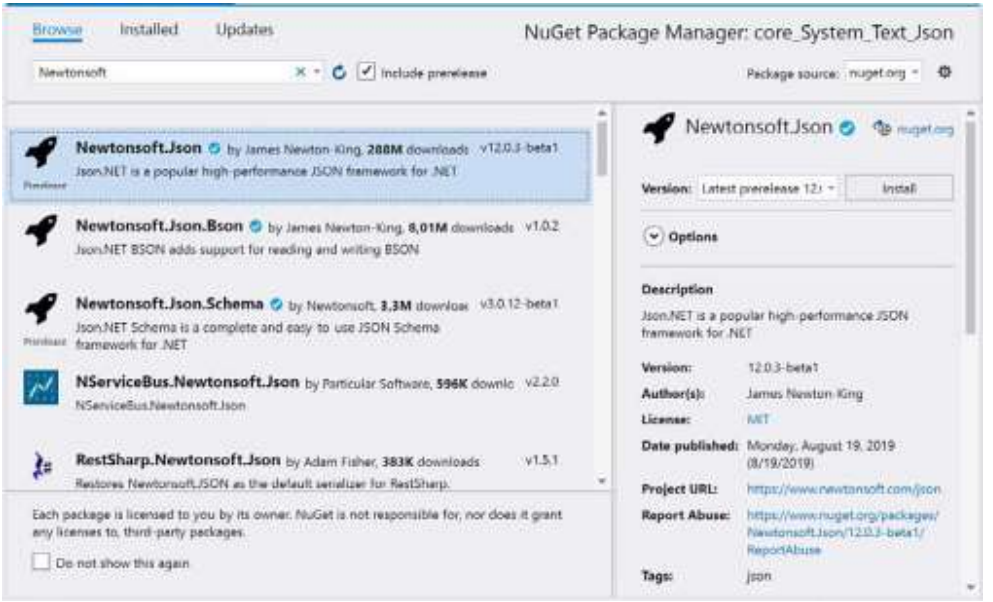
{"Meslek":"Öğretmen", "Branş":"Bilişim Teknolojileri"}

Meslek ve Branş anahtarları temsil ederken, Öğretmen ve Bilişim Teknolojileri ise değerleri temsil etmektedir.

.NET Core 3.0 sürümü öncesinde JSON işlemlerinde Newtonsoft Json.NET kütüphanesini yükleyerek projelerimizde kullanabiliyorduk. .NET Core 3.0 sürümünden itibaren haricî bir kütüphane yüklemeye gerek kalmadan *JSON oluşturma, okuma, yazma vb. işlemleri* **daha performanslı** bir biçimde gerçekleştirebiliyoruz.

Her iki kullanım şeklini de görmek için bir json tanımlaması yaparak içerisindeki

Project / Manage NuGet Packages... menüsü altındaki **NuGet Package Maneger** penceresini açarak **Browse** bölümüne **Newtonsoft.Json** yazarak, **Newtonsoft Json.NET** kütüphanesini yüklüyoruz.

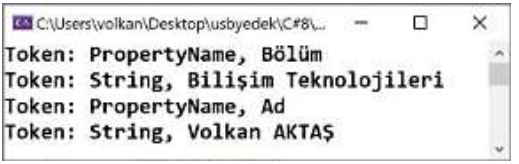


Kodumuz aşağıdaki gibi olacak:

```
using Newtonsoft.Json;
using System;
using System.IO;
namespace newtonsoft_json
{
  class Program
  {
```

```
    static void Main(string[] args)
    {
      var json = "{ \"Bölüm\": \"Bilişim Teknolojileri\",
          \"Ad\": \"Volkan AKTAŞ\" }";
      JsonTextReader reader =
          new JsonTextReader(new StringReader(json));
      while (reader.Read())
      {
        if (reader.Value != null)
        {
          Console.WriteLine("Token: {0}, {1}",
              reader.TokenType, reader.Value);
        }
      }
      Console.ReadLine();
    }
  }
}
```

Kodumuzda **`json`** nesnesi içerisinde **`string`** değerleri " " (**çift tırnak**) işaretleri arasında tanımladık. Değerlerin başlangıç ve bitişlerini "\" (**ters bölü**) işaretleri ile belirliyoruz. **`reader`** isimli **`JsonTextReader`** sınıfının **`Read`** metodu ile anahtar ve değerleri okuyabiliriz. **`TokenType`** özelliği okunan özelliğin türünü belirtirken, **`Value`** ise **anahtar** ve **değerleri okumak amacıyla** kullanılmaktadır.

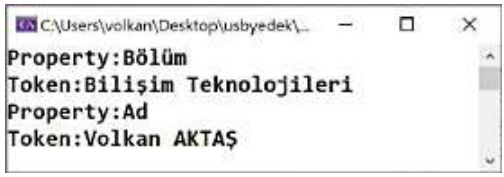


`System.Text.Json` sınıfını kullanarak aynı kodu aşağıdaki gibi yazabiliriz:

```
using System;
using System.Text;
using System.Text.Json;
namespace System_Text_Json
{
  class Program
  {
```

```
    static void Main(string[] args)
    {
      var json = "{ \"Bölüm\": \"Bilişim Teknolojileri\",
          \"Ad\": \"Volkan AKTAŞ\" }";
      byte[] data = Encoding.UTF8.GetBytes(json);
      Utf8JsonReader reader = new Utf8JsonReader(data,
          isFinalBlock: true, state: default);
      while (reader.Read())
      {
        JsonTokenType tokenType = reader.TokenType;
        switch (tokenType)
        {
          case JsonTokenType.PropertyName:
            Console.WriteLine("Property:" + reader.GetString());
            break;
          case JsonTokenType.String:
            Console.WriteLine("Token:"+ reader.GetString());
            break;
        }
      }
      Console.ReadLine();
    }
  }
}
```

Kodumuzda **reader** isimli **Utf8JsonReader** sınıfının **Read** metodu ile anahtar ve değerleri okuyabiliriz. **TokenType** özelliği okunan özelliğin türünü belirtirken, **GetString** ise anahtar ve değerleri okumak amacıyla kullanılmaktadır.



Json nesnelerini yazmak için **System.Text.Json** kütüphanesine ait **Utf8JsonWriter** sınıfını kullanabiliriz.

```
using System;
using System.IO;
```

```
using System.Text;
using System.Text.Json;
namespace core_System_Text_Json_Writer
{
  class Program
  {
    static void Main(string[] args)
    {
      using (var stream = new MemoryStream())
      {
        using (var writer = new Utf8JsonWriter(stream,
            new JsonWriterOptions { Indented = true }))
        {
          writer.WriteStartObject();
          writer.WriteString("Bölüm", "Bilişim Teknolojileri");
          writer.WriteString("Ad", "Volkan AKTAŞ");
          writer.WriteNumber("Yaş", 43);
          writer.WriteEndObject();
          writer.Flush();
        }
        string json = Encoding.UTF8.GetString(stream.ToArray());
        Console.WriteLine(json);
      }
      Console.ReadLine();
    }
  }
}
```

Kodumuzda öncelikle

```
using (var stream = new MemoryStream())
```

satırında **`stream`** adlı bir **`MemoryStream`** nesnesi tanımlıyoruz.

```
using (var writer = new Utf8JsonWriter(stream,
    new JsonWriterOptions { Indented = true }))
```

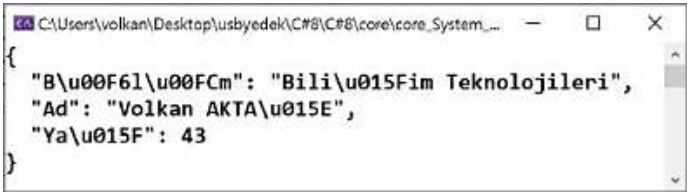
satırında ise **`writer`** isimli bir **`Utf8JsonWriter`** nesnesi tanımlıyoruz. **`Utf8JsonWriter`** nesnesi **2 parametre alır**. İlk parametre **`stream`** ismini belirtirken, 2. Parametre olan **`JsonWriterOptions`** parametresi ile **`json`**

nesnesi oluşturma esnasında kullanılacak olan seçenekler belirtilir. Burada kullandığımız "`Indented=true`" parametresi ile `Json` çıktısının girintiler, anahtarlar ve değerler arasına **boşluklar** konularak biçimlendirilmesini sağlar.

`WriteString()` metodu ile stringleri, `WriteNumber()` metodu ile ise sayıları `json` nesnesi içerisine yazarız. Ayrıca her `json` nesnesinin **başlangıç** ve **bitişini** belirlemek için `WriteStartObject()` ve `WriteEndObject()` metotlarını kullanırız.

```
string json = Encoding.UTF8.GetString(stream.ToArray());
Console.WriteLine(json);
```

Satırlarında ise `stream` nesnesini diziye dönüştürüp içerisindeki değerleri okuyarak **konsol ekranına yazdırıyoruz**.



Uygulamamızın ekran çıktısına baktığımızda Türkçe karakterlerin yazdırılmadığını görürüz. Bunun için `Utf8JsonWriter` nesnesinin `JsonWriterOptions` parametresinde `System.Text.Encodings.Web` sınıfının `JavaScriptEncoder.Create()` metodunu kullanabiliriz.

Öncelikle

```
using System.Text.Encodings.Web;
using System.Text.Unicode;
```

sınıflarını projemize ekliyoruz. Artık daha fazla `JsonWriterOptions` parametresi ekleyeceğimiz için bunları aşağıda görüldüğü gibi yazıyoruz.

```
var options = new JsonWriterOptions
{
    Indented = true,
    Encoder = JavaScriptEncoder.Create(UnicodeRanges.All)
};
```

Kodumuzda daha önce tanımlamış olduğumuz **`Indented`** parametresine ek olarak, **`Encoder`** parametresini de ekleyerek belirttiğimiz **Encoder** seçeneğinin ayarlanmasını sağlıyoruz. Bu parametre **`null`** olduğunda varsayılan **Encoder ayarları** uygulanır.

Kodumuzda

```
using (var writer = new Utf8JsonWriter(stream,
    new JsonWriterOptions { Indented = true }))
```

şeklinde yazmış olduğumuz **`Utf8JsonWriter`** nesnesi oluşturma cümlemizi aşağıda görüldüğü gibi

```
using (var writer = new Utf8JsonWriter(stream, options))
```

tanımlamış olduğumuz **`options`** isimli **`JsonWriterOptions`** parametrelerini çağıracak şekilde değiştiriyoruz. Uygulamamızı tekrar çalıştırdığımızda artık **Türkçe karakterlerin doğru biçimde yazdırıldığını görebiliriz**.



`Json` nesnelerini serialize etmek için **`System.Text.Json`** kütüphanesine ait **`JsonSerializer`** sınıfının **`Serialize`** metodunu kullanabiliriz. Serialization (*Serileştirme*) herhangi bir **`object`** türündeki nesneyi depolamak ya da kullanmak amacıyla istenilen **`json`** formatına dönüştürmedir. Örnek olarak bir sınıf tanımlayarak bu sınıftan bir nesne oluşturup değer girelim ve değerlere **Serialization işlemi** ile erişelim.

```
class Ogretmen
{
  public string Bolum { get; set; }
  public string Ad { get; set; }
  public int Yas { get; set; }
}
```

```
class Program
{
  static void Main(string[] args)
  {
    Ogretmen ogretmen = new Ogretmen
    {
      Bolum = "Bilişim Teknolojileri",
      Ad = "Volkan AKTAŞ",
      Yas = 43
    };
    Console.WriteLine(JsonSerializer.Serialize<Ogretmen.
Ad>(ogretmen));
    Console.ReadLine();
  }
}
```

Kodumuzda **öğretmen** isimli bir sınıf tanımlayarak **Main** metodumuzun içerisinde bir **öğretmen** nesnesi oluşturuyoruz ve değerlerimizi aktarıyoruz. **JsonSerializer** nesnesinin Serialize metoduyla **öğretmen** nesnesinin **Ad** özelliğine erişerek değerini ekrana yazdırıyoruz.



Türkçe karakter problemi yaşamamak için daha önceki örnek uygulamamızda yaptığımız gibi **Encoding ayarlarını** yapmamız gerekiyor.

```
using System.Text.Encodings.Web;
using System.Text.Unicode;
```

sınıflarını projemize ekleyerek **JsonSerializerOptions** parametrelerini aşağıda görüldüğü gibi yazıyoruz.

```
var options = new JsonSerializerOptions
{
    WriteIndented = true,
    Encoder = JavaScriptEncoder.Create(UnicodeRanges.All)
};
```

Kodumuzda daha önce **Utf8JsonWriter** sınıfında kullanmış olduğumuz

Indented parametresini **JsonSerializer** sınıfında **WriteIndented** şeklinde kullanıyoruz.

Kodumuzda sadece bir değeri değil de tüm anahtar ve değerlere erişerek yazdırmak istiyorsak,

```
Console.WriteLine(JsonSerializer.Serialize<Ogretmen.Ad>
    (ogretmen, options));
```

Satırını

```
Console.WriteLine(JsonSerializer.Serialize<Ogretmen>
(ogretmen, options));
```

Şeklinde düzenlemeliyiz.



Json nesnelerini deserialize etmek için **System.Text.Json** kütüphanesine ait **JsonSerializer** sınıfının **Deserialize** metodunu kullanabiliriz. **Deserialization json** formatındaki bir veriyi bir nesneyi depolamak ya da kullanmak amacıyla istenilen bir **object** tipine dönüştürmedir. Örnek olarak bir **json** nesnesini tanımladığımız bir sınıftaki modele dönüştürelim.

```
class Ogretmen
{
  public string Bolum { get; set; }
  public string Ad { get; set; }
  public int Yas { get; set; }
}
class Program
{
  static void Main(string[] args)
  {
    var json = "{ \"Bölüm\": \"Bilişim Teknolojileri\",
```

```
        \"Ad\": \"Volkan AKTAŞ\",\"Yaş\": \"43\"  }";
    var ogretmen = JsonSerializer.Deserialize<Ogretmen>(json);
    Console.WriteLine(ogretmen.Ad);
    Console.ReadLine();
  }
}
```

Kodumuzda tanımlamış olduğumuz **json** nesnesini **Ogretmen** adlı **Class** içerisindeki modele uygun bir biçime dönüştürerek **ogretmen.Ad** alanının değeri olan "**Volkan AKTAŞ**"ı konsol ekranına yazdırdık.

Json nesnesi içerisindeki alanlara

ogretmen.Ad= "Bünyamin KARAHAN"

Şeklinde değer akarımı da yapabiliriz.

VERİ TABANI İŞLEMLERİ

27

C# veri tabanları üzerinde işlemler gerçekleştirmek üzere DAO, ADO, RDO, ODBC gibi teknolojilere sahiptir. Biz, bu bölümde bu teknolojilerden en çok kullanılan ADO.NET'e değineceğiz. ADO.NET ile gerek Access, SQL, Oracle, IBM DB2 gibi çeşitli veritabanları üzerinde kayıt oluşturma, ekleme, güncelleme, sorgulama, silme gibi işlemler kolaylıkla gerçekleştirilebilir.

Veritabanları üzerinde işlem yapmak için kullanılabilecek veri tabanı nesneleri şunlardır:

- **Connection nesnesi:** Veritabanına bağlantı kurmak için kullanılır.
- **Command:** Veritabanı işlemleri ile ilgili kullanılan sorguları çalıştırmak için kullanılır.
- **DataReader:** Veritabanı bağlantısı olduğu müddetçe, veritabanı içerisindeki verileri okuma amacı ile kullanılır.
- **DataSet:** Veritabanı bağlantısı olduğu müddetçe veri tabanı içerisindeki verileri okuma ve yazma amacı ile kullanılır.
- **DataAdapter:** Veritabanı bağlantısı olmasa bile veriler üzerinde işlemler gerçekleştirilebilir.

SORGULAR

SQL sorguları yardımıyla veritabanı içerisindeki tablolar ve kayıtları yöneterek, tablo ve kayıtlar üzerinde düzenlemeler yapabiliriz. SQL diline ait sorguları MSSQL, Access, Oracle, Firebird gibi veritabanları üzerinde kullanabiliriz.

Aşağıdaki gibi örnek bir Access tablosu üzerinde temel SQL sorgularını uygulayarak sonuçlarını görelim.

Bilgiler

id	ad	soyad	sınıf	numara	yaş
1	Cihan	Dindar	A10B	8	16
2	İbrahim	Demir	A10B	15	15
3	Burak	Yalçın	A10B	147	16
4	Ufuk	Aygün	A10B	149	16
5	Baran	Yılmaz	T10A	1	16
6	Büşra	Toksoy	T10A	2	15
7	Murat	Aktaş	T10A	26	15
8	Muhammed	Doğan	T10A	27	16

- **`FROM`**: Üzerinde işlem yapılacak olan tabloyu belirtir.
- **`SELECT`**: Veritabanındaki kayıtları seçmek amacıyla kullanılır.

Örneğin;

```
SELECT * FROM Bilgiler
```

Sql sorgusu ile **`Bilgiler`** tablosundaki bütün kayıtları seçeriz. **`*`** karakteri bütün kayıtlar manasına gelmektedir.

```
SELECT ad,soyad FROM Bilgiler
```

Sql sorgusu ise, **`Bilgiler`** tablosundaki **`ad`** ve **`soyad`** alanlarındaki tüm kayıtları seçer.

Sorgu1

ad	soyad
Cihan	Dindar
İbrahim	Demir
Burak	Yalçın
Ufuk	Aygün
Baran	Yılmaz
Büşra	Toksoy
Murat	Aktaş
Muhammed	Doğan

- **`DISTINCT`**: Belirtilen bir şarta uyan kayıtlardan birden fazla varsa, bunlardan yalnızca birinin seçilmesini sağlar.

Örneğin;

```
SELECT DISTINCT sınıf FROM Bilgiler
```

Sorgusu, sınıf isimlerinden yalnız birer adet listelenmesini sağlar. Dolayısı ile sorgu sonucu yandaki gibi olacaktır.

Sorgu1

sınıf
A10B
T10A

- **WHERE**: Belirtilen bir şarta göre verileri seçmek amacıyla kullanılır.

Örneğin;

```
SELECT * FROM Bilgiler WHERE sınıf= 'T10A'
```

Sorgusu, **Bilgiler** tablosundaki sınıfı **T10A** olan kayıtları listelemektedir.

Sorgu1

id	ad	soyad	sınıf	numara	yaş
5	Baran	Yılmaz	T10A	1	16
6	Büşra	Toksoy	T10A	2	15
7	Murat	Aktaş	T10A	26	15
8	Muhammed	Doğan	T10A	27	16

```
SELECT * FROM Bilgiler WHERE yaş>15
```

Sorgusu ise; **Bilgiler** tablosundaki yaşı **15**'den büyük olan kayıtları listelemektedir.

Sorgu1

id	ad	soyad	sınıf	numara	yaş
1	Cihan	Dindar	A10B	8	16
3	Burak	Yalçın	A10B	147	16
4	Ufuk	Aygün	A10B	149	16
5	Baran	Yılmaz	T10A	1	16
8	Muhammed	Doğan	T10A	27	16

- **LIKE**: Bir ya da daha fazla tablo içerisindeki kayıtlar üzerinde karşılaştırma işlemleri yapmak amacıyla kullanılır. Karşılaştırma işlemleri yapılırken joker karakterlerden faydalanılabilir. Kullanılabilecek başlıca karakterler şunlardır:
- ***** : Birden fazla değeri ifade etmektedir.
- **?** : Bir tek karakteri ifade etmektedir.
- **[ ]** : Bir değer aralığını ifade etmek amacıyla kullanılır. **[A-Z]** A ile Z arasındaki harfler gibi.

Örneğin;

```
SELECT * FROM Bilgiler WHERE ad LIKE 'M*'
```

Sorgusu, `Bilgiler` tablosundaki adı **`M`** ile başlayan kayıtları (Örnek tablomuza göre **`Murat`** ve **`Muhammed`** kayıtları) listelemektedir.

Sorgu1

id	ad	soyad	sınıf	numara	yaş
7	Murat	Aktaş	T10A	26	15
8	Muhammed	Doğan	T10A	27	16

```
SELECT * FROM Bilgiler WHERE ad LIKE 'Mu???'
```

Sorgusu, **`Bilgiler`** tablosundaki ilk iki harfi **`Mu`** olan **`5`** harfli kayıtları listelemektedir. Örnek tablomuza göre şekildeki gibi sadece ad alanında **`Murat`** yazan değer seçilmiştir.

Sorgu1

id	ad	soyad	sınıf	numara	yaş
7	Murat	Aktaş	T10A	26	15

```
SELECT * FROM Bilgiler WHERE ad LIKE '[B-İ]*'
```

Sorgusu, **`Bilgiler`** tablosundaki adları **`B`** ile **`İ`** harfleri arasındaki harfler ile başlayan kayıtları (Örnek tablomuza göre **`Burak`**, **`Baran`**, **`Büşra`**, **`Cihan`** ve **`İbrahim`** kayıtlarını) listelemektedir.

Sorgu1

id	ad	soyad	sınıf	numara	yaş
1	Cihan	Dindar	A10B	8	16
2	İbrahim	Demir	A10B	15	15
3	Burak	Yalçın	A10B	147	16
5	Baran	Yılmaz	T10A	1	16
6	Büşra	Toksoy	T10A	2	15

- **`COUNT`**: Belirtilen bir ya da daha fazla tablodaki kayıt sayısını bulmak amacıyla kullanılır.

Örneğin;

```
SELECT Count(*) FROM Bilgiler
```

Sorgusu, **Bilgiler** tablosundaki toplam kayıt sayısı olan **8** değerini geriye döndürecektir.

BETWEEN: Bir değer aralığını belirtmek amacıyla kullanılır.

Örneğin;

```
SELECT * FROM Bilgiler WHERE numara BETWEEN 1 AND 8
```

Sorgusu, **Bilgiler** tablosundaki numarası 1 ile 8 arasında olan kayıtları listelemektedir. Sql sorgusunun örnek tablomuz üzerinde çalıştırılması sonucunda, aşağıdakine benzer bir sonuç elde edilecektir.

Sorgu1

id	ad	soyad	sınıf	numara	yaş
1	Cihan	Dindar	A10B	8	16
5	Baran	Yılmaz	T10A	1	16
6	Büşra	Toksoy	T10A	2	15

- **ORDER BY:** Kayıtları belirtilen bir alana göre sıralamak amacıyla kullanılır.

Örneğin;

```
SELECT * FROM Bilgiler ORDER BY ad
```

Sorgusu, **Bilgiler** tablosundaki kayıtları ad alanına göre küçükten büyüğe doğru sıralayarak listeler.

Sorgu1

id	ad	soyad	sınıf	numara	yaş
5	Baran	Yılmaz	T10A	1	16
3	Burak	Yalçın	A10B	147	16
6	Büşra	Toksoy	T10A	2	15
1	Cihan	Dindar	A10B	8	16
2	İbrahim	Demir	A10B	15	15
8	Muhammed	Doğan	T10A	27	16
7	Murat	Aktaş	T10A	26	15
4	Ufuk	Aygün	A10B	149	16

- **`INSERT INTO`**: Bir tablo içerisinde belirtilen alan ya da alanlara kayıt eklemek amacıyla kullanılır.

  Örneğin;

```
INSERT INTO Bilgiler ( ad, soyad, sınıf, numara, yaş )
VALUES ('Ahmet', 'Demir', '10TA', 16, 17)
```

  Sorgusu, **`Bilgiler`** tablosundaki **`ad`**, **`soyad`**, **`sınıf`**, **`numara`** ve **`yaş`** alanlarına **`VALUES`** ifadesinden sonra belirtilen kayıtları ekler. Sorgu satırında metin içeren bilgiler tek tırnak içerinde belirtilmelidir. SQL sorgusu çalıştırıldıktan sonra tablomuz aşağıdaki gibi olacaktır.

Bilgiler

id	ad	soyad	sınıf	numara	yaş
1	Cihan	Dindar	A10B	8	16
2	İbrahim	Demir	A10B	15	15
3	Burak	Yalçın	A10B	147	16
4	Ufuk	Aygün	A10B	149	16
5	Baran	Yılmaz	T10A	1	16
6	Büşra	Toksoy	T10A	2	15
7	Murat	Aktaş	T10A	26	15
8	Muhammed	Doğan	T10A	27	16
9	Ahmet	Demir	10TA	16	17

- **`UPDATE`**: Bir tablo içerisinde belirtilen alan ya da alanlardaki kayıtlar üzerinde değişiklik yaparak güncellemek amacıyla kullanılır.

  Örneğin;

```
UPDATE Bilgiler SET yaş = 16 WHERE ad='Ahmet'
```

  Sorgusu, **`Bilgiler`** tablosundaki adı **`Ahmet`** olan kaydın yaşını **`16`** olarak değiştirerek kaydı günceller.

Bilgiler

id	ad	soyad	sınıf	numara	yaş
1	Cihan	Dindar	A10B	8	16
2	İbrahim	Demir	A10B	15	15
3	Burak	Yalçın	A10B	147	16
4	Ufuk	Aygün	A10B	149	16
5	Baran	Yılmaz	T10A	1	16
6	Büşra	Toksoy	T10A	2	15
7	Murat	Aktaş	T10A	26	15
8	Muhammed	Doğan	T10A	27	16
9	Ahmet	Demir	10TA	16	16

- **`DELETE`:** Bir tablo içerisinde bir ya da daha fazla kaydı silmek amacıyla kullanılır.

Örneğin;

```
DELETE FROM Bilgiler WHERE ad='Ahmet'
```

Sorgusu, **`Bilgiler`** tablosundaki adı **`Ahmet`** olan kaydı siler.

Bilgiler

id	ad	soyad	sınıf	numara	yaş
1	Cihan	Dindar	A10B	8	16
2	İbrahim	Demir	A10B	15	15
3	Burak	Yalçın	A10B	147	16
4	Ufuk	Aygün	A10B	149	16
5	Baran	Yılmaz	T10A	1	16
6	Büşra	Toksoy	T10A	2	15
7	Murat	Aktaş	T10A	26	15
8	Muhammed	Doğan	T10A	27	16
#####	#Silindi	#Silindi	#Silindi	#Silindi	######

Access Veri Tabanı İşlemleri

Access veritabanlarına erişim için kullanılabilecek veritabanı nesneleri şunlardır;

- **`OleDBConnection` nesnesi**
- **`OleDBCommand` nesnesi**
- **`OleDbDataAdapter`**
- **DataReader**
- **DataSet**

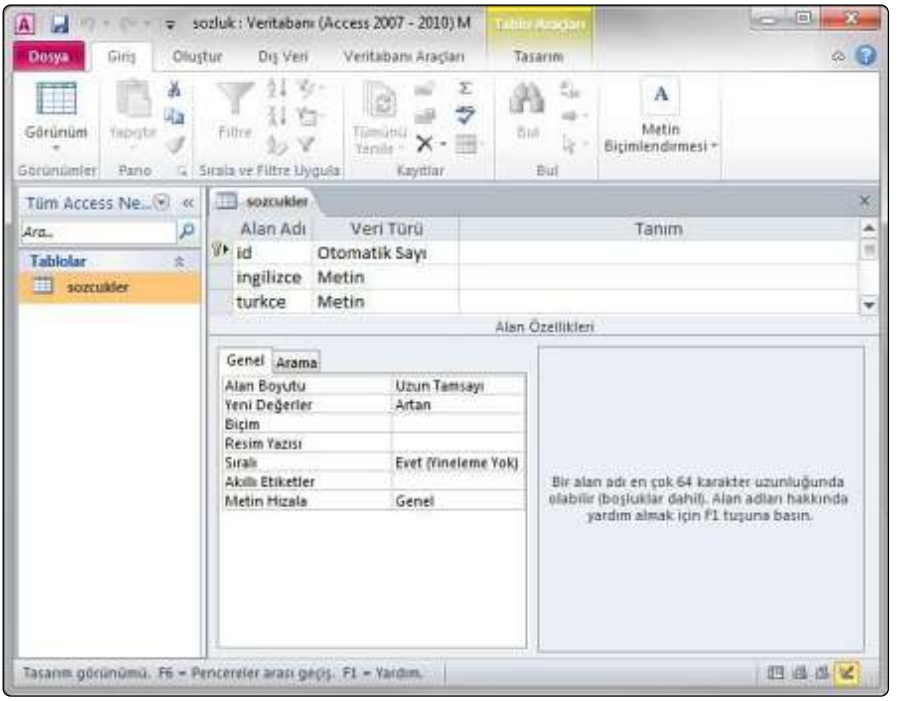
Şimdi bu veritabanı nesnelerini kısaca açıklayalım…

- **`OleDBConnection` nesnesi:** Bağlantı açmak ve kapatmak için kullanılan nesnedir. Bu nesne ile veritabanının türü ve veritabanının fiziksel olarak bulunduğu yol belirtilir.
- **`OleDBCommand` nesnesi:** Veritabanı üzerinde SQL deyimleri çalıştırmak için kullanılan nesnedir. **`Select`**, **`Insert`**, **`Update`** ve **`Delete`** gibi SQL deyimleri bu nesne üzerinde çalıştırılabilir.
- **`OleDbDataAdapter`**: **`Connected`** ve **`Disconnected`** (*bağlantılı ve bağlantısız*) özelliği ile veriler arasında köprü görevi görür. Veritabanından çekilen veriyi **DataSet**'e aktarmak ya da **DataSet**'te güncellenmiş veriyi veritabanına aktarmada kullanılır.
- **DataReader:** Veritabanından kayıt çekmek için kullanılır. Kayıtlar **`OleDBCommand`** nesnesinin **`ExecuteReader`** metodu ile **DataReader**'a aktarılır. Bu nesne ile çekilen veriler sırası ile okunabilir.
- **DataSet:** Veritabanından bağlantısız olarak çalışan bir nesnedir. **`OleDbDataAdapter`** ile veritabanından çekilen veriler **DataSet** içerisinde tablolar şeklinde saklanır. **DataSet**'teki veriler üzerinde güncellemeler yapıldıktan sonra **`OleDbDataAdapter`** ile tekrar veritabanına aktarılır. **DataSet**; **DataTable**, **DataColumn**, **DataRow** ve **DataRelation** nesnelerinden oluşur.

Öncelikle kod yazmadan bir veri tabanına erişimi nasıl sağlarız, onunla ilgili bir uygulama yapalım. Örnek bir Access veri tabanı oluşturalım.

- Veri tabanı adı: **`sozluk`**
- Tablo adı: **`sozcukler`**
- Alan adları: **`id`**, **`ingilizce`**, **`turkce`**

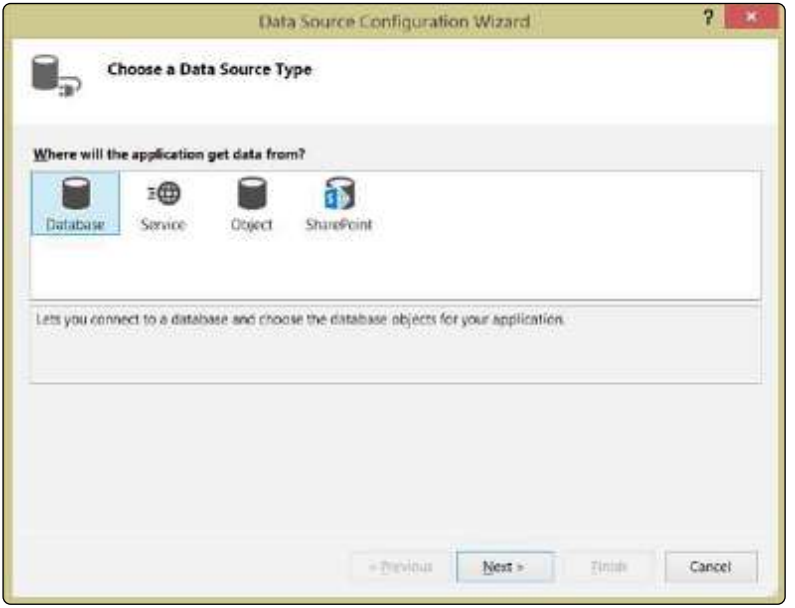
… şeklinde olsun.



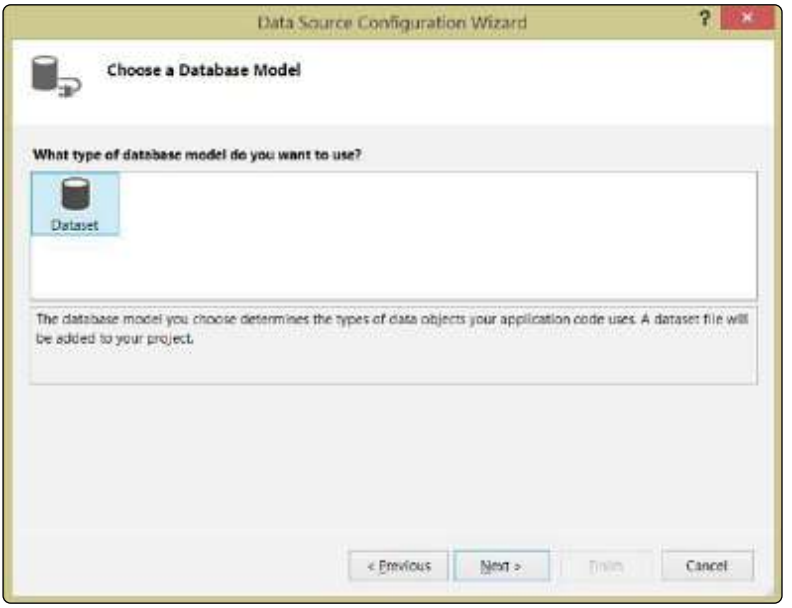


Oluşturmuş olduğumuz veritabanındaki kayıtları çekebilmek için **DataSources** penceresini kullanarak yeni bir **DataSource** oluşturalım. Eğer **DataSources** penceresi görüntülenmiyorsa, **View / Other Windows / Data Sources** seçenekleri ile görüntüleyebilirsiniz. Bu penceredeki **Add New DataSource** simgesini tıklıyoruz.

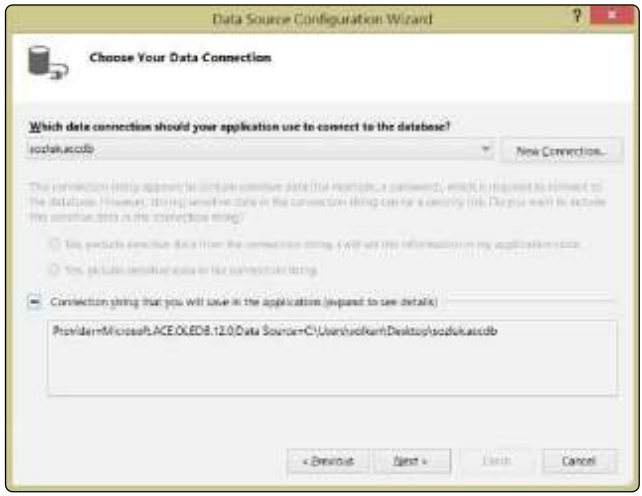
Gelen pencerede **Database** seçeneğini seçip **Next** butonuna basalım.



DataSet simgesini seçerek **Next** butonunu tıklayalım.



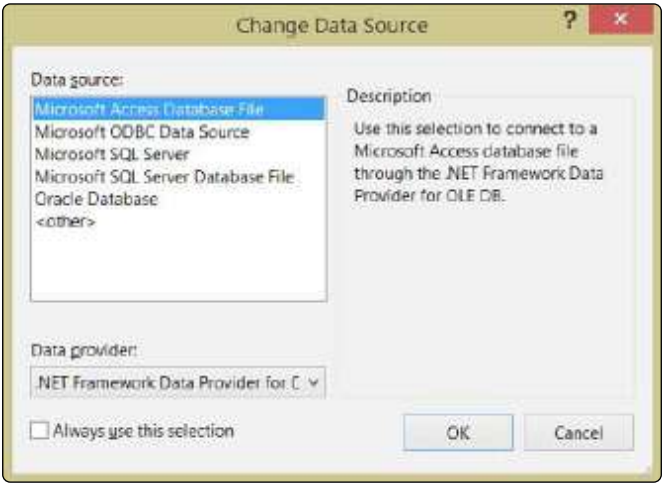
Gelen pencerede daha önce oluşturmuş olduğumuz Access veri tabanımızın ismi hazır olarak eklenmiş olacaktır. Bu pencerede Connection String'in (veri tabanı bağlantı cümlesi) otomatik olarak oluşturulduğunu görebilirsiniz.



Yeni bir veri tabanı bağlantısı oluşturmak için bu penceredeki **"New Connection"** butonuna tıklıyoruz. Bu pencerede varsayılan olarak **"Microsoft Access Database File (OLE DB)"** seçili durumdadır.

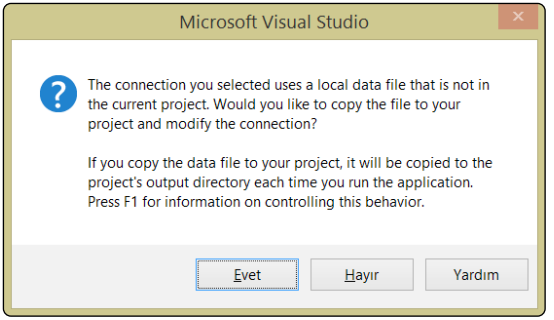


Eğer değiştirmek istersek **Change** butonuna tıklayarak karşımıza gelen **Choose Data Source** penceresinden istediğimiz veri tabanı türünü seçerek **OK** butonuna tıklarız.



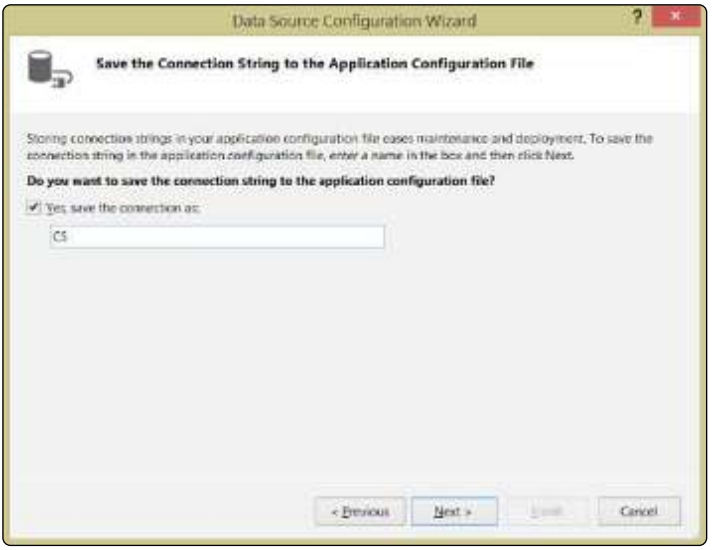
Veri tabanı türünü seçerek **OK** butonuna tıklarız. Eğer veri tabanı için kullanıcı adı ve şifre belirlemişsek **"User name"** isimli metin kutusuna kullanıcı adı, **"Password"** isimli metin kutusuna ise veritabanı şifremizi bu bölümde gireriz.

Evet butonuna tıklayarak veritabanını projemize ekliyoruz.

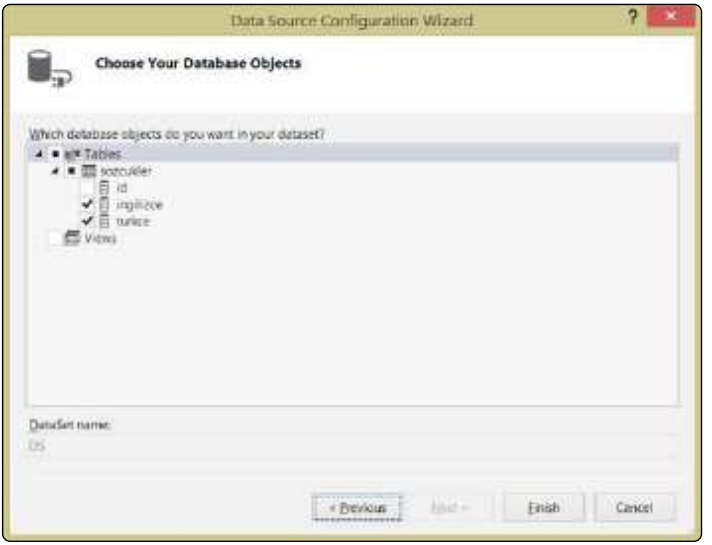




Gelen pencerede **connection string** ismini belirliyoruz.



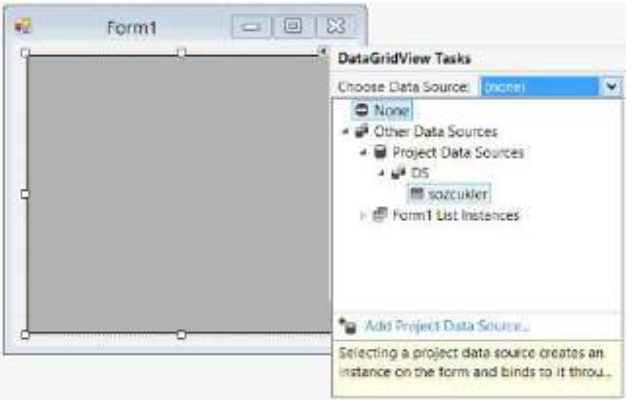
Gerekli ismi verip **Next** butonuna tıkladığımızda, karşımıza **DataSet** içerisinde hangi tablo ve alanların kullanılacağının seçildiği bir pencere gelir. Burada kullanmak istediğimiz tablo ve alanları seçip, **DataSet** ismini belirleyerek **Finish** butonuna basarak, veritabanı bağlama işlemini bitirmiş oluruz.





Data Sources penceresi içerisine baktığımızda seçtiğimiz tabloların eklendiğini görebiliriz.

Form'umuzun üzerine **ToolBox** içerisinden **DataGridView** nesnesi ekleyelim. **DataGridView**'e oluşturmuş olduğumuz **DataSet**'i bağlamak için **DataGridView**'in sağ üst köşesindeki kulakçığa tıklayarak **DataSet**'i seçelim.



Dataset bağlantı işleminin bitmesinin ardından proje içerisine **DataSet**, **TableAdapter**, **BindingSource** nesnelerinin eklendiğini görürüz.



DataSet nesnesi ile veritabanı bağlantısının yapılacağı veri tabanı seçilir.

TableAdapter nesnesi ile verilerin çekileceği tablo belirtilir.



BindingSource nesnesi ile veri tabanındaki tabloların içerisinde bulunan alanlar ile form üzerindeki nesnelerin bağlantısı yapılır.

Uygulamamızı çalıştırdığımızda veritabanındaki değerlerin **DataGridView** içerisinde listelendiğini görürüz.

OLEDBCOMMAND

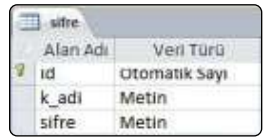
Veritabanı işlemleri ile ilgili kullanılan sorguları çalıştırmak için kullanılır. Komutları yürütmek için 4 çeşit **Execute** metoduna sahiptir:

- **`ExecuteReader`:** Yürütülen komut sonrasında geriye DataReader nesnesi döndürür.
- **`ExecuteNonQuery`:** Ekleme, silme, güncelleme gibi sorgulardan etkilenen satır sayısını geriye döndürür.
- **`ExecuteScalar`:** Bu metod geriye tek bir değer döndüren sorgular için kullanılır. Bu değer kayıt sayısı, bir alanın değeri gibi değerler olabilir.
- **`ExecuteXMLReader`:** Yürütülen komut sonrasında geriye **XML Reader** nesnesi döndürür.

Veritabanı işlemlerimizde kullanacağımız örnek bir veritabanı tasarlayalım. Veritabanımız bir sözlük uygulamasının veritabanı olacak. Veri tabanımızın adını **`sozluk`** olarak belirliyoruz.

Veritabanımızda **`sozcukler`** ve **`sifre`** tablolarını oluşturuyoruz. Bu tablolarda aşağıda görülen alanları oluşturuyoruz.





`OleDbConnection`, **`OleDbCommand`** ve **`OleDbAdapter`** nesnelerini kodumuzda kullanabilmek için öncelikle **`System.Data`** ve **`System.Data.OleDb`** sınıflarını projemizde dahil etmemiz gerekmektedir.

Daha sonra yeni bir **Connection String** oluşturmamız gerekmektedir.

Access 97-2003 **.mdb** uzantılı veri tabanları için bağlantı cümlesi aşağıdaki biçimde yazılır:

```
OleDbConnection baglantı
= new OleDbConnection ("Provider=Microsoft.Jet.Oledb.4.0;
Data Source = " + Application.StartupPath + "/sozluk.mdb")
```

Access 2003 sonrası **.accdb** uzantılı veri tabanları için ise bağlantı cümlesi aşağıdaki biçimde yazılır:

```
OleDbConnection baglantı
= new OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0;
Data Source = " + Application.StartupPath + "/sozluk.accdb")
```



Buradaki "OLEDB.12.0" ifadesi kullanılan Access versiyonuna göre değişiklik göstermektedir. Örneğin bu ifade Access 2019 versiyonu için "OLEDB.16.0" şeklinde yazılır.

Access 2003 sonrası **.accdb** uzantılı veri tabanlarına erişim sağlayabilmek için bilgisayarda ilgili ofis sürümünün ya da "**Microsoft Access Database Engine**" isimli bileşenin yüklü olması gerekiyor.



.mdb uzantılı veri tabanı dosyaları ile çalışmak için gerekli olan **Microsoft.Jet.Oledb.4.0** sürücüsünü çalıştırabilmek için Visual Studio içerisinde **Project Properties** penceresinde **Build** sekmesinde **Platform Target** bölümünde **x86** seçilmelidir.

.accdb uzantılı veri tabanı dosyaları ile çalışmak için gerekli olan **Microsoft.ACE.OLEDB** sürücüsünü çalıştırabilmek için ise Visual Studio içerisinde **Project Properties** penceresinde **Build** sekmesinde **Platform Target** bölümünde **x64** seçilmelidir.

Connection string içerisindeki "**`Application.StartupPath`**" deyimi, programın çalıştığı yolu belirtir. Yukarıdaki bağlantı cümlesindeki "**`Data Source`**" deyimi ile veri tabanımızın; programımız ile aynı klasörde bulunduğunu belirttik.

Veritabanına bağlantı yaptıktan sonra, yapılacak işlemler için açılması, işlemler sonrasında da kapatılması gerekir. Veritabanını kullanıma açmak için kullanılan kod satırı aşağıdaki gibi olmalıdır.

```
baglantı.Open()
```

Yapılan işlemler sonrasında veritabanının kapatılması için kullanılacak kod satırı şu şekildedir:

```
baglantı.Close()
```

Veritabanı kapatılmadan tekrar açılmak istendiğinde ise hata mesajı verecektir.

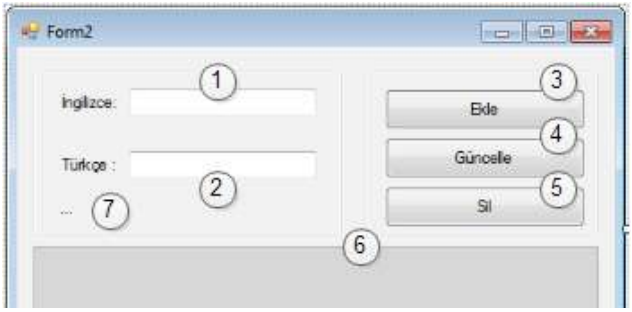
`OleDBConnection` nesnesi ile veritabanına bağlantı yapıldıktan sonra; kayıt ekleme, kayıt silme ve kayıt güncelleme gibi işlem komutları **`Command`** nesnesi

ile tanımlanmalıdır. Örneğin; **Select**, **Insert**, **Update**, **Delete** gibi SQL komutları, **Command** nesnesi ile çalıştırılabilir.

Veritabanı ile ilgili işlemler aşağıdaki sıraya uygun olarak yapılmalıdır.

- **OleDbConnection** kullanılarak, veritabanı bağlantı nesnesi tanımlanır.
- Veritabanı üzerinde işlem yapılmak üzere **Open()** metodu ile kullanıma açılır.
- **OleDbCommand** kullanılarak, yeni bir **Command** nesnesi tanımlanır ve veritabanı üzerinde işlem görecek SQL ifadeleri **Command** nesnesine atanır.
- **Command** nesnesine atanan SQL ifadelerinin çalıştırılması **Insert**, **Update**, **Delete** deyimlerinde **ExecuteNonQuery()** metodu ile **Select** deyiminde ise **ExecuteReader()** metodu ile gerçekleştirilir.
- SQL ifadesi ile alınan değerler kullanıldıkdan sonra veritabanı bağlantısı, **Close()** metodu ile kapatılır.

OleDBCommand kullanımına dair yapacağımız örneğimizde iki adet form bulunacak. İlk form ile şifreli giriş işlemi yapılacak. Eğer şifre doğru girilmiş ise,



ikinci formumuz olan sözlük kodlarının yer aldığı form ekrana gelecek.

Öncelikle ikinci formumuz olan sözlük kodlarımızın yer aldığı formda gerçekleştirilen veritabanı işlemlerini sırasıyla inceleyelim.

Formumuzun tasarımı aşağıdaki gibi olacak:

(1) **txt_ingilizce**
(2) **txt_turkce**
(3) **btnEkle**
(4) **btnGuncelle**
(5) **btnSil**
(6) **dataGridView1**
(7) **lblKayıtSayısı**

Kayıt Ekleme

Yeni bir kayıt eklemek için **insert** sorgusunu kullanıyoruz. **Insert** sorgusu bir tablodaki belirtilen bir alana bir ya da daha çok değeri eklemek amacı ile kullanılır. Ayrıca veritabanına yeni kayıt eklemek için **OleDbCommand** nesnesinin **ExecuteNonQuery()** metodunu kullanıyoruz.

```
using System;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;
public partial class Form1 : Form
{
 OleDbConnection baglantı = new OleDbConnection
   ("Provider= Microsoft.ACE.OLEDB.12.0;Data Source = " +
   Application.StartupPath + "/sozluk.accdb");
 OleDbCommand komut = new OleDbCommand();

 private void btnEkle_Click(object sender, EventArgs e)
 {
  try
  {
    DataSet ds = new DataSet();
    if (baglantı.State == ConnectionState.Closed) baglantı.Open();
    ds.Clear();

    OleDbCommand komut = new OleDbCommand
("INSERT INTO sozcukler (ingilizce,turkce) VALUES ('" + txt_ingilizce.Text
+ "','" + txt_turkce.Text + "')", baglantı);
    komut.ExecuteNonQuery();
    dataGridView1.Update();
    baglantı.Close();
    MessageBox.Show("Kayıt Eklendi!");
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message);
    baglantı.Close();
  }
 }
}
```

Kodumuzda öncelikle **baglantı** adlı yeni bir **Connection String** (*bağlantı cümlesi*) tanımladık. Örneğimizde veritabanı olarak Access 2010 veritabanı kullanıldığı için bağlantı cümlemizde **Microsoft.ACE.OLEDB.12.0** sınıfını kullandık.

Veritabanındaki kayıtlar üzerinde okuma ya da yazma yapmak için ds isimli yeni bir dataset nesnesi tanımladık.

```
if (baglantı.State == ConnectionState.Closed)
baglantı.Open();
```

… satırında, eğer bağlantı durumu kaplı ise; veritabanı bağlantısının açılmasını sağlıyoruz. Aksi takdirde bağlantı açıkken, bağlantıyı yeniden açmaya kalktığımızda hata durumu meydana gelecektir.

```
ds.Clear();
```

Bu satırı **DataSet**'in içerisini temizliyoruz.

```
OleDbCommand komut = new OleDbCommand
  ("INSERT INTO sozcukler (ingilizce,turkce) VALUES ('" +
  txt_ingilizce.Text + "','" + txt_turkce.Text + "')", baglantı);
```

… satırında **OleDbCommand** nesnesini kullanarak **INSERT** sorgusunu çalıştırıp, belirtilen text kutularındaki değerleri veritabanına ekliyoruz.

```
komut.ExecuteNonQuery();
```

… satırında **OleDbCommand** nesnesi içerisindeki sorgumuzun çalıştırılmasını sağlıyoruz. Geriye herhangi bir değer döndürülmeyip, sadece sorgu sonucunda kayıt ekleme işlemi gerçekleştirileceği için **ExecuteNonQuery** metodunu kullandık.

```
dataGridView1.Update();
```

Kayıt ekleme işlemi sonrasında **DataGridView** içerisinde listelenen kayıtların güncellenmesini sağladık.

```
baglantı.Close();
```

Veritabanı bağlantımızı kapattık.

Kayıt Güncelleme

Kayıtlar üzerinde güncelleme yapmak için **update** sorgusunu kullanıyoruz. **Update** sorgusu bir tablodaki belirtilen bir alana ait veri üzerinde yapılan değişiklikler uygulanarak alanda bulunan veri güncellenir. Ayrıca veritabanındaki kayıtları güncellemek için **OleDbCommand** nesnesinin **ExecuteNonQuery()** metodunu kullanıyoruz.

```
using System;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;
public partial class Form1 : Form
{
 OleDbConnection baglantı = new OleDbConnection
   ("Provider= Microsoft.ACE.OLEDB.12.0;Data Source = "
+ Application.StartupPath + "/sozluk.accdb");

 private void btnGuncelle_Click(object sender, EventArgs e)
 {
  try
  {
    DataSet ds = new DataSet();
    if (baglantı.State == ConnectionState.Closed) baglantı.Open();
    ds.Clear();

    OleDbCommand komut = new OleDbCommand
  ("UPDATE sozcukler SET ingilizce='" + txt_ingilizce.Text +
     "', turkce='" + txt_turkce.Text + "' WHERE ingilizce='" +
     txt_ingilizce.Text + "'", baglantı);
    komut.ExecuteNonQuery();
    dataGridView1.Update();
    baglantı.Close();
    MessageBox.Show("Kayıt Güncellendi!");
  }
  catch (Exception ex)
```

```
  {
    MessageBox.Show(ex.Message);
    baglantı.Close();
  }
 }
}
```

Kodumuzda…

```
OleDbCommand komut = new OleDbCommand
("UPDATE sozcukler SET ingilizce='" + txt_ingilizce.Text +
   "', turkce='" + txt_turkce.Text + "' WHERE ingilizce='" +
   txt_ingilizce.Text + "'", baglantı);
```

… satırında **`OleDbCommand`** nesnesini kullanarak **`UPDATE`** sorgusunu çalıştırıp, belirtilen text kutularındaki değerler kullanılarak, veritabanındaki kayıtların güncellenmesini sağlıyoruz.

Kayıt Silme

Kayıtları silmek için **`delete`** sorgusunu kullanıyoruz. **`Delete`** sorgusu, bir tablodaki belirtilen bir alana ait bir ya da daha fazla veriyi silmek amacı ile kullanılır. Ayrıca veritabanındaki kayıtları silmek için **`OleDbCommand`** nesnesinin **`ExecuteNonQuery()`** metodunu kullanıyoruz.

```
using System;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;
public partial class Form1 : Form
{
 OleDbConnection baglantı = new OleDbConnection
   ("Provider= Microsoft.ACE.OLEDB.12.0;Data Source = "
+ Application.StartupPath + "/sozluk.accdb");

 private void btnSil_Click(object sender, EventArgs e)
 {
  try
  {
    DataSet ds = new DataSet();
    if (baglantı.State == ConnectionState.Closed) baglantı.Open();
```

```
    ds.Clear();

    OleDbCommand komut = new OleDbCommand
("DELETE FROM sozcukler WHERE ingilizce='" + txt_ingilizce.Text +
"'", baglantı);
    komut.ExecuteNonQuery();
    dataGridView1.Update();
    dataGridView1.Refresh();
    baglantı.Close();
    MessageBox.Show("Kayıt Silindi!");
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message);
    baglantı.Close();
  }
 }
}
```

Kodumuzda…

```
OleDbCommand komut = new OleDbCommand
("DELETE FROM sozcukler WHERE ingilizce='" + txt_ingilizce.Text +
"'", baglantı);
```

… satırında **`OleDbCommand`** nesnesini kullanarak **`DELETE`** sorgusunu çalıştırıp, belirtilen text kutusundaki değer veritabanında aranarak, bulunan kaydın silinmesini sağlıyoruz.

Şifreli giriş formumuzun tasarımı aşağıdaki gibi olacak:



(1) `txtKul_Adi`

(2) `txtSifre`

(3) `btnTamam`

(4) `btniptal`

Şifreli giriş formunun kodu:

```
using System;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;
namespace OleDbCommand_Sozluk
{
 public partial class frmSifre : Form
 {
  public Form1()
  {
    InitializeComponent();
  }
  OleDbConnection baglantı = new OleDbConnection
   ("Provider= Microsoft.ACE.OLEDB.12.0;Data Source = "
+ Application.StartupPath + "/sozluk.accdb");
  OleDbCommand komut = new OleDbCommand();
  string deger;

  private void btnTamam_Click(object sender, EventArgs e)
  {
    if (baglantı.State == ConnectionState.Closed)
      baglantı.Open();
    komut = new OleDbCommand("SELECT * FROM sifre where sifre='" +
      txtSifre.Text + "' ", baglantı);
    try
    {
      deger = komut.ExecuteScalar().ToString();
      if (Convert.ToInt32(deger) == 1)
      {
        MessageBox.Show("Şifre doğru");
        baglantı.Close();
        this.Hide();
        Form2 frm2 = new Form2();
        frm2.Show();
      }
      else
        MessageBox.Show("Şifre yanlış");
      baglantı.Close();
    }
```

```
        Catch { }
    }
    private void btniptal_Click(object sender, EventArgs e)
    {
     Application.Exit();
    }
  }
}
```



Şifre giriş formunu yapmaktaki asıl amacımız; **`OleDbCommand`** nesnesinin **`ExecuteScalar`** metodunun kullanımını görmektir. Veritabanımızda tek kullanıcı ve buna ait tek şifre olması nedeni ile **`ExecuteScalar`** metodunu kullandık. Çünkü **`ExecuteScalar`** metodu geriye tek bir değer gönderen sorgular için kullanılmaktadır. Eğer **`ExecuteScalar`** metodu ile çalıştırılan sorgu doğru sonuç verirse, geriye **`1`** değerini göndermektedir. Aksi takdirde **`null`** değerini göndermektedir. Sorgu doğru sonuç vermediğinde geriye dönen null değeri, programda hata oluşumuna neden olduğundan dolayı **`try catch`** bloğu kullandık.

Bağlantı cümlesi tanımlandıktan sonra…

```
BağlantıCümlesiAdı.Open()
```

… şeklinde veritabanı bağlantısını açıyoruz. Ancak eğer bağlantı daha önceden açıksa, bu durum hata oluşumuna neden olacaktır. Bunu engellemek için programımızda yazdığımız gibi…

```
if (baglantı.State == ConnectionState.Closed)
    baglantı.Open();
```

… öncelikle **`BağlantıcümlesiAdı.State`** metodu ile bağlantı durumunu

kontrol ederek, eğer bağlantı durumu **`ConnectionState.Closed`** (*bağlantı durumu kapalı*) ise, bağlantıyı açıyoruz.

Kayıt ekleme, **güncelleme** ve **silme** işlemlerine dair kodlarımızı birleştirerek sözlük formumuzun kodlarını yazalım.

```
using System;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;
namespace OleDbCommand_Sozluk
{
  public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
    }
    OleDbConnection baglantı = new OleDbConnection
      ("Provider= Microsoft.ACE.OLEDB.12.0;Data Source = "
+ Application.StartupPath + "/sozluk.accdb");
    OleDbCommand komut = new OleDbCommand();
    OleDbDataAdapter adaptor = new OleDbDataAdapter();
    DataSet ds = new DataSet();
    byte i;

    private void Form1_Load(object sender, EventArgs e)
    {
      if ((baglantı.State == ConnectionState.Closed)) baglantı.Open();

      komut = new OleDbCommand("SELECT * FROM sozcukler ", baglantı);

      OleDbDataReader reader = komut.ExecuteReader();
      reader.Read();

      if (reader.HasRows)
      {
        txt_turkce.Text = reader["turkce"].ToString();
        txt_ingilizce.Text = reader.GetString(1).ToString();
      }

      adaptor.SelectCommand = new OleDbCommand
("SELECT ingilizce,turkce FROM sozcukler ", baglantı);
```

```
            adaptor.Fill(ds);
            dataGridView1.DataSource = ds.Tables[0];

            komut = new OleDbCommand("SELECT COUNT(*) FROM sozcukler", baglantı);
            lblKayıtSayısı.Text = "Veritabanında " +
        (int)komut.ExecuteScalar() + " tane kelime var.";
            baglantı.Close();
        }
        private void dataGridView1_CellClick(object sender,
        DataGridViewCellEventArgs e)
        {
            int satirno;
            satirno = e.RowIndex;
            txt_ingilizce.Text = dataGridView1.Rows[satirno].
        Cells[0].Value.ToString();
            txt_turkce.Text = dataGridView1.Rows[satirno].
        Cells[1].Value.ToString();
        }
        private void btnEkle_Click(object sender, EventArgs e)
        {
            try
            {
                DataSet ds = new DataSet();
                if (baglantı.State == ConnectionState.Closed) baglantı.Open();
                ds.Clear();

                OleDbCommand komut = new OleDbCommand
        ("INSERT INTO sozcukler (ingilizce, turkce) VALUES
        (‘" + txt_ingilizce.Text + "’,’" + txt_turkce.Text + "’)", baglantı);
                komut.ExecuteNonQuery();
                dataGridView1.Update();
                baglantı.Close();
                MessageBox.Show("Kayıt Eklendi!");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                baglantı.Close();
            }
        }
        private void btnGuncelle_Click(object sender, EventArgs e)
```

```
    {
      try
      {
        DataSet ds = new DataSet();
        if (baglantı.State == ConnectionState.Closed) baglantı.Open();
        ds.Clear();

        OleDbCommand komut = new OleDbCommand
("UPDATE sozcukler SET ingilizce='" + txt_ingilizce.Text + "', turkce='" +
txt_turkce.Text + "' WHERE ingilizce='" + txt_ingilizce.Text + "'",
baglantı);
        komut.ExecuteNonQuery();
        dataGridView1.Update();
        baglantı.Close();
        MessageBox.Show("Kayıt Güncellendi!");
      }
      catch (Exception ex)
      {
        MessageBox.Show(ex.Message);
        baglantı.Close();
      }
    }
    private void btnSil_Click(object sender, EventArgs e)
    {
      try
      {
        DataSet ds = new DataSet();
        if (baglantı.State == ConnectionState.Closed) baglantı.Open();
        ds.Clear();

        OleDbCommand komut = new OleDbCommand
("DELETE FROM sozcukler WHERE ingilizce='" + txt_ingilizce.Text +
"'", baglantı);
        komut.ExecuteNonQuery();
        dataGridView1.Update();
        dataGridView1.Refresh();
        baglantı.Close();
        MessageBox.Show("Kayıt Silindi!");
      }
      catch (Exception ex)
      {
```

```
            MessageBox.Show(ex.Message);
            baglantı.Close();
        }
    }
  }
}
```



Form oluşturulurken;

- Sql komutlarını çalıştırmak için **`komut`** adında **`OleDbCommand`** nesnesi,
- Veritabanından çekilen veriyi **DataSet**'e aktarmak ve **DataSet**'te güncellenmiş veriyi veritabanına aktarmada kullanılmak üzere **`adaptor`** adında **`OleDbDataAdapter`** nesnesi,
- **`OleDbDataAdapter`** ile veritabanından çekilen verilerin **DataSet** içerisinde tablolar şeklinde saklanması ve veritabanı ile ilişkilendirilmesi için **`ds`** adında **DataSet** nesnesi tanımlıyoruz.

Formun **Load** olayı esnasında veritabanındaki kayıtları çekerek **DataGridView** içerisine ve **TextBox** nesneleri içerisine veritabanındaki değerleri ekliyoruz.

Kayıtları okumak için **`Select`** sorgusunu kullanıyoruz. **`Select`** sorgusu veritabanındaki kayıtları çekmek amacı ile kullanılır. Kodumuzda

```
SELECT * FROM sozcukler
```

Sorgusu ile **`sozcukler`** tablosundaki bütün kayıtları çektik. **`*`** karakteri ile veritabanındaki bütün kayıtlar çekilir.

Aynı sorguyu…

```
SELECT id, ingilizce, turkce FROM sozcukler
```

… şeklinde sozcukler tablosundaki bütün alan isimlerini yazarak da çalıştırabiliriz.

Veritabanındaki kayıtları çekmek için **`OleDbCommand`** nesnesinin **`ExecuteReader()`** metodunu kullanıyoruz.

`reader.Read()` metodu ile kayıt çekme işlemini başlatıyoruz.

`HasRows` metodu ile sorgumuz çalıştırıldıktan sonra kayıt olup olmadığını sorgulayarak, formun **`Load`** olayı esnasında **DataGridView** nesnesinin birinci satırındaki değerleri, formdaki **TextBox** nesnelerine yüklüyoruz.

`txt_turkce.Text = reader["turkce"].ToString()` satırı ile sorgumuzun çalışması sonucunda oluşan tabloda, ilk satırdaki **`turkce`** alanında bulunan değeri **`txt_turkce`** isimli metin kutusuna ekleniyor.

`txt_ingilizce.Text = reader.GetString(1).ToString()` satırı ile de sorgumuzun çalışması sonucunda oluşan tabloda, ilk satırın ikinci sutunundaki (İngilizce karşılığı) değer, **`txt_ingilizce`** isimli metin kutusuna ekleniyor. Bu işlemi…

```
txt_ingilizce.Text = reader["ingilizce"].ToString( )
```

… kod satırı ile de gerçekleştirebiliriz.

Veritabanından tüm kayıtları çekip formda kullanmaya hazırlamak için;

```
adaptor.SelectCommand = new OleDbCommand
("SELECT ingilizce,turkce FROM sozcukler ", baglantı)
```

… komut satırını kullandık. **`Adoptor`** nesnesi ile veritabanımızdan çekilen bilgileri **DataSet**'e aktarmak için ise aşağıdaki komut satırını kullandık.

```
adaptor.Fill(ds)
```

Bir **DataSet**'te birden fazla tablo bilgisi tutulabildiğinden dolayı **DataSet**'teki

ilk tablonun formdaki **DataGridView**'e yükleme işlemini ise aşağıdaki komut satırı ile yapıyoruz.

```
dataGridView1.DataSource = ds.Tables[0]
```

Ayrıca formun **Load** olayı esnasında veritabanında bulunan kayıt sayısını bularak görüntülüyoruz. Bu işlem için aşağıdaki sorguyu kullanıyoruz.

```
SELECT COUNT(*) FROM sozcukler
```

Bu şekilde **sozcukler** tablosundaki kayıt sayısını buluyoruz. Geriye tek bir değer dönmesi nedeni ile de **OleDbCommand** nesnesinin **ExecuteScalar()** metodunu kullanıyoruz.

Kayıt eklemek için kodumuzda…

```
INSERT INTO sozcukler (ingilizce, turkce) VALUES ('"
+ txt_ingilizce.Text + "','" + txt_turkce.Text + "')
```

… sorgusu ile **sozcukler** tablosundaki **ingilizce** ve **turkce** adlı alanlara **txt_ingilizce** ve **txt_turkce** adlı text kutularında yazılı olan değerleri aktardık. Ayrıca veritabanına yeni kayıt eklemek için **OleDbCommand** nesnesinin **ExecuteNonQuery()** metodunu kullandık.

```
dataGridView1.Update()
```

… komut satırı ile veritabanında yapılan değişikliklerin **DataGridView**'e aktarılmasını sağlıyoruz.

Kayıtlar üzerinde güncelleme yapmak için kodumuzda…

```
UPDATE sozcukler SET ingilizce='" + txt_ingilizce.Text + "', turkce='"
+ txt_turkce.Text + "' WHERE ingilizce='" + txt_ingilizce.Text + "'
```

… sorgusu ile sozcukler tablosundaki **ingilizce** ve **turkce** adlı alanlardaki veriler, eğer **ingilizce** adlı alandaki veri **txt_ingilizce** adlı text kutusundaki veriye eşit ise, **txt_ingilizce** ve **txt_turkce** adlı text kutularında yazılı olan yeni değerlere göre güncellenir.

Ayrıca veritabanındaki kayıtları güncellemek için **OleDbCommand** nesnesinin **ExecuteNonQuery()** metodunu kullandık.

Kayıtları silmek için kodumuzda…

```
DELETE FROM sozcukler WHERE İngilizce = '" + txt_ingilizce.Text + "'
```

Sorgusu ile **sozcukler** tablosundaki **ingilizce** adlı alandaki veri **txt_ingilizce** adlı text kutusundaki değere eşit ise bu satırdaki veri silinir.

Ayrıca veritabanındaki kayıtları silmek için **OleDbCommand** nesnesinin **ExecuteNonQuery()** metodunu kullandık.

C# içerisinde her nesnenin çalışma anında kendine ait olayları (tıklanma, imlecin nesne üzerindeki konumu, üzerine sürükleme gibi) bulunmaktadır. **DataGridView**'inde kendine has olayları bulunmaktadır. Bu olaylar **DataGridViewCellEventArgs** sınıfının **e** değişkenine aktarılması ile yakalanabilir.

```
satirno = e.RowIndex
```

… komut satırı ile tabloda tıklanan hücrenin satır numarası alınmakta ve…

```
txt_ingilizce.Text = DataGridView1.Rows[satirno].Cells[0].Value.ToString()
```

… komut satırı ile tıklanan satırın ilk hücresindeki değer, **txt_ingilizce** isimli text kutusuna aktarılmakta ve Türkçe karşılığı ise…

```
txt_turkce.Text = dataGridView1.Rows[satirno].Cells[1].Value.ToString()
```

… kodu ile de **txt_turkce** isimli text kutusuna aktarılmaktadır.

OLEDBDATAADAPTER

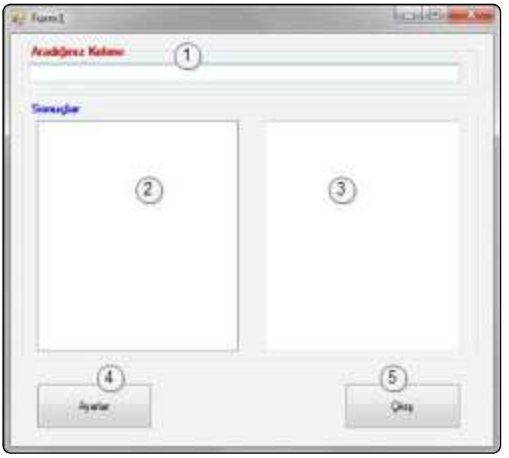
Veritabanı bağlantısı olmasa bile veriler üzerinde işlemler gerçekleştirilebilir. Veritabanına bağlanıp veriler çekilir. Daha sonra bağlantı kesilse bile, daha önceden çekilmiş veriler üzerinde işlem yapılmaktadır. Üzerinde değişiklik yapılmış veriler daha sonra veritabanına gönderilir.

OleDbAdapter nesnesini kodumuzda kullanabilmek için öncelikle **System.Data** ve **System.Data.OleDb** sınıflarını projemize dahil etmemiz gerekmektedir.

`OleDBDataAdapter` kullanımına dair yapacağımız örneğimizde iki adet form bulunacak. İlk formumuz ile sözlük veritabanımızda arama işlemi yapılacak. Bu formdaki **Ayarlar** butonuna tıklanınca, ikinci formumuz olan `frmAyarlar` isimli form görüntülenecektir. Bu formda ise kayıt ekleme, güncelleme, silme işlemlerini gerçekleştireceğiz.

Öncelikle sözlük veritabanında arama işlemlerinin yapıldığı ilk formumuzda gerçekleştirilen veritabanı işlemlerini sırasıyla inceleyelim.

Formumuzun tasarımı aşağıdaki gibi olacak:



(1) `txtaranan`

(2) `lstsonuc`

(3) `txtsonuc`

(4) `btnAyar`

(5) `btnCikis`

Kayıt Listeleme

Veritabanındaki kayıtları listelemek için `Select` sorgusunu kullanıyoruz. `Select` sorgusu kullanılarak bir tablodaki belirtilen bir alandaki bir ya da daha çok veriler çekilir ve çekilen veriler `OledbDataAdapter` içerisine doldurulur. Daha sonra çekilen bu veriler **DataSet**'e aktarılır. **DataSet**'teki veriler de form üzerindeki kontrollere bağlanarak aktarılır.

```
using System.Data;
using System.Data.OleDb;
public partial class Form1 : Form
{
 OleDbConnection baglantı = new OleDbConnection
("Provider= Microsoft.ACE.OLEDB.12.0;Data Source = "
+ Application.StartupPath + "/sozluk.accdb");
```

```
 private void Form1_Load(object sender, EventArgs e)
 {
  try
  {
   DataSet ds = new DataSet();
   if ((baglantı.State == ConnectionState.Closed))
     baglantı.Open();

   OleDbDataAdapter adp = new OleDbDataAdapter
  ("SELECT * FROM sozcukler", baglantı);

   adp.Fill(ds);
   lstsonuc.DataSource = ds.Tables[0];
   lstsonuc.DisplayMember = "ingilizce";
   txtsonuc.DataBindings.Clear();
   txtsonuc.DataBindings.Add(new System.Windows.Forms.Binding
  ("text", ds.Tables[0], "turkce", true,
System.Windows.Forms.DataSourceUpdateMode.OnPropertyChanged));
   baglantı.Close();
   lstsonuc.SelectedIndex = 0;
   txtingilizce.Text = lstsonuc.Text;
   txtturkce.Text = txtsonuc.Text;
  }
  catch (Exception ex)
  {
   MessageBox.Show(ex.Message);
   baglantı.Close();
  }
 }
}
```

Kayıtların listelenmesi esnasında form üzerindeki kontrollerin veritabanı bağlantısını kurduk.

Yeni bir dataset tanımlayarak **DataAdapter** nesnesi içerisindeki verileri **DataSet** içerisine aktarıyoruz. **DataSet** veritabanından bağlantısız çalışmaktadır. Bir defa bağlanıp veriler çekildikten sonra, veritabanı ile bağlantı kesilir. **DataAdapter**, **DataSet** ile veritabanında aracılık vazifesi görmektedir. Veritabanındaki verileri **DataSet**'e aktarmak, **DataSet** içerisinde güncellenmiş olan verileri de veritabanına aktarmak amacı ile kullanılır.

ListBox nesnesinin veritabanı ile bağlantısını kurmak için **`DataSource`** özelliğine, **DataSet** içerisindeki verilerden belirtilen tablo ismini çekiyoruz. **`Tables[0]`** kullanımı ile ilk sıradaki tablo bağlantısı yapılmıştır. Bunun yerine **`Tables["sozcukler"]`** şeklinde de kullanılabilir. **`DisplayMember`** özelliği ile bağlantı kurulacak alan ismi belirtilmiştir.

TextBox nesnesinin veritabanındaki ilgili alanla bağlantısını kurmak için **`DataBindings.Add`** metodu kullanılabilir.

Örneğimizde görüldüğü gibi txtsonuc adlı **TextBox** nesnesini, text veri biçimi ile `0` nolu tablonun (sözcükler tablosuna) turkce adlı alanına bağladık.

Kayıt Ekleme

Yeni bir kayıt eklemek için **`insert`** sorgusunu kullanıyoruz. **`Insert`** sorgusu bir tablodaki belirtilen bir alana bir ya da daha çok değeri eklemek amacı ile kullanılır. Eklenen veriler önce **OledbDataAdapter**'e aktarılmakta, buradan da **DataSet**'e aktarılmaktadır.

```
using System.Data;
using System.Data.OleDb;
public partial class Form1 : Form
{
 OleDbConnection baglantı = new OleDbConnection
("Provider= Microsoft.ACE.OLEDB.12.0;Data Source=" +
Application.StartupPath + "/sozluk.accdb ");
 private void btnEkle_Click(object sender, EventArgs e)
 {
  if (string.IsNullOrEmpty(txtingilizce.Text) |
string.IsNullOrEmpty(txtturkce.Text))
  {
    MessageBox.Show("Lütfen Bütün Alanları Doldurunuz", "Hata");
    return;
  }
  try
  {
    DataSet ds = new DataSet();
    baglantı.Open();
    ds.Clear();
    OleDbDataAdapter adp = new OleDbDataAdapter
```

```
  ("INSERT INTO sozcukler (ingilizce,turkce) VALUES ('" +
  txtingilizce.Text + "','" + txtturkce.Text + "')", baglantı);
    adp.Fill(ds);
    baglantı.Close();
    MessageBox.Show("Kayıt Eklendi", "Bilgi");
    txtingilizce.Text = "";
    txtturkce.Text = "";
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message);
    baglantı.Close();
  }
 }
}
```

Kayıt Güncelleme

Kayıtlar üzerinde güncelleme yapmak için **update** sorgusunu kullanıyoruz. **Update** sorgusu bir tablodaki belirtilen bir alana ait veri üzerinde yapılan değişiklikler uygulanarak, alanda bulunan veri güncellenir. Güncellenen veriler önce **OledbDataAdapter**'e aktarılmakta, buradan da **DataSet**'e aktarılmaktadır. Daha sonra veritabanı ile bağlantı kurulduğunda değişmiş olan veriler, veri tabanındaki eski değerleri ile değiştirilir.

```
using System.Data;
using System.Data.OleDb;
public partial class Form1 : Form
{
 OleDbConnection baglantı = new OleDbConnection
("Provider= Microsoft.ACE.OLEDB.12.0;Data Source=" +
Application.StartupPath + "/sozluk.accdb ");
 private void btnGuncelle_Click(object sender, EventArgs e)
 {
  try
  {
    DataSet ds = new DataSet();
    baglantı.Open();
    ds.Clear();
```

```
    OleDbDataAdapter adp = new OleDbDataAdapter
  ("UPDATE sozcukler SET ingilizce='" + txtingilizce.Text +
  "', turkce='" + txtturkce.Text + "' WHERE ingilizce='" +
  txtingilizce.Text + "'", baglantı);
    adp.Fill(ds);
    baglantı.Close();
    MessageBox.Show("Kayıt Güncellendi.", "Bilgi");
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message);
    baglantı.Close();
  }
 }
}
```

Kayıt Silme

Kayıtları silmek için **delete** sorgusunu kullanıyoruz. **Delete** sorgusu bir tablodaki belirtilen bir alana ait bir ya da daha fazla veriyi silmek amacı ile kullanılır.

```
using System.Data;
using System.Data.OleDb;
public partial class Form1 : Form
{
 OleDbConnection baglantı = new OleDbConnection
("Provider= Microsoft.ACE.OLEDB.12.0;Data Source=" +
Application.StartupPath + "/sozluk.accdb");
 private void btnSil_Click(object sender, EventArgs e)
 {
  try
  {
    DataSet ds = new DataSet();
    baglantı.Open();
    ds.Clear();
    OleDbDataAdapter adp = new OleDbDataAdapter
  ("DELETE FROM sozcukler WHERE ingilizce='" +
  txtingilizce.Text + "'", baglantı);
```

```
    adp.Fill(ds);
    baglantı.Close();
    MessageBox.Show("Kayıt Silindi","Bilgi");
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message);
    baglantı.Close();
  }
 }
}
```

Kayıt Arama

Kayıtlar içerisinde arama yapmak için **like** sorgusunu kullanabiliriz. **Like** sorgusu bir tablodaki belirtilen bir alandaki veriler, aranan değer ile karşılaştırılarak bulunan değerler geriye döndürülür.

Aşağıdaki örnekte…

```
SELECT * FROM sozcukler WHERE ingilizce LIKE '" + txtaranan.Text + "%'
```

… komut satırı ile **txtaranan** isimli metin kutusuna girilen değer ile başlayan kayıtları getiriyoruz.

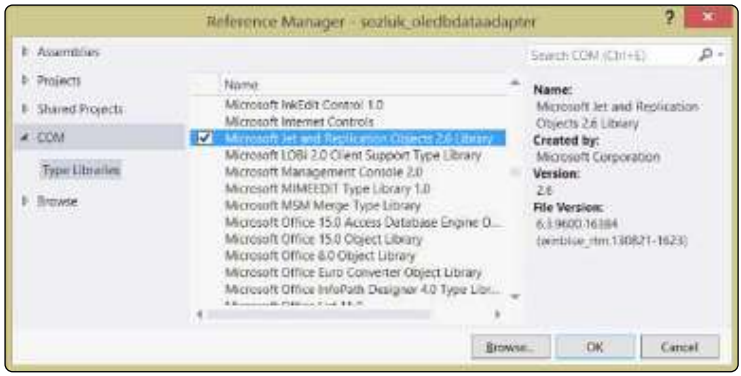
```
using System.Data;
using System.Data.OleDb;
public partial class Form1 : Form
{
 OleDbConnection baglantı = new OleDbConnection
  ("Provider= Microsoft.ACE.OLEDB.12.0;Data Source = "
+ Application.StartupPath + "/sozluk.accdb");
 private void txtaranan_TextChanged(object sender, EventArgs e)
 {
  try
  {
    DataSet ds = new DataSet();
    if ((baglantı.State == ConnectionState.Closed))
    baglantı.Open();
    OleDbDataAdapter adp = new OleDbDataAdapter
```

```
      ("SELECT * FROM sozcukler WHERE ingilizce LIKE '" +
txtaranan.Text + "%'", baglantı);
    adp.Fill(ds);
    lstsonuc.DataSource = ds.Tables[0];
    lstsonuc.DisplayMember = "ingilizce";
    txtsonuc.DataBindings.Clear();
    txtsonuc.DataBindings.Add(new System.Windows.Forms.Binding
  ("text", ds.Tables[0], "turkce", true,
System.Windows.Forms.DataSourceUpdateMode.OnPropertyChanged));
    baglantı.Close();
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message);
    baglantı.Close();
  }
 }
}
```

Veri Tabanını Düzenleme

Access veritabanları kullanıldıkça kayıt ekleme ve silme işlemleri sonucunda, kayıt sayısına bağlı olmayan boyut artması meydana gelmektedir. Bu da zamanla veritabanına erişimin ve veritabanı üzerindeki işlemlerin yavaşlamasına neden olacaktır. Bunun için veritabanını onaran, hatalarını gideren ve boyutunu küçülten **Compact Database** işleminin uygulanması gerekmektedir.

Öncelikle projemize **Microsoft Jet and Replication Objects 2.x Library** adlı **dll** dosyasını projemize referans olarak eklemeliyiz. Dll dosyasını **Com** sekmesinden seçerek ya da **Browse** butonu ile ***C:\Program Files\Common Files\System\ado\msjro.dll*** dosyasını seçerek ekleyebiliriz.




```
using System.IO;
public partial class Form1 : Form
{
 private void mdb_duzenle()
 {
  JRO.JetEngine jro = default(JRO.JetEngine);
  jro = new JRO.JetEngine();

  jro.CompactDatabase("Provider= Microsoft.ACE.OLEDB.12.0;Data Source=" +
    Application.StartupPath + "/sozluk.accdb",
    "Provider= Microsoft.ACE.OLEDB.12.0;Data Source=" +
    Application.StartupPath + "/yenisozluk.accdb;" +
    "Jet OLEDB:Engine Type=5");
  File.Delete(Application.StartupPath + "/sozluk.mdb");
  File.Copy(Application.StartupPath + "/yenisozluk.mdb",
    Application.StartupPath + "/sozluk.mdb");
  File.Delete(Application.StartupPath + "/yenisozluk.mdb");
  MessageBox.Show("Veritabanına Bakım ve Düzenleme İşlemi Yapıldı");
 }
}
```

Projemizin iki formdan oluşmakta olduğunu konumuzun başında söylemiştik. İlk formda veritabanındaki kayıtlar üzerinde arama işlemi gerçekleştirilerek İngilizce sözcükler ve buna karşılık gelen Türkçe sözcükler listelenmektedir. Ayrıca ilk formda veritabanı üzerinde düzenleme işlemi gerçekleştirilmektedir. Ancak bu işlem her ayın ilk günü bir defa gerçekleştirilecek. **`frmAyarlar`** isimli ikinci formda ise kayıt ekleme, güncelleme, silme işlemleri yapılmaktadır.

İlk formumuzun kodu:

```
using System;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;
using Microsoft.Win32;
using System.IO;
namespace OleDbDataAdapter_Sozluk
{
 public partial class Form1 : Form
 {
  public Form1()
  {
    InitializeComponent();
  }
  OleDbConnection baglantı = new OleDbConnection
("Provider= Microsoft.ACE.OLEDB.12.0;Data Source = "
+ Application.StartupPath + "/sozluk.accdb");

  private void txtaranan_TextChanged(object sender, EventArgs e)
  {
   try
   {
    DataSet ds = new DataSet();
    if ((baglantı.State == ConnectionState.Closed))
    baglantı.Open();

    OleDbDataAdapter adp = new OleDbDataAdapter
("SELECT * FROM sozcukler WHERE ingilizce LIKE '" + txtaranan.Text +
"%'", baglantı);

    adp.Fill(ds);
    lstsonuc.DataSource = ds.Tables[0];
    lstsonuc.DisplayMember = "ingilizce";
    txtsonuc.DataBindings.Clear();
    txtsonuc.DataBindings.Add(new System.Windows.Forms.Binding
("text", ds.Tables[0], "turkce", true,
System.Windows.Forms.DataSourceUpdateMode.OnPropertyChanged));
    baglantı.Close();
   }
   catch (Exception ex)
```

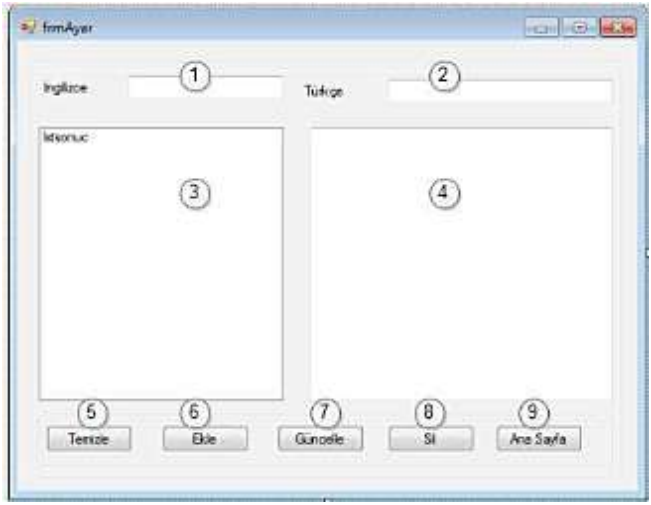
```
   {
    MessageBox.Show(ex.Message);
    baglantı.Close();
   }
  }
  private void btnAyar_Click(object sender, EventArgs e)
  {
   frmAyar frmayar=new frmAyar();
   frmayar.Show();
  }
  private void btnCikis_Click(object sender, EventArgs e)
  {
   Application.Exit();
  }
  private void Form1_Load(object sender, EventArgs e)
  {
   RegistryKey anahtar = Registry.CurrentUser.OpenSubKey
("Software", true);
   if (anahtar.OpenSubKey("Sözlük") != null)
   {
    byte tekrar = Convert.ToByte(anahtar.OpenSubKey
("Sözlük", true).GetValue("CalismaSayisi").ToString());
    anahtar.OpenSubKey("Sözlük", true).SetValue("CalismaSayisi",
tekrar + 1, RegistryValueKind.String);
    if (DateTime.Now.ToString().Substring(0, 2) == "01")
    {
     RegistryKey duzenkey = Registry.CurrentUser.OpenSubKey
("Software", true).OpenSubKey("Sözlük", true);
     if (duzenkey.GetValue("DuzenlemeZamanı") == null)
     {
      mdb_duzenle();
      duzenkey.SetValue("DuzenlemeZamanı",
DateTime.Now.ToString().Substring(1, 10));
     }
    }
    else
    {
     RegistryKey duzenkey = Registry.CurrentUser.OpenSubKey
("Software", true).OpenSubKey("Sözlük", true);
     if (duzenkey.GetValue("DuzenlemeZamanı") != null)
```

```
      {
       duzenkey.DeleteValue("DuzenlemeZamanı");
      }
     }
    }
    else
    {
     anahtar.CreateSubKey("Sözlük");
     anahtar.OpenSubKey("Sözlük", true).SetValue
("CalismaSayisi", 1, RegistryValueKind.String);
     MessageBox.Show("Program Bu Bilgisayarda İlk Defa Çalışıyor");
    }
    anahtar.Close();
   }

   private void mdb_duzenle()
   {
    JRO.JetEngine jro = default(JRO.JetEngine);
    jro = new JRO.JetEngine();

    jro.CompactDatabase
("Provider= Microsoft.ACE.OLEDB.12.0;Data Source=" +
Application.StartupPath + "/sozluk.accdb ",
"Provider= Microsoft.ACE.OLEDB.12.0;Data Source=" +
Application.StartupPath + "/yenisozluk.accdb;" +
"Jet OLEDB:Engine Type=5");
    File.Delete(Application.StartupPath + "/sozluk.accdb");
    File.Copy(Application.StartupPath + "/yenisozluk.accdb",
Application.StartupPath + "/sozluk.accdb");
    File.Delete(Application.StartupPath + "/yenisozluk.mdb");
    MessageBox.Show("Veritabanına Bakım ve Düzenleme İşlemi Yapıldı");
   }
  }
}
```

Kayıt ekleme, güncelleme, silme gibi işlemleri yaptığımız **frmAyar** isimli ikinci formumuzun tasarımı aşağıdaki gibi olacak:



(1) `txtingilizce`

(2) `txtturkce`

(3) `lstsonuc`

(4) `txtsonuc`

(5) `btnTemizle`

(6) `btnEkle`

(7) `btnGuncelle`

(8) `btnSil`

(9) `btnAnasayfa`

Formumuzun kodu:

```
using System;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;
namespace OleDbDataAdapter_Sozluk
{
 public partial class frmAyar : Form
 {
  public frmAyar()
  {
   InitializeComponent();
  }
  OleDbConnection baglantı = new OleDbConnection
 ("Provider= Microsoft.ACE.OLEDB.12.0;Data Source=" +
 Application.StartupPath + "/sozluk.accdb");

  private void frmAyar_Load(object sender, EventArgs e)
  {
   try
   {
```

```
        DataSet ds = new DataSet();
        if ((baglantı.State == ConnectionState.Closed))
          baglantı.Open();

        OleDbDataAdapter adp = new OleDbDataAdapter
    ("SELECT * FROM sozcukler", baglantı);

        adp.Fill(ds);
        lstsonuc.DataSource = ds.Tables[0];
        lstsonuc.DisplayMember = "ingilizce";
        txtsonuc.DataBindings.Clear();
        txtsonuc.DataBindings.Add(new System.Windows.
    Forms.Binding("text", ds.Tables[0], "turkce", true,
    System.Windows.Forms.DataSourceUpdateMode.OnPropertyChanged));
        baglantı.Close();
        lstsonuc.SelectedIndex = 0;
        txtingilizce.Text = lstsonuc.Text;
        txtturkce.Text = txtsonuc.Text;
       }
       catch (Exception ex)
       {
        MessageBox.Show(ex.Message);
        baglantı.Close();
       }
      }

      private void lstsonuc_SelectedIndexChanged(object sender, EventArgs e)
      {
       lstsonuc.SelectedItem.ToString();
       txtingilizce.Text = lstsonuc.Text;
       txtturkce.Text = txtsonuc.Text;
      }

      private void btnTemizle_Click(object sender, EventArgs e)
      {
       txtingilizce.Clear();
       txtturkce.Clear();
      }

      private void btnEkle_Click(object sender, EventArgs e)
      {
       if (string.IsNullOrEmpty(txtingilizce.Text) |
      string.IsNullOrEmpty(txtturkce.Text))
```

```
    {
     MessageBox.Show("Lütfen Bütün Alanları Doldurunuz", "Hata");
     return;
    }
    try
    {
     DataSet ds = new DataSet();
     baglantı.Open();

     ds.Clear();

     OleDbDataAdapter adp = new OleDbDataAdapter
("INSERT INTO sozcukler (ingilizce,turkce) VALUES ('" +
txtingilizce.Text + "','" + txtturkce.Text + "')", baglantı);
     adp.Fill(ds);

     baglantı.Close();
     MessageBox.Show("Kayıt Eklendi", "Bilgi");
     txtingilizce.Text = "";
     txtturkce.Text = "";
    }
    catch (Exception ex)
    {
     MessageBox.Show(ex.Message);
     baglantı.Close();
    }
   }
   private void btnGuncelle_Click(object sender, EventArgs e)
   {
    try
    {
     DataSet ds = new DataSet();
     baglantı.Open();
     ds.Clear();
  OleDbDataAdapter adp = new OleDbDataAdapter
    ("UPDATE sozcukler SET ingilizce='" + txtingilizce.Text +
    "', turkce='" + txtturkce.Text + "' WHERE ingilizce='" +
    txtingilizce.Text + "'", baglantı);
     adp.Fill(ds);

     baglantı.Close();
     MessageBox.Show("Kayıt Güncellendi.", "Bilgi");
```

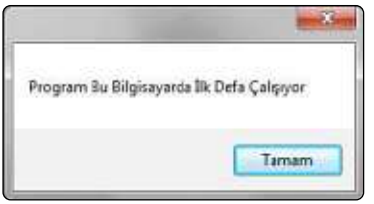
```
    }
    catch (Exception ex)
    {
     MessageBox.Show(ex.Message);
     baglantı.Close();
    }
   }
   private void btnSil_Click(object sender, EventArgs e)
   {
    try
    {
     DataSet ds = new DataSet();
     baglantı.Open();
     ds.Clear();

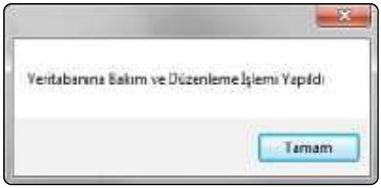
     OleDbDataAdapter adp = new OleDbDataAdapter
      ("DELETE FROM sozcukler WHERE ingilizce='" +
      txtingilizce.Text + "'", baglantı);
     adp.Fill(ds);

     baglantı.Close();
     MessageBox.Show("Kayıt Silindi", "Bilgi");
    }
    catch (Exception ex)
    {
     MessageBox.Show(ex.Message);
     baglantı.Close();
    }
   }

   private void btnAnasayfa_Click(object sender, EventArgs e)
   {
    this.Close();
   }
  }
}
```

Programın ilk çalıştırılışı esnasında ekrana "*Program Bu Bilgisayarda İlk Defa Çalışıyor*" şeklinde bir mesaj penceresi gelecektir.



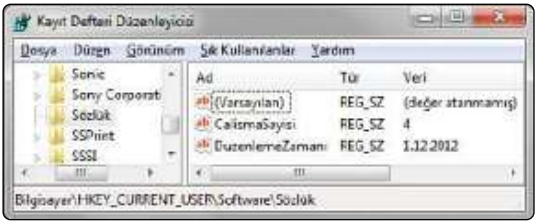


Ayrıca her ayın ilk gününde veri tabanı düzenlenip ekrana "*Veritabanına Bakım ve Düzenleme İşlemi Yapıldı*" şeklinde bir mesaj penceresi gelecektir.

Veritabanının düzenleme öncesi boyutu **612 kb** iken, düzenleme işlemi sonrası **304 kb**'a inmiştir.

sozluk - Kopya.accdb	07.11.2012 07:02	612 KB
sozluk.accdb	01.12.2012 18:20	304 KB

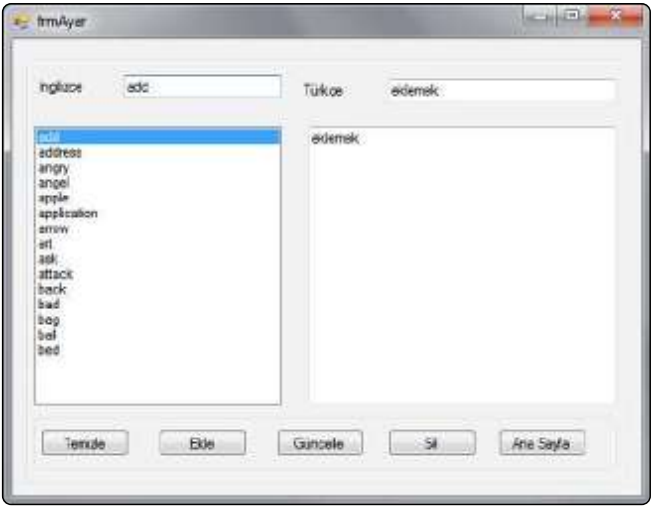
Registry içerisine baktığımızda düzenleme zamanı, çalışma sayısı bilgilerinin kaydedildiğini görürüz.



İlk formumuzda herhangi bir kelimenin ilk harfini yazdığımızda, o harfe dair sonuçların listelendiğini görürüz.



Ayarlar butonuna tıklayarak kayıt ekleme, düzenleme ve silme işlemlerini gerçekleştireceğimiz **`frmAyar`** adındaki formumuzu görüntüleriz.

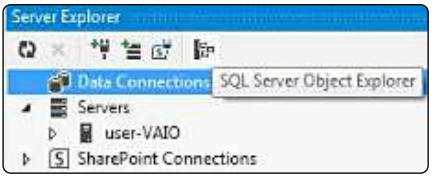


SQL Veri Tabanı İşlemleri

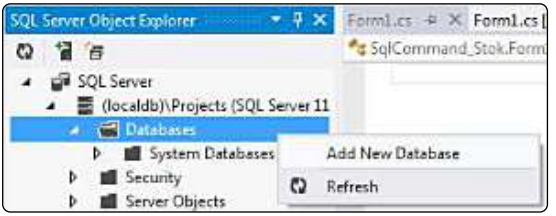
SQL veritabanlarına erişim için kullanılabilecek veritabanı nesneleri şunlardır:

- **SqlConnection nesnesi**
- **SqlCommand nesnesi**
- **SqlDataAdapter**
- **DataReader**
- **DataSet**

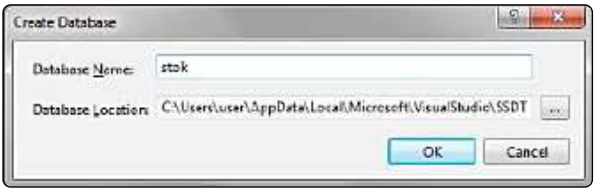
Veritabanı işlemlerimizde kullanacağımız örnek bir SQL veritabanı tasarlayalım. Veritabanımız bir stok takip uygulamasının veritabanı olacak. **SQL Server Object Explorer** kullanarak SQL veritabanı oluşturup, içerisinde düzenleme ve kayıt ekleme işlemleri yapabilirsiniz. **Server Explorer** içerisinde **SQL Server Object Explorer** butonuna tıklayalım.



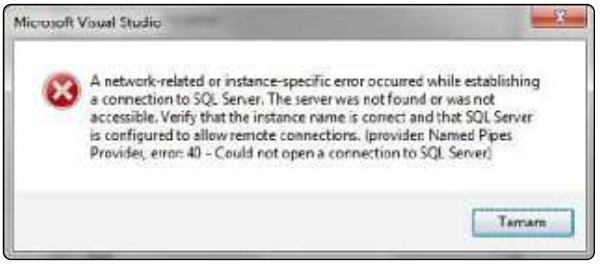
SQL Server Object Explorer içerisinde yerel veritabanları `localdb` başlığı altında yer almaktadır. Biz de SQL veritabanını yerel olarak oluşturacağımız için `localdb` başlığı altındaki **Databases** ismine farenin sağ tuşu ile tıklayarak, gelen menüden **Add New Database** seçeneğini seçelim.



Karşımıza gelen pencerede veritabanı ismini yazalım.



NOT Eğer SQL veritabanı oluşturma esnasında remote connection hatası meydana gelirse, nedeni Sql server içerisinde remote connection izni verilmediğinden olabilir.



Bu izni vermek için **Başlat / Programlar / Microsoft SQL Server 20XX / Configuration Tools / SQL Server Configuration Manager** aracını açıyoruz.

NOT Bu programı komut satırından "**SQLServerManager14.msc**" dosyasını çalıştırarak da açabilirsiniz. 14 rakamı SQLServer versiyonuna göre değişiklik göstermektedir. Örneğin **SQLServer2016** için bu rakam **13** iken, **SQLServer2017** için **14**'tür. Dosya yolu "*C:\Windows\SysWOW64\SQLServerManager14.msc*" biçimindedir. "*Sql Server Manager*" penceresini görüntülemek için bilgisayarınızda SQLServer uygulaması yüklü olmalıdır.

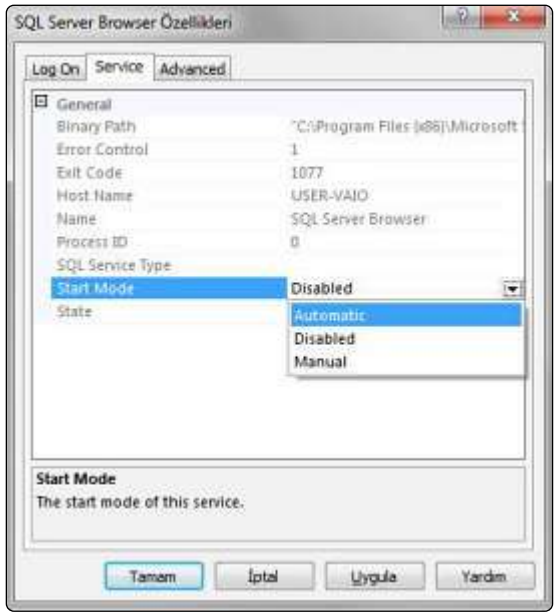
Görüldüğü gibi **Sql Server Browser** çalışma modunda (**Running**) değildir. Bunu çalıştıracağız.



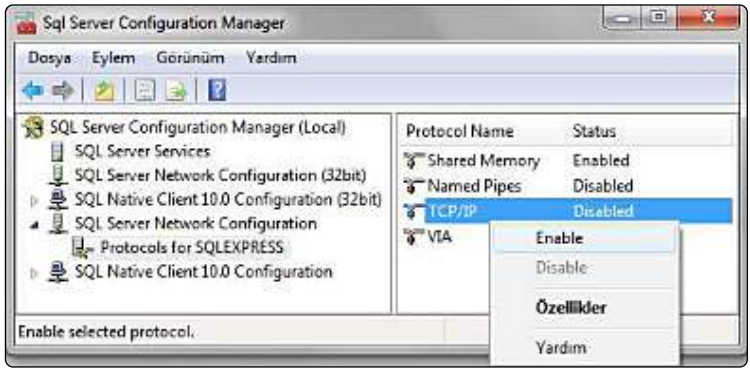
SQL Server Browser ismine çift tıklayarak **Özellikler** penceresini görüntülüyoruz. **Log On** sekmesinde **Built-in account** seçeneğinden **Local Service** seçeneğini seçiyoruz.



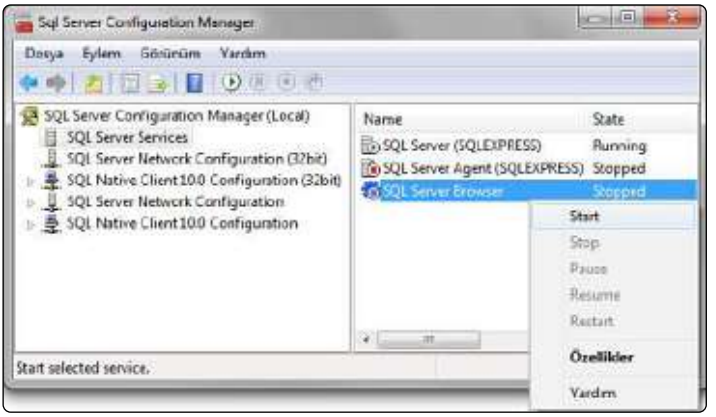
Service sekmesinde **Disabled** modunda olan **Start Mode** seçeneğini **Automatic** olarak ayarlıyoruz.



Özellikler penceresini kapatarak **Configuration Manager** penceresinde **SQL Server Network Configuration** seçeneğine tıklayarak, **Protocols for SQLEXPRESS** seçeneğini görüntülüyoruz. Bu seçeneğe tıklayarak sağ panelde **Disabled** konumunda olan **TCP/IP** seçeneğini **Enable** konumuna getiriyoruz.



SQL Server Browser'a farenin sağ tuşu ile tıklayarak, **Start** seçeneğini seçip **SQL Server Browser**'ı başlatıyoruz.

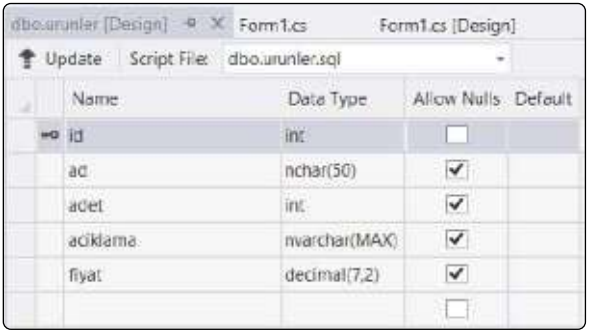


Bu işlemden sonra **Sql Server Configuration Manager** penceresini kapatarak SQL veritabanı oluşturma işlemini yeniden başlattığımızda, hata meydana gelmeden boş bir Sql veritabanı oluşturulacaktır.

SQL Server Object Explorer içerisinde Databases bölümünde oluşturmuş olduğumuz veritabanı ismine tıklıyoruz. Açılan listede **Tables** seçeneğine farenin sağ tuşu ile tıklayarak, **Add New Table** seçeneğini tıklayıp, veritabanımızda tablo ve alanları oluşturacağız.



Tablomuzdaki alanları aşağıdaki gibi oluşturuyoruz.

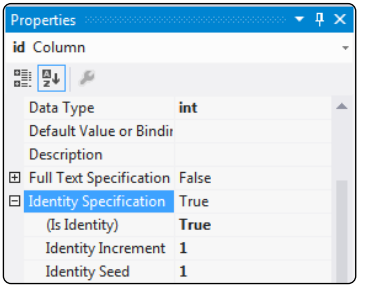


`id` alanında **Allow Nulls** seçeneğinin işaretini kaldırarak boş değer girilmesini engelliyoruz. Diğer alanları da işaretleyerek boş değer giriş izni veriyoruz.

`id` alanındaki değerlerin otomatik olarak arttırılması için **Properties** penceresinde **Identity Specification** seçeneğini **Yes** olarak seçiyoruz.

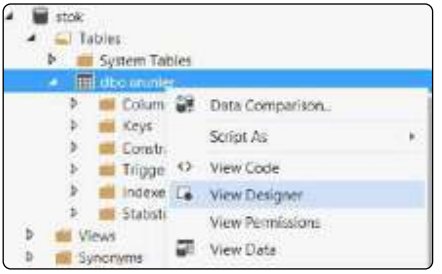
Identity Increment seçeneği artış değerinin kaçar kaçar olacağını belirtir.

Identity Seed seçeneği ise, artış işleminin hangi rakamdan itibaren başlatılacağını belirtir.



`id` alanına farenin sağ tuşu ile tıklayarak, **Set Primary Key**'i seçip, birincil anahtar olarak belirlenmesini sağlıyoruz.

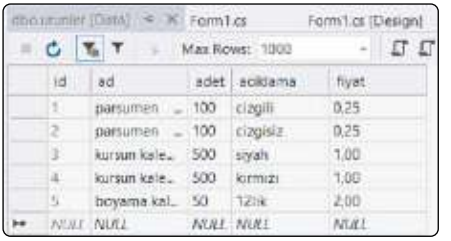
Tablomuzu kayıt edip `urunler` ismini veriyoruz. Tüm işlemler bittikten sonra tabloyu güncellemek için **Update** butonunu tıklıyoruz.



Alanlar üzerinde daha sonra düzenleme yapmak için tablo adına farenin sağ tuşu ile tıklayarak **View Designer** seçeneğini kullanabiliriz.

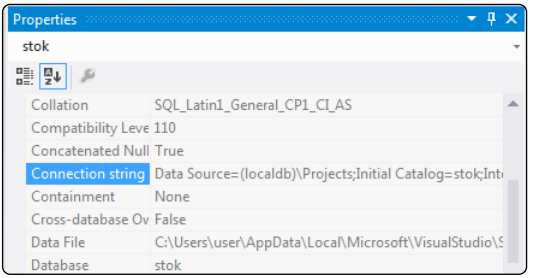
Alanlara kayıt girmek için tablo adına farenin sağ tuşu ile tıklayarak, **View Data** seçeneğini kullanabiliriz.

`SQLConnection`, `SQLCommand` ve `SQLDataAdapter` nesnelerini kodumuzda kullanabilmek için öncelikle `System.Data` ve `System.Data.SqlClient` sınıflarını projemizde dahil etmemiz gerekmektedir.



Daha sonra yeni bir **Connection String** oluşturmamız gerekmektedir.

Oluşturmuş olduğumuz SQL veritabanı nesnesinin bağlantı cümlesini öğrenmek için **SQL Server Object Explorer** içerisindeki veri tabanı ismine tıklayıp, **Properties** penceresinde **Connection String** özelliğine bakabiliriz.



Stok örneği için kullanacağımız bağlantı cümlesi şu şekildedir:

```
SqlConnection baglantı = new SqlConnection("Data Source=(localdb)\\
Projects; Initial Catalog=stok; Integrated Security=True; Connect
Timeout=30; Encrypt=False; TrustServerCertificate=False");
```

SQLCOMMAND

SQL veritabanı işlemleri ile ilgili kullanılan sorguları çalıştırmak için kullanılır. `OleDbCommand` nesnesinde olduğu gibi komutları yürütmek için 4 çeşit **execute** metoduna sahiptir:

- **`ExecuteReader`**: Yürütülen komut sonrasında geriye `DataReader` nesnesi döndürür.
- **`ExecuteNonQuery`**: Ekleme, silme, güncelleme gibi sorgulardan etkilenen satır sayısını geriye döndürür.
- **`ExecuteScalar`**: Bu metot, geriye tek bir değer döndüren sorgular için kullanılır. Bu değer kayıt sayısı, bir alanın değeri gibi değerler olabilir.
- **`ExecuteXMLReader`**: Yürütülen komut sonrasında geriye `XML Reader` nesnesi döndürür.

Oluşturmuş olduğumuz stok veritabanındaki ürünler tablosunda listeleme, ekleme, günceleme ve silme işlemlerini `SqlCommand` kullanarak gerçekleştirecek örnek bir uygulama yaparak, gerçekleştirilen veritabanı işlemlerini sırasıyla inceleyelim.

Formumuzun tasarımı aşağıdaki gibi olacak:



(1) `txt_ad`

(2) `txt_fiyat`

(3) `btnEkle`

(4) `btnGuncelle`

(5) `btnSil`

(6) `dataGridView1`

(7) `lblrastgeleurun`

(8) `lblKayıtSayısı`

Kayıt Listeleme

Veri tabanındaki kayıtları listelemek için **`select`** sorgusunu **`SqlCommand`** nesnesinin **`ExecuteReader()`** metodu ile birlikte kullanıyoruz.

```
using System.Data;
using System.Data.SqlClient;
public partial class Form1 : Form
{
 SqlConnection baglantı = new SqlConnection
   ("Data Source=(localdb)\\Projects; Initial Catalog=stok;
   Integrated Security=True; Connect Timeout=30; Encrypt=False;
   TrustServerCertificate=False");
 SqlCommand komut = new SqlCommand();
 SqlDataAdapter adaptor = new SqlDataAdapter();

 private void Form1_Load(object sender, EventArgs e)
 {
  DataSet ds = new DataSet();
  if ((baglantı.State == ConnectionState.Closed)) baglantı.Open();
  komut = new SqlCommand("SELECT * FROM urunler ", baglantı);
  SqlDataReader reader = komut.ExecuteReader();
  reader.Read();

  if (reader.HasRows)
  {
   txt_ad.Text = reader["ad"].ToString();
   txt_fiyat.Text = reader.GetDecimal(4).ToString();
  }
```

```
  reader.Close();
  adaptor.SelectCommand = new SqlCommand
   ("select ad,fiyat from urunler ", baglantı);
  adaptor.Fill(ds);
  dataGridView1.DataSource = ds.Tables[0];
  baglantı.Close();
 }
}
```

TextBox nesnelerinin veritabanı ile bağlantısını kurmak için **`SQLCommand`** nesnesinin **`ExecuteReader`** metodu ile veritabanından verileri çekiyoruz. **`Reader.Read()`** metodu ile okuma işlemini başlatıyoruz. **`Reader.HasRows`** metodu ile okunan değerin boş olup olmadığının kontrolü yapılır. **`Reader["ad"]`** ve **`GetDecimal(4)`** metodlarının her ikisi ile birden okunan verilerin textBox içerisine aktarımı yapılır.

Kayıt Ekleme

Yeni bir kayıt eklemek için **`insert`** sorgusunu kullanıyoruz. Ayrıca veritabanına yeni kayıt eklemek için **`SqlCommand`** nesnesinin **`ExecuteNonQuery()`** metodunu kullanıyoruz.

```
using System.Data;
using System.Data.SqlClient;
public partial class Form1 : Form
{
 SqlConnection baglantı = new SqlConnection
   ("Data Source=(localdb)\\Projects;Initial Catalog=stok;
   Integrated Security = True; Connect Timeout = 30; Encrypt
   = False; TrustServerCertificate=False");
 SqlCommand komut = new SqlCommand();
 SqlDataAdapter adaptor = new SqlDataAdapter();

 private void btnEkle_Click(object sender, EventArgs e)
 {
  try
  {
   DataSet ds = new DataSet();
   if (baglantı.State == ConnectionState.Closed) baglantı.Open();
   ds.Clear();
```

```
   SqlCommand komut = new SqlCommand
     ("INSERT INTO urunler (ad, fiyat) VALUES ('" +
     txt_ad.Text + "','" + txt_fiyat.Text + "')", baglantı);
   komut.ExecuteNonQuery();
   dataGridView1.Update();
   baglantı.Close();
   MessageBox.Show("Kayıt Eklendi!");
  }
  catch (Exception ex)
  {
   MessageBox.Show(ex.Message);
   baglantı.Close();
  }
 }
}
```

Kayıt Güncelleme

Kayıtlar üzerinde güncelleme yapmak için **update** sorgusunu kullanıyoruz. Ayrıca veritabanındaki kayıtları güncellemek için **SqlCommand** nesnesinin **ExecuteNonQuery()** metodunu kullanıyoruz.

```
using System.Data;
using System.Data.SqlClient;
public partial class Form1 : Form
{
 SqlConnection baglantı = new SqlConnection
   ("Data Source=(localdb)\\Projects;Initial Catalog = stok;
   Integrated Security = True; Connect Timeout = 30; Encrypt
   = False; TrustServerCertificate = False");
 SqlCommand komut = new SqlCommand();
 SqlDataAdapter adaptor = new SqlDataAdapter();

 private void btnGuncelle_Click(object sender, EventArgs e)
 {
  try
  {
   DataSet ds = new DataSet();
   if (baglantı.State == ConnectionState.Closed) baglantı.Open();
   ds.Clear();
```

```
    SqlCommand komut = new SqlCommand
   ("UPDATE urunler SET ad = '" + txt_ad.Text + "', fiyat='" +
   txt_fiyat.Text + "' WHERE ad = '" + txt_ad.Text +
    "'", baglantı);
    komut.ExecuteNonQuery();
    dataGridView1.Update();
    baglantı.Close();
    MessageBox.Show("Kayıt Güncellendi!");
   }
   catch (Exception ex)
   {
    MessageBox.Show(ex.Message);
    baglantı.Close();
   }
  }
}
```

Kayıt Silme

Kayıtları silmek için **delete** sorgusunu kullanıyoruz. Ayrıca veritabanındaki kayıtları silmek için **SqlCommand** nesnesinin **ExecuteNonQuery()** metodunu kullanıyoruz.

```
using System.Data;
using System.Data.SqlClient;
public partial class Form1 : Form
{
SqlConnection baglantı = new SqlConnection
   ("Data Source=(localdb)\\Projects;Initial Catalog = stok;
   Integrated Security = True; Connect Timeout = 30; Encrypt
   = False; TrustServerCertificate = False");
 SqlCommand komut = new SqlCommand();
 SqlDataAdapter adaptor = new SqlDataAdapter();

 private void btnSil_Click(object sender, EventArgs e)
 {
  try
  {
   DataSet ds = new DataSet();
   if (baglantı.State == ConnectionState.Closed) baglantı.Open();
   ds.Clear();
```

```
   SqlCommand komut = new SqlCommand
  ("DELETE FROM urunler WHERE ad='" + txt_ad.Text + "'", baglantı);
   komut.ExecuteNonQuery();
   dataGridView1.Update();
   dataGridView1.Refresh();
   baglantı.Close();
   MessageBox.Show("Kayıt Silindi!");
  }
  catch (Exception ex)
  {
   MessageBox.Show(ex.Message);
   baglantı.Close();
  }
 }
}
```

Kayıt ekleme, güncelleme ve silme işlemlerine dair kodlarımızı birleştirelim.

Uygulamamızın kodu:

```
using System;
using System.Windows.Forms;
using System.Data;
using System.Data.SqlClient;
namespace SqlCommand_Stok
{
 public partial class Form1 : Form
 {
  public Form1()
  {
    InitializeComponent();
  }
  SqlConnection baglantı = new SqlConnection
   ("Data Source=(localdb)\\Projects;Initial Catalog=stok;
   Integrated Security=True;Connect Timeout=30; Encrypt
   =False;TrustServerCertificate=False");
  SqlCommand komut = new SqlCommand();
  SqlDataAdapter adaptor = new SqlDataAdapter();
  DataSet ds = new DataSet();
```

```
private void Form1_Load(object sender, EventArgs e)
{
 if ((baglantı.State == ConnectionState.Closed)) baglantı.Open();

 komut = new SqlCommand("SELECT * FROM urunler ", baglantı);

 SqlDataReader reader = komut.ExecuteReader();
 reader.Read();
 if (reader.HasRows)
 {
  txt_ad.Text = reader["ad"].ToString();
  txt_fiyat.Text = reader.GetDecimal(4).ToString();
 }
 reader.Close();

 adaptor.SelectCommand = new SqlCommand
("SELECT ad,fiyat FROM urunler ", baglantı);
 adaptor.Fill(ds);
 dataGridView1.DataSource = ds.Tables[0];

 komut = new SqlCommand
("SELECT TOP 1 ad FROM urunler ORDER BY NEWID()", baglantı);
 lblrastgeleurun.Text = "Günün indirimli ürünü " +
   komut.ExecuteScalar();

 komut = new SqlCommand("SELECT COUNT(*) FROM urunler", baglantı);
 lblKayıtSayısı.Text = "Veritabanında " +
  (int)komut.ExecuteScalar() + " tane ürün var.";
 baglantı.Close();
}

private void dataGridView1_CellClick(object sender,
DataGridViewCellEventArgs e)
{
 int satirno;
 satirno = e.RowIndex;

 txt_ad.Text = dataGridView1.Rows[satirno].Cells[0].Value.ToString();
 txt_fiyat.Text = dataGridView1.Rows[satirno].Cells[1].Value.ToString();
}

private void btnEkle_Click(object sender, EventArgs e)
{
 try
 {
```

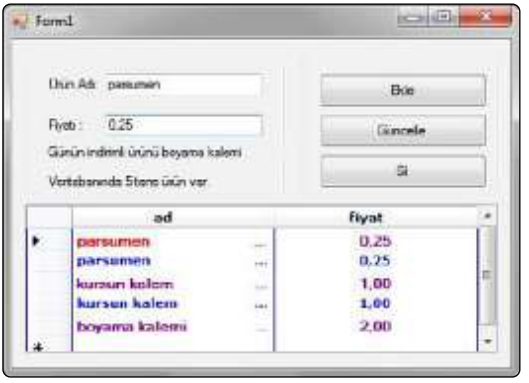
```
    DataSet ds = new DataSet();
    if (baglantı.State == ConnectionState.Closed) baglantı.Open();
    ds.Clear();

    SqlCommand komut = new SqlCommand
   ("INSERT INTO urunler (ad, fiyat) VALUES (‘" + txt_ad.Text +
"’,’" + txt_fiyat.Text + "’)", baglantı);
    komut.ExecuteNonQuery();
    dataGridView1.Update();
    baglantı.Close();
    MessageBox.Show("Kayıt Eklendi!");
   }
   catch (Exception ex)
   {
    MessageBox.Show(ex.Message);
    baglantı.Close();
   }
  }
  private void btnGuncelle_Click(object sender, EventArgs e)
  {
   try
   {
    DataSet ds = new DataSet();
    if (baglantı.State == ConnectionState.Closed) baglantı.Open();
    ds.Clear();

    SqlCommand komut = new SqlCommand
   ("UPDATE urunler SET ad=’" + txt_ad.Text + "’,
   fiyat=’" + txt_fiyat.Text + "’ WHERE ad=’" +
   txt_ad.Text + "’", baglantı);
    komut.ExecuteNonQuery();
    dataGridView1.Update();
    baglantı.Close();
    MessageBox.Show("Kayıt Güncellendi!");
   }
   catch (Exception ex)
   {
    MessageBox.Show(ex.Message);
    baglantı.Close();
   }
  }
```

```
  private void btnSil_Click(object sender, EventArgs e)
  {
   try
   {
    DataSet ds = new DataSet();
    if (baglantı.State == ConnectionState.Closed)
      baglantı.Open();
    ds.Clear();

    SqlCommand komut = new SqlCommand
   ("DELETE FROM urunler WHERE ad='" + txt_ad.Text + "'", baglantı);
    komut.ExecuteNonQuery();
    dataGridView1.Update();
    dataGridView1.Refresh();
    baglantı.Close();
    MessageBox.Show("Kayıt Silindi!");
   }
   catch (Exception ex)
   {
    MessageBox.Show(ex.Message);
    baglantı.Close();
   }
  }
 }
}
```



Formun `Load` olayı esnasında veritabanındaki kayıtları çekerek **DataGridView** içerisine ve **TextBox** nesneleri içerisine veritabanındaki değerleri ekliyoruz.

Kayıtları okumak için **Select** sorgusunu kullanıyoruz. **Select** sorgusu veritabanındaki kayıtları çekmek amacı ile kullanılır. Kodumuzda…

```
SELECT * FROM urunler
```

… sorgusu ile urunler tablosundaki bütün kayıtları çektik. * karakteri ile veri tabanındaki bütün kayıtlar çekilir.

Veritabanındaki kayıtları çekmek için **SqlCommand** nesnesinin **ExecuteReader()** metodunu kullanıyoruz.

Ayrıca formun **Load** olayı esnasında, veritabanında bulunan kayıt sayısını bularak görüntülüyoruz. Bu işlem için…

```
SELECT COUNT(*) From urunler
```

… sorgusunu kullanıyoruz. Bu şekilde **urunler** tablosundaki kayıt sayısını buluyoruz. Geriye tek bir değer dönmesi nedeni ile de **SqlCommand** nesnesinin **ExecuteScalar()** metodunu kullanıyoruz.

Yine formun **Load** olayı esnasında rastgele bir ürünü seçerek, günün indirimli ürünü olarak görüntülüyoruz. Bu işlem için…

```
SELECT TOP 1 ad FROM urunler ORDER BY NEWID()
```

… sorgusunu kullanıyoruz. Bu şekilde **urunler** tablosundaki **ad** alanından rastgele bir değer çekiyoruz. Geriye tek bir değer dönmesi nedeni ile de **SqlCommand** nesnesinin **ExecuteScalar()** metodunu kullanıyoruz.

Kayıt eklemek için kodumuzda…

```
INSERT INTO urunler (ad, fiyat) Values ('" + txt_ad.Text + "',"
+ txt_fiyat.Text + "')
```

… sorgusu ile **urunler** tablosundaki **ad** ve **fiyat** adlı alanlara **txt_ad** ve **txt_fiyat** adlı text kutularında yazılı olan değerleri aktardık.

Ayrıca veritabanına yeni kayıt eklemek için **SqlCommand** nesnesinin **ExecuteNonQuery()** metodunu kullandık.

Kayıtlar üzerinde güncelleme yapmak için kodumuzda…

```
UPDATE urunler SET ad='" + txt_ad.Text + "', fiyat='" + txt_fiyat.Text
+ "' WHERE ad='" + txt_ad.Text + "'
```

… sorgusu ile **urunler** tablosundaki **ad** ve **fiyat** adlı alanlardaki veriler; eğer **ad** alanındaki veri **txt_ad** adlı text kutusundaki veriye eşit ise, **txt_ad** ve **txt_fiyat** adlı text kutularında yazılı olan yeni değerlere göre güncellenir.

Ayrıca veritabanındaki kayıtları güncellemek için **SqlCommand** nesnesinin **ExecuteNonQuery()** metodunu kullandık.

Kayıtları silmek için kodumuzda…

```
DELETE FROM urunler WHERE ad='" + txt_ad.Text + "'
```

… sorgusu ile **urunler** tablosundaki **ad** alanındaki veri, **txt_ad** adlı text kutusundaki değere eşit ise, bu satırdaki veri silinir. Ayrıca veritabanındaki kayıtları silmek için **SqlCommand** nesnesinin **ExecuteNonQuery()** metodunu kullandık.

SQLDATAADAPTER

Veritabanı bağlantısı olmasa bile veriler üzerinde işlemler gerçekleştirilebilir. Veritabanına bağlanıp veriler çekilir. Daha sonra bağlantı kesilse bile daha önceden çekilmiş veriler üzerinde işlem yapılmaktadır. Üzerinde değişiklik yapılmış veriler daha sonra veritabanına gönderilir.

SQLDataAdapter nesnesini kodumuzda kullanabilmek için öncelikle **System.Data** ve **System.Data.SqlClient** sınıflarını projemizde dahil etmemiz gerekmektedir.

SqlDataAdapter nesnesini **SqlCommand** sınıfı ile birlikte kullanarak 4 farklı komut biçimi ile seçme, ekleme, güncelleme ve silme işlemlerini gerçekleştirebiliriz.

- **SelectCommand:** Kayıtları listelemek amacı ile kullanılır.
- **InsertCommand:** Kayıt eklemek için kullanılır propertydir.
- **UpdateCommand:** Kayıtları güncellemek amacı ile kullanılır.
- **DeleteCommand:** Kayıt silmek amacı ile kullanılır.

Oluşturmuş olduğumuz stok veritabanındaki **urunler** tablosunda listeleme, ekleme, güncelleme ve silme işlemlerini **SqlDataAdapter** kullanarak gerçekleştirecek örnek bir uygulama yaparak, gerçekleştirilen veritabanı işlemlerini sırasıyla inceleyelim.

Formumuzun tasarımı aşağıdaki gibi olacak:



(1) txt_ad

(2) txt_fiyat

(3) btnEkle

(4) btnGuncelle

(5) btnSil

(6) dataGridView1

(7) lblenyuksekfiyat

(8) lblendusukfiyat

Kayıt Listeleme

Veritabanındaki kayıtları listelemek için **select** sorgusunu kullanıyoruz. **Select** sorgusu bir tablodaki belirtilen bir alandaki bir ya da daha çok veriyi çekmek amacı ile kullanılır.

```
using System.Data;
using System.Data.SqlClient;
public partial class Form1 : Form
{
  SqlConnection baglantı = new SqlConnection
   ("Data Source=(localdb)\\Projects;Initial Catalog=stok;
   Integrated Security = True; Connect Timeout = 30;
 SqlCommand komut = new SqlCommand();
 SqlDataAdapter adaptor = new SqlDataAdapter();

 private void Form1_Load(object sender, EventArgs e)
 {
  try
  {
   adaptor = new SqlDataAdapter("SELECT ad,fiyat FROM urunler", baglantı);
   ds = new DataSet();
   adaptor.Fill(ds);
   dataGridView1.DataSource = ds.Tables[0];
```

```
    txt_ad.DataBindings.Clear();
    txt_ad.DataBindings.Add(new System.Windows.Forms.
      Binding("text", ds.Tables[0], "ad", true,
      System.Windows.Forms.DataSourceUpdateMode.OnPropertyChanged));
    txt_fiyat.DataBindings.Clear();
    txt_fiyat.DataBindings.Add(new System.Windows.Forms.
      Binding("text", ds.Tables[0], "fiyat", true,
      System.Windows.Forms.DataSourceUpdateMode.OnPropertyChanged));
   }
   catch (Exception ex)
   {
    MessageBox.Show(ex.Message);
   }
  }
}
```

Kayıtların listelenmesi esnasında form üzerindeki kontrollerin veritabanı bağlantısını kurduk.

Yeni bir **DataSet** tanımlayarak **DataAdapter** nesnesi içerisindeki verileri **DataSet** içerisine aktarıyoruz. **DataSet** veritabanından bağlantısız çalışmaktadır. Bir defa bağlanıp veriler çekildikten sonra veritabanı ile bağlantı kesilir. **DataAdapter DataSet** ile veritabanında aracılık vazifesi görmektedir. Veritabanındaki verileri **DataSet**'e aktarmak, **DataSet** içerisinde güncellenmiş olan verileri de veritabanına aktarmak amacı ile kullanılır.

ListBox nesnesinin veri tabanı ile bağlantısını kurmak için **`DataSource`** özelliğine **DataSet** içerisindeki verilerden belirtilen tablo ismini çekiyoruz. **`Tables[0]`** kullanımı ile ilk sıradaki tablo bağlantısı yapılmıştır. Bunun yerine **`Tables["urunler"]`** şeklinde de kullanılabilir

TextBox nesnesinin veritabanındaki ilgili alanla bağlantısını kurmak için **`DataBindings.Add`** metodu kullanılabilir.

Örneğimizde görüldüğü gibi **`txt_ad`** ve **`txt_fiyat`** adlı **TextBox** nesnelerini text veri biçimi ile 0 nolu tablonun (**`urunler`** tablosuna) **`ad`** ve **`fiyat`** adlı alanlarına bağladık.

Kayıt Ekleme

Yeni bir kayıt eklemek için **insert** sorgusunu kullanıyoruz. **Insert** sorgusunu bir **SqlCommand** nesnesi ile tanımlayarak oluşturacağımız bir **SqlDataAdapter** nesnesinin **SelectCommand** özelliğine aktarırız.

```
using System.Data;
using System.Data.SqlClient;
public partial class Form1 : Form
{
  SqlConnection baglantı = new SqlConnection
   ("Data Source=(localdb)\\Projects;Initial Catalog=stok;
   Integrated Security=True;Connect Timeout=30;
 SqlCommand komut = new SqlCommand();
 SqlDataAdapter adaptor = new SqlDataAdapter();
 DataSet ds = new DataSet();

 private void btnEkle_Click(object sender, EventArgs e)
 {
  try
  {
   SqlDataAdapter adaptor = new SqlDataAdapter();
   string insertSQL = "INSERT INTO urunler (ad, fiyat) " +
    "VALUES (@ad, @fiyat)";

   SqlCommand insertCMD = new SqlCommand(insertSQL, baglantı);
   adaptor.SelectCommand = insertCMD;

   insertCMD.Parameters.Add("@ad", SqlDbType.NVarChar, 15).
    Value = txt_ad.Text;
   insertCMD.Parameters.Add("@fiyat", SqlDbType.Float, 15).
    Value = txt_fiyat.Text;

   adaptor.Fill(ds, "urunler");
   MessageBox.Show("Kayıt eklendi!");
  }
  catch (Exception ex)
  {
   MessageBox.Show(ex.Message);
  }
 }
}
```

Kayıt eklemek için kodumuzda…

```
INSERT INTO urunler (ad, fiyat) " + "VALUES (@ad, @fiyat)
```

… sorgusunu kullandık. Bu şekilde **urunler** tablosundaki **ad** ve **fiyat** alanlarına karşılık **ad** ve **fiyat** adlı parametrelerin değerlerini aktaracağını belirttik. **@** işareti kendisinden sonra gelen ifadenin parametre olacağını belirtir.

Parametrelerimize değer aktarmak için ise **Parameters.Add** metodu kullanılır.

Kullanımı şu şekildedir:

```
SqlCommandNesnesi.Parameters.Add("@Parametre", SqlDbType.VeriTürü,
Alan uzunluğu).Value = AktarılacakNesne
```

Kayıt Güncelleme

Kayıtlar üzerinde güncelleme yapmak için **update** sorgusunu kullanıyoruz

```
using System.Data;
using System.Data.SqlClient;
public partial class Form1 : Form
{
  SqlConnection baglantı = new SqlConnection
   ("Data Source=(localdb)\\Projects;Initial Catalog=stok;
   Integrated Security=True;Connect Timeout=30;
 SqlCommand komut = new SqlCommand();
 SqlDataAdapter adaptor = new SqlDataAdapter();
 DataSet ds = new DataSet();

 private void btnGuncelle_Click(object sender, EventArgs e)
 {
  try
  {
   baglantı.Open();
   ds.Clear();
   SqlDataAdapter adp = new SqlDataAdapter
   ("UPDATE urunler SET ad='" + txt_ad.Text + "',
   fiyat='" + txt_fiyat.Text + "' WHERE ad='" +
   txt_ad.Text + "'", baglantı);

   adp.Fill(ds);
```

```
   baglantı.Close();
   MessageBox.Show("Kayıt Güncellendi");
  }
  catch (Exception ex)
  {
   MessageBox.Show(ex.Message);
   baglantı.Close();
  }
 }
}
```

KAYIT SİLME

Kayıtları silmek için **delete** sorgusunu kullanıyoruz.

```
using System.Data;
using System.Data.SqlClient;
public partial class Form1 : Form
{
  SqlConnection baglantı = new SqlConnection
   ("Data Source=(localdb)\\Projects;Initial Catalog=stok;
   Integrated Security=True;Connect Timeout=30;
 SqlCommand komut = new SqlCommand();
 SqlDataAdapter adaptor = new SqlDataAdapter();
 DataSet ds = new DataSet();

 private void btnSil_Click(object sender, EventArgs e)
 {
  try
  {
   baglantı.Open();
   ds.Clear();
   SqlDataAdapter adp = new SqlDataAdapter
    ("DELETE FROM urunler WHERE ad='" + txt_ad.Text + "'", baglantı);
   adp.Fill(ds);
   baglantı.Close();
   MessageBox.Show("Kayıt Silindi");
  }
  catch (Exception ex)
  {
   MessageBox.Show(ex.Message);
```

```
   baglantı.Close();
  }
 }
}
```

Kayıt ekleme,güncelleme ve silme işlemlerine dair kodlarımızı birleştirelim. **SqlCommand** konusunda kullandığımız **stok** adlı SQL veritabanını bu uygulamamızda da kullanacağız.

Uygulamamızın kodu:

```
using System;
using System.Windows.Forms;
using System.Data;
using System.Data.SqlClient;
namespace sqldataadapter_stok
{
 public partial class Form1 : Form
 {
  public Form1()
  {
   InitializeComponent();
  }
   SqlConnection baglantı = new SqlConnection
   ("Data Source=(localdb)\\Projects;Initial Catalog=stok;
   Integrated Security=True;Connect Timeout=30;
  SqlCommand komut = new SqlCommand();
  SqlDataAdapter adaptor = new SqlDataAdapter();
  DataSet ds = new DataSet();

  private void Form1_Load(object sender, EventArgs e)
  {
   try
   {
    adaptor = new SqlDataAdapter
    ("SELECT ad,fiyat FROM urunler", baglantı);
    adaptor.Fill(ds);
    dataGridView1.DataSource = ds.Tables[0];

    txt_ad.DataBindings.Clear();
    txt_ad.DataBindings.Add(new System.Windows.Forms.
```

```
    Binding("text", ds.Tables[0], "ad", true,
    System.Windows.Forms.DataSourceUpdateMode.OnPropertyChanged));
   txt_fiyat.DataBindings.Clear();
   txt_fiyat.DataBindings.Add(new System.Windows.Forms.
    Binding("text", ds.Tables[0], "fiyat", true,
    System.Windows.Forms.DataSourceUpdateMode.OnPropertyChanged));

   baglantı.Open();

   komut = new SqlCommand
    ("SELECT MAX(fiyat) AS maksimum FROM urunler", baglantı);
   lblenyuksekfiyat.Text = "En yüksek fiyat " + komut.ExecuteScalar();

   komut = new SqlCommand
    ("SELECT MIN(fiyat) AS minimum FROM urunler", baglantı);
   lblendusukfiyat.Text = "En düşük fiyat " + komut.ExecuteScalar();

   baglantı.Close();
  }
  catch (Exception ex)
  {
   MessageBox.Show(ex.Message);
   baglantı.Close();
  }
}

private void btnEkle_Click(object sender, EventArgs e)
{
 try
 {
  SqlDataAdapter adaptor = new SqlDataAdapter();
  string insertSQL = "INSERT INTO urunler (ad, fiyat) " +
   "VALUES (@ad, @fiyat)";

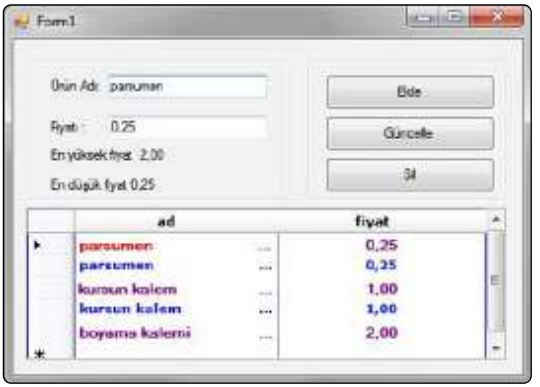
  SqlCommand insertCMD = new SqlCommand(insertSQL, baglantı);
  adaptor.SelectCommand = insertCMD;

  insertCMD.Parameters.Add("@ad", SqlDbType.NVarChar, 15).
   Value = txt_ad.Text;
  insertCMD.Parameters.Add("@fiyat", SqlDbType.Float, 15).
   Value = txt_fiyat.Text;

  adaptor.Fill(ds, "urunler");
  MessageBox.Show("Kayıt eklendi!");
 }
```

```
        catch (Exception ex)
        {
         MessageBox.Show(ex.Message);
        }
       }
       private void btnGuncelle_Click(object sender, EventArgs e)
       {
        try
        {
         baglantı.Open();
         ds.Clear();
         SqlDataAdapter adp = new SqlDataAdapter
          ("UPDATE urunler SET ad='" + txt_ad.Text + "',
          fiyat='" + txt_fiyat.Text + "' WHERE ad='" +
          txt_ad.Text + "'", baglantı);
         adp.Fill(ds);
         baglantı.Close();
         MessageBox.Show("Kayıt Güncellendi");
        }
        catch (Exception ex)
        {
         MessageBox.Show(ex.Message);
         baglantı.Close();
        }
       }
       private void btnSil_Click(object sender, EventArgs e)
       {
        try
        {
         baglantı.Open();
         ds.Clear();
         SqlDataAdapter adp = new SqlDataAdapter
          ("DELETE FROM urunler WHERE ad = `" + txt_ad.Text + "'", baglantı);
         adp.Fill(ds);
         baglantı.Close();
         MessageBox.Show("Kayıt Silindi");
        }
        catch (Exception ex)
        {
```

```
            MessageBox.Show(ex.Message);
            baglantı.Close();
        }
    }
}
```



Formun **Load** olayı esnasında veritabanındaki kayıtları çekerek **DataGridView** içerisine ve **TextBox** nesneleri içerisine veritabanındaki değerleri ekliyoruz.

Kayıtları okumak için **Select** sorgusunu kullanıyoruz.

Kodumuzda..

```
SELECT ad, fiyat FROM urunler
```

… sorgusu ile **urunler** tablosundaki **ad** ve **fiyat** alanlarını çektik.

Veritabanındaki kayıtları çekmek için **SqlCommand** nesnesinin **ExecuteReader()** metodunu kullanıyoruz.

Ayrıca formun **Load** olayı esnasında veritabanında bulunan kayıt sayısını bularak görüntülüyoruz. Bu işlem için formun **Load** olayı esnasında fiyatı en düşük ve en yüksek olan ürünleri seçerek görüntülüyoruz. Fiyatı en yüksek olan ürünü bulmak için **MAX**, en düşük olanı bulmak için **MIN** sorgu deyimi kullanılır.

Bu işlem için örneğimizde…

```
SELECT MAX(fiyat) AS maksimum FROM urunler
SELECT MIN(fiyat) AS minimum FROM urunler
```

… sorgularını kullandık. Bu şekilde **urunler** tablosundaki **fiyat** alanındaki değerlerden en yüksek ve en düşük olan değerleri çekiyoruz. Geriye tek bir değer dönmesi nedeni ile de **SqlCommand** nesnesinin **ExecuteScalar()** metodunu kullanıyoruz.

Kayıt eklemek için kodumuzda…

```
INSERT INTO urunler (ad, fiyat) " + "VALUES (@ad, @fiyat)
```

… sorgusunu kullandık. Bu şekilde **urunler** tablosundaki **ad** ve **fiyat** alanlarına karşılık **ad** ve **fiyat** adlı parametrelerin değerlerini aktaracağını belirttik. **@** işareti kendisinden sonra gelen ifadenin parametre olacağını belirtir.

Parametrelerimize değer aktarımını ise…

```
insertCMD.Parameters.Add("@ad", SqlDbType.NVarChar, 15).Value = txt_ad.Text
insertCMD.Parameters.Add("fiyat", SqlDbType.Float, 15).Value = txt_fiyat.Text
```

… kod satırlarında görüldüğü gibi **Parameters.Add** metodu ile yaptık.

Kayıtlar üzerinde güncelleme yapmak için kodumuzda…

```
UPDATE urunler SET ad='" + txt_ad.Text + "', fiyat='" + txt_fiyat.Text
+ "' WHERE ad = '" + txt_ad.Text + "'
```

… sorgusu ile **urunler** tablosundaki **ad** ve **fiyat** adlı alanlardaki veriler, eğer **ad** alanındaki veri **txt_ad** adlı text kutusundaki veriye eşit ise; **txt_ad** ve **txt_fiyat** adlı text kutularında yazılı olan yeni değerlere göre güncellenir.

Kayıtları silmek için kodumuzda…

```
DELETE FROM urunler WHERE ad='" + txt_ad.Text + "'
```

…. sorgusu ile **urunler** tablosundaki **ad** alanındaki veri **txt_ad** adlı text kutusundaki değere eşit ise, bu satırdaki veri silinir.

LINQ (LANGUAGE INTEGRATED QUERY)

28

LINQ; kelime olarak **Language Integrated Query**, yani dil ile bütünleştirilmiş sorgu anlamını taşımaktadır. Burada dikkatinizi çekmek istediğim nokta dil ile bütünleştirilmiş olmasıdır. Bildiğiniz gibi veritabanı işlemleri için SQL kullanıyorduk. LINQ ile artık bu sorgulama işlemleri için ayrı bir dil kullanmaya gerek kalmıyor. Ayrıca dizileri, koleksiyon yapılarını, dosyaları, XML yapılarını da veritabanı gibi sorgulayabiliyoruz.

LINQ'in kelime yapısını inceleyecek olursak SQL'e benzer bir yapısı vardır. Örneklerde de göreceğiniz üzere **SELECT** ifadesinin sona gelmesi haricinde pek de bir farkı yok.

```
int[] sayilar= {1071, 1453, 1799, 1923, 2009, 1988};
var sayi1 = from veri in sayilar where veri > 1900 select veri;
foreach(int gelenler in sayi1)
  Console.WriteLine(gelenler.ToString());
```

Yukarıdaki kodu inceleyecek olursak, öncelikle **sayilar** adli bir dizi oluşturduk ve eleman atadık. Hemen ardından LINQ sorgumuzu yazdık. Sorgu sonucu dönecek değerler **sayi1** adlı değer dizimizde tutulacak. LINQ sorgumuzda **veri** adlı bir değişken kullandık, bu değişken ile dizimizdeki elemanları seçtik. **in** ifadesinden sonra veri kaynağımızı seçtik, **WHERE** ifadesi ile de filtreleme işlemlerini gerçekleştirdik. En son ise **SELECT** işlemi ile seçimimizi yapıp sorgumuzu sonlandırıyoruz. **foreach** döngümüz ile **sayi1** değer dizisindeki verileri okuyoruz.



Görüldüğü gibi 1900'dan büyük sayılar geldi. Fakat biz istiyoruz ki bu sayılar sıralı gelsin. Bunun için LINQ kodumuzu aşağıdaki gibi yapıyoruz:

```
var sayi1 = from veri in sayilar where veri > 1900 orderby veri
ascending select veri;
```



Değişen kısımları inceleyecek olursak, **`veri`** adlı değişkene integer tip verdik, bir de sıralama yapması için **`orderby`** **veri ascending** ifadesini ekledik.

LINQ olmasaydı, yukarıdaki işlemi yapmak için akla ilk gelen yöntem olarak, döngü içinde tüm dizi elemanlarını okuyup, sonra **`if`** kontrolü ile 1900'den büyük olanları alacaktık. Sonra bu elemanları bir sıralama algoritması ile sıraya koyabiliriz veya **`if`** içinde gelen verileri **`ArrayList`** koleksiyonuna atıp, bu sınıfın hazır metotlarından da yararlanabiliriz.

LINQ Operatörleri

Linq içerisinde kullanılan başlıca operatör türleri şunlardır:

- **Aggregation Operators (Yığın operatörleri):** Bir koleksiyon ya da dizi içerisindeki değeri hesaplamak için kullanılır. Örneğin; bir dizi içerisindeki rakamların toplamı ya da ortalaması gibi.

```
(Count, LongCount, Sum, Max, Min, Average, Aggregate)
```

- **Projection Operators (Seçim Operatörleri):** Seçim işlemleri için kullanılan operatörlerdir.

```
(Select, Selectmany)
```

- **Concatenation Operators (Birleştirme Operatörleri):** Bir diziyi bir diğeri ile birleştirmek amacı ile kullanılır.

```
(Concat)
```

- **Element Operators (Eleman Operatörleri):** Bir diziden bir elemanı döndürmek amacı ile kullanılır. Örneğin; bir dizinin ilk elemanı, son elemanı ya da index numarası belirtilen bir elemanını bulmak gibi.

```
(DefaultIfEmpty, ElementAt, ElementAtOrDefault, First,
FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault)
```

- **Conversion Operators (Dönüşüm Operatörleri):** Belirtilen bir nesneyi başka bir türe dönüştürmek amacı ile kullanılır.

```
(AsEnumerable, Cast, OfType, ToArray, ToDictionary, ToList, ToLookUp)
```

- **Equality Operators (Eşitlik Operatörleri):** İki diziyi karşılaştırarak birbirine eşit olup olmadığını anlamak amacı ile kullanılır.

```
(SequenceEqual)
```

- **Generation Operators (Üretim Operatörleri):** Yeni bir değerler dizisi oluşturmak amacı ile kullanılır.

```
(Empty, Range, Repeat)
```

- **Grouping Operators (Gruplama Operatörleri):** Ortak bir niteliğe sahip elemanları gruplandırmak amacı ile kullanılır.

```
(GroupBy)
```

- **Join Operators (Birleştirme Operatörleri):** Ortak bir niteliğe sahip farklı veri kaynaklarındaki değerleri birleştirmek amacı ile kullanılır.

```
(Join, GroupJoin)
```

- **Partitioning Operators (Bölümleme Operatörleri):** Bir diziyi iki veya daha fazla bölüme ayırmak için kullanılır.

```
(Skip, SkipWhile, Take, TakeWhile)
```

- **Quantifers Operators (Belirleyici Operatörler):** Bir dizi içerisindeki elemanların var olup olmadığı ile ilgili işlemler için kullanılır.

```
(All, Any, Contains)
```

- **Restriction Operators (Kısıtlama Operatörleri):** Bir değer ya da değerler dizisi üzerinde sorgu yolu ile kısıtlamalar yapmak amacı ile kullanılır.

```
(Where, OfType)
```

- **Set Operators:** Sorgu yolu ile üretilen değerler üzerinde çeşitli ayarlamalar yapmak amacı ile kullanılır. Örneğin; distinct operatörü ile üretilen sonuçlardan birden fazla varsa, bir adet listelenmesi gibi.

```
(Distinct, Except, Intersect, Union)
```

- **Ordering Operators:** Sıralama amacı ile kullanılır.

```
(OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse)
```

Bu operatörlerden sık kullanılan bazı operatörler ile ilgili örnek uygulamalar yapalım.

WHERE OPERATÖRÜ

Sorgu ifadesinden önce kullanılarak, belirtilen sorguya göre işlem yapılmasını sağlar.

SELECT OPERATÖRÜ

Belirtilen şarta uyan değerlerin listelenmesini sağlar.

`WHERE` ve **`SELECT`** operatörlerini bir arada kullanarak bir örnek uygulama yapalım. Öncelikle bir tane **struct** oluşturup değer atayalım. Bu başlık altındaki örneklerimizi, oluşturacağımız struct üzerinden yapacağız.

```
using System;
using System.Linq;
using System.Text;
namespace linq_select
{
 class Program
 {
  public struct ogrenci
  {
    public string ad;
    public string soyad;
```

```
    public int numara;
    public string sınıf;
  }
  static void Main(string[] args)
  {
    ogrenci[] ogrenciler = new ogrenci[3];
    ogrenciler[0].ad = "Talha";
    ogrenciler[0].soyad = "AKTAŞ";
    ogrenciler[0].numara = 151;
    ogrenciler[0].sınıf = "3-C";

    ogrenciler[1].ad = "Sümeyye";
    ogrenciler[1].soyad = "AKTAŞ";
    ogrenciler[1].numara = 133;
    ogrenciler[1].sınıf = "1-A";

    ogrenciler[2].ad = "Musab";
    ogrenciler[2].soyad = "ÇELİK";
    ogrenciler[2].numara = 144;
    ogrenciler[2].sınıf = "3-C";

    var gelenogr = from ogr in ogrenciler where ogr.sınıf == "3-C"
select ogr.ad;
    foreach(string isim in gelenogr)
      Console.WriteLine(isim.ToString());
    Console.ReadKey();
  }
 }
}
```



Yapı üzerindeki ilk LINQ kodumuzu bu şekilde yazalım. Bu LINQ sorgusundan bize dönecek değerler 3-C sınıfındaki öğrencilerin adıdır.

Yukarıdaki örneğimizde öğrencilerin yalnızca ad bilgisine erişmiştik. Ad, soyad, sınıf, numara bilgilerinin hepsine birden erişebilmek için select kısmını biraz değiştiriyoruz.

```
var gelenogr = from ogr in ogrenciler
               where
                 ogr.sınıf == "3-C"
               select ogr;
```

Sorgumuzu yukarıdaki gibi değiştirdikten sonra döngü içerisindeki kodumuzu yazmaya başlayalım.



Gördüğünüz gibi ad, soyad, numara ve sınıf bilgilerinin hepsine birden erişebiliyoruz.

Kodumuzun son hali aşağıdaki gibi olacak:

```
var gelenogr = from ogr in ogrenciler
               where
                 ogr.sınıf == "3-C"
               select ogr;
foreach(var isim in gelenogr)
  Console.WriteLine(isim.ad.ToString() + " " + isim.soyad.ToString());
```



Birden fazla filtre seçeneği için **WHERE** ifadesinden sonra yazdığımız ifadeler arasına **&** (*and*) veya **|** (*or*) yazıyoruz.

```
var gelenogr = from ogr in ogrenciler where ogr.soyad == "AKTAŞ" &
ogr.numara > 140 select ogr.ad ;
foreach (var yazogr in gelenogr)
  Console.WriteLine(yazogr);

var gelenogr = from ogr in ogrenciler where ogr.soyad == "AKTAŞ" |
ogr.numara > 140 select ogr.ad ;
foreach (var yazogr in gelenogr)
  Console.WriteLine(yazogr);
```

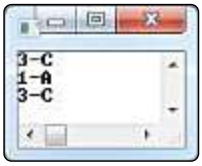
Birinci sorgumuzda dönen değer *Talha* iken, ikinci sorgumuzda dönen değer *Talha*, *Sümeyye*, *Musab* olacaktır. Çünkü ilk sorgumuzda "*Soyadı AKTAŞ olan ve numarası 140'tan büyük olan isimleri*", ikinci sorgumuzda "*Soyadı AKTAŞ olan veya numarası 140'tan büyük olan isimleri*" sorguladık.

DISTINCT OPERATÖRÜ

Listelenen bir değerin birden fazla yazılmasını engellemek amacı ile kullanılır.

ogrenci isimli struct içerisindeki bilgilere göre okulda hangi sınıfların olduğunu öğrenmek istersek…

```
var gelenogr = from ogr in ogrenciler select ogr;
foreach(var yazsınıf in gelenogr)
  Console.WriteLine(yazsınıf.sınıf.ToString());
```



… sorgusu ile sınıflar bizim karşımıza çıkar. Fakat *Talha* ve *Musab* aynı sınıfa gittiği için *3-C* yazısı iki kez yazılır. Bu ise istenilen bir durum değildir.

Bunu engellemek için **distinct** operatörünü kullanıyoruz.

```
var gelenogr = (from ogr in ogrenciler select ogr.sınıf).Distinct();
foreach (var yazsınıf in gelenogr)
  Console.WriteLine(yazsınıf);
```



CONTAINS OPERATÖRÜ

Belirtilen değerin mevcut olup olmadığını bulmak amacı ile kullanılır.

```
var gelenogr = from ogr in ogrenciler
               where
                 ogr.ad.Contains("a")
               select new {ogr.ad, ogr.soyad};
foreach (var yazogr in gelenogr)
  Console.WriteLine(yazogr.ad.ToString() + " " + yazogr.soyad.ToString());
```



Örneğimizde adlarında **a** harfi geçen öğrencilerin ad ve soyadlarını getiriyoruz.

MAX, MIN OPERATÖRLERİ

Bir koleksiyon içerisindeki en yüksek ve en düşük değerleri bulmak amacı ile kullanılırlar.

```
var gelenogr = (from ogr in ogrenciler select ogr.numara).Max();
Console.WriteLine(" Numaralardan en büyüğü= " + gelenogr.ToString());
```




```
var gelenogr = (from ogr in ogrenciler select ogr.numara).Min();
Console.WriteLine(" Numaralardan en küçüğü= " + gelenogr.ToString());
```

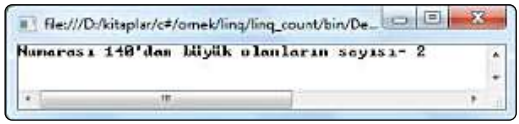


COUNT Operatörü

Bir koleksiyon içerisindeki belirtilen sorgu kriterine uyan kayıtların sayısını bulmak amacı ile kullanılır.

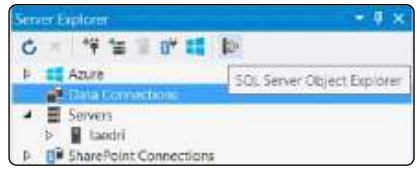
```
var sayı = (from ogr in ogrenciler where
      ogr.numara>140 select ogr.numara).Count();
Console.WriteLine
      ("Numarası 140'dan büyük olanların sayısı= " +sayı);
Console.ReadKey();
```

Örneğimiz ile `ogrenci` adlı yapı içerisindeki numarası 140'dan büyük olanları sayısını bularak ekrana yazdırıyoruz.

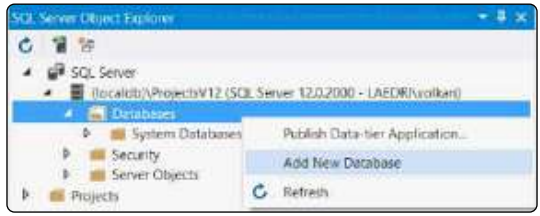


SQL ve LINQ

Veritabanı işlemlerini şu ana kadar SQL ile yaptık. Bu konu altında bu işlemleri LINQ kullanarak yapacağız. Veritabanı işlemlerinde LINQ kullanmanın özel adı **DLINQ**'dir. Öncelikle örneğimiz için bir veritabanı oluşturalım. **SQL Server Object Explorer** kullanarak SQL veritabanı oluşturup, içerisinde düzenleme ve kayıt ekleme işlemleri yapabilirsiniz. **Server Explorer** içerisinde **SQL Server Object Explorer** butonuna tıklayalım.



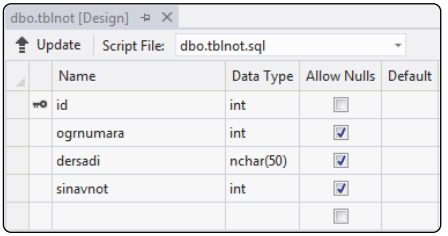
SQL Server Object Explorer içerisinde yerel veritabanları **localdb** başlığı altında yer almaktadır. Biz de SQL veritabanını yerel olarak oluşturacağımız için **localdb** başlığı altındaki **Databases** ismine farenin sağ tuşu ile tıklayarak, gelen menüden **Add New Database** seçeneğini seçelim.

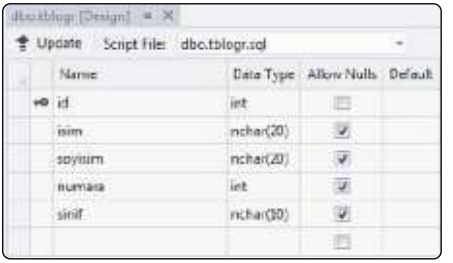


Karşımıza gelen pencerede veritabanı ismini yazalım.

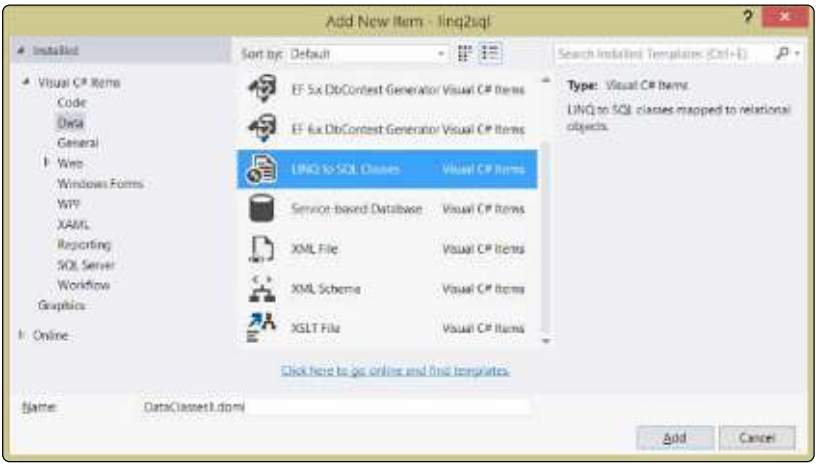


Veritabanımızın ismini **mydb** olarak belirledik. Veritabanı içerisinde **tblnot** ve **tblogr** adlı iki adet tablo oluşturuyoruz. Tablo içerisindeki alanları ise aşağıdaki gibi oluşturuyoruz.

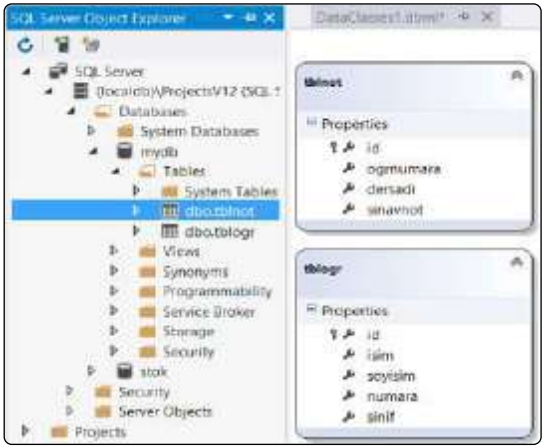




Projemize sağ tıklayıp **Add > New Item** dedikten sonra bir adet **LINQ to SQL Classes** ekliyoruz.



Tablolarımızı oluşan dosya üzerine sürükleyerek atıyoruz.



Örneğimizin form tasarımı aşağıdaki gibi olacak:



(1) `txtAd`

(2) `txtSoyad`

(3) `txtNo`

(4) `cmbsinif`

(5) `btnEkle`

(6) `btnOku`

(7) `btnSınıfDeg`

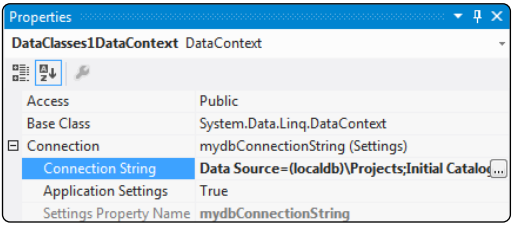
(8) `btnSil`

Artık kod kısmına geçebiliriz.

Kayıt ekleme kodumuz aşağıdaki gibi olacak:

```
string cs = "Data Source=(localdb)\\Projects;Initial
 Catalog=mydb;Integrated Security=True;Connect Timeout=30";
DataClasses1DataContext veriler = new DataClasses1DataContext(cs);
tblogr ogr = new tblogr() { isim=txtAd.Text , soyisim=txtSoyad.Text,
 numara=Convert.ToInt32(txtNo.Text) , sinif=cmbsinif.Text };
veriler.tblogrs.InsertOnSubmit(ogr);
veriler.SubmitChanges();
MessageBox.Show("Kayıt Eklendi");
```

Kodumuzda görüldüğü gibi öncelikle veritabanı ile bağlantımız için öncelikle bir **connection string** oluşturduk. Connection string'i **dbml** uzantılı **Linq To SQL** sınıfımız seçiliyken, **Properties** penceresinden connection **string** özelliğine bakarak görebiliriz.



Daha sonra oluşan sınıftan bir `DataContext` nesnesi türetiyoruz.

```
DataClasses1DataContext veriler = new DataClasses1DataContext(cs);
```

`InsertOnSubmit` metodu ile yeni bir kayıt ekleyip, `SubmitChanges` metodu ile de yaptığımız değişikliklerin uygulanmasını sağlıyoruz.



Örnek olarak bir kayıt ekleyelim.

166 numaralı öğrencinin sınıfını bulalım.

```
DataClasses1DataContext veriler = new DataClasses1DataContext();
var ogr = from veri in veriler.tblogrs where veri.numara
== 166 select veri.sinif;
foreach (var gelen in ogr)
   MessageBox.Show(gelen.ToString());
```



Numara alanına 166 değerini girip **Oku** butonuna tıklayarak, sonucu görüntüleyelim.

166 nolu öğrencinin sınıfı olan 1-A'yı değiştirerek 1-B yapalım:

```
DataClasses1DataContext veriler = new DataClasses1DataContext();
tblogr tabloogr = (from ogr in veriler.tblogrs where ogr.numara
== 166 select ogr).SingleOrDefault();
tabloogr.sinif = "1-B";
veriler.SubmitChanges();
```



Numara alanına 166 değerini girip **Oku** butonuna tıklayarak, güncellenmiş sonucu görüntüleyelim. Update olayında dikkat edilirse mevcut kaydı alıp, üzerinde değişiklik yapmak sureti ile update olayını gerçekleştirdik.

166 numaralı öğrenciyi silelim:

```
string cs = "Data Source=(localdb)\\Projects;Initial
 Catalog=mydb;Integrated Security=True;Connect Timeout=30";
DataClasses1DataContext veriler = new DataClasses1DataContext(cs)
tblogr tabloogr = (from ogr in veriler.tblogrs where ogr.numara
== 166 select ogr).SingleOrDefault();
veriler.tblogrs.DeleteOnSubmit(tabloogr);
veriler.SubmitChanges();
```

Silme işlemi için **`DeleteOnSubmit`** metodunu kullanıp, **`SubmitChanges`** metodu ile de yaptığımız değişikliklerin uygulanmasını sağladık.

XML VE LINQ

XML yapılarında da LINQ'i etkin bir şekilde kullandığımızı belirtmiştim. Şimdi bunu örneklerle pekiştirelim. XML verileri üzerinde LINQ ile işlem yapmanın özel adı **XLINQ**'dir.

Üzerinde işlem yapacağımız XML dökümanı aşağıdaki gibidir.

```
<?xml version="1.0" encoding="utf-8" ?>
<Arkadaslar>
  <Kisi>
    <isim>Tahsin Sevinç</isim>
    <site>http://tahsin.sevinch.net/</site>
  </Kisi>
  <Kisi>
    <isim>Erdem Tüzen</isim>
    <site>http://erdem.tuzen.net/</site>
  </Kisi>
  <Kisi>
    <isim>Selçuk Ermaya</isim>
    <site>http://www.selcukermaya.com/</site>
  </Kisi>
</Arkadaslar>
```

Öncelikle XML ve LINQ işlemleri için…

```
System.Xml;
System.Xml.Linq;
System.Linq;
```

… sınıflarını ekliyoruz. Öncelikle XML dökümanından linq ile veri okuyacağız.

```
XDocument myXML = XDocument.Load("list.xml");
var arkadaslar = from kisi in myXML.Descendants("Kisi")
  select kisi.Element("isim").Value;

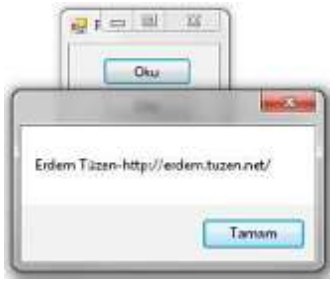
foreach (var veri in arkadaslar)
  MessageBox.Show(veri.ToString());
```



İkinci secenekte ise kişilerin isimlerinin yanında site adreslerini de görüntüleyelim:

```
XDocument myXML = XDocument.Load("list.xml");
var arkadaslar = from kisi in myXML.Descendants("Kisi") select kisi;

foreach (var veri in arkadaslar)
  MessageBox.Show(veri.Element("isim").Value.ToString() +
   "-" + veri.Element("site").Value.ToString());
```



XML dökumanlarımıza LINQ ile veri eklemek aynı şekilde gayet basit.

```
XDocument myXML = XDocument.Load("list.xml");
XElement xmlekle = new XElement("Kisi");
xmlekle.Add(new XElement("isim", "Volkan AKTAŞ"));
xmlekle.Add(new XElement("site", "http://www.volkanaktas.com/"));
myXML.Element("Arkadaslar").Add(xmlekle);
myXML.Save("list.xml");
```

Burada yaptığımız işlemde öncelikle **`xmlekle`** adında **`Kişi`** düğümünü oluşturacak bir değişken tanımlıyoruz. Daha sonra bu düğüme ekliyeceğimiz elemanları ve değerleri belirtiyoruz. Düğümü oluşturduktan sonra…

```
myXML.Element("Arkadaslar").Add(xmlekle);
```

… ile bu düğümün ekleneceği kök düğümü belirtiyoruz. En son ise yapılan değişiklikleri listemize ekliyoruz.

Ve sonuç:

```
<?xml version="1.0" encoding="utf-8"?>
<Arkadaslar>
  <Kisi>
    <isim>Tahsin Sevinç</isim>
    <site>http://tahsin.sevinch.net/</site>
  </Kisi>
  <Kisi>
    <isim>Erdem Tüzen</isim>
    <site>http://erdem.tuzen.net/</site>
  </Kisi>
  <Kisi>
    <isim>Selçuk Ermaya</isim>
    <site>http://www.selcukermaya.com/</site>
  </Kisi>
  <Kisi>
    <isim>Volkan AKTAŞ</isim>
    <site>http://www.volkanaktas.com/</site>
  </Kisi>
</Arkadaslar>
```

Güncelleme işleminde ise yapılan öncelikle LINQ sorgumuz ile güncelleme yapacağımız düğümü seçiyoruz. Daha sonra buradan gelen sonuca göre istediğimiz güncelleme işlemini yapıyoruz.

```
XDocument myXML = XDocument.Load("list.xml");
var eleman = (from veri in myXML.Descendants("Kisi")
       where
         veri.Element("isim").Value == "Selçuk Ermaya"
       select veri).SingleOrDefault();

((XElement)eleman).Element("site").SetValue("http://www.blogsa.net/");
myXML.Save("list.xml");
```

Burada dikkat çekmek istediğim nokta **`SingleOrDefault`**'dur. Bunu eklememizin sebebi; sadece bir adet sonuc istememizdir. Bu ifadeyi kullandığımızda bize ilk bulduğu sonucu verir.

```
<Kisi>
  <isim>Selçuk Ermaya</isim>
  <site>http://www.blogsa.net/</site>
</Kisi>
```

Silme işlemimizde yine güncellemeye benzemekle birlikte, sadece güncelleme işlemi sırasında bulunan düğümü **`Remove()`** ile siliyoruz.

```
XDocument myXML = XDocument.Load("list.xml");
var eleman = (from veri in myXML.Descendants("Kisi")
       where
         veri.Element("isim").Value == "Selçuk Ermaya"
       select veri).SingleOrDefault();
```

```
eleman.Remove();
myXML.Save("list.xml");
```

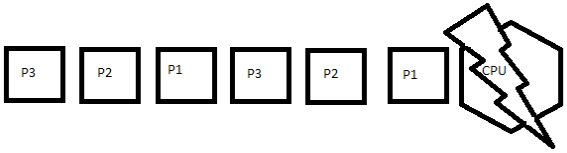
Tüm bu işlemler sonucu, elimizde kalan XML belgesi şu şekilde olur:

```
<?xml version="1.0" encoding="utf-8"?>
<Arkadaslar>
  <Kisi>
    <isim>Tahsin Sevinç</isim>
    <site>http://tahsin.sevinch.net/</site>
  </Kisi>
  <Kisi>
    <isim>Erdem Tüzen</isim>
    <site>http://erdem.tuzen.net/</site>
  </Kisi>
  <Kisi>
    <isim>Volkan AKTAŞ</isim>
    <site>http://www.volkanaktas.com/</site>
  </Kisi>
</Arkadaslar>
```

THREADING

29

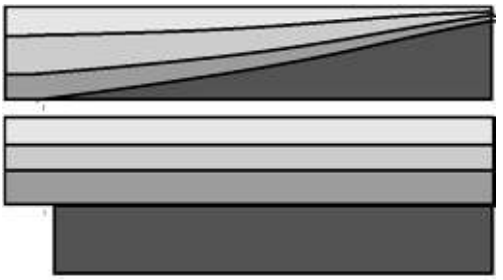
Thread kavramından önce thread kullanmanın gerekliliğini ve olayların arkaplanını inceleyelim. Öncelikle DOS kullanıldığında işletim sistemi üzerinde sadece bir işlem yapılıyordu. Windows ile Word de yazı yazarken aynı anda müzik dinleme lüksüne kavuştuk. İşin donanım kısmına inecek olursak esasında pekte değişen bir şey yok işlemci yine aynı anda tek işlem yapıyor. Sadece gelen iş parçacıklar halinde geliyor. P1, P2, P3 adlı üç tane işlemimizin olduğunu varsayarsak bu işlemlerin her biri bir parçalanarak işlemcide işlenmesi için sıraya konur ve sırası gelen iş parçacığı işlenerek uygulamamız devam eder. Fakat bir işlem süresi ortalama 20ms olarak gerçekleştiği için aradaki farkı biz hissetmeyiz.



Fakat işlemler her zaman bu şekilde sıra ile gelmeyebilir. İşlem önceliği olan bir işlem varsa bu sıra değişebilir. Zaten bir programın yanıt vermediği anlarda bilgisayarımızdaki diğer işlemlerin yavaşlamasının sebebi de budur.

Her bir işlemde içinde işlemin gerçekleşmesi için bir kanal oluşur. Buna **Main Thread** adı verilir. Fakat bu ana kanal içerisinde olan işlemlerden yoğun çalışanlar olduğunda diğer işlemlerin çalışmasını engeller. Bu durumu tek şeritli bir yolda bir tırın trafiği tıkamasına benzetebiliriz. Bu durumda yazacağımız programlarda **multi threading** (*çok kanallı yapı*) kullanma ihtiyacı

hissediyoruz. Yani trafiğin aynı hızda işlemesi için tıra yeni bir yol açıyoruz. Bahsettiğimiz bu durumu bir de program üzerinde örnekleyelim. Bir internet cafe programı düşünün. Ana bilgisayardan gelecek olan komutu işleyecek bir client programı nasıl yazarız? Programımız öncelikle veri alacağı portu sürekli tarar ve bir bilgi okuduğunda aldığı veriyi, işleyecek kısma gönderir. Fakat port okuma kısmı bilgisayar açıldığı andan, kapandığı ana kadar süreceği için burada bir sonsuz döngü olacak. Bu sonsuz döngüde birkaç dakika sürmeden programımızın kilitlenmesine sebep olacak. İşte bu noktada multi threading yardımımıza koşuyor ve port okuma kısmını ayrı bir thread içinde yapıyoruz. Bu şekilde programımızın kilitlenmesinin önüne geçiyoruz.



I. Grafikte thread kullanılmaması sonucu t anında çalışmaya başlayan koyu gri renkli işlemin sistem kaynaklarını kullanmasının diğer işlemleri kısıtlaması şematize edilmiştir.

II. Grafikte ise t anında çalışmaya başlayan koyu gri renkli gri işlemin, ayrı bir thread içerisinde çalışması şematize edilmiştir.

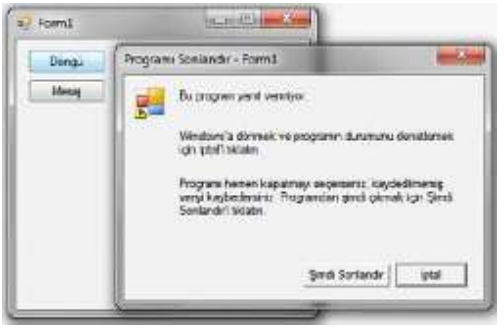
Threadler İle Çalışmak

Projemizde thread kullanmak için öncelikle **`System.Threading`** isim uzayını eklemeliyiz. Öncelikle thread kullanmadan şu şekilde basit bir kod yazalım:

```
private void btnDongu_Click(object sender, EventArgs e)
{
  Dongu();
}
private void Dongu()
{
  decimal i = default(decimal);
  for (i = 0; i <= 1000000000; i++)
```

```
  {
  }
  MessageBox.Show("İşlem Bitti");
}
private void btnMesaj_Click(object sender, EventArgs e)
{
  MessageBox.Show("Tıkladın");
}
```

Önce **btnDongu** isimli butona basıp döngüyü çalıştırdığınızda, **btnMesaj** isimli butona bastığınızda işlemin gerçekleşmediğini göreceksiniz. Çünkü o anda **mainthread** içerisinde **Dongu** çok fazla işlem gerçekleştirmektedir.



Aynı işlemi bir de thread kullanarak yapalım. Bu sefer programımızın kodları aşağıdaki gibi olacaktır.

```
using System;
using System.Windows.Forms;
using System.Threading;
namespace Mesaj_Thread
{
 public partial class Form1 : Form
 {
  public Form1()
  {
    InitializeComponent();
  }
  private void btnDongu_Click(object sender, EventArgs e)
  {
    Thread kanal1 = new Thread(Dongu);
```

```
        kanal1.Start();
    }
    private void Dongu()
    {
        decimal i = default(decimal);
        for (i = 0; i <= 1000000000; i++)
        {
        }
        MessageBox.Show("İşlem Bitti");
    }
    private void btnMesaj_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Tıkladın");
    }
  }
}
```

Kodları çalıştırıp **btnDongu** isimli butona bastığınızda ve ardından **btnMesaj** isimli bastığınızda önceki örnekte karşılaşılan sorunun gerçekleşmediğini ve "*Tıkladın*" mesajının çıktığını göreceksiniz. Çünkü…

```
Thread kanal1 = new Thread(Dongu);
    kanal1.Start();
```

… satırlarında **kanal1** isimli yeni bir thread oluşturarak **Dongu** isimli metodumuzu bu kanal içerisinde çalıştırıyoruz.



Sleep Metodu

Kanal içerisinde yapılan işlemi bekletmek amacıyla **`Thread.Sleep`** metodu kullanılır. Örneğin; yapılan işlemi 1 saniye bekletmek için **`Thread.Sleep(1000)`** şeklinde kullanılır. Buradaki 1000, milisaniye değeridir.

Örnek olarak bir form üzerine 2 adet **Label** nesnesi ekleyelim ve bu Label'ler içerisinde iki ayrı sayım işlemi gerçekleştirelim. İki ayrı sayım işlemi için iki ayrı thread tanımlayacağız.

```
private void btnBasla_Click(object sender, EventArgs e)
{
  Thread kanal1 = new Thread( new ThreadStart(sayici1));
  Thread kanal2 = new Thread(new ThreadStart(sayici2));
  CheckForIllegalCrossThreadCalls = false;

  kanal1.Start();
  kanal2.Start();
}
private void sayici1()
{
  for (int i = 1; i <=30; i++)
  {
    Thread.Sleep(500);
    label1.Text=i + " ";
  }
}
private void sayici2()
{
  for (int i = 1; i <= 30; i++)
  {
    Thread.Sleep(500);
    label2.Text = i + " ";
    if (i == 15)
      Thread.Sleep(3000);
  }
}
```



Her bir rakam arttırma işleminden önce 500 milisaniyelik bir bekleme süresi ile rakam artma aralığını, yarım saniye olarak belirliyoruz. Ayrıca **`sayici2`** adlı

thread içerisinde yazdırılan rakam 15 ise, 3000 milisaniyelik ek bir bekleme işlemi daha yaptırıyoruz.

```
CheckForIllegalCrossThreadCalls = false;
```

… ifadesi ile **cross thread exception** hatalarının göz ardı edilmesini sağladık. Uygulamamızda her iki thread içerisinden **Label** nesnelerine erişip değer görüntülemek istediğimizden dolayı, hata meydana gelmektedir. .NET içerisinde başka bir thread üzerinden form üzerindeki kontrollerin özelliklerini değiştirmeye kalktığınızda bu hatayı alırsınız. Ancak bu yöntem sağlıklı bir yöntem değildir. Bunun yerine delegate yapısının kullanılmasını tavsiye ederim. Aynı işlemi delegate yapısını kullanarak yeniden yazalım:

```
int j,k;
private void btnBasla_Click(object sender, EventArgs e)
{
  Thread kanal1 = new Thread(new ThreadStart(sayici1));
  Thread kanal2 = new Thread(new ThreadStart(sayici2));

  kanal1.Start();
  kanal2.Start();
}
private void sayici1()
{
  for (int i = 1; i <= 30; i++)
  {
    Thread.Sleep(500);
    label1.Invoke(new sayici1_delegate(yeni_sayici1));
  }
}
private void sayici2()
{
  for (int i = 1; i <= 30; i++)
  {
    Thread.Sleep(500);
    label2.Invoke(new sayici2_delegate(yeni_sayici2));
    if (i == 15)
      Thread.Sleep(3000);
  }
}
private delegate void sayici1_delegate();
```

```
private void yeni_sayici1()
{
  j++;
  label1.Text = j + " ";
}
private delegate void sayici2_delegate();
private void yeni_sayici2()
{
  k++;
  label2.Text = k + " ";
}
```

Invoke metodunu kullanarak delegate aracılığı ile tanımladığımız **yenisayici1** ve **yenisayici2** metodlarını çağırırız.

Thread.Sleep metodunu **TimeSpan** ile birlikte kullanabiliriz.

Thread.Sleep(TimeSpan.FromMilliseconds(1000));	1 saniye bekletir
Thread.Sleep(TimeSpan.FromMinutes(1);	1 dakika bekletir.
Thread.Sleep(TimeSpan.FromHours(1));	1 saat bekletir.
Thread.Sleep(TimeSpan.FromDays(1));	1 gün bekletir.

Suspend ve Resume Metodları

Thread.Suspend metodu ile thread içerisinde gerçekleştirilen işlemleri duraklatabiliriz. **Thread.Resume** metodu ile de işlem kaldığı yerden devam ettirilir.

```
private void btnDurdur_Click(object sender, EventArgs e)
{
  if (btnDurdur.Text == "Durdur")
  {
    btnDurdur.Text = "Devam Et";
    kanal1.Suspend();
    kanal2.Suspend();
  }
  else
  {
    btnDurdur.Text = "Durdur";
    kanal1.Resume();
    kanal2.Resume();
  }
}
```

Abort Metodu

Abort metodunu kullanarak yürütülmekte olan bir thread iptal edilir. İptal işlemi sırasında oluşan hatalar, **`ThreadAbortException`** istisnasını oluşturur.

```
private void btnBitir_Click(object sender, EventArgs e)
{
  try
  {
    kanal1.Abort();
    kanal2.Abort();
  }
  catch (ThreadAbortException ex)
  {
    MessageBox.Show(ex.ToString(), "Hata");
  }
}
```

Join Metodu

Yürütülmekte olan bir thread'i belirtilen bir süre kadar bekletmek ya da belirtilen başka bir thread içerisindeki işlemlerin bitirilmesine kadar bekletmek amacıyla kullanılır.

Yukarıda anlattığımız başlatma, durdurma, devam etme işlemleriyle birlikte join işlemini de kullanan bir örnek yapalım:

```
Thread kanal1,kanal2;
private void btnBasla_Click(object sender, EventArgs e)
{
  kanal1 = new Thread( new ThreadStart(sayici1));
  kanal2 = new Thread(new ThreadStart(sayici2));
  CheckForIllegalCrossThreadCalls = false;

  kanal1.Start();
  kanal2.Start();
}
private void sayici1()
{
  for (int i = 1; i <= 30; i++)
  {
```

```
    Thread.Sleep(500);
    label1.Text = i + " ";
  }
}
private void sayici2()
{
  kanal1.Join();
  for (int i = 1; i <= 30; i++)
  {
    Thread.Sleep(500);
    label2.Text = i + " ";
    if (i == 15)
      Thread.Sleep(3000);
  }
}
private void btnDurdur_Click(object sender, EventArgs e)
{
  if (btnDurdur.Text == "Durdur")
  {
    btnDurdur.Text = "Devam Et";
    kanal1.Suspend();
    kanal2.Suspend();
  }
  else
  {
    btnDurdur.Text = "Durdur";
    kanal1.Resume();
    kanal2.Resume();
  }
}
private void btnBitir_Click(object sender, EventArgs e)
{
  try
  {
    kanal1.Abort();
    kanal2.Abort();
  }
  catch (ThreadAbortException ex)
  {
    MessageBox.Show(ex.ToString(), "Hata");
  }
}
```

Örneğimizi çalıştırıp **Başla** butonuna bastığımızda öncelikle `kanal1` içerisindeki sayma işleminin gerçekleştirildiğini, `kanal2` isimli thread içerisindeki sayma işleminin `kanal1` içerisindeki sayma işleminin bitmesinin ardından başladığını görüyoruz.



PARALEL PROGRAMLAMA

30

Artık standart bir hâl alan çok çekirdekli işlemciler, tek çekirdekli işlemcilere göre oldukça **yüksek hız** ve **performans artışları** getirdiler. Donanım alanındaki bu gelişmelere paralel olarak programların birden fazla işlemci çekirdeği üzerinde çalıştırılması gerekmekteydi. Ancak bu işlemi gerçekleştirecek algoritmaları yazmak oldukça zordu. Aslında pek çok uygulama, çok çekirdekli işlemcilerde çalıştırılsa bile tek çekirdeği kullanmakta ve dolayısı ile herhangi bir hız ve performans artışı gerçekleşmemektedir. İşte bu amaçla Microsoft tarafından **TPL** (*Task Parallel Library*) geliştirilmiştir. Bu kütüphane, çok çekirdekli işlemcileri kullanmak ve kolayca kod yazabilmek amacıyla tasarlanmıştır.

.NET içerisinde paralel programlama başlığı altında **Parallel LINQ**, **Parallel sınıfı** ve **Task** yapıları bulunmaktadır. Bu yapılar `System.Threading.Tasks` sınıfı altında yer almaktadır. Ayrıca **C# 5** sürümünden itibaren **async** ve **await** ifadeleri ile asenkron metotlar da kullanılmaya başlanmıştır.

Parallel Sinifi

Parallel sınıfı altında 3 temel metot bulunmaktadır. Bunlar:

- `Parallel.Invoke`
- `Parallel.For`
- `Parallel.ForEach` metotlarıdır.

Parallel.Invoke Metodu

Bir metot dizisini aynı anda çalıştırmak amacıyla kullanılır. Kullanım biçimi şu şekildedir:

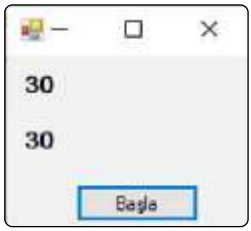
```
Invoke(ParallelOptions, Action())
```

`Parallel.Invoke` metodu ile yürütülecek işlemler **`=>`** operatöründen sonra yazılmaktadır. Yürütülecek her bir işlem, **virgüle** ayrılmaktadır.

Örnek olarak bir form üzerine iki adet **`label`** nesnesi ekleyelim ve bu labeller içerisinde iki ayrı sayım işlemi gerçekleştirelim.

```
using System;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
namespace Parallel_Invoke
{
    public partial class Form1 : Form
    {
      public Form1()
      {
          InitializeComponent();
      }
      private void btnBasla_Click(object sender, EventArgs e)
      {
          CheckForIllegalCrossThreadCalls = false;
          Parallel.Invoke(
              () =>
              {
                  for (int i = 1; i <= 30; i++)
                  {
                      Thread.Sleep(500);
                      label1.Text = i + " ";
                  }
              },
              () =>
              {
```

```
                for (int i = 1; i <= 30; i++)
                {
                    Thread.Sleep(500);
                    label2.Text = i + " ";
                }
            }
            );
        }
    }
}
```



Her bir rakam arttırma işleminden önce:

```
Thread.Sleep(500);
```

Satırı ile **`Thread.Sleep`** metodunu kullanarak **`500`** milisaniyelik bir bekleme süresi ile rakam artma aralığını **yarımşar saniye** olarak belirliyoruz.

```
CheckForIllegalCrossThreadCalls = false;
```

İfadesi ile **`cross thread exception`** hatalarının göz ardı edilmesini sağladık. Uygulamamızda yürütülen her iki paralel işlem içerisinden **`label`** nesnelerine erişip değer görüntülemek istediğimizden dolayı **hata** meydana gelmektedir. **.NET** içerisinde başka bir **`thread`** üzerinden form üzerindeki kontrollerin özelliklerini değiştirmeye kalktığınızda bu hatayı alırsınız.

Parallel.Invoke ile sadece metot ismini çağırarak da işlem yaptırmamız mümkündür. Aşağıdaki örnekte hem işlem yapan kodu **Parallel.Invoke** içerisinde tanımlayarak çalıştırdık hem de işlem yapan diğer bir kodu metot içerisinde tanımlayarak ismini çağırmakla çalıştırdık.

```
private void btnBasla_Click(object sender, EventArgs e)
{
    CheckForIllegalCrossThreadCalls = false;
    Parallel.Invoke(
    () =>
    {
        Thread.Sleep(1000);
        for (int i = 1; i <= 30; i++)
        {
            Thread.Sleep(500);
            label1.Text = i + " ";
        }
    },sayici2);
}
private void sayici2()
{
    for (int j = 1; j <= 30;j++)
    {
        Thread.Sleep(500);
        label2.Text = j + " ";
    }
}
```

Parallel.For ve Parallel.ForEach Metotları

For ve **ForEach** komutları yardımıyla paralel işlemler gerçekleştirmek amacıyla kullanılan metotlardır. **Parallel.For** metodunun kullanımı aşağıdaki biçimdedir:

```
Parallel.For(Başlangıç, Bitiş, İşlemler);
```

Parallel.ForEach metodunun kullanımı ise aşağıdaki biçimdedir:

```
Parallel.ForEach(Değer dizisi, İşlemler);
```

Örnek olarak 0 ile 10 arasındaki sayıları konsol ekranına yazdıran uygulamayı **Parallel.For** metodu ile yazalım.

```
using System;
using System.Threading.Tasks;
using System.Text;
namespace Parallel_For
{
    class Program
    {
        static void Main(string[] args)
        {
            Parallel.For(0, 10, i =>
            {
                Console.WriteLine(i);
            });
            Console.ReadLine();
            }
    }
}
```



Örneğimizde **i=>** ifadesinin ardından işlem satırlarını yazarak **0** ile **10** arasındaki yazıların görüntülenmesini sağladık.

İç içe birden fazla **Parallel.For** metodu kullanmamız da mümkündür.

```
static void Main(string[] args)
{
    Parallel.For(0, 20, i =>
    {
        Parallel.For(0, 10, j => Console.WriteLine
      (i.ToString(), j.ToString()));
    });
    Console.ReadLine();
}
```

Kodumuzda **0**'dan **20**'ye kadar bir sayma işlemi gerçekleştirilirken diğer yandan da **0**'dan **10**'a kadar ayrı bir sayma işlemi gerçekleştirilmektedir.

Bir dizi içerisindeki elemanları konsol ekranına yazdıran uygulamayı **Parallel.ForEach** ile yazalım.

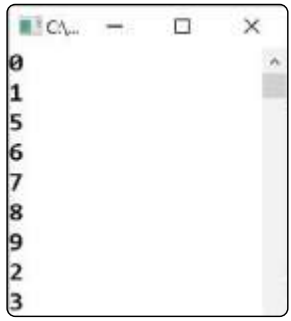
```
static void Main(string[] args)
{
    string[] adsoyad = { "Talha", "Sümeyye", "Azra", "Murat" };
    Parallel.ForEach(adsoyad, i =>
    {
        Console.WriteLine(i);
    }
    );
    Console.ReadLine();
}
```



ParallelLoopState Nesnesi

ParallelLoopState nesnesinin **Break** ya da **Stop** özelliklerini kullanarak yürütülmekte olan bir işlemi **durdurabiliriz**.

```
static void Main(string[] args)
{
    Parallel.For(0, 20, (i,durum) =>
    {
        if (i == 10)
            durum.Break();
        else
            Console.WriteLine(i);
    });
    Console.ReadLine();
}
```



Örneğimizde sayım işleminin 20'ye kadar gerçekleştirilmesi gerekirken sayım işlemi 10'a geldiğinde **durum.Break** ile sayım işlemi durdurulmuştur.

Task Yapısı

Paralel olarak yürütülen işlemlerin oluşturulması, zamanlanması ve senkronize edilmesi gibi işlemleri gerçekleştirmek **task** yapısı sayesinde kolayca gerçekleştirilebilir. **Task** yapısı **System.Threading.Tasks** isim uzayı altında yer alır.

Yeni Bir Task Olusturmak

Yeni bir **task** oluşturmak için **Start** metodu ya da **Task.Factory** nesnesinin **StartNew** metodu kullanılır.

Start metodunun kullanım biçimi şu şekildedir:

```
Task TaskİsmI = new Task(( ) => İşlemler);
TaskİsmI.Start();
```

StartNew metodunun kullanım biçimi ise şu şekildedir:

```
Task TaskİsmI = Task.Factory.StartNew(( ) İşlemler);
```

Örnek olarak form üzerinde iki adet **label** yardımıyla iki ayrı saydırma işlemi gerçekleştirelim.

```
using System;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
namespace Task_Olusturmak
{
    public partial class Form1 : Form
    {
      public Form1()
      {
          InitializeComponent();
      }
      private void btnBasla_Click(object sender, EventArgs e)
      {
          CheckForIllegalCrossThreadCalls = false;
          Task task1 = Task.Factory.StartNew(() =>
              {
```

```
            for (int i = 1; i <= 30; i++)
            {
                Thread.Sleep(500);
                label1.Text = i + " ";
            }
            });
        Task task2 = Task.Factory.StartNew(() =>
        {
            for (int j = 1; j <= 30; j++)
            {
                Thread.Sleep(500);
                label2.Text = j + " ";
            }
        });
    }
  }
}
```



Kodumuzda **task1** ve **task2** adlı iki adet ayrı ayrı paralel işlem tanımlayarak bu paralel işlemler içerisinde sayma işlemlerinin yürütülmesini sağladık.

CheckForIllegalCrossThreadCalls = false; satırını kullanarak **cross thread exception** hatalarının göz ardı edilmesini sağladığımızı belirtmiştik. Her ne kadar yazım kolaylığından dolayı bunu tercih etsek de hatayı göz ardı etmek yerine **delegates** yapısını kullanmanızı tavsiye ederim.

Aynı işlemi **delegate** yapısını kullanarak yeniden yazalım:

```
int j, k;
private void btnBasla_Click(object sender, EventArgs e)
{
    Task task1 = Task.Factory.StartNew(() =>
    {
        sayici1();
    });
    Task task2 = Task.Factory.StartNew(() =>
    {
        sayici2();
    });
}
private void sayici1()
{
    for (int i = 1; i <= 30; i++)
    {
        Thread.Sleep(500);
        label1.Invoke(new sayici1_delegate(yeni_sayici1));
    }
}
private void sayici2()
{
    for (int i = 1; i <= 30; i++)
    {
        Thread.Sleep(500);
        label2.Invoke(new sayici2_delegate(yeni_sayici2));
    }
}
private delegate void sayici1_delegate();
private void yeni_sayici1()
{
    j++;
    label1.Text = j + " ";
}
private delegate void sayici2_delegate();
```

```
private void yeni_sayici2()
{
    k++;
    label2.Text = k + " ";
}
```

Invoke metodunu kullanarak **delegate** aracılığı ile tanımladığımız **yenisayici1** ve **yenisayici2** metotlarını çağırırız.

Task İçerisinde Yürütülen İşlemleri Durdurmak

Task içerisinde yürütülen işlemleri durdurmak amacıyla **Wait**, **WaitAll** ve **WaitAny** kullanılır.

Wait: Belirtilen bir işlemi durdurmak amacıyla kullanılır.

```
TaskİsmiWait( )
```

Biçiminde kullanılır.

WaitAll: Belirtilen işlemleri durdurmak amacıyla kullanılır.

```
Task.WaitAll(TaskİsmiTaskİsmi,…)
```

Biçiminde kullanılır.

WaitAny: Herhangi bir işlemi bekletmek amacıyla kullanılır.

```
Task.WaitAny(TaskİsmiTaskİsmi,…)
```

Biçiminde kullanılır.

Sayma işlemi yapan kodumuzu **Wait** kullanarak yeniden düzenleyelim.

```
private void btnBasla_Click(object sender, EventArgs e)
{
    CheckForIllegalCrossThreadCalls = false;
    Task task1 = new Task(() =>
    {
        for (int i = 1; i <= 30; i++)
        {
            Thread.Sleep(500);
            label1.Text = i + " ";
        }
    });
    Task task2 = new Task(() =>
    {
        for (int j = 1; j <= 30; j++)
        {
            Thread.Sleep(500);
            label2.Text = j + " ";
        }
    });
    task1.Start();
    task2.Wait();
}
```

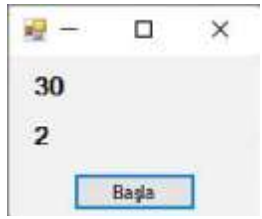


Kodumuzda **task1** işlemi başlatılırken **Wait** ile **task2** işlemini durduruyoruz.

ContinueWith( )

Task içerisinde yürütülen bir işlemin ardından başka bir işlemi başlatmak amacıyla kullanılır.

```
private void btnBasla_Click(object sender, EventArgs e)
{
    CheckForIllegalCrossThreadCalls = false;
    Task task1 = Task.Factory.StartNew(() =>
    {
        for (int i = 1; i <= 30; i++)
        {
            Thread.Sleep(500);
            label1.Text = i + " ";
        }
    })
    .ContinueWith((task2) =>
    {
        for (int j = 1; j <= 30; j++)
        {
            Thread.Sleep(500);
            label2.Text = j + " ";
        }
    });
}
```

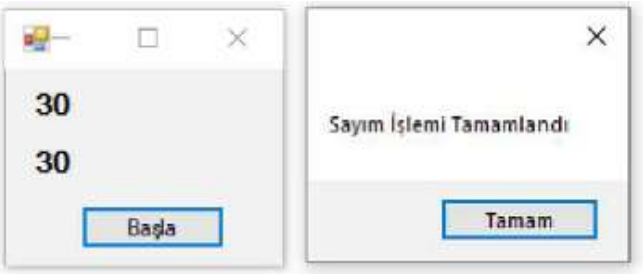


Kodumuzda **task1** işleminin tamamlanmasının ardından **ContinueWith** kullanılarak **task2** işlemi başlatılmaktadır.

IsCompleted( )

`Task` içerisinde yürütülen bir işlemin tamamlanıp tamamlanmadığını öğrenmek amacıyla kullanılır. **`Boolean`** değer alır. Eğer **`true`** değer almışsa işlem **tamamlanmış**, **`false`** değer almışsa işlem **tamamlanmamış** demektir.

```
private void btnBasla_Click(object sender, EventArgs e)
{
    CheckForIllegalCrossThreadCalls = false;
    Task task1 = Task.Factory.StartNew(() =>
    {
        for (int i = 1; i <= 30; i++)
        {
            Thread.Sleep(500);
            label1.Text = i + " ";
        }
    })
    .ContinueWith((task2) =>
    {
        for (int j = 1; j <= 30; j++)
        {
            Thread.Sleep(500);
            label2.Text = j + " ";
        }
      if (task2.IsCompleted == true)
            MessageBox.Show("Sayım İşlemi Tamamlandı");
    });
}
```

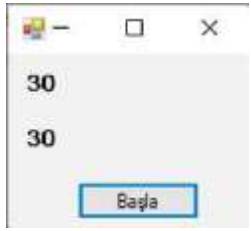


Kodumuzda **`IsCompleted`** kullanılarak **`task2`** işleminin tamamlanmasının ardından ekranda sayım işleminin bittiğine dair bir mesaj penceresi görüntülenmiştir.

Child Task

Bir **task** başlatıldığı zaman bir başka **task**'ı da başlatmak üzere o **task**'a iliştirmek mümkündür. İliştirilen **task**'a **child task**, **child task**'ın iliştirildiği **task**'a ise **parent task** adı verilir. Bir **task**'ın **child task** olabilmesi için **TaskCreationOptions.AttachedToParent** seçeneğinin belirtilmesi gerekir.

```
private void btnBasla_Click(object sender, EventArgs e)
{
    CheckForIllegalCrossThreadCalls = false;
    Task parent = Task.Factory.StartNew(() =>
    {
        Task.Factory.StartNew(() => //Parent Task
        {
            MessageBox.Show("Parent task saymaya başladı");
            for (int i = 1; i <= 30; i++)
            {
                Thread.Sleep(500);
                label1.Text = i + " ";
            }
        });
        Task.Factory.StartNew(() => // Child task
        {
            MessageBox.Show("Child task saymaya başladı");
            for (int j = 1; j <= 30; j++)
            {
                Thread.Sleep(500);
                label2.Text = j + " ";
            }
        }, TaskCreationOptions.AttachedToParent);
    });
}
```



Kodumuzda **parent task**'ın sayma işlemini yürütmeye başlamasıyla birlikte **child task** da diğer sayma işlemini yürütmeye başlamıştır.

Asenkron Metotlar

Asenkron metotlar bir işlemin gerçekleştirilmesi esnasında, diğer işlemlerin de bu ilk işlemin sonuçlanmasını beklemeden yürütülebilmesi amacıyla geliştirilmişlerdir. İşlemler **multi threading** de olduğu gibi farklı thread'ler içerisinde değil, tek bir thread içerisinde gerçekleştirilmektedir.

async ve await

C# 5 ile birlikte kullanmaya başladığımız `async` ve `await` ifadeleri asenkron metotların tanımlanmasında kullanılırlar. `async` ifadesi asenkron bir metodu tanımlamak amacıyla kullanılmaktadır. `await` ifadesi ise **asenkron** olarak belirtilmiş bir metoddaki hangi komutların asenkron çalışacağını belirler. Bir asenkron metodda birden fazla `await` ifadesi kullanılabilir. `async` ifadesi kullanılarak tanımlanan metotlar içerisinde `out` ve `ref` parametreleri kullanılamaz. `async` ifadesi kullanılarak tanımlanan metodun dönüş biçimi aşağıdaki biçimlerden biri olmalıdır:

- `Task`: Eğer metot geriye herhangi bir değer göndermiyorsa kullanılır.
- `Task<TResult>`: Eğer metot geriye `TResult` türünde bir değer gönderiyorsa kullanılır.
- `void`: Dönüş biçimi olarak `void` kullanılırsa `await` ifadesi metot içerisinde kullanılamaz ve oluşan herhangi bir istisna durumu yakalanamaz. `void` metotlarını, `async` ifadesi kullanarak tanımlayabilir ve gerekli olayları içerisinde yazarak kullanabilirsiniz.

Belirtilen bir web adresindeki **mail adreslerini** çeken örnek bir uygulama yapalım. Örneğimiz için form üzerine bir adet `textBox`, bir adet `listBox`, bir adet de `buton` nesnesi ekliyoruz. `textBox` içerisinde web adresleri yer alacak. Sıra ile web adresleri okunarak bulunan mailler `listBox` içerisine aktarılacak.

Örneğimizi öncelikle **async** ve **await** ifadelerini kullanmadan yapalım. Kodumuz aşağıdaki gibi olacak:

```
using System;
using System.Windows.Forms;
using System.Net;
using System.Text.RegularExpressions;
namespace task_await
{
  public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
    }
    private void btnBasla_Click(object sender, EventArgs e)
    {
      foreach (string adresler in textBox1.Lines)
          islemyap(adresler);
    }
    private void islemyap(string adres)
    {
Regex mailDeseni = new Regex
("[\\w-]+(?:\\.[\\w-]+)*@(?:[\\w-]+\\.)+[a-zA-Z]{2,7}");
var client = new WebClient();
foreach (Match isim in
mailDeseni.Matches(client.DownloadString(adres)))
                listBox1.Items.Add(isim.Value);
    }
  }
}
```

Kodumuzda **islemyap** adında bir metot tanmlıyoruz. Metot içerisinde, **adres** parametresine verilen URL adresi içeriğindeki mail adresleri ayıklanmaktadır. **islemyap** metodu **btnBasla** isimli buton içerisinden çağırılmaktadır.

```
foreach (string adresler in textBox1.Lines)
          islemyap(adresler);
```

Satırında **for** döngüsü ile web adreslerinin yer aldığı **textBox1** içerisindeki satırlar sıra ile okunarak **islemyap** metoduna parametre olarak gönderilmektedir.

islemyap metodundaki:

```
Regex mailDeseni = new Regex
      ("[\\w-]+(?:\\.[\\w-]+)*@(?:[\\w-]+\\.)+[a-zA-Z]{2,7}");
```

Satırında web adresinden okunan içeriğin ayıklanmasında kullanılacak **regular expression pattern** (*düzenli ifade deseni*) tanımlanıyor.

```
var client = new WebClient();
```

Satırında web sayfalarına bağlanıp veri çekmek amacıyla **System.Net** sınıfı altında yer alan bir **WebClient** nesnesi tanımlıyoruz.

```
foreach (Match isim in mailDeseni.Matches(client.
DownloadString(adres)))
listBox1.Items.Add(isim.Value);
```

Satırında **foreach** döngüsü içerisinde **DownloadString()** metoduyla, belirtilen web adresinin kaynak kodu indirilerek düzenli ifade desenine uyan e-mail değeri form üzerindeki **listBox**'a ekleniyor. Uygulamamızı bu biçimde çalıştırdığımızda tüm değerler **listBox** içerisine eklenene kadar form penceresi kilitlenir ve form penceresi üzerinde bir işlem gerçekleştiremezsiniz. Ayrıca **textBox** içerisindeki tüm adreslerin taranması bitmeden **listBox** içerisine ekleme yapılamamaktadır.

async ve **await** ifadelerini kullanarak kodumuzu yeniden yazalım.

```
using System;
using System.Net;
using System.Text.RegularExpressions;
using System.Windows.Forms;
namespace task_await
{
  public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
    }
    private void btnBasla_Click(object sender, EventArgs e)
    {
      foreach (string adresler in textBox1.Lines)
          islemyap(adresler);
    }
    private async void islemyap(string adres)
    {
Regex mailDeseni = new Regex
            ("[\\w-]+(?:\\.[\\w-]+)*@(?:[\\w-]+\\.)+[a-zA-Z]{2,7}");
      var client = new WebClient();
      foreach (Match isim in mailDeseni.Matches(await
client.DownloadStringTaskAsync(adres)))
                listBox1.Items.Add(isim.Value);
    }
  }
}
```

Kodumuzda aşağıdaki değişiklikleri yaptık:

```
private void islemyap(string adres)
```

Şeklindeki metot tanımlamamızı **async** ifadesini kullanarak:

```
private async void islemyap(string adres)
```

Şeklinde yeniden tanımladık.

```
foreach (Match isim in mailDeseni.Matches(client
DownloadString(adres)))
```

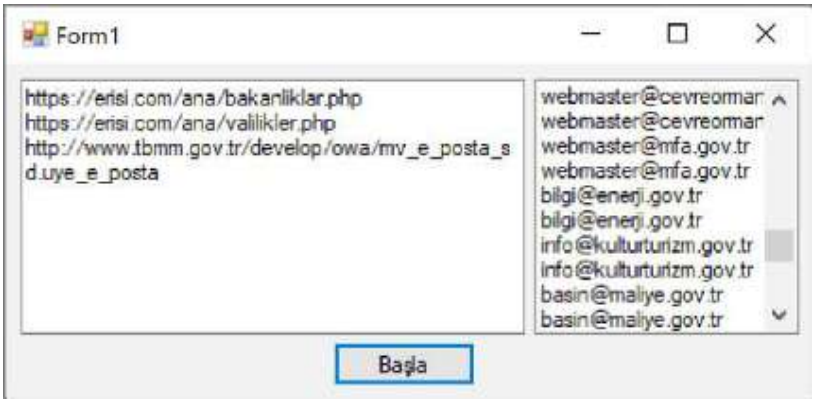
Satırında:

`DownloadString()` metodunu asenkron karşılığı olan **`DownloadStringTaskAsync()`** metodu ile değiştirdik ve metodu **`await`** ifadesini kullanarak çalıştırdık.

```
foreach (Match isim in mailDeseni.Matches(await
client.DownloadStringTaskAsync(adres)))
            listBox1.Items.Add(isim.Value);
```

Sonuç olarak kodumuzda **`islemyap`** metodumuzu başına **`async`** ifadesini ekleyerek bu metot içerisinde **asenkron** metotlar kullanılacağını derleyiciye bildirdik. Bu şekilde kullanılan asenkron metotların yürütülmesi ve işlem sonuçlarının uygulamaya geri döndürülmesi derleyici tarafından yürütülecektir. **`islemyap`** metodu içerisinde **`DownloadStringTaskAsync()`** asenkron metodunu **`await`** ifadesini kullanarak çalıştırdık. Bu metodun çalıştırılması bitinceye kadar uygulamaya geri dönülerek diğer program satırlarının çalıştırılmasına devam edilir. **`await`** ile çağırılan metot bittiğinde ise bir sonraki program satırından uygulama çalıştırılmaya devam edilir.

Uygulamamızı çalıştırdığımızda artık mail adresleri taranırken form kilitlenmeyecek ve **`textBox`** içerisindeki adreslerden her birinin taranması bittikçe bulunan değerler **`listBox`** içerisine eklenecektir.



ASYNC MAIN

C# 7.1 versiyonu öncesinde **Main** metodu **asenkron** olarak kullanamıyorduk. C# 7.1 versiyonundan itibaren **Main** metodu **Task** veya **Task<int>** şeklinde **async** olarak kullanabiliyoruz. Öncelikle **async await** bölümünde örnek olarak yaptığımız uygulamayı **konsol** uygulaması olarak yeniden düzenleyelim. Kodumuz aşağıdaki gibi olacak:

```
static void Main(string[] args)
{
  string[] webadresleri = {"https://erisi.com/ana/bakanliklar.php",
"https://erisi.com/ana/valilikler.php",                "http://www.
tbmm.gov.tr/develop/owa/mv_e_posta_sd.uye_e_posta"
  };
  foreach (string webadresi in webadresleri)
    islemyap(webadresi);
  Console.ReadLine();
}
public static async void islemyap(string adres)
{
  Regex mailDeseni = new Regex
    ("[\\w-]+(?:\\.[\\w-]+)*@(?:[\\w-]+\\.)+[a-zA-Z]{2,7}");
  var client = new WebClient();
  foreach (Match isim in mailDeseni.Matches(await
client.DownloadStringTaskAsync(adres)))
          Console.WriteLine(isim.Value);
}
```

Kodumuzda **Main** metot içerisinde **webadresleri** dizisinde, üzerinde işlem yapılacak web sitesi adresleri bulunmaktadır. Bu dizi içerisindeki web siteleri:

```
foreach (string webadresi in webadresleri)
    islemyap(webadresi);
```

Satırlarında **for** döngüsü ile sıra ile okunarak **islemyap** metoduna parametre olarak gönderilmektedir. **İslemyap** metodu içerisinde **adres** parametresine verilen URL adresi içeriğindeki mail adresleri ayıklanmaktadır. **islemyap** metodu **Main** metot içerisinden çağırılmaktadır.

islemyap metodundaki;

```
foreach (Match isim in mailDeseni.Matches(await
client.DownloadStringTaskAsync(adres)))
            Console.WriteLine(isim.Value);
```

Satırlarında **foreach** döngüsü içerisinde **DownloadStringTaskAsync()** asenkron metodunu **await** ifadesini kullanarak çalıştırdık. **DownloadStringTaskAsync** metoduyla, belirtilen web adresinin kaynak kodu indirilerek düzenli ifade desenine uyan e-mail değeri konsol ekranında yazdırılıyor.

Yukarıdaki yazdığımız kodu, **C# 7.1** versiyonundan itibaren geçerli olan **async Main** metodu kullanacak şekilde yeniden düzenleyelim. Kodumuz aşağıdaki gibi olacak:

```
static async Task Main(string[] args)
{
  Regex mailDeseni = new Regex
     ("[\\w-]+(?:\\.[\\w-]+)*@(?:[\\w-]+\\.)+[a-zA-Z]{2,7}");
  string[] webadresleri = {"https://erisi.com/ana/bakanliklar.php",
"https://erisi.com/ana/valilikler.php",                "http://www.
tbmm.gov.tr/develop/owa/mv_e_posta_sd.uye_e_posta"
  };
  var client = new WebClient();
  foreach (string webadresi in webadresleri)
  {
    foreach (Match isim in mailDeseni.Matches(await
client.DownloadStringTaskAsync(webadresi)))
                  Console.WriteLine(isim.Value);
  }
  Console.ReadLine();
}
```

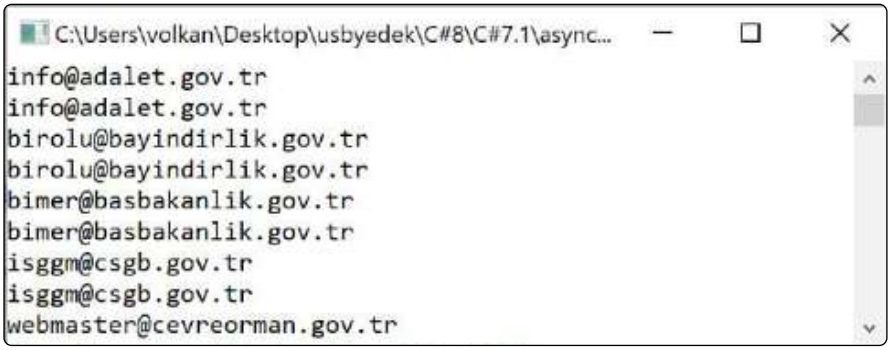
Kodumuzda öncelikle **`main`** metodumuzu:

```
static async Task Main(string[] args)
```

Satırında görüldüğü gibi **`async Task`** metoduna dönüştürüyoruz. Bu şekilde; ayrıca bir asenkron metot tanımlamaya gerek kalmadan doğrudan **`Main`** metodumuzu **asenkron** olarak tanımladık. Web site adreslerini okuyan ve mail adreslerini ayıklayan döngülerimizi ise;

```
foreach (string webadresi in webadresleri)
{
  foreach (Match isim in mailDeseni.Matches(await
    client.DownloadStringTaskAsync(webadresi)))
          Console.WriteLine(isim.Value);
  }
```

Şeklinde iç içe döngüler olacak şekilde yeniden düzenledik.




Muhammed MASTAR - Süha ERİŞ
KODLAB.TV
21. BASKI
ÇOK SATAN
OKU. İZLE. DİNLE. ÖĞREN!
C++
İNTERAKTİF EĞİTİM VİDEOLARI KODLAB'DE!
KODLAB
www.kodlab.com


Bülent ÇOBANOĞLU
KODLAB
2.
BASKI
C/C++
PROGRAMLAMA
KODLAB
www.kodlab.com