Volkan AKTAŞ

HERKES İÇİN!



7. BASKI

HER YÖNÜYLE C# 4.0

KODLAB
www.kodlab.com

HER YÖNÜYLE C#

VOLKAN
AKTAŞ

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

KODLAB® 31

HER YÖNÜYLE C#

VOLKAN AKTAŞ

ISBN 978-605-4205-29-5
Yayıncılık Sertifika No: 13206

1. Baskı: Eylül 2010
2. Baskı: Aralık 2010
3. Baskı: Mart 2011
4. Baskı: Eylül 2011
5. Baskı: Kasım 2011
6. Baskı: Mart 2012
7. Baskı: Ekim 2012

Yayın Yönetmeni: Uğur Gelişken
Grafik Tasarım: Mustafa Said Şahin
Satış Sorumlusu: Can Üstünel
Baskı: İnkilap Kitabevi Yayın San. Tic. A.Ş. Tel: (0212) 496 11 81
Sertifika No: 10614

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.
Alemdar Mah. Salkım Söğüt Sok. Ağa İş Hanı
No: 16 Kat:1 D: 3 Sultanahmet / İSTANBUL

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

Volkan Aktaş

Volkan AKTAŞ, 1976 Mersin doğumlu olup aslen Diyarbakır/Ergani'lidir. Lise öğrenimini Mersin Atatürk Endüstri Meslek Lisesi'nde Teknik Lise Bilgisayar Bölümü'nden, üniversite öğrenimini ise Gazi Üniversitesi Bilgisayar Öğretmenliği bölümünden mezun olarak tamamlamıştır. 1999-2003 yılları arasında Şanlıurfa Endüstri Meslek Lisesi'nde bilgisayar öğretmeni olarak görev yapmış olup, 2003 yılından bu yana da Mersin'de görevine devam etmektedir.

Visual Basic.NET, C#, ASP.NET, VBA, Python dilleri ile aktif olarak uygulamalar geliştirmekte olup bu dillerle yazdığı proje ve makalelerini *www.volkanaktas.com*, *www.vbnetnedir.com*, *www.gelecekcoknet.net*, *www.cyber-warrior.org* sitelerinde yayınlamaktadır. Ayrıca Bilişim Güvenliği ve Bilişim Suçlarına Karşı Mücadele Derneği'nin Mersin temsilciliğini yürütmektedir.

Yazarımızın daha önce KODLAB'tan Visual Basic.NET 2010, Visual Studio 2010, MVC3 Razor ve MVC4 kitapları çıkmıştır.

TEŞEKKÜR

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

Ayrıca içerik ve örneklerin kontrolünde destek olan çalışma arkadaşlarım Bünyamin KARAHAN, Suat ÜRE, Ergün ŞAHİN, Hale KATMER'e, yazım sürecinde bana destek olan Aziz BEKTAŞ, Tahsin SEVİNÇ ve Vedat TAYLAN'a, manevi desteklerini eksik etmeyen Mustafa AKSU, Enver ARSLAN, İsmail GÜVEN, Sait KOTAY, Ayhan ÇELİK, Ahmet IŞIK'a, adlarını burada sayamadığım dostlarıma ve öğrencilerime teşekkür ederim.

ÖNSÖZ

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 yazarlık ve yöneticilik yaptığım web sitelerinde yazdığım makaleler de etkili olmuştur.

.NET yapısının sağladığı güçlü alt yapı birçok projemi C#. NET ile geliştirmemde etkili olmuştur. .NET altındaki diğer dillerden farklı olarak tamamen .NET için geliştirilen bir dil olması nedeniyle, .NET alt yapısını 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 anlatılan C# .NET versiyonu olarak C# .NET 4 versiyonu tercih edilmiştir. Bununla C# .NET'i yeni öğrenmeye başlayan bir okuyucunun en son versiyonundan başlayarak öğrenmeye başlaması amaçlanmıştır. Genel olarak C# .NET temel ve orta seviye konularıyla birlikte C# .NET 4 versiyonuna 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. Hemen her konu örneklerle desteklenerek anlatılmıştır. Anlatılan her örnek denenmiş ve çalıştığı onaylandıktan sonra kitaba eklenmiştir. Kitapta özellikle Console uygulamalarından ziyade Windows Forms uygulamalarına yer vererek konuları açıklamaya gayret ettim. Çünkü kullanıcıların en çok kullandığı uygulama tipleri Windows Forms uygulamaları ile yapılmış projelerdir. Yapılan örnek uygulamalarda 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.

Bu kitabı, kitabın yazım sürecinde
desteğini eksik etmeyen eşime ithaf ediyorum.

İÇİNDEKİLER

1 C#.NET KOD GELİŞTİRME ORTAMI 1

2 KOD YAZMAYA GİRİŞ 17

Framework Nedir? 17
Bir Programın Derleme Süreci 19
Yeni Proje Açmak 22
Kayıtlı Projeleri Açmak 27
Projeleri Kaydetmek 27
Projeleri Derlemek 28
Kod Yazmak 29

3 KONSOL EKRANI 35

Konsol Ekranında Değer Yazma ve Okuma 38
Write ve WriteLine Metodları 38
Read ve ReadLine Metodları 40
Console.In ve Console.Out Özellikleri 42
BackgroundColor ve ForegroundColor Metodları 42
ConsoleKeyInfo Metodu 43

4 SABİTLER VE DEĞİŞKENLER 47

Sabitler 47
Değişkenler 50
Değişken Tipleri 50
Değer Tipleri 50
Referans Tipleri 51
Tanımlandıkları Yerlere Göre Değişken Türleri 54
Kullanıcı Tanımlı Veri Tipleri 58
Structure (Yapı) Oluşturmak 58
Structure İçinde Structure Tanımlamak 62
ENUM Oluşturmak 63
IsDefined Metodu 64
Enum İçerisindeki Elemanlara Erişim 64
System.Enum Sınıfı 72
Değer ve Referans Tipi Değişken Tanımlama 74
Boxing ve Unboxing 76

Değişken Tiplerini Öğrenmek 79
GetType Metodu 79
TypeOf Metodu 80
Değişken Tipleri Arasında Dönüşüm Yapmak 81
Parse Metodu 84
TryParse Metodu 87
Convert Metodu 88

5 OPERATÖRLER 91
Aritmetik Operatörler 91
Aktarma Operatörleri 92
Mantıksal Operatörler 93
& (Ve) Operatörü 93
| (Veya) Operatörü 93
&& (Ve Değil) Operatörü 93
|| (Veya Değil) Operatörü 94
^ (Özel veya) Operatörü 94
! (Değil) Operatörü 94
?? (Null Coalescing) Operatörü 94
?: (Koşul) Operatörü 95
Karşılaştırma Operatörleri 95
Birleştirme Operatörleri 97

6 DİZİLER 99
Dizi Oluşturmak 99
Dizileri Yeniden Boyutlandırmak 101
Dizi Değişkenlerinde Kullanılan Metodlar 102
CreateInstance Metodu 103
IsArray Metodu 103
SetValue Metodu 104
GetValue Metodu 105
Join Metodu 105
Copy Metodu 106
Diziler İçerisinde Arama İşlemleri 107
Contains Metodu ile Arama Yapmak 107
IndexOf Metodu ile Arama Yapmak 108

LastIndexOf Metodu ile Arama Yapmak 110
BinarySearch Metodu ile Arama Yapmak 111
ArrayList Sınıfı ile Dizi Tanımlama ve Bu Sınıf İçerisinde Kullanılan Çeşitli Metod ve Özellikler 113
Add Metodu 113
Insert Metodu 114
Sort Metodu 115
Count Özelliği 116
Remove Metodu 117
RemoveAt Metodu 118
Clear Metodu 118
Reverse Metodu 119
Capacity Özelliği 119
AddRange Metodu 121
InsertRange Metodu 121
RemoveRange Metodu 122
CopyTo Metodu 123
Clone Metodu 126
ToArray Metodu 127
List Nesnesi 126

7 DİYALOG PENCERELERİ 131
Inputbox (Giriş Penceresi) 131
Messagebox (Mesaj Penceresi) 134
File Dialog Pencereleri 140
Color Dialog 140
Folder Browser Dialog 141
Font Dialog 143
Open File Dialog 145
Save File Dialog 150

8 ŞARTLI DALLANMA YAPILARI 153
if else Yapısı 153
switch case Yapısı 156

9 DÖNGÜLER 159
for Döngüsü 159

foreach Döngüsü 163
while Döngüsü 165

10 STRING İŞLEMLERİ 167
String Sınıfı 167
TOSTRING Metodu 167
COPY Metodu 168
CONCAT Metodu 168
EMPTY Metodu 169
COMPARE Metodu 169
SPLIT Metodu 174
JOIN Metodu 178
INDEXOF, INDEXOFANY, LASTINDEX ve LASTINDEXOFANY Metodları 179
SUBSTRING Metodu 182
FORMAT Metodu 183
Custom Numeric Format Strings 185
Standart Date and Time Format Strings 186
Custom Date and Time Format Strings 187
STRINGBUILDER Sınıfı 189
STRINGBUILDER Sınıfı Özellikleri 192
LENGTH Özelliği 192
CAPACITY Özelliği 192
ENSURECAPACITY Özelliği 192
MAXCAPACITY Özelliği 193
STRINGBUILDER Sınıfı Metodları 193
APPEND ve APPENDLINE Metodları 193
APPENDFORMAT Metodu 194
INSERT Metodu 195
REMOVE Metodu 197
REPLACE Metodu 198
CLEAR Metodu 199
MICROSOFT.VISUALBASIC Sınıfı 199
CHR Metodu 200
LEN Metodu 202

MID METODU 202
LEFT ve RIGHT Metodları 204
TRIM, LTRIM ve RTRIM Metodları 205
INSTR Metodu 206
INSTRREV Metodu 208
STRCOMP Metodu 210
UCASE ve LCASE Metodları 212
STRCONV Metodu 214

11 REGULAR EXPRESSIONS (DÜZENLİ İFADELER) 217
Regex Sınıfı ve Metodları 218
IsMatch Metodu 218
Match Metodu 221
Matches Metodu 223
Replace Metodu 228
Split Metodu 229

12 HATALARI BULMAK VE KONTROL ETMEK 231
Try Catch Finally Yapısı 231
Exception Nesnesi 232
Finally Deyimi 236
Throw Deyimi 237
Kullanıcı Tanımlı Exception Nesnesi Oluşturmak 239

13 FORMLAR VE KONTROLLER 243
Formlar 244
Formlar Arası Geçiş 245
Form Nesnesinin Özellikleri 251
Form Boyutu ile İlgili Özellikler 251
Form Olayları 254
Owned Formlar 262
MDI (Multiple Document Interface) Formlar 263
Diğer Kontroller 267
Label Kontrolü 267
LinkLabel Kontrolü 268
TextBox Kontrolü 270
TextBox Kontrolünün Özellikleri 270

TextBox Kontrolünün Metodları 271
TextBox Kontrolünün Olayları 274
Button Kontrolü 274
Button Kontrolünün Özellikleri 275
Button Kontrolünün Olayları 275
ComboBox Kontrolü 276
ComboBox Kontrolünün Özellikleri 276
ComboBox Kontrolünün Metodları 277
ListBox Kontrolü 279
Drag And Drop (Fare İle Sürükleme ve Bırakma) Olayları 283
TreeView Kontrolü 285
TreeView Kontrolünün Özellikleri 285
TreeView Kontrolünün Metodları 287
TreeView Kontrolünün Olayları 287
GroupBox Nesnesi 291
Panel Kontrolü 293
TabControl Nesnesi 295
TabControl Kontrolünün Özellikleri 295
TabControl Kontrolünün Metodları 296
TabControl Nesnesi İçerisine Sekme Eklemek ve Silmek 297
CheckBox Kontrolü 298
CheckBox Kontrolünün Özellikleri 298
CheckBox Kontrolünün Olayları 299
CheckedListBox Kontrolü 300
CheckedListBox Kontrolünün Özellikleri 300
CheckedListBox Kontrolünün Metodları 300
RadioButton Nesnesi 302
RadioButton Kontrolünün Özellikleri 302
Timer Kontrolü 304
Timer Kontrolünün Özellikleri 304
Timer Kontrolünün Metodları 305
Timer Kontrolünün Olayları 305
ProgressBar Kontrolü 311
ProgressBar Kontrolünün Özellikleri 311
PictureBox Kontrolü 312

PictureBox Kontrolünün Özellikleri 312
PictureBox Kontrolünün Metodları 313
MenuStrip Kontrolü 315
MenuStrip Kontrolünün Özellikleri 315
ContextMenuStrip Kontrolü 321
Notifyicon Kontrolü 323
Notifyicon Kontrolünün Özellikleri 323
Notifyicon Kontrolünün Metodları 323
ImageList Kontrolü 326
ImageList Kontrolünün Özellikleri 326
ToolStrip Kontrolü 327
ToolStrip Kontrolünün Özellikleri 327
StatusStrip Kontrolü 333
ToolTip Kontrolü 337
Tooltip Kontrolünün Özellikleri 337
Tooltip Kontrolünün Metodları 339
ErrorProvider Kontrolü 341
ErrorProvider Kontrolünün Özellikleri 341
ErrorProvider Kontrolünün Metodları 342
HelpProvider Kontrolü 344
HelpProvider Kontrolünün Özellikleri 344
HelpProvider Kontrolünün Metodları 345
DateTimePicker Kontrolü 349
DateTimePicker Kontrolünün Özellikleri 349
DateTimePicker Kontrolünün Olayları 353
MonthCalendar Kontrolü 353
MonthCalendar Kontrolünün Özellikleri 353
MonthCalendar Kontrolünün Metodları 357
MonthCalendar Kontrolünün Olayları 357
DriveListBox Kontrolü 358
DriveListBox Kontrolünün Özellikleri 359
DirListBox Kontrolü 359
DirListBox Kontrolünün Özellikleri 359
FileListBox Kontrolü 359
FileListBox Kontrolünün Özellikleri 359

14 RESOURCE DOSYALARI **363**
Resource Dosyaları 363
Resource Dosyası İşlemleri 368
ResourceManager 368
ResXResourceReader 369
GetEnumerator Metodu 369
ResXResourceWriter 369
AddResource Metodu 369

15 ÇİZİM İŞLEMLERİ **375**
Form Üzerine Yazı Çizimi 375
Form Üzerine Şekil Çizimi 380
Şekil ve Yazının Birlikte Kullanımı 391

16 YAZDIRMA İŞLEMLERİ **393**
PrintDocument Sınıfı 393
PrintDocument Sınıfının Özellik ve Metodları 393
PrintDialog Sınıfı 405

17 METODLAR **409**
Değer Döndürmeyen Metodlar 409
Metod İçerisinde Parametre Kullanımı 410
Değer Döndüren Metodlar 417
Kontrollere Metod Bağlamak (Event Handler) 422
Metodların Aşırı Yüklenmesi (Overloading) 423
Temsilciler (Delegates) 425
Anonim (Anonymous) Metotlar 429

18 GENİŞLETME METODLARI (EXTENSION METHODS) **431**
Genişletme Metodlarının Aşırı Yüklenmesi (Overloading) 435

19 CLASS (SINIF) YAPISI **441**
Property Kullanımı 451
Inheritence (Miras) 456
XML Yorum Satırı Eklemek 459
Access Modifiers (Erişim Belirleyiciler) 462
Encapsulation 465

20 USER CONTROL HAZIRLAMA **469**

21 REGISTRY İŞLEMLERİ **475**
Registry Anahtar ve Değerlerine Erişim 476
Registry'e Anahtar ve Değer Ekleme 477
Anahtar ve Değer Sayısını Bulma 481
Anahtar ve Değer Listesini Alma 482
Anahtar ve Değerleri Silme 484
22 WORD ve EXCEL PROGRAMLARINI KULLANMA **489**
Word Programını Kullanma 492
Belge Açma, Kapama ve Kaydetme Olayları 492
Yazı Ekleme ve Düzenleme İşlemleri 498
Range Nesnesi Kullanarak 498
Sentences Nesnesi Kullanarak 499
Paragraph Nesnesi Kullanarak 500
Selection Nesnesi Kullanarak 502
Önizleme Yapma ve Yazdırma İşlemleri 502
Tablo İşlemleri 507
Grafik İşlemleri 511
Excel Programını Kullanma 515
Belge Açma, Kapama ve Kaydetme Olayları 515
Hücreye Erşlim, Yazı Ekleme ve Biçimlendirme İşlemleri 522
Seçim İşlemleri 525
Önizleme Yapma ve Yazdırma İşlemleri 526
Arama İşlemleri 530
Grafik İşlemleri 533
23 WINDOWS API'LERİNİ KULLANMA **537**
Disk Toplam Alanını ve Boş Alanını Bulma 538
Mikroişlemci Hakkında Bilgi Alma 540
RAM Bellek Hakkında Bilgi Almak 542
Masaüstü Resmini Değiştirmek 545
Ekran Çözünürlüğünü Bulma ve Değiştirmek 546
Geri Dönüşüm Kutusunu Kullanmak 552
İnternet Bağlantısını Test Etmek 555
Ses Dosyalarını Çalmak 557
Ses Ayarlarını Değiştirmek 561
Video Dosyalarını Yürütmek 563

Windows 7 API'lerini Kullanmak 566
Diyalog Pencereleri 567
Dosya Açma 568
Klasör Seçme 570
ExplorerBrowser Control 576
Power Manager 581
NetWorkListManager 585
24 DOSYA VE KLASÖR İŞLEMLERİ 587
System.IO Sınıfı Kullanılarak Dosya ve Klasör İşemlerini Gerçekleştirme 587
Dosyalar Üzerinde Yapılan İşlemler 588
File Nesnesi ve Metodları 588
File.Create Metodu 590
File.WriteAll Metodu 592
File.ReadAllLines Metodu 593
Dosya Özelliklerini Elde Etme ve Değiştirme 594
Dosyaları Kopyalama, Taşıma ve Silme 596
FileStream Nesnesi 597
FileStream Nesnesinin Özellikleri 599
FileStream Nesnesinin Metodları 599
Klasörler Üzerinde Yapılan İşlemler 604
Klasör Oluşturma, Silme 606
Dosya İsimlerini Listeleme 607
Klasör Özelliklerini Elde Etme ve Değiştirme 608
Klasörleri Kopyalama ve Taşıma 612
Isolated Storage File Sınıfı ile Dosya ve Klasör İşlemlerini Gerçekleştirme 613
Isolated Storage File Sınıfı Özellikleri 613
Isolated Storage File Sınıfı Metodları 614
Klasör Oluşturma 616
Dosya Oluşturma 617
Dosyaya Yazma 620
Dosyadan Okuma Yapma 621
Dosya ve Klasör Silme 621
25 VERİTABANI İŞLEMLERİ 627
Access Veritabanı İşlemleri 628
OLEDBCOMMAND 635
Kayıt Ekleme 636

Kayıt Güncelleme 637
Kayıt Silme 638
OLEDBDATAADAPTER 647
Kayıt Listeleme 647
Kayıt Ekleme 649
Kayıt Güncelleme 650
Kayıt Silme 650
Kayıt Arama 651
Veritabanını Düzenleme 652
SQL Veritabanı İşlemleri 653
SQLCOMMAND 670
Kayıt Listeleme 671
Kayıt Ekleme 672
Kayıt Güncelleme 673
Kayıt Silme 674
SQLDATAADAPTER 681
Kayıt Listeleme 681
Kayıt Ekleme 683
Kayıt Güncelleme 684
Kayıt Silme 685

26 LINQ (LANGUAGE INTEGRATED QUERY) 693
LINQ Operatörleri 695
WHERE Operatörü 696
SELECT Operatörü 696
DISTINCT Operatörü 699
CONTAINS Operatörü 700
MAX, MIN Operatörü 700
COUNT Operatörü 700
SQL ve LINQ 701
XML ve LINQ 705

27 CRYSTAL REPORTS 709
Report Wizard (Rapor Sihirbazı) Kullanarak Rapor Oluşturma 711
Standard 711
Cross-Tab 713
Manuel Olarak Rapor Oluşturma 725

Grafik Eklemek 738
Oluşturulmuş Olan Rapor Dosyalarını
CrystalReportViewer ile Görüntüleme 743
Rapor Dosyasının İsmini Belirterek Rapor Görüntüleme 743
ReportDocument Nesnesini Kullanarak Rapor Görüntüleme 746
CrystalReportViewer Nesnesinin Özellikleri 747
CrystalReportViewer Nesnesinin Metodları 747

28 THREADING 749

Threadler ile Çalışmak 751
Sleep Metodu 753
Suspend ve Resume Metodları 756
Abort Metodu 756
Join Metodu 757

29 PARALEL PROGRAMLAMA 759

Parallel Sınıfı 760
Parallel.Invoke Metodu 760
Parallel.For ve Parallel.ForEach Metodları 762
ParallelLoopState Nesnesi 764
Task Yapısı 765
Yeni Bir Task Oluşturmak 765
Task İçerisinde Yürütülen İşlemleri Durdurmak 768
ContinueWith( ) Metodu 769
IsCompleted( ) Metodu 770
Child Task 771

30 SETUP HAZIRLAMA İŞLEMLERİ 773

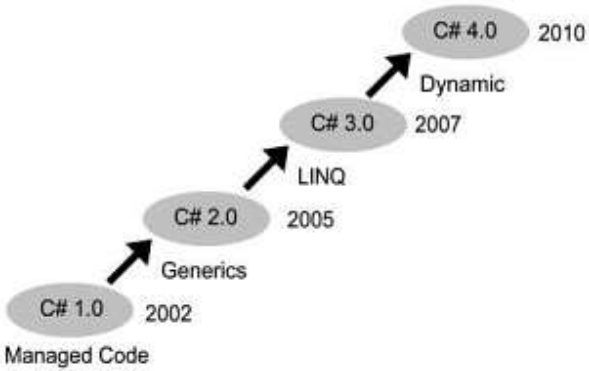
Application Folder 775
User's Desktop 775
User's Programs Menu 775

1

C#.NET KOD GELİŞTİRME ORTAMI

Microsoft firması tarafından 1999 yılı 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 metodlar*) gibi programlama yapıları kullanılmaya başlandı.

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

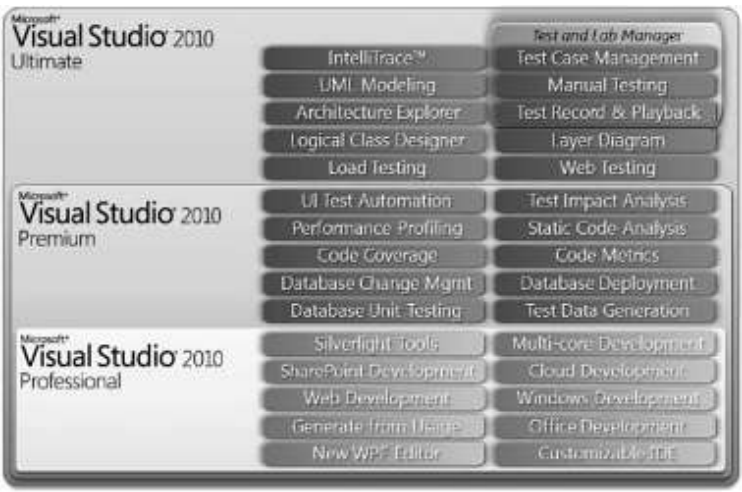
4.0 versiyonu ile birlikte **dynamic veri tipi**, **named** ve **optional parameters** gibi programlama yapıları kullanılmaya başlandı.

C# programlama dili, TurboPascal, Delphi, J++ gibi dillerinin 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# dilleri ve web sitesi tasarımı, kodlaması yapabileceğimiz ASP.Net adlı web programlama dili mevcuttur.

Visual Studio 2010'un **ultimate**, **premium** ve **professional** adlı üç sürümü mevcuttur. Bu sürümler dışında ücretsiz olarak indirip kullanabileceğimiz bazı özellikleri kısıtlanmış olan express sürümü de mevcuttur.

Aşağıdaki resimde görüldüğü gibi en kapsamlı özelliklere sahip olan sürümü **Ultimate** sürümüdür. Diğer sürümlerdeki özelliklere ek olarak debugging, test ve analiz araçlarına sahiptir. **Premium** sürümü ise Professional sürümdeki özelliklere ek olarak analiz araçlarına ve gelişmiş veritabanı desteğine sahiptir.



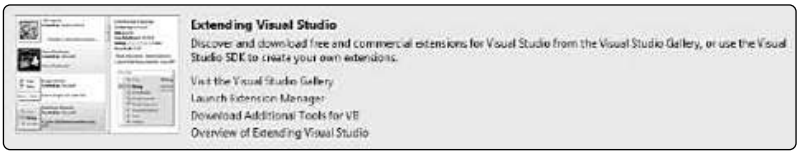
Visual Studio .NET ilk açıldığında karşımıza **Start Page** adlı başlangıç ekranı gelir. Bu bölümde **GetStarted**, **Guidance and Resources** ve **Latest News** adlı üç ana başlık vardır.



Welcome başlığı altında,

- Visual Studio 2010'un ve .Net Framework 4.0'ın genel özelliklerinin açıklandığı,
- Visual Studio .NET ile ilgili yardım ve destek alabileceğimiz ve Visual Studio .NET ile yapılmış kod ve projeleri bulabileceğimiz,
- Extension'lar ile ilgili bilgi alabileceğimiz,
- Visual Studio 2010 ile ilgili haberler, etkinlikler ve ürün bilgilerine erişebileceğimiz Microsoft'un ilgili web site adreslerinin linkleri bulunur.

Extension'lar ile ilgili bilgi alabilmek ve extension'lar ile ilgili işlemleri yapabilmek için **Welcome** başlığı altındaki **Extending Visual Studio** seçeneği kullanılabilir.



Launch Extension Manager linki tıklanarak Visual Studio içerisinde yüklü olan extension ve güncellemeleri görebileceğiniz ya da yükleyebileceğiniz **Extension Manager** penceresi açılır. Bu menüye **Tools** menüsünden **Extension Manager** seçeneği ile de erişilebilinir.

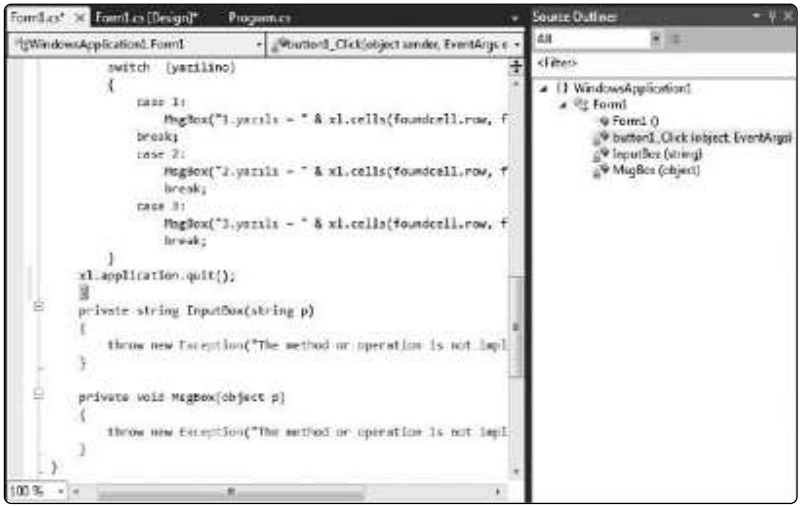
Online Gallery bölümünde extension seçip yükleyebiliriz.



Installed Extensions bölümünde ise yüklü olan extension isimlerini seçip pasif yapabilir ya da Visual Studio içerisinden tamamen kaldırabiliriz.



Eklemiş olduğumuz extension Visual Studio kapatılıp yeniden açıldığında aktifleşecektir.



Yüklenen extension *C:\Users\bilgisayaradı\AppData\Local\Microsoft\VisualStudio\10.0\Extensions\Microsoft* klasörüne yüklenecektir. Bu klasörde mevcut olan extension'lar içerisinden açık kaynak kodlu olanlar üzerinde istenilen düzenlemeler yapılabilir.



Updates kısmında ise güncellemeleri görüp, gerekli güncellemeleri gerçekleştirebiliriz.

Windows başlığı altında Windows Forms, WPF, Silverlight gibi teknolojileri kullanarak Visual Studio 2010 ile Windows uygulamaları geliştirme konusundaki bilgilere ve kaynaklara erişebileceğimiz, Microsoft'un ilgili forum ve web site adreslerinin linkleri bulunur.

Web başlığı altında, ASP.NET dilini kullanarak Visual Studio 2010 ile Web uygulamaları geliştirmek konusundaki bilgilere ve kaynaklara erişebileceğimiz, Microsoft'un ilgili forum ve web site adreslerinin linkleri bulunur.

Cloud başlığı altında, Windows Azure platformunu kullanarak Visual Studio 2010 ile cloud uygulamaları geliştirmek konusundaki bilgilere ve kaynaklara erişebileceğimiz, gerekli araç ve yardım dosyalarını indirebileceğimiz, Microsoft'un ilgili forum ve web site adreslerinin linkleri bulunur.

Cloud computing, kullanıcıların uygulamaları kendi sistemlerine kurmadan sunucu sistemleri vasıtası ile web tarayıcıları üzerinde çalıştırabildiği sistemdir. Bu işlemlerin yönetilmesi **Windows Azure** adlı işletim sistemi vasıtası ile gerçekleştirilmektedir. Visual Studio 2010 vasıtası ile Azure platformu ile cloud uygulamaları geliştirmek mümkündür.

Office başlığı altında, **VSTO** (**V***isual* **S***tudio* **T***ool for* **O***ffice*) alt yapısını kullanarak Visual Studio 2010 ile ofis uygulamaları geliştirmek konusundaki bilgilere ve kaynaklara erişebileceğimiz, Microsoft'un ilgili forum ve web site adreslerinin linkleri bulunur.

VSTO (**V***isual* **S***tudio* **T***ool for* **O***ffice*) Word, Excel gibi ofis yazılımları için uygulamalar geliştirmek üzere şablon ve kod desteği sağlar.

Sharepoint başlığı altında, Sharepoint alt yapısını kullanarak Visual Studio 2010 ile intranet uygulamaları geliştirmek konusundaki bilgilere ve kaynaklara erişebileceğimiz, Microsoft'un ilgili forum ve web site adreslerinin linkleri bulunur.

Sharepoint özellikle iş uygulamaları geliştirmek üzere kullanılan bir sistemdir. Sharepoint ile doküman ve içerik yönetimi, portal uygulamala-

rı geliştirme, raporlama, projeler üzerinde ekip halinde çalışabilme kurumsal nitelikli bir çok işlemler gerçekleştirebilirsiniz.

Data başlığı altında, **ADO.NET**, **LINQ** gibi yapılar kullanarak Visual Studio 2010 ile veritabanı uygulamaları geliştirmek konusundaki bilgilere ve kaynaklara erişebileceğimiz, Microsoft'un ilgili forum ve web site adreslerinin linkleri bulunur.

Guidance And Resources başlığı altında bulunan seçenekler şunlardır:

- **Overview of Development Processes:** Kod geliştirme süreçleri ile ilgili Microsoft'un ilgili web adreslerini açar.



- **Managing your source code in Visual Studio: Team Foundation Server** ile ilgili geniş bilgi alabileceğimiz Microsoft'un ilgili web adresini açar. Team Foundation Server takım halinde projeler geliştirmek, proje sürecini kısaltmak, verimli hale getirmek, proje yönetimi sağlamak gibi amaçlarla kullanılan ve Microsoft tarafından üretilmiş bir üründür.



- **Unit Testing in Visual Studio:** Visual Studio 2010 ile gerçekleştirilebilecek **unit testing** (*birim testi*) işlemleri hakkında geniş bilgi alabileceğimiz Microsoft'un ilgili web adresini açar. **Unit testing** sayesinde proje içerisindeki birimlerin doğru şekilde çalışıp çalışmadığı test edilir.



- **Use Visual Studio for Test Driven Development:** Visual Studio 2010 ile gerçekleştirilebilecek **test driven development** (*test güdümlü yazılım geliştirme*) işlemleri hakkında geniş bilgi alabileceğimiz Microsoft'un ilgili web adresini açar. Test driven development ile belirlenen test koşullarına uygun şekilde proje süreci gerçekleştirilir.



Latest New başlığı altında ise abone olunan sitelerin RSS bilgileri görüntülenir.

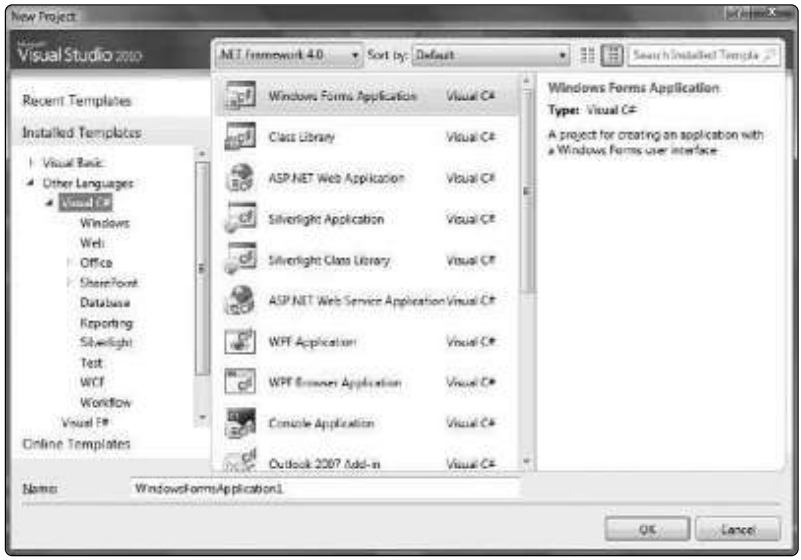


Projelerle ilgili işlemleri ise **Start Page** sayfasının sol tarafındaki menüden gerçekleştirebiliriz. Burada yer alan seçenekler şunlardır:

Open Project seçeneği ile daha önceden mevcut olan projelerden biri seçilerek açılır.

New Project seçeneği ile yeni bir proje açılır. Bu pencerede bulunan diller içerisinden ve mevcut olan proje şablonlarından birisi seçilerek proje geliştirilebilir.





Recent Templates ile son kullanılan proje şablonları seçilebilir.

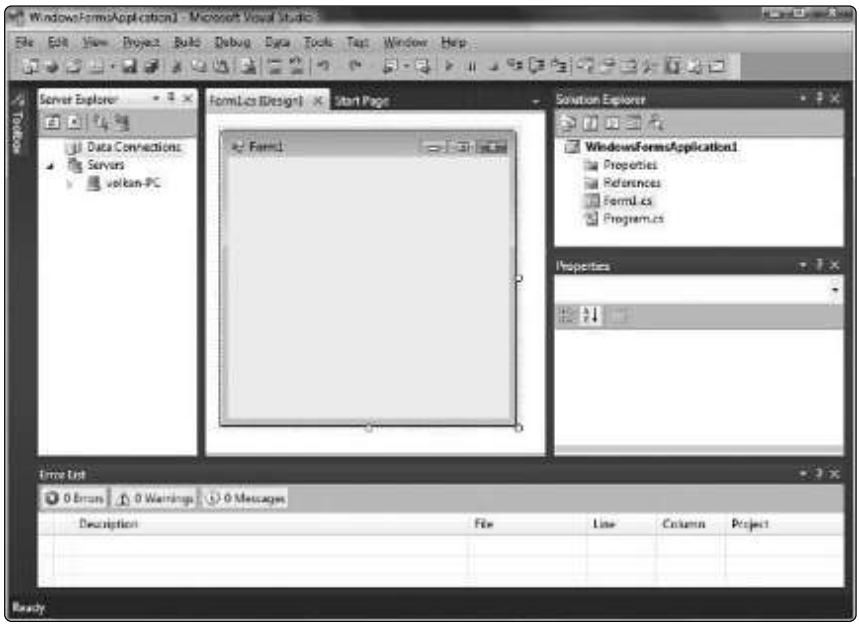
Installed Templates ile yüklü olan proje şablonları listelenir.

Online Templates ile internet ortamında mevcut bulunan proje şablonları listelenir.

Recent Projects seçeneği ile son kullanılan proje listesine erişilebilir. Burada varsayılan olarak son 10 adet proje listelenmektedir. Bu sayıyı değiştirmek için **Tools** menüsünden **Options** seçenekleri ile gelen penceredeki **Recent Files** başlığı altındaki "10 items shown in Window menu" seçeneği kullanılır. "10 items shown in recently used lists" seçeneği ile ise **File** menüsü altındaki **Recent Files** seçeneği ile görüntülenen son kullanılan proje listesi değiştirilebilir.



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



1. **Menü Window:** İki bölümden oluşur:

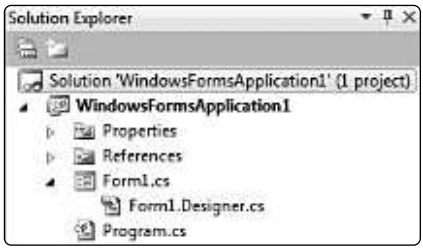
   a. Menü Çubuğu



b. Araç Simgesi Çubuğu



2. **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 yüklü olan form, modül gibi proje bileşenlerini görüntülemek amacı ile **Solution Explorer** adlı Visual Studio bileşeni kullanılır.

Bir proje form ve kod olarak iki temel kısımdan oluşur. Form görünümünde projemizin görsel öğelerini kod yazmadan ekleyip düzenleyebiliriz. Kod görünümünde ise projemizin gerçekleştirmesi gereken işlere dair kod satırlarını yazarız.

Kod görünümüne geçiş yapmak için ise şu yöntemler kullanılabilir:

a. **Solution Explorer** penceresindeki  düğmesi tıklanarak,

b. **View** menüsünden **Code** seçeneği tıklanarak,

c. **F7** kısayol tuşu ile.

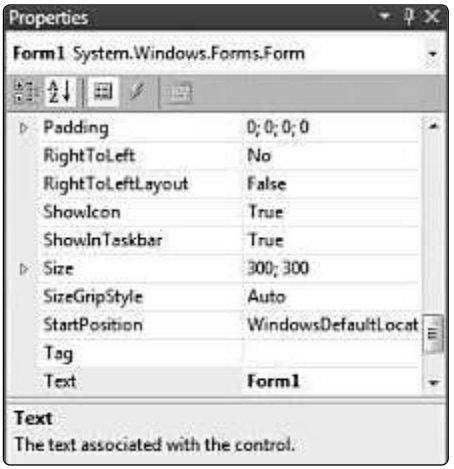
Form görünümüne geçiş yapmak için ise şu yöntemler kullanılabilir:

a. **Solution Explorer** penceresindeki  düğmesi tıklanarak,

b. **View** menüsünden **Designer** seçeneği tıklanarak,

c. **Shift+F7** kısayol tuşu ile.

3. **Properties Window:** Seçili olan nesnenin özelliklerini kod yazmadan değiştirmek amacı ile kullanılır.



Bu pencere üzerinde bulunan butonlar yardımı ile **Properties** penceresi üzerinde çeşitli işlemler yapılabilir.

Bu butonlar ve görevleri şu şekildedir:

Categorized: Properties penceresindeki özelliklerin benzer özelliklerine göre gruplandırılarak listelenmesini sağlar.

Alphabetical: Properties penceresindeki özelliklerin alfabetik sıraya göre gruplandırılarak listelenmesini sağlar.

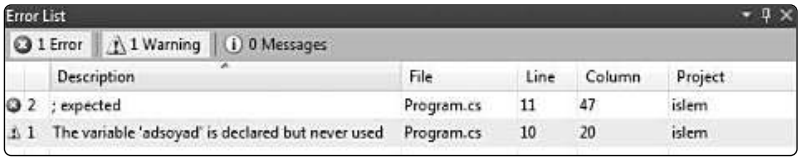
Properties: Properties penceresinde seçili olan nesneye ait özelliklerin listelenmesini sağlar.

Events: Properties penceresinde seçili olan nesneye ait olayların listelenmesini sağlar.

4. **Server Explorer:** Yeni bir veritabanı oluşturmak, oluşturulmuş olan veritabanları içerisinde düzenlemeler yapmak amacı ile kullanılır.



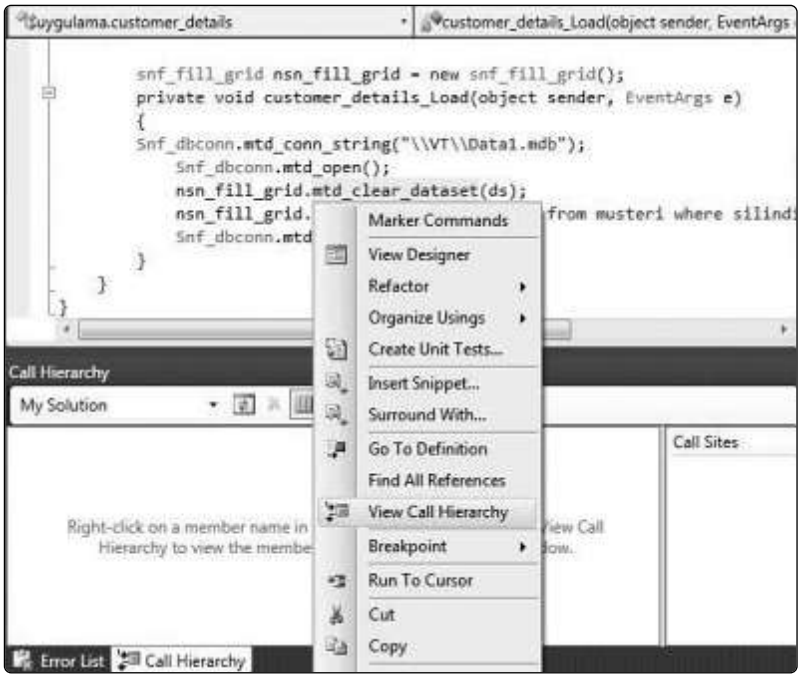
5. **Error List:** Proje süreci içerisinde oluşan hata ve uyarı mesajlarının görüntülendiği penceredir. Programın çalışmasını kesintiye uğratacak hatalar (x) işareti ile, programın çalışmasını kesintiye uğratmayıp programcıya yol gösterme amaçlı uyarılar ise (!) işareti ile gösterilir.



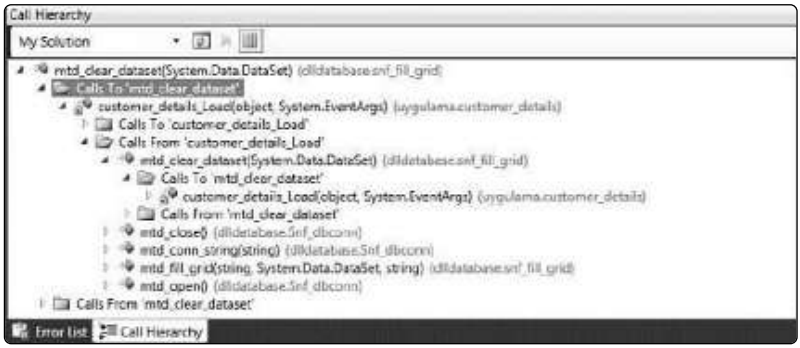
Örneğin yukarıdaki **Error List** penceresini incelediğimizde program içerisinde tanımlanıp kullanılmayan adsoyad değişkeni programın çalışmasını engellemediği için uyarı biçiminde görüntülenmektedir. Ancak bir komut satırının sonuna noktalı virgül konulmaması programın çalışmasını engellediği için hata biçiminde görüntülenmektedir.

6. **Call Hierarchy:** Bir sınıfın, bir tipin, bir alt programın metotlarını, özelliklerini, parametrelerini görüntüleyerek nerelerden çağırıldığı, hangi sınıfa ait olduğu gibi bilgileri verir.

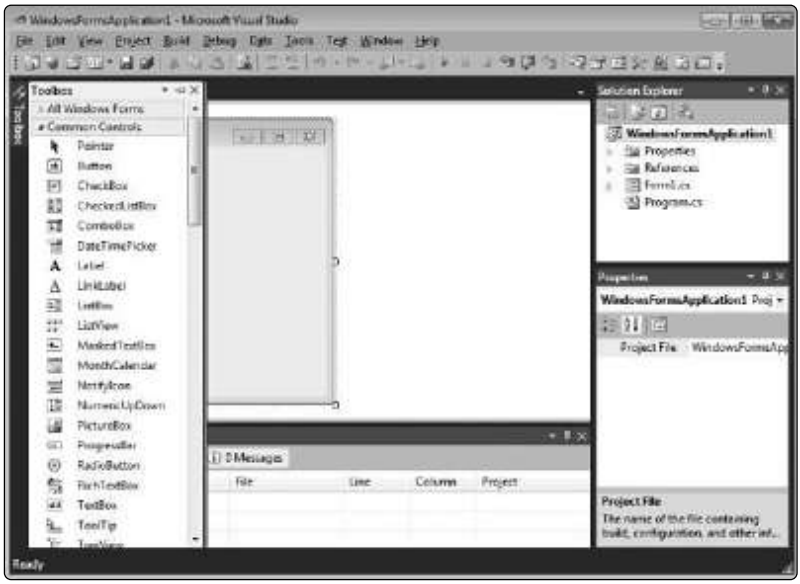
   Örneğin kodumuzdaki bir sınıfa ait bir metoda farenin sağ tuşu ile tıklayarak **View Call Hierarchy** seçeneğine tıklayalım.



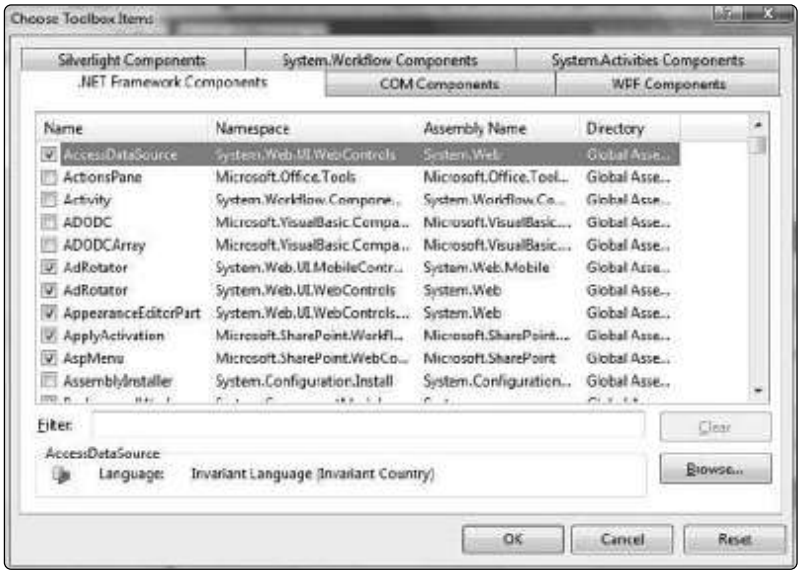
Bu metodun çağırıldığı yer ile bilgilerin listelendiğini görürüz.



7. **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.


Choose Toolbox Items
Silverlight Components
System.Workflow Components
System.Activities Components
.NET Framework Components
COM Components
WPF Components
Name
Namespace
Assembly Name
Directory
AccessDataSource
System.Web.UI.WebControls
System.Web
Global Asse...
ActionsPane
Microsoft.Office.Tools
Microsoft.Office.Tool...
Global Asse...
Activity
System.Workflow.Compone...
System.Workflow.Co...
Global Asse...
ADODC
Microsoft.VisualBasic.Compa...
Microsoft.VisualBasic...
Global Asse...
ADODCArray
Microsoft.VisualBasic.Compa...
Microsoft.VisualBasic...
Global Asse...
AdRotator
System.Web.UI.MobileContr...
System.Web.Mobile
Global Asse...
AdRotator
System.Web.UI.WebControls
System.Web
Global Asse...
AppearanceEditorPart
System.Web.UI.WebControls...
System.Web
Global Asse...
ApplyActivation
Microsoft.SharePoint.Workfl...
Microsoft.SharePoint...
Global Asse...
AspMenu
Microsoft.SharePoint.WebCo...
Microsoft.SharePoint
Global Asse...
AssemblyInstaller
System.Configuration.Install
System.Configuration...
Global Asse...
Filter:
Clear
AccessDataSource
Language:
Invariant Language (Invariant Country)
Browse...
OK
Cancel
Reset

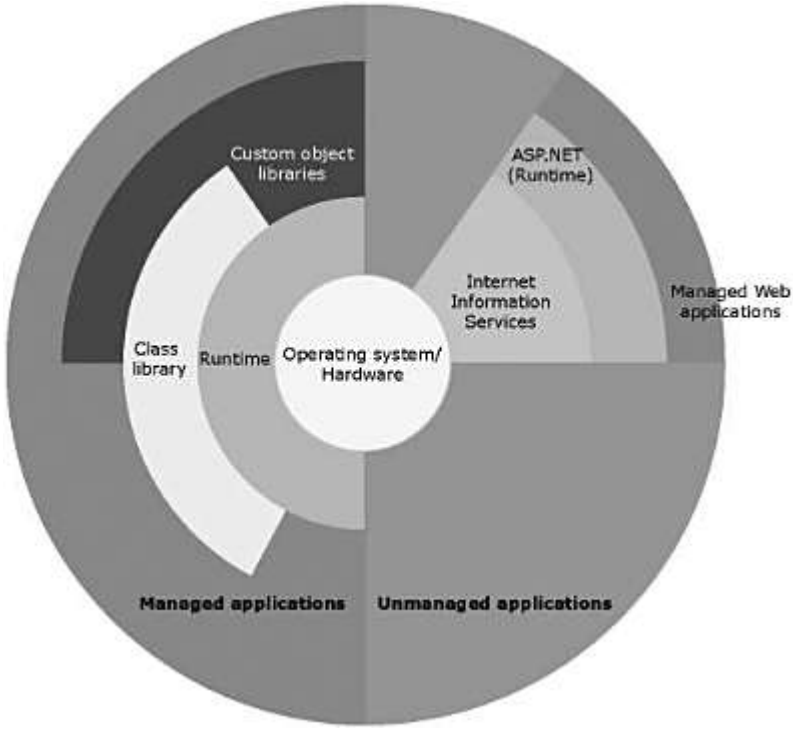
2

KOD YAZMAYA GİRİŞ

Visual Studio arabirimi (IDE) bizlere form tasarımı, kod yazımı, hata yakalama, derleme, çalıştırma gibi işlemler için gerekli araçları sağlamaktadır. Bazı işlemler kod yazmadan, bazıları kod yazılarak bazıları da her iki şekilde Visual Studio arabirimi ile gerçekleştirilebilmektedir.

FRAMEWORK NEDİR?

Framework yeni nesil Windows uygulamaları oluşturulması, çalıştırılması ve yönetilmesi gibi işlemler için gerekli alt yapıyı sağlayan bir platformdur. Windows uygulamaları ile Windows işletim sistemleri arasında bir katman görevi görür. İşletim sistemleri ile bağlantıyı ve entegrasyonu sağlayan kütüphaneler ve araçlara sahiptir. Bu sayede işletim sisteminden ve donanımdan bağımsız olarak çalışmayı ve uygulama taşınabilirliğini sağlar.



Managed Applications (*Yönetilebilir Uygulamalar*) .NET ile yapılan uygulamalardır. **Unmanaged Applications** (*Yönetilmeyen Uygulamalar*) ise .NET dışındaki uygulamalardır.

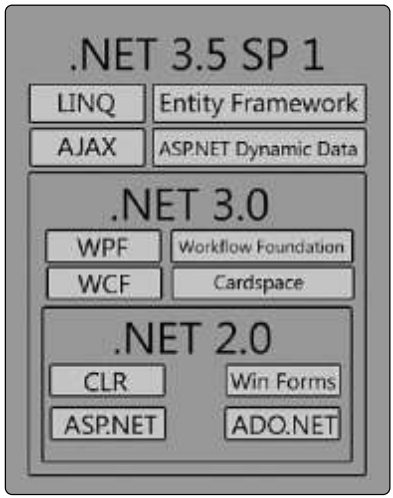
Managed uygulamalarda bellek yönetimi .NET tarafından **Garbage Collector** gibi bir mekanizma ile yürütülerek işlemler için hafızada yer açma, işlemler bittiğinde ya da artık kullanılmadığında hafızayı temizleme gibi işlemler gerçekleştirilir.

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

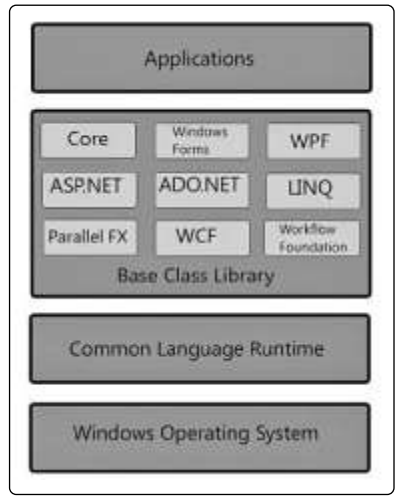
Framework yalnızca .NET dillerine değil bunun yanı sıra .NET dışında Python, Pascal, C++, Cobol, Java, Scheme gibi 50 civarında programlama diline de destek vermektedir. Desteklenen tüm dillerin isimlerine *http://en.wikipedia.org/wiki/CLI_Languages* adresinden erişebilirsiniz.

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 Vista Framework 2.0 kurulu olarak kullanıma sunulurken, Windows 7 Framework 3.5 kurulu olarak kullanıma sunulmaktadır.

Visual Studio kurulduğu zaman kurulan sürüme göre Framewok sürümleri yüklenmektedir. Örneğin Visual Studio 2008 Framework 3.5 versiyonu ile birlikte gelirken, Visual Studio 2010 ise Framework 4.0 versiyonu ile birlikte gelmektedir.



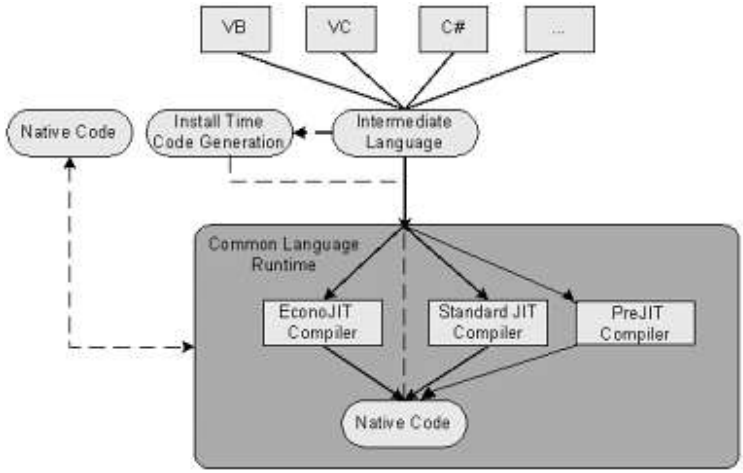
Framework 4 öncesi sürümleri



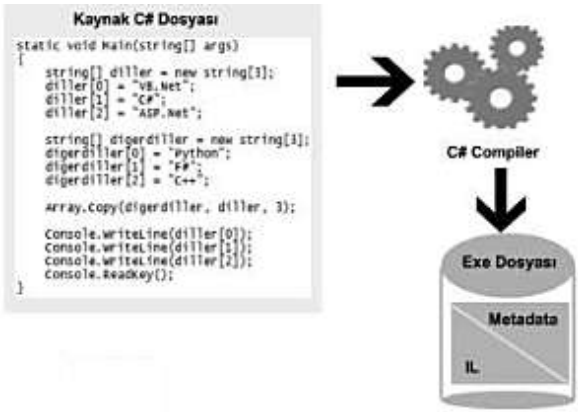
Framework 4

Bir Programın Derleme Süreci

C# içerisinde bir programı derlemek için **F5** tuşuna basarız ve programımız çalışarak ekranda sonuçları görüntüler. Acaba bu süreçte ne olmaktadır. Bu sürece kısaca değinelim.



Program kodu derleme anında **IL** (*Microsoft* **I***ntermediate* **L***anguage-Microsoft Ara Dili*) 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** (**C***ommon* **L***anguage* **R***untime-Ortak Dil Çalışma Zamanı Platformu*) içerisinde **JIT** (**J***ust* **I***n* **T***ime*) tarafından derlenerek makine diline çevrilir ve hafızaya yüklenerek çalıştırılır. Oluşturulan kod içerisinde yalnızca IL değil aynı zamanda metadata bilgileri de bulunur. Metadata içerisinde veri tipleri, kullanılan metodlar ve sınıflarla ilgili bilgiler bulunur.



Varsayılan olarak derleme işlemi **Standart JIT Compiler** tarafından gerçekleştirilirken bunun dışında **EconoJIT** ve **PreJIT** derleyiciler de kullanılabilir.

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

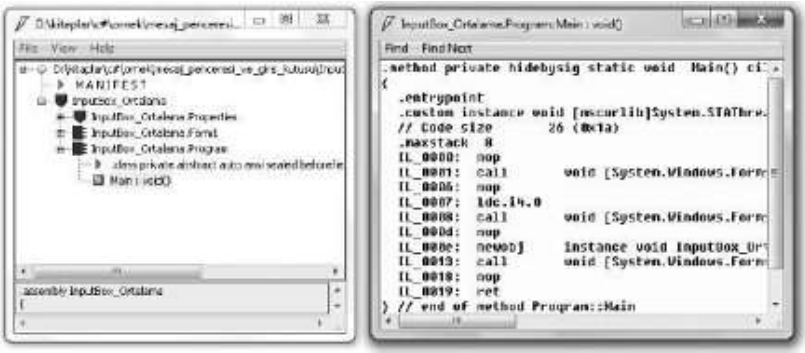
EconoJIT Compiler ise sınırlı sistem kaynakları için dizayn edilmiştir. Ram belleğin yeterli olmadığı durumlarda kullanılabilir. Derlenen metodlar ihtiyaç duyulmadığında silinir. Derlenen kısımlar bellekte depolanmayıp her defasında yeniden çalıştırıldığı için hafıza kaynaklarını az kullanır ancak Standart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

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

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

.NET ile yazdığımız herhangi bir uygulamanın MSIL kodunu görmek için "Programlar / Microsoft Visual Studio / Microsoft Windows SDK Tools" menüsü altında bulunan **IL Disassembler** aracını kullanabiliriz. **IL Disassembler** içerisinde **File** menüsünden **Open** seçeneği ile uygulamamızın exe dosyasını açarak IL kodunu görüntületebiliriz.

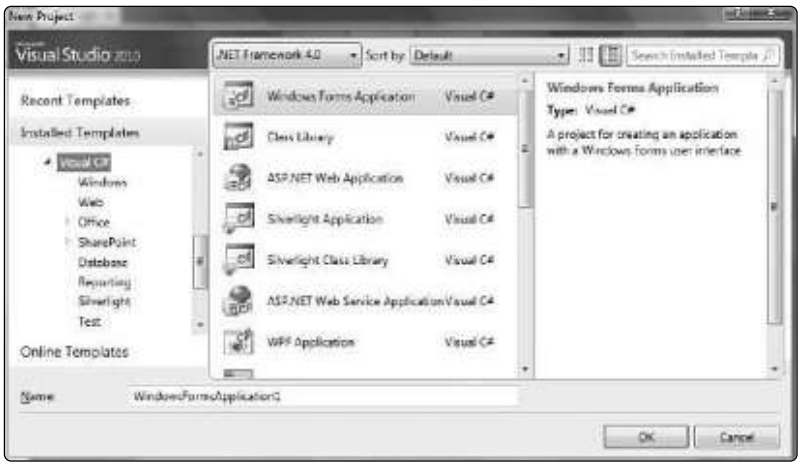


YENİ PROJE AÇMAK

Yeni bir proje açmak için şu seçenekler kullanılabilir:

- **Start Page** ekranından **Projects** başlığı altındaki **New Project** simgesi tıklanabilir.
- File menüsünden **New Project** seçeneği kullanılabilir.
- **Ctrl+N** tuşları kullanılabilir.
- Araç çubuğundan  simgesi tıklanabilir.

Bu seçeneklerden herhangi biri kullanılarak **New Project** penceresi görüntülenir. Bu pencereden mevcut olan proje şablonlarından birisi seçilerek yeni bir proje oluşturulabilir.



C# .NET ile kullanılabilecek başlıca proje tipleri şunlardır:

Windows Forms Application: İçerisinde windows formları bulunan masaüstü uygulamaları geliştirmek için kullanılır.

Console Application: İçerisinde windows formları bulunmayan komut satırı uygulamaları geliştirmek için kullanılır.

WPF Application: Görsel açıdan üstün niteliklere sahip **WPF** (**W***indows* **P***resentation* **F***oundation*) masaüstü uygulamaları geliştirmek için kullanılır.

ASP.NET Web Application: İnternet ortamında çalışacak ASP.NET uygulamaları geliştirmek için kullanılır.

Silverlight Application: Görsel açıdan üstün niteliklere sahip Silverlight internet uygulamaları geliştirmek için kullanılır.

Class Library: Dll dosyalarından oluşan sınıflar geliştirmek için kullanılır.

Windows Forms Control Library: Windows form uygulamalarınızda kullanabileceğiniz kontroller geliştirmek için kullanılır.

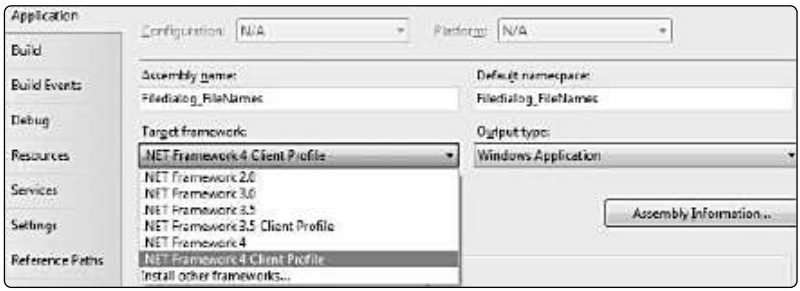
Windows Service: Arkaplanda çalışan Windows servisleri geliştirmek için kullanılır.

Burada listelenen şablon sayısı seçili olan Framework versiyonuna göre değişmektedir.

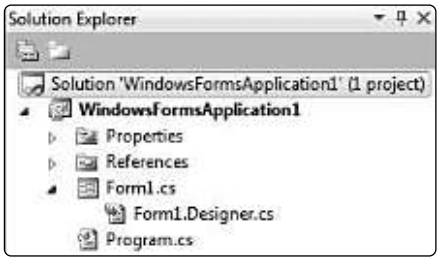


Örneğin Framework 2.0 versiyonunda yaklaşık 10 civarında şablon mevcutken, Framework 4.0 versiyonunda MVC, LINQ, WPF, Cloud Computing, Sharepoint gibi teknolojilerinde devreye girmesiyle şablon sayısı 30 civarına yükselmiştir.

Framework versiyonunu yeni bir proje oluştururken yukarıdaki resimde görüldüğü gibi pencerenin üst tarafındaki açılan liste içerisinden istediğimiz framework versiyonunu seçip değiştirebilirken, daha önceden yazdığımız programların framework versiyonunu değiştirmek için ise 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.
- Bu işlemin ardından projemiz otomatik olarak kapatılarak seçtiğimiz framework versiyonu ile yeniden açılacaktır.



Yeni bir proje oluşturup solution explorer içerisini incelediğimizde standart olarak bazı dosyaların oluşturulduğunu görürüz. Bunlar; *Form1.cs*, *Form1.Designer.cs* ve *Program.cs* dosyalarıdır.

Form1.cs: Formumuzun kodlarını yazdığımız kısımdır.

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

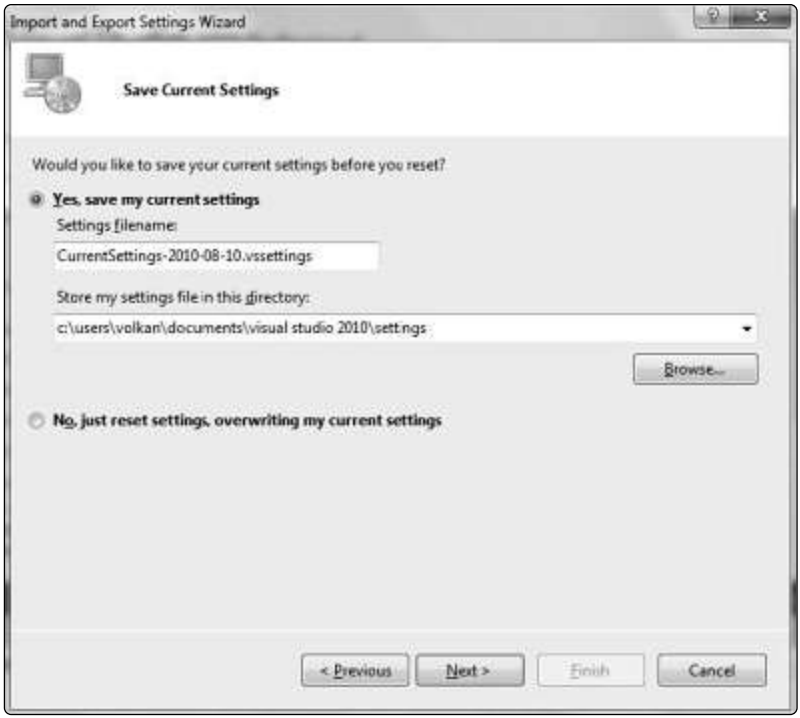
Program.cs: Projemizin ortam ayarları ile ilgili işlemlerin yapıldığı dosyadır. Örneğin bu dosya içerisinden proje içerisinde birden fazla form var ise bunlardan hangisinin başlangıç formu olarak görüntüleneceğini belirleyebiliriz.

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, ayar dosyası ismi ve klasörü seçilerek **Finish** butonu tıklanır.



KAYITLI PROJELERİ AÇMAK

Daha önceden mevcut olan projeleri açmak için şu seçenekler kullanılabilir:

- **Start Page** ekranından **Projects** başlığı altındaki **Open Solution** simgesi tıklanabilir.
- **File** menüsünden **Open Project** seçeneği kullanılabilir.
- **`Ctrl+O`** tuşları kullanılabilir.
- Araç çubuğundan  simgesi tıklanabilir.

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

PROJELERİ KAYDETMEK

Bir projeyi kaydetmek için şu seçenekler kullanılabilir:

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

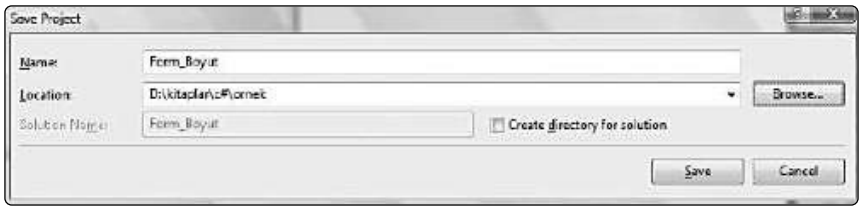
Bu seçeneklerden herhangi biri kullanılarak **Save Project** penceresi görüntülenir. Varsayılan olarak solution ile aynı ada sahip bir proje klasörü oluşturulur.



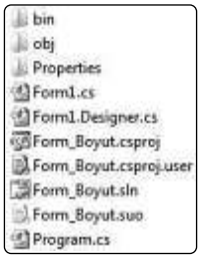
Eğer solution içerisinde birden fazla proje mevcutsa her proje için ayrı bir klasör oluşturulur.



Kayıt esnasında eğer solution ile proje dosyalarının aynı klasörde olması istenirse **Save Project** penceresinde yer alan "Create directory for solution" seçeneğinin işareti kaldırılmalıdır.



Proje klasörü içerisine baktığımızda farklı uzantılara sahip dosyalar görürüz. C# içerisinde kullanılan başlıca dosya uzantıları şunlardır:



.csproj: Proje dosyası uzantısıdır.

.csproj.user: Bir C# projesi için kullanıcı seçeneklerini içerir.

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

.suo: Solution dosyası ile ilgili kullanıcı bilgilerini içerir.

.cs: Proje içerisinde kullanılan formların kaynak kodlarını içerir.

.resx: Resource dosyaları tanımlamak ve düzenlemek amacı ile kullanılır.

.rpt: Crystal Report dosyası uzantısıdır.

PROJELERİ DERLEMEK

Bir projeyi derlemek için şu seçenekler kullanılabilir:

- **Debug** menüsünden **Start Debugging** seçeneği kullanılabilir.
- **F5** tuşu kullanılabilir.
- Araç çubuğundan  simgesi tıklanabilir.

Derleme esnasında projeyi duraklatmak için ise...

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

Derleme işlemi esnasında oluşan exe dosyası proje klasörü içerisindeki **Bin** klasörü altında bulunan **Debug** klasörü içerisinde oluşturulur.

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

Build menüsü altındaki **Build** seçeneği ile Dll dosyası derlenir.

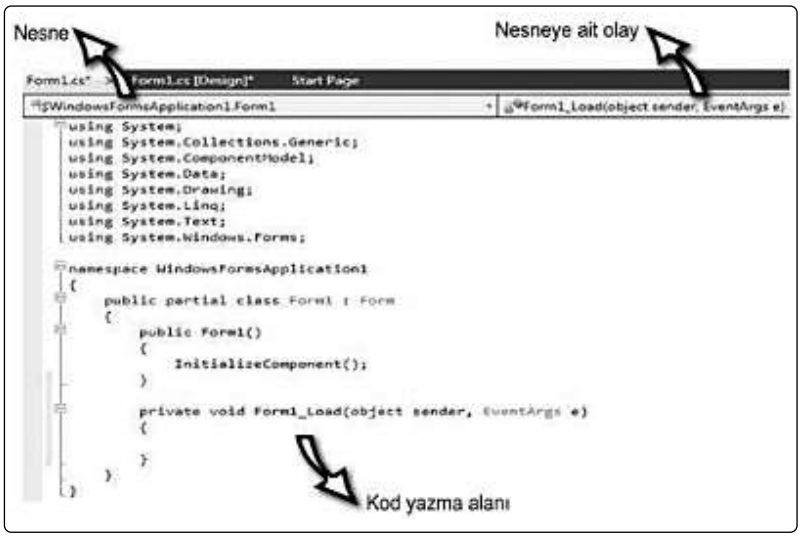
Rebuild seçeneği ile proje üzerinde yapılan değişiklikler uygulanarak proje yeniden derlenir.

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

Derleme işlemi esnasında oluşan dll dosyası proje klasörü içerisindeki **Bin** klasörü altında bulunan **Release** klasörü içerisinde oluşturulur.

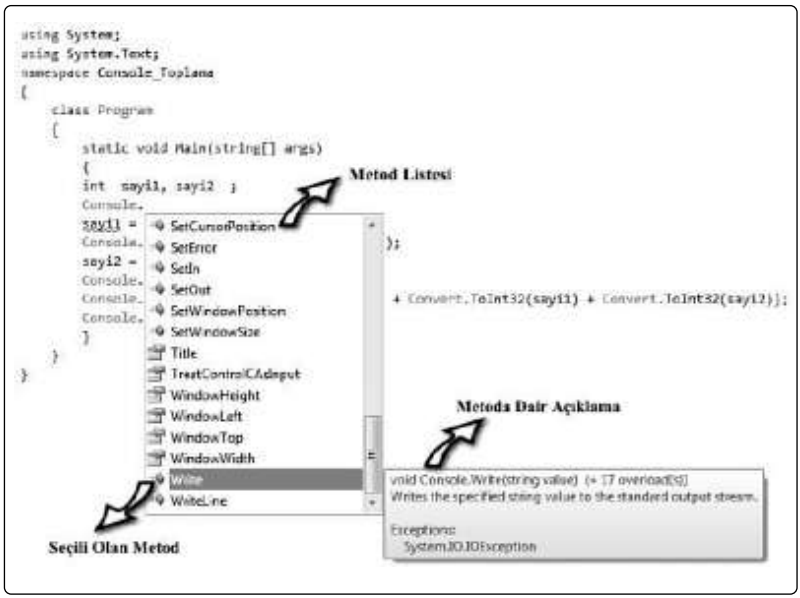
KOD YAZMAK

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.

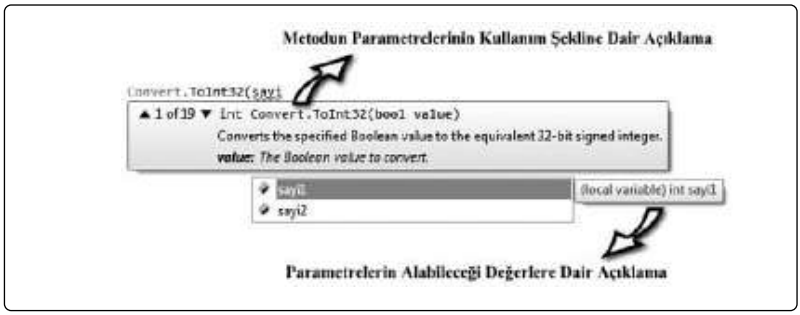


Kod yazarken Visual Studio IDE'si bize kullanacağımız metodlar ile ilgili intellisense desteğini sağlamaktadır. Kullanabileceğimiz metodlar

listelenmekte ve bu metodların kullanım şekilleri hakkında kullanıcı bilgi pencereleri ile bilgilendirilmektedir.



Kod yazımı esnasında kullandığımız metodların parametrelerinin kullanım şekilleri ve bu parametrelerin alabilecekleri değerlere dair açıklamalar hakkında da kullanıcı bilgi pencereleri ile bilgilendirilmektedir.



Projelerimizde kullanacağımız metod ve nesnelere ait sınıfları using ifadesi ile projemize dahil ederiz. Örneğin konsol uygulamaları için standart olarak `System` ve `System.Text` sınıflarını projemize dahil etmemiz

gerekirken, Windows Form uygulamaları için ise standart olarak `System` ve `System.Windows.Forms` sınıflarını projemize dahil etmemiz gerekmektedir. Ekleyeceğimiz sınıflar projelerimizde kullanacağımız metodlara, nesnelere göre değişiklik gösterecektir.

Kodlarımız varsayılan olarak proje ismi ile aynı olan bir namespace içerisinde yer almaktadır. Kullanıcı isterse kodlama esnasında bu namespace ismini değiştirebilir. Namespace kullanacağımız metod ya da sınıfları daha iyi bir şekilde organize edebilmek amacıyla .NET içerisinde oluşturulmuş olan bir yapıdır.

Örneğin varsayılan namespace'e ek olarak yeni bir namespace ekleyerek içerisinde bir **structure** (*yapı*) tanımlayalım ve içerisindeki elemanlara erişelim.

```
using System;
using System.Text;
using metodlar;
namespace islem
{
    class Program
    {
        static void Main(string[] args)
        {
            ogrenci ogrenciler = new ogrenci();
            ogrenciler.ad = "Talha Osman";
            ogrenciler.soyad = "AKTAŞ";
            ogrenciler.numara = 15;
            ogrenciler.sınıf = "2-C";

            Console.WriteLine(ogrenciler.ad);
            Console.WriteLine(ogrenciler.soyad);
            Console.WriteLine(ogrenciler.numara);
            Console.WriteLine(ogrenciler.sınıf);

            Console.ReadLine();
        }
    }
}
namespace metodlar
```

```
{
    struct ogrenci
    {
        public string ad;
        public string soyad;
        public int numara;
        public string sınıf;
    }
}
```

Görüldüğü gibi `using` ifadesi ile metodlar isimli namespace'i projemize dahil ederek metodlar isimli namespace içerisinde bulunan `ogrenci` adlı structure içerisindeki elemanlara erişip konsol ekranında listeletiyoruz.

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

Yazacağımız metodları ve kod bloklarını `{ }` işaretleri arasında yazarız.

`static void Main` ifadesi ile Main bir metod tanımladık. Main metod olmadan programlarımız çalışmaz. `Void` ifadesi genel olarak tek başına kullanıldığında değer göndermeyen metodları 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 amacı ile kullanıyoruz.

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

```
private void button1_Click(object sender, EventArgs e)
{
    int i = 0;
    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.

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

3

KONSOL EKRANI

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

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

blokları arasına yazılır.

C# .NET komut satırı derleyicisini kullanarak not defteri programıyla bile kod yazıp bunları komut satırında derleyebilirsiniz. C# .NET komut satırı derleyicisi *C:\Windows\Microsoft.NetFramework\FrameworkSürümü* 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();
        }
    }
}
```

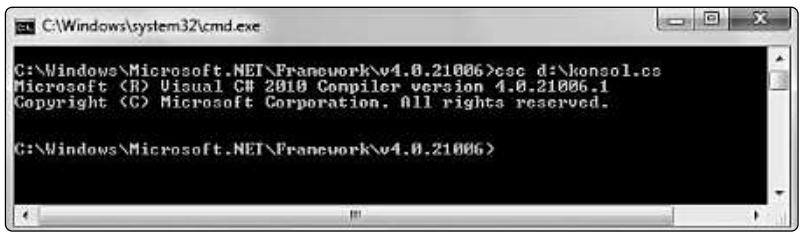
şeklindeki kodu not defterinde yazıp **konsol.cs** adı ile kaydedelim.

Başlat\Çalıştır\cmd ile komut satırını açıp *C:\Windows\Microsoft.Net\Framework\v4.0.20506* şeklindeki klasör içerisinde geçiş yaptım. Bendeki framework sürümü 4.0 olduğundan v4.0.20506 şeklinde klasör mevcut. Siz de bilgisayarınızda yüklü olan framework versiyonuna göre uygun klasörü seçebilirsiniz.

Komut satırında iken

csc d:\konsol.cs

şeklinde yazmış olduğumuz ve **D** sürücüsünde mevcut bulunan konsol.cs. uygulamamızı derliyoruz. Eğer herhangi bir hata bulunmazsa aşağıda görüldüğü gibi kodumuz derlenir ve **exe** uzantılı hale getirilir.



Kodumuzu çalıştırarak test ettiğimizde yazmış olduğumuz kod satırlarının hatasız biçimde çalıştığını görürüz:



CSC komut satırı derleyicisi ile birlikte aşağıdaki parametreleri de kullanabiliriz:

/out: Varsayılan olarak derlenecek olan **cs** uzantılı dosyamızın ismi ile derleme sonucunda oluşan dosyamız aynı isimlere sahiptir. Eğer farklı isimde bir dosya oluşturulmak istenirse **out** parametresi kullanılmalıdır.

Örneğin;

csc /out:d:\ornek.exe d:\konsol.cs

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

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

Örneğin;

csc /platform:x64 d:\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.

NOT **/target** parametresi **/t** şeklinde de kullanılabilir.

Örneğin;

csc /target:exe d:\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.



/reference parametresi **/r** şeklinde de kullanılabilir.

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

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



Birden fazla dll dosyasını eklemek için aralarında "," işareti konularak yazılabilir.

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

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

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

Konsol Ekranında Değer Yazma ve Okuma

Konsol ekranında yazdırma işlemleri için `Write` ve `WriteLine` metodları ya da `Console.Out` özelliği kullanılırken, okuma işlemleri için ise `Read` ve `ReadLine` metodları ya da `Console.In` özelliği kullanılır.

Write ve WriteLine Metodları

`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 metoddur.

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

şeklinde yazdığımızda:

```
Volkan Aktaş
```

şeklinde çıktı verecek ve imleç cümlenin sonunda bekleyecektir.

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

şeklinde yazdığımızda:

```
Volkan
Aktaş
```

şeklinde çıktı verecek ve imleç ikinci kelimenin sonunda bekleyecektir.

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

şeklinde yazdığımızda ise:

```
Volkan
Aktaş
```

şeklinde çıktı verecek ve imleç "*Aktaş*" kelimesinin bir alt satırında bekleyecektir.

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

```
using System;
using System.Text;
namespace Console_Parametre
{
    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}` şeklinde `WriteLine` komutu ile yazdırıyoruz.

Read ve ReadLine Metodları

`Read` metodu; konsol ekranından değer okumak için kullanılan bir metoddur. String değişken tipleri dışındaki değerleri okuma amacı ile kullanılır.

`ReadLine` metodu ise, `Read` metodu gibi konsol ekranından değer okumak için kullanılan bir metoddur. Farklı olarak `String` değişken tiplerindeki değerleri okuma amacı ile kullanılır.

Örneğin;

```
using System;
using System.Text;

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

Örneğin girilen "*volkan*" ismindeki değeri `Console.ReadLine` metodu ile okuyarak ad değişkenine aktarıp konsol ekranına yazdırdığımızda girilen "*volkan*" ismi ekrana yazdırılmıştır. `Console.Read` metodu ile aynı işlemi yapmaya kalktığımızda uygulamamız *"Cannot implicitly convert type int to string"* şeklinde dönüşüm hatası verecektir.

Kodumuzda da görüldüğü gibi `Console.Clear()` metodu, konsol ekranını temizlemek için kullanılır.

`Console.ReadLine` metodu ile okunabilecek en fazla karakter sınırı `256`'dır. Bunu arttırmak için aşağıdaki şekilde görüldüğü gibi buffer değeri arttırılarak daha fazla miktarda karakter girişi sağlanabilir.

```
using System;
using System.Text;
using System.IO;
namespace Console_Buffer
{
    class Program
    {
        static void Main(string[] args)
        {
            string okunandeger;
            byte[] Buffer = new byte[1025];
            Stream stream_ayar = Console.
                 OpenStandardInput(Buffer.Length);
            Console.SetIn(new StreamReader(stream_ayar));
            Console.Write("Okunacak değeri giriniz: ");
            okunandeger = Console.ReadLine();
            Console.Clear();
            Console.Write(okunandeger);
            Console.ReadLine();
        }
    }
}
```

Örneğimizde son satırlarda yazdığımız `Console.ReadLine( )` metodu, işlemi **Enter** tuşuna basılıncaya kadar bekletmek amacıyla da kullanılabilir.

Console.In ve Console.Out Özellikleri

Console.In özelliği ile konsol ekranından değer okuma işlemi yapılabilirken, **Console.Out** özelliği ile konsol ekranına değer yazdırma işlemi yapılabilir. **Console.In** özelliği **System.IO.TextReader** türünden değer alabilir. **Console.Out** özelliği ise **System.IO.TextWriter** türünden değer alabilir.

```
using System;
using System.Text;
using System.IO;
namespace Console_Characters
{
    class Program
    {
        static void Main(string[] args)
        {
            TextWriter out_text = Console.Out;
            TextReader inp_text = Console.In;

            out_text.WriteLine("Adınızı Girin");
            String isim = inp_text.ReadLine();
            out_text.WriteLine("Girilen isim = {0}", isim);
            inp_text.ReadLine();
        }
    }
}
```



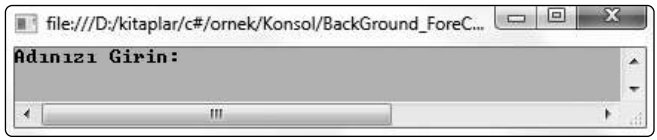
Görüldüğü gibi **Console.In** özelliği ile **Read** ve **ReadLine** özelliklerini kullanabilirken, **Console.Out** özelliği ile de **Write** ve **WriteLine** özelliklerini kullanabiliyoruz.

BackgroundColor ve ForegroundColor Metodları

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

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

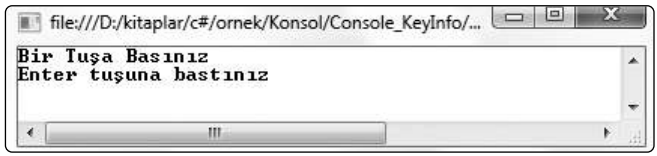
Zemin rengini tüm konsol ekranında geçerli kılmak için `Console.Clear` metodunu kullanarak ekranı temizledik.



ConsoleKeyInfo Metodu

Konsol ekranında basılan tuş ile ilgili bilgi almak amacı ile kullanılır. System sınıfına ait olan bir metoddur.

```
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();
}
```



Konsol ekranında **`Ctrl`**, **`Alt`** ve **`Shift`** tuşlarına basılıp basılmadığını algılamak için `ConsoleKeyInfo` metodunun `Modifiers` özelliğinden faydalanabiliriz.

```
static void Main(string[] args)
{
```

```
    Console.TreatControlCAsInput = true;
    System.ConsoleKeyInfo KeyInfo;
    KeyInfo = Console.ReadKey();
    Console.WriteLine("Bir Tuşa Basınız");

    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();
}
```

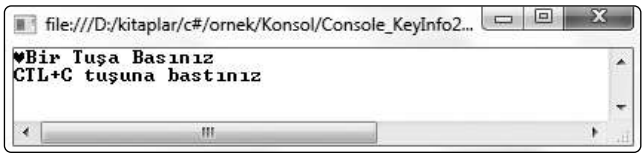
Örneğimizde...

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 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.



4

SABİTLER VE DEĞİŞKENLER

SABİTLER

İçeriği sabit olan değer ve ifadelerin saklanması amacı ile kullanılırlar. Bir sabitin değeri uygulamanın çalıştırılması sırasında kesinlikle değiştirilemez. `const` ifadesi ile tanımlanırlar.

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.

```
using System;
using System.Windows.Forms;
using Microsoft.VisualBasic;

namespace sabit_ucgen
{
    public partial class Form1 : Form
    {
        public Form1()
```

```
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
             // PI değerini sabit olarak tanımladık
             const double PI = 3.142857143;
             //değişkenler
             Double alan;
             Double çevre ;
             String mes;
             Double yarıçap;

             mes = "Yarıçapı giriniz ?";
             //Microsoft.VisualBasic sınıfını
             //referans olarak eklemeyi unutmayalım
             yarıçap = Convert.ToInt32
             //kullanıcıdan bilgi almak amacıyla
             //InputBox nesnesini kullanıyoruz.
             (Microsoft.VisualBasic.Interaction.InputBox(mes));

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

             mes = " dairenin çevresi ";
             mes = mes + çevre + " cm. Alanı ise: ";
             mes = mes + alan + " cm. dir.";

             Microsoft.VisualBasic.Interaction.MsgBox(mes);
        }
    }
}
```

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

Ayrıca kullanıcıdan bilgi almak amacıyla InputBox nesnesini kullandık. Ancak C# içerisinde InputBox nesnesi olmadığı için Microsoft.VisualBasic sınıfı içerisindeki InputBox nesnesini kullanıyoruz. Bunun için öncelikle **Project /Add Reference** seçeneği ile ya da Solution Explorer

içerisinde **References** bölümüne sağ tıklayarak gelen menüden **Add Reference** seçeneği ile **Microsoft.VisualBasic** sınıfını projemize referans olarak eklemeliyiz.

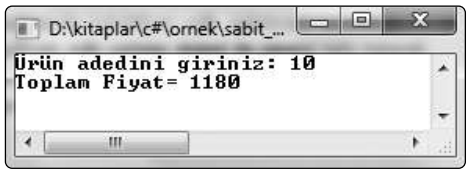
Daha sonra

```
using Microsoft.VisualBasic;
```

satırı ile Microsoft.VisualBasic sınıfını proje içerisinde kullanıma açıyoruz.

Bir sabiti tanımlarken başka bir sabitin değerini kullanarak tanımlamak da mümkündür.

```
using System;
namespace Sabit2
{
    class Program
    {
        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.

DEĞİŞKENLER

Program içerisinde üretilen değerleri geçici olarak saklamak amacı ile kullanılırlar.

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

şeklinde tanımlanırlar.

Örneğin; `string adsoyad` şeklinde ilk değer ataması yapılmadan tanımlanabileceği gibi `string adsoyad = "Volkan AKTAŞ"` şeklinde ilk değer ataması yapılarak da kullanılabilir.

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)

DEĞİŞKEN TİPLERİ

Başlıca değişken tipleri aşağıdaki gibidir.

DEĞER TİPLERİ

Değişken Tipi	Kapladığı Alan	Değer Aralığı
byte	8 bit işaretsiz tamsayı	0 - 255
sbyte	8 bit işaretli tamsayı	-128 - 127
short	16 bit işaretli tamsayı	-32.768 - 32.767

ushort	16 bit işaretsiz tamsayı	0 : 65.535
int	32 bit işaretli tamsayı	-2.147.483.648 - 2.147.483.647
uint	32 bit işaretsiz tamsayı	0 : 4.294.967.295
long	64 bit işaretli tamsayı	-9.223.372.036.854.775.808 ile - 9.223.372.036.854.775.807 arası
ulong	64 bit işaretsiz tamsayı	0 : 18.446.744.073.709.551.615
float	32 bit tek kayan sayı	+ya da - 1,5*10-45 : + ya da - 3,4*1038
double	64 bit çift kayan sayı	+yada - 5*10-324 : + ya da - 1,7*10308
decimal	128 bit ondalıklı sayı	+yada - 1,5*10-28 : + ya da - 7,9*1028
bool		true ya da false
char	Karakterleri temsil eder	16 Unicode karakterleri

REFERANS TİPLERİ

Değişken Tipi	Açıklama
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.

NOT

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 tanımlayarak string değer ataması yaparak adsoyad değişkeninin veri türünün string olarak belirlenmesini sağladık.

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 halen 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.



dynamic veri türü C# 4.0'a ait bir veri türü olduğundan dolayı dynamic veri türünü kullanacağınız projelerde **Framework 4.0** versiyonunu seçmelisiniz.

Tanımlandıkları Yerlere Göre Değişken Türleri

Değişkenler tanımlandıkları yerlere göre global ve local olmak üzere ikiye ayrılırlar.

1. Bir değişkeni global olarak tanımlamak için aşağıdaki yöntemler kullanılabilir;

   a. Tanımlanan değişkenin yalnızca aktif form içerisinde geçerli olması için formun

   ```
   Public Class FormAdı
   ```

   satırının altında tanımlama işlemi yapılır.

   b. **`[Public] [Static] [Değişken tipi] [Değişken adı]`** şeklinde bir tanımlama yapıldığında değişken nerede tanımlanmış olursa olsun tanımlanan değişken bütün proje içerisinde geçerli olur. **Public** ile tanımlanan değişkenlere de bir proje içerisindeki bütün formlardan erişmek mümkündür. Ancak başka form içerisinde tanımlanmış ve üzerinde değişiklik yapılmış bir değişkeni kullanabilmek için **Public Static** olarak tanımlamak gerekmektedir.

2. Bir değişkeni local olarak tanımlama işlemi ise aşağıda görüldüğü gibi yapılır;

   ```
   [Değişken tipi]  [Değişken adı] ;
   ```

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

   Global ve local değişken kullanımına dair örnek bir uygulama yapalım:

   Form üzerine bir adet buton ekleyerek butona tıklanma sayısını bulalım. Öncelikle local olarak dinamik değişken tanımlanması 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 dinamik değişken tanımlaması yapılmıştır. Değişken ilk 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"* şeklinde mesajı görüntülenmektedir.

Değişkenimizi butona tıklanma olayının dışında form kodlarının içerisinde global 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 içerisinde `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 erişebilmek için `Form2.i` şeklinde kullanabilmekteyiz. `Form1` içerisindeki butona her tıklandığında `Form2` değişkeni içerisinde tanımlanmış olan `i` değişkeninin değerini `1` arttırarak bu değişkenin içeriğini bir mesaj penceresi içerisinde görüntülüyoruz.

KULLANICI TANIMLI VERİ TİPLERİ

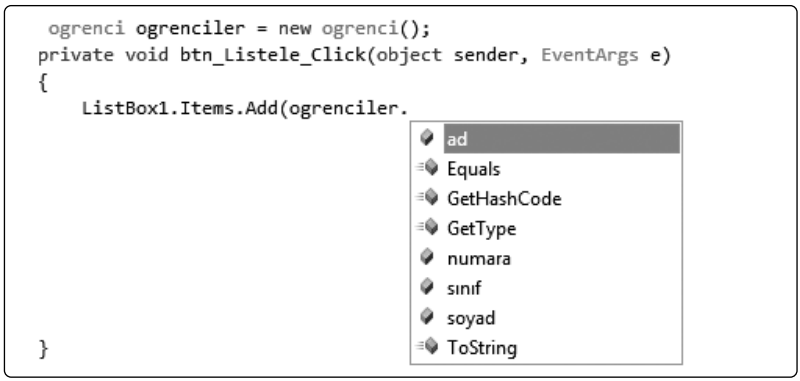
STRUCTURE (YAPI) OLUŞTURMAK

Structure veri türü, farklı tiplerden değişken tiplerini bir çatı altında toplayarak gruplandırmak amacı ile kullanılan veri türü yapısıdır. Değer türünden oldukları için küçük miktarda verilerin saklanmasında daha yüksek performans sağlarlar. Structure içerisindeki her eleman tek tek tanımlanmalıdır. Yapının boyutu, içerisinde tanımlanmış olan elemanların toplam boyutu kadardır. Örnek olarak `ogrenci` adında bir structure oluşturup içerisinde elemanlar tanımlayalım.

```
struct ogrenci
        {
            public string ad;
            public string soyad;
            public int numara;
            public string sınıf;
        }
```

Yeni bir değişken tanımlayarak değişkenin tipini oluşturduğumuz `ogrenci` adlı structure tipinden verelim.

```
ogrenci ogrenciler = new ogrenci( );
```

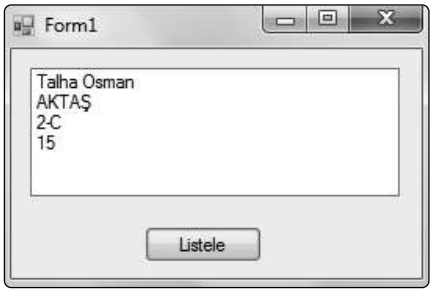


Daha sonra sıra ile structure içerisindeki elemanlara değer aktaralım. `ogrenciler` adlı structure türünden değişkenimizin elemanlarının listelendiğini görebiliriz.

Sonuç olarak kodumuz aşağıdaki gibi olacak:

```
using System;
using System.Windows.Forms;
namespace Structure_Olusturmak
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        public 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 = 15;
            ogrenciler.sınıf = "2-C";
        }
    }
}
```



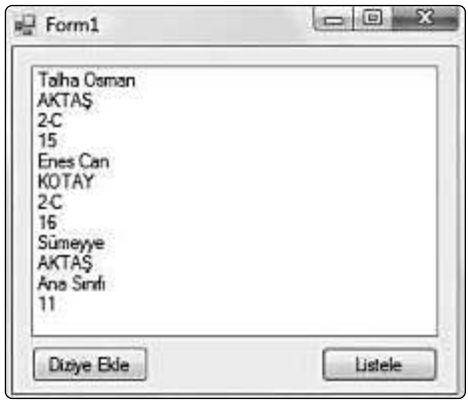
Structure'ı bir diziye aktararak diziler üzerinde uyguladığımız işlemleri structure'a da uygulayabiliriz.

```
using System;
using System.Windows.Forms;
using Microsoft.VisualBasic;
namespace Structure_To_Array
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        public 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"));
            }
        }
    }
}
```



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 metodları kullanabilmekteyiz. Örneğimizde dizi elemanlarına index numaralarını kullanarak erişmemiz ya da dizinin eleman sayısını bulmakta kullandığımız `Length` metodunda olduğu gibi.

Structure İçinde Structure Tanımlamak

Bir structure içerisinde bir ya da birden fazla structure tanımlamak mümkündür.

```
using System;
using System.Windows.Forms;
namespace Structure_İcinde_Structure
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        public 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 = 15;
            ogrenciler.sınıf = "2-C";
            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 değişkene aktardık.

Daha sonra `veli.adı`, `veli.ev_telefonu` şeklinde bu yapının elemanlarına erişim sağladık.

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.

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.

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 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.

`ToString` metodu ile `diziEnum` elemanlarına erişirken kullanacağımız bazı karakterler ile dizi 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 değerini 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** (*onaltılık*) 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();
        }
    }
}
```



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) 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) 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 **C** değeri görüntülendi.

`[Flags( )]` niteliğini kullanarak `ToString( )` ya da `ToString ("G")` metodları 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 yapmamız gerekmektedir. Programımızı bu eklemeyi yapıp çalıştırdığımızda



şeklinde ekran çıktısı elde ederiz.

Enum içerisindeki elemanlara verdiğim 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 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 SINIFI

Enum içerisindeki elemanlar üzerinde işlem yapmak için **System.Enum** sınıfından faydalanılabilir.

Örneğin oluşturmuş olduğumuz uzmanlık adlı Enum içerisindeki ilk elemana ulaşabilmek için `GetName` metodundan faydalanabiliriz. `GetName` Enum içerisinde mevcut olan bir elemana erişmek amacıyla kullanılan metoddur.

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
}
```

GetName metodunu...

```
GetName(typeof(uzmanlık), 3)
```

şeklinde son elemanın sıra numarasını belirterek kullanarak CSharp değerini görüntüleyebilirdik.

Enum içerisindeki eleman dizisine erişebilmek için ise GetNames metodundan faydalanılır. Örneğin uzmanlık adlı Enum içerisindeki elemanları bir listbox'a aktarmak için;

```
foreach (var uzmanlıkalanı in Enum.GetNames(typeof(uzmanlık)))
{
    listBox1.Items.Add(uzmanlıkalanı);
}
```

şeklinde GetNames metodundan faydalanarak aktarım yapabiliriz.

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 = 3,
            CSharp = 4
        }
        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 =4"* şeklinde sonuç üretecektir.

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.

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

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 olu-

şacak 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 metodla birlikte `TypeOf` metodu kullanılabilir. `GetType` metodu ile direkt olarak değişkenin tipi öğrenilebilecekken, `TypeOf` metodu ile bir değişkenin belirtilen veri türüne ait olup olmadığı belirlenebilir.

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 bir örnek uygulama yapalım. Örneğimizde değişkenlerin tipini belirtmeden değer ataması yapalım. Değişkenlere içerdikleri değerlere 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
```

ifadelerinin görüntülendiğini görürüz.

TypeOf Metodu

Herhangi bir değişkenin 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` 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
İnt	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** işlemlerinde kullanılan metodlarla 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, 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 `2.5` değerinin 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 türünde ise dönüştürme işleminin hangi türe yapılacağı kullanıcı tarafından çeşitli metodlar 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** ile otomatik olarak gerçekleştirdiğimiz dönüşüm işlemini, **Açık dönüşüm** 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 olduğunda **InvalidCastException** hatası oluşacaktır.

Örneğin;

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

şeklinde yazacağımız kod *"Cannot convert type 'int' to 'string'"* hatasını verecektir.

```
object nesne = 123;
string yazi1 = (string)nesne;
```

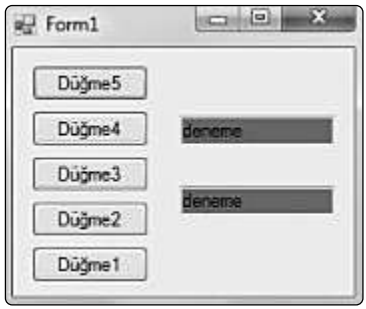
şeklinde yazacağımız kod ise hata vermeden çalışacaktır.

Dönüşüm esnasında oluşacak bir hata durumunda geriye hata döndürmek yerine **boş değer** (*null*) döndürmek için `as` metodunu kullanabiliriz.

Örnek olarak bir form üzerindeki kontrollere erişerek onların özelliklerini değiştirecek bir uygulama 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 dugme in this.Controls) {
        Button btn = dugme as Button;
```

```
            if (btn == null) {
                dugme.BackColor = Color.SteelBlue;
                dugme.Text = "deneme";
            }
            else {
                btn.BackColor = Color.Aquamarine;
                sayi = sayi + 1;
                btn.Text = "Düğme" + sayi;
            }
        }
}
```



Kodumuzu çalıştırdığımızda form üzerindeki butonların rengi değiştirilerek üzerlerine `Düğme1..Düğme5` 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 hata 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.

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.

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 = "24/01/2009";
    MessageBox.Show((System.DateTime.Parse(deger)). ↓
                ToLongDateString());
}
```

Kodumuzu çalıştırdığımızda aşağıdaki 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

```
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() ;
}
```

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.

TryParse Metodu ile

`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, 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.

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 = "24/01/2009";
     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.

Convert Metodu ile

`Convert` metodu şu alt metotlara sahiptir;

Metod	Açıklama
`Convert.ToBase64String()`	Verilen bir stringi 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.ToInt64()	UInt64 tipine dönüştürme yapar.

String tipteki bir veriyi `Parse` ve `TryParse` metodları ile tarih veri türüne dönüştürme işlemini yapmıştık. Şimdi de `Convert.ToDateTime( )` metodu ile yapalım.

```
private void button1_Click(object sender, EventArgs e)
{
     string deger = "24/01/2009";
     MessageBox.Show(Convert.ToDateTime(deger).ToLongDateString());
}
```

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.

```
static void Main(string[] args)
{
     string sayi1 = "123";
     string sayi2 = null;
     string sayi3 = "123.45";
     string sayi4 = "123456789101112131415161718192O";
     Console.WriteLine(Convert.ToInt32(sayi1));
     Console.WriteLine(Convert.ToInt32(sayi2));
     Console.WriteLine(Convert.ToInt32(sayi3));
     Console.WriteLine(Convert.ToInt32(sayi4));
```

```
    Console.ReadLine();
}
```

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.

5 OPERATÖRLER

C# .NET dilinde aritmetiksel, aktarma, mantıksal, karşılaştırma ve birleştirme operatörü olarak beş çeşit operatör kullanılmaktadır.

Aritmetik Operatörler

Matematiksel işlemlerin gerçekleştirilmesi amacı ile kullanılan operatörlerdir.

Aritmetik operatörler başlıca şu şekildedir:

+ Toplama
- Çıkarma
* Çarpma
^ Üs alma
/ Bölme
% Mod

Örneğin bölme ile ilgili örnek bir uygulama 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` görüntülenmektedir.

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 değişkenin sağındaki değişkene bölerek kalanı solundaki değişkene aktarır.

```
byte a = 7;
byte b = 4;
a %= b;
```

`a` 3 değerini alır.

MANTIKSAL OPERATÖRLER

Aynı zamanda **boolean operatörler** adını da almaktadır. 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
```

&& (VE DEĞİL) 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
```

|| (VEYA DEĞİL) 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.

?: (Koşul) Operatörü

```
x ?: y : z
```

Üç adet operanta sahiptir. Eğer x true ise y değişkeni hesaplanır, false ise z değişkeni hesaplanır.

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

`a` `“32”` değerini alır.

6

DİZİLER

Dizi Oluşturmak

Diziler aynı türdeki verilerin sıralanarak gruplandırılmasında kullanılan yapılardır. Diziler içerisinde veriler kaydedilirken `0` numaralı sıra değerinden itibaren kaydedilmeye başlanılır.

`Değişken Türü [ ] Dizi değişkeni adı;`

şeklinde tanımlanırlar.

Örneğin;

```
string[ ] sınıf;
```

şeklinde bir tanımlama ile `sınıf` adında bir string dizi tanımlamış oluyoruz. Dizi boyutunu belirlemek için ise `new` anahtar sözcüğünü kullanabiliriz.

Örneğin;

```
int[ ] numara;
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 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 üç elemanlı bir `adsoyad` dizisi tanımlayarak program içerisinde dizinin elemanlarına atama yapılmıştır.

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.

İlk değer ataması yapacağımız çok boyutlu dizi tanımlamalarında `","` karakteri kullanılarak boyut sayısı belirtilir.

```
string[,] adsoyad = { { "Volkan", "Aktaş" }, { "Mustafa",
"Aksu" }, { "Enver", "Arslan" } };
```

Örneğin yukarıdaki `adsoyad` dizisi iki boyutlu bir dizi olarak tanımlanmıştır. Bu dizi içerisindeki `"Aksu"` elamanına `adsoyad[1,1]` şeklinde ulaşılabilir.

Karakter dizilerini tanımlarken ise diziyi oluşturan elemanları

```
char[] karakterler = { '!', '?'};
```

örneğinde olduğu gibi, `""` (çift tırnak) karakterleri arasına değil de, `''` (tek tırnak) karakterleri arasına yazarız.

DİZİLERİ YENİDEN BOYUTLANDIRMAK

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 `Array.Resize` metodu ile dizi boyutu belirlenebilir. Belirlenmişse de yeniden belirlenerek dizi boyutu değiştirilebilir.

Örneğin;

```
string[ ] adsoyad = new string[2];
adsoyad[0] = "Talha";
adsoyad[1] = "Sümeyye";
```

Yukarıda tanımladığımız iki elemanlı `adsoyad` dizisindeki ilk elemanın içeriğine

```
adsoyad[0]
```

şeklinde eriştiğimizde içeriğinin `"Talha"` olduğunu görürüz.

Bu diziyi;

```
Array.Resize(ref adsoyad, 3);
adsoyad[2] = "Ömer";
```

şeklinde `Array.Resize` metodu ile yeniden boyutlandırarak üç elemanlı bir dizi haline çevirdik.

`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.

Kodumuzda daha sonra dizinin 3. elemanına da `"Ömer"` değerini aktardık.

`Adsoyad[2]` ifadesi ile dizinin son elemanını görüntülemek istediğimizde `"Ömer"` değerini görebiliriz. Ayrıca yukarıda görüntülediğimiz gibi `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";

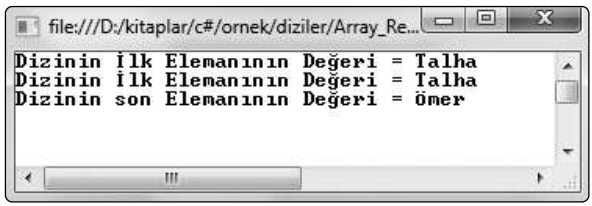
       Console.WriteLine("Dizinin İlk Elemanının Değeri = "
         + adsoyad[0]);

       Array.Resize(ref adsoyad, 3);

       adsoyad[2] = "Ömer";

       Console.WriteLine("Dizinin İlk Elemanının Değeri = "
         + adsoyad[0]);
       Console.WriteLine("Dizinin Son Elemanının Değeri = "
         + adsoyad[2]);
       Console.ReadKey();
}
```

Uygulamayı çalıştırdığımızda aşağıdaki çıktıyı elde ederiz;



Ekran çıktısında da görüldüğü gibi dizimizi yeniden boyutlandırdıktan sonra dizi içerisindeki elemanların değerleri muhafaza edildiği için *"Dizinin İlk Elemanının Değeri=Talha"* olarak görüntülenmiştir.

Dizi Değişkenlerinde Kullanılan Metodlar

Diziler üzerinde kullanılan pek çok metod vardır. Bu metodlar genellikle **System.Array** sınıfına ait metodlardır.

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();
 }
```

Kodumuzu çalıştırdığımızda `"Teknik"` sonucunu üretecektir. `okulturu` değişkeni object olarak geriye döndüğünden dolayı;

```
string[] okulturleri = (string[])okulturu;
```

satırı ile okulturleri dizisine dönüştürüyoruz.

`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();
}
```

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();
}
```

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.

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);
}
```



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 `","` 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)`

```
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();
}
```

Örneğimizde copy metodu ile `digerdiller` dizisinin elemanları diller dizisi içerisine kopyalanarak 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 iki elemanını kopyalasa idik, diller dizisinin son elemanı olan *"ASP.Net"*, diller dizisinin içerisinde mevcut olmaya devam edecekti. Bu şekilde kodumuzu çalıştırdığımızda ise *"Python, F#, ASP.Net"* şeklinde sonuç verirdi.

Diziler İçerisinde Arama İşlemleri

Contains Metodu ile Arama Yapmak

Bir dizi içerisinde bir elemanın var olup olmadığını bulma amacı ile kullanılır. Geriye boolean bir değer döndürür. `System.Array` sınıfına ait bir metod olduğu için `System.Collections` sınıfını using anahtar sözcüğü ile projemize dahil etmemiz gerekmektedir.

Kullanımı şu şekildedir:

Contains(aranan değer)

```
static void Main(string[] args)
{
        string aranan;
        ArrayList isimler = new ArrayList();
        isimler.Add("Hasan");
        isimler.Add("Murat");
        isimler.Add("Talha");
        isimler.Add("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();
}
```

IndexOf Metodu ile Arama Yapmak

Dizi içerisinde aranan elemanın index numarasını geriye döndüren metoddur.

Kullanımı şu şekildedir:

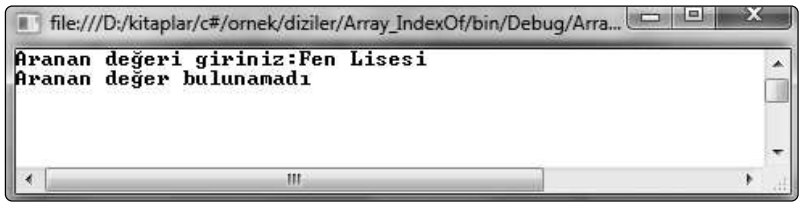
IndexOf(Arama yapılacak dizi değişkeninin adı, aranacak değer, aramanın başlayacağı index değeri)

`IndexOf` metodu geriye, eğer 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 console 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 aşağıdaki gibi *"Aranan değer bulunamadı"* mesajı yazılacaktır.



`"Endüstri Meslek"` değerini verdiğimizde ise eleman dizi içerisinde var olduğundan index numarası olan `0` görüntülenir.



Örneğin arama işlemine 1 index numaralı elemandan başlanılmasını isteseydik arama işlemini

```
indexno = Array.IndexOf(okulturu, aranan.ToString(),1);
```

şeklinde gerçekleştirirdik.

LastIndexOf Metodu ile Arama Yapmak

`IndexOf` metodunda olduğu gibi arama diziler içerisinde arama yapmak amaçlı olarak kullanılır. Farklı olarak eğer aranan elemandan birden fazla mevcut ise bu elemanlardan sonuncusunun index değerini verir.

Kullanımı şu şekildedir:

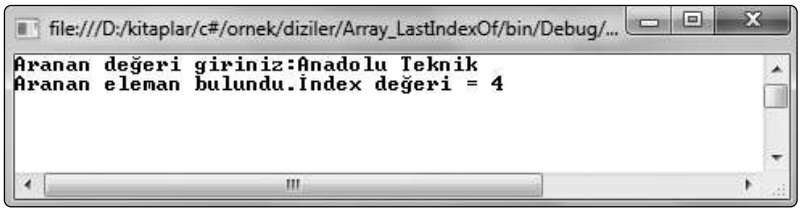
`LastIndexOf(Arama yapılacak dizi değişkeninin adı, aranacak değer, aramanın başlayacağı index değeri)`

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 sahip iken son index değeri olan 4 ekrana getirilmiştir.



BinarySearch Metodu İle Arama Yapmak

Dizi içerisinde aranan elemanın index numarasını geriye döndüren metoddur.

Kullanımı şu şekildedir:

BinarySearch(Arama yapılacak dizi değişkeninin adı, aranacak değer, aramanın başlayacağı index değeri)

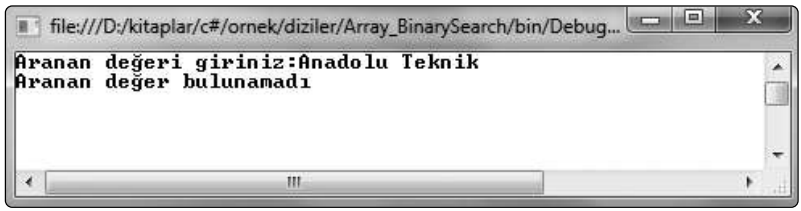
IndexOf metodu geriye, eğer aranan değer bulunursa o değerin index değerini, bulunamazsa negatif bir değer döndürür. Bu metod ile arama yapabilmek için üzerinde arama işlemi yapılacak dizinin sıralı olması gerekir.

Örneğin;

```
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.

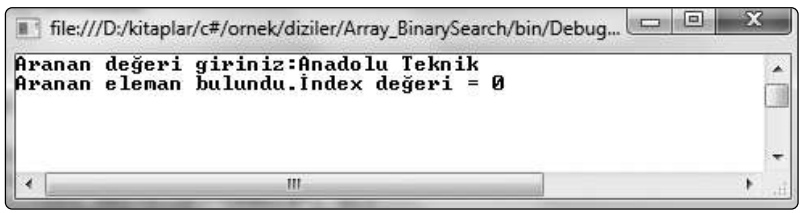


Görüldüğü gibi `"Anadolu Teknik"` elemanı dizi içerisinde bulunmasına rağmen *"Aranan değer bulunamadı"* mesajı döndürüldü.

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.

ArrayList Sınıfı ile Dizi Tanımlama ve Bu Sınıf İçerisinde Kullanılan Çeşitli Metod ve Özellikler

`ArrayList` sınıfı, eleman sayısını belirlemeden dizi tanımlamak amacı ile kullanılır. Herhangi bir eleman sınırı konulmadan tanımlanır. `ArrayList`'in boyutu, içerisine eklenen eleman sayısına göre otomatik olarak belirlenir. İçerisine aktarılan değerler, boxing işlemi ile object türüne dönüştürülerek tanımlanır. `Collections` sınıfı içerisinde yer alır. Dolayısı ile bu metodları kullanabilmek için projemize `System.Collections` sınıfını, using anahtar sözcüğünü kullanarak eklememiz gerekmektedir.

```
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 nickname = new ArrayList();
        nickname.Add("Talha");
        nickname.Add(123);
        nickname.Add("a");

        Console.Write(nickname[1]);
        Console.ReadKey();
}
```

`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. Eklenen 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.

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("Hasan");
         isimler.Add("Murat");
         isimler.Add("Talha");
         isimler.Add("Onur");

         isimler.Sort();
         Console.Write(isimler[0]);
         Console.Write(isimler[1]);
         Console.Write(isimler[2]);
         Console.Write(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.

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 yararlanarak 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 (i = 0; i <= rakamlar.Count - 1; i++)
         {
             Console.WriteLine(rakamlar[i]);
         }
         Console.ReadKey();
}
```

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 `123` elemanını sildiğimiz için `1` nolu index değerine sahip elemanı yazdırdığımızda `"a"` değeri yazdırılır.

RemoveAt Metodu

Dizi içerisinden, indeks değeri belirtilen elemanları silmek amacı ile kullanılır.

```
static void Main(string[] args)
{
        ArrayList nickname = new ArrayList();
        nickname.Add("Talha");
        nickname.Add(123);
        nickname.Add("a");
        Console.WriteLine(nickname[nickname.Count - 1]);
        nickname.RemoveAt(nickname.Count - 1);
        Console.WriteLine(nickname[nickname.Count - 1]);
        Console.ReadKey();
}
```

Örneğimizde

```
Console.WriteLine(nickname[nickname.Count - 1]);
```

satırında `Count` metodunu kullanarak `nickname` dizisinin son elemanı olan "`a`" değerini görüntüledik.

Ardından,

```
nickname.RemoveAt(nickname.Count - 1);
```

satırında dizinin son elemanı olan "a" değerini sildik.

Son olarak,

```
Console.WriteLine(nickname[nickname.Count - 1]);
```

satırında dizinin son elemanını yeniden yazdırdığımızda 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.Write(isimler[0]);
        Console.Write(isimler[1]);
        Console.Write(isimler[2]);
        Console.Write(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.

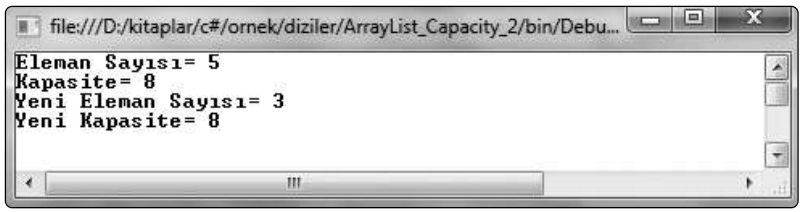
Dizimize...

```
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 iki adet eleman silmemize rağmen `capacity` değeri değişmeyerek `8` değeri görüntülenecektir.

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);

        Console.WriteLine(diller[0]);
        Console.WriteLine(diller[1]);
        Console.WriteLine(diller[2]);
        Console.WriteLine(diller[3]);
        Console.WriteLine(diller[4]);
        Console.WriteLine(diller[5]);
        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);

        Console.WriteLine(diller[0]);
        Console.WriteLine(diller[1]);
        Console.WriteLine(diller[2]);
        Console.WriteLine(diller[3]);
        Console.WriteLine(diller[4]);
        Console.WriteLine(diller[5]);
        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.

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 iki 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. `System.Array` sınıfına ait olan `Copy` metoduna göre daha gelişmiş seçeneklere sahiptir.

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)`

şeklindeki 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);

        Console.WriteLine(diller.GetValue(0));
        Console.WriteLine(diller.GetValue(1));
        Console.WriteLine(diller.GetValue(2));
        Console.WriteLine(diller.GetValue(3));
        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.

`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 iki 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);
```

`diger`diller dizisinin ilk elemanından itibaren iki 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 iki 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 dolayısı ile aynı bellek bölgesini paylaşmaktadırlar. Dolayısı ile diller dizisinde yap-

tığımız 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));
```

satırında 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.

List Nesnesi

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` nesnesi **`System.Collections.Generic`** sınıfına ait bir nesnedir. Dolayısı ile bu metodları kullanabilmek için projemize **`System.Collections.Generic`** sınıfını, `using` anahtar sözcüğünü kullanarak eklememiz gerekmektedir.

`List<Değişken Türü> isim = new List<Değişken Türü>()`

şeklinde tanımlanırlar.



List nesnesi için **`ArrayList`** sınıfının generic karşılığıdır şeklinde bir nitelendirme yapabiliriz. Hemen hemen tüm metodları ortaktır. Dolayısı ile bu metodları tekrar açıklamayacağız. Yalnızca List nesnesinin 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.

7

DİYALOG PENCERELERİ

INPUTBOX (GİRİŞ PENCERESİ)

C# içerisinde `inputbox` nesnesi mevcut değildir. Bunu kullanabilmek için Microsoft.VisualBasic sınıfını kullanmamız gerekmektedir. **Project / Add Reference** seçeneği ile **.Net** sekmesi altından **Microsoft.VisualBasic** sınıfını projemize referans olarak eklemeliyiz. Ayrıca using anahtar sözcüğünü kullanarak **Microsoft.VisualBasic** sınıfını projemize eklemeliyiz.

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)
```

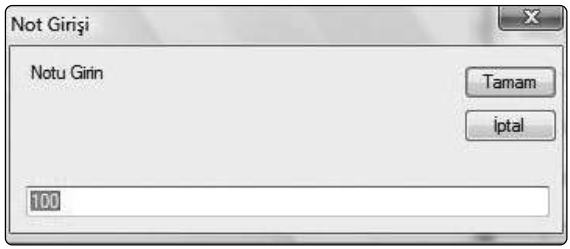
Inputbox içerisinde görüntülenecek olan mesajın uzunluğu en fazla `1024` karakter olabilir.

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("Notu Girin",
                    "Not Girişi", "100", 10, 10));
```

```
        byte not2 = Convert.ToByte(Interaction.InputBox("Notu Girin",
                    "Not Girişi", "100", 10, 10));
        MessageBox.Show("Ortalama= " + (not1 + not2) / 2);
}
```

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 `End` komutu ile programın çalışmasını sona erdiriyoruz.

Örneğimizde not1 ve not2 değişkenleri artık string tipte olduğu için Convert.ToByte metodu ile bu değişkenlerin aldıkları string değerleri byte değişken türüne çeviriyoruz.

MESSAGEBOX (MESAJ PENCERESİ)

Program içerisinde üretilen sonuçların ekranda gösterilmesi amacı ile kullanılan pencerelerdir.

Kullanım biçim 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 birkaç 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.

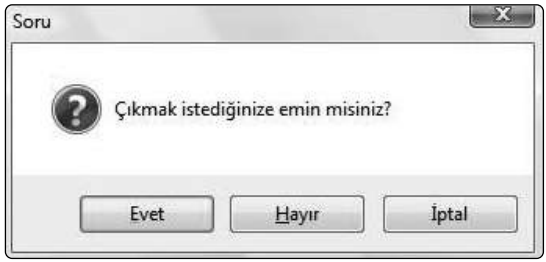


Sıradaki örneğimizde ise üzerinde **soru** işareti ve **evet**, **hayır**, **iptal** düğmeleri olan bir mesaj penceresi görüntületelim.

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;



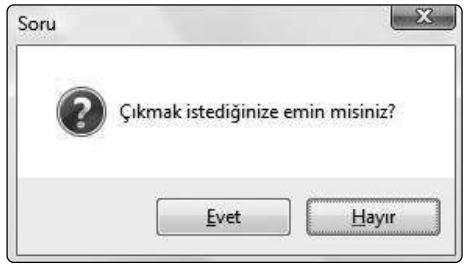
Mesaj penceresi üzerindeki butonlardan herhangi birini varsayılan buton olarak seçip aktifleştirmek mümkündür. Bunun için aşağıdaki değer ya da sabitler kullanılabilir;

Değer	Görevi
MessageBoxDefaultButton.DefaultButton1	ilk 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 yandaki 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:



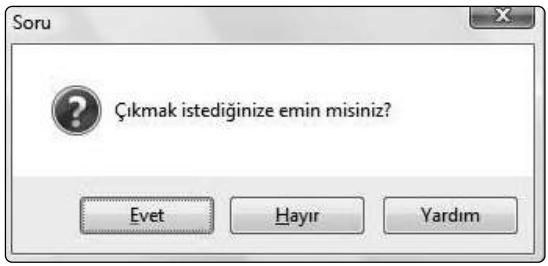
sonucunu üretecektir.

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.

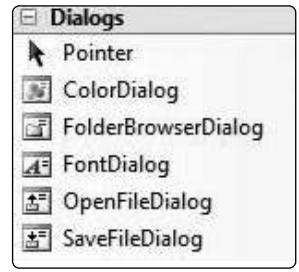
Mesaj penceresi üzerindeki butonlardan hangisine basılıp basılmadığını anlayabilmek için `DialogResult` nesnesini kullanabiliriz. `DialogResult` bize basılan tuşu döndürecektir.

```
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 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 beş adet dialog kontrolü vardır.



Diyalog pencerelerini görüntülemek için `ShowDialog` metodu kullanılırken, diyalog penceresi üzerinde basılan tuş `DialogResult` özelliği ile geriye döndürülmektedir.

Color Dialog

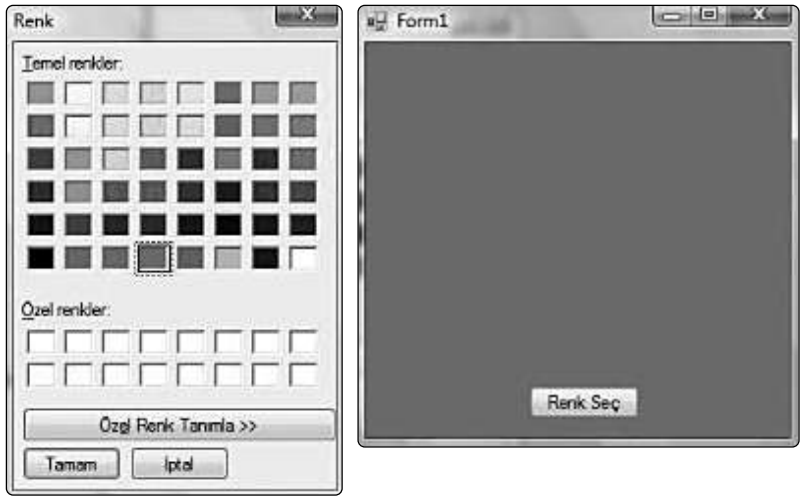
Renk seçme amaçlı olarak kullanılan diyalog penceresidir. Örneğin bir formun rengini color dialog nesnesi yardımı ile değiştiren örnek bir uygulama yapalım.

Form üzerine bir adet buton ve bir adet color dialog 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;
}
```



Rengimizi seçip **Tamam** düğmesini tıkladığımızda formun renginin seçtiğimiz renkle değiştirildiğini görürüz.

FOLDER BROWSER DIALOG

Klasör seçmek amacı ile kullanılır. Örnek olarak folder browser dialog nesnesi yardımı ile seçilecek bir klasördeki word belgelerini listeleyecek olan bir örnek uygulama yapalım.

Form üzerine bir adet buton, bir adet listbox ve bir adet de folder browser dialog 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); }
                }
            }
        }
}
```

Kodumuzu çalıştırıp klasör seç düğmesine basalım ve bir klasör seçelim. Örneğin ben **Belgeler** klasörünü seçtim.



Daha sonra **Tamam** düğmesine bastığımızda 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 bir adet buton, bir adet de textbox ekleyeceğiz.

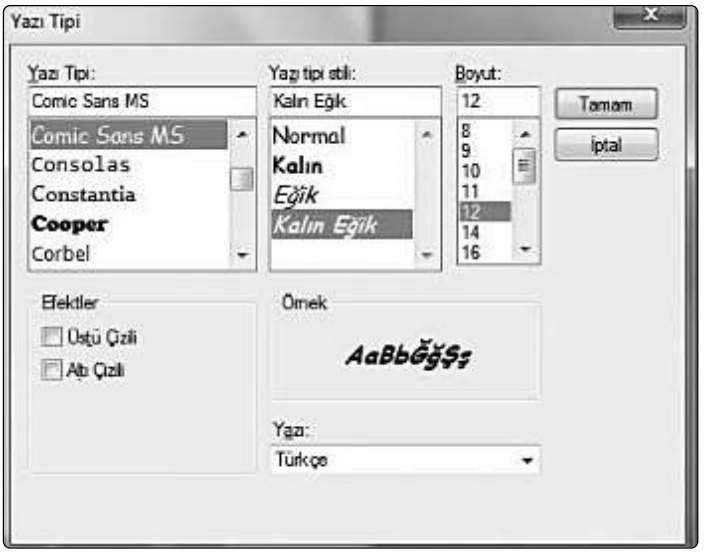
Kodumuz aşağıdaki gibi olacak:

```
private void btnYazıTipi_Click(object sender, EventArgs e)
{
        DialogResult basilan_tus;
        basilan_tus = fontDialog1.ShowDialog();
        if (basilan_tus == DialogResult.OK)
                textBox1.Font = fontDialog1.Font;}
}
```

Kodumuzu çalıştıralım. Textbox içerisine yazdığımız yazının ilk hali yandaki 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 açmak 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 bir adet buton, bir adet picturebox nesnesi ve bir adet de `OpenFileDialog` nesnesi ekleyelim.

Kodumuz aşağıdaki gibi olacak:

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

namespace OpenFileDialog
{
        public partial class Form1 : Form
        {
```

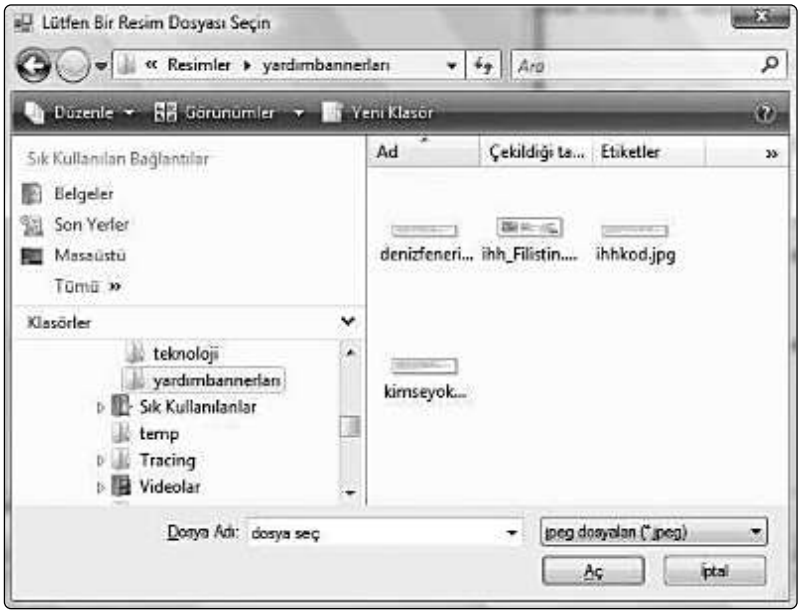
```
            public Form1()
            {
                InitializeComponent();
            }
            private void btnResimSec_Click(object sender,
            EventArgs e)
            {
                DialogResult basilan_tus;
                basilan_tus = openFileDialog1.ShowDialog();
                if (basilan_tus == DialogResult.OK)
                {
                    string yol = System.IO.Path.GetFullPath
                                (openFileDialog1.FileName);

                    System.IO.FileStream fs =
                                new System.IO.FileStream
                                (yol, FileMode.Open,
                                FileAccess.Read);

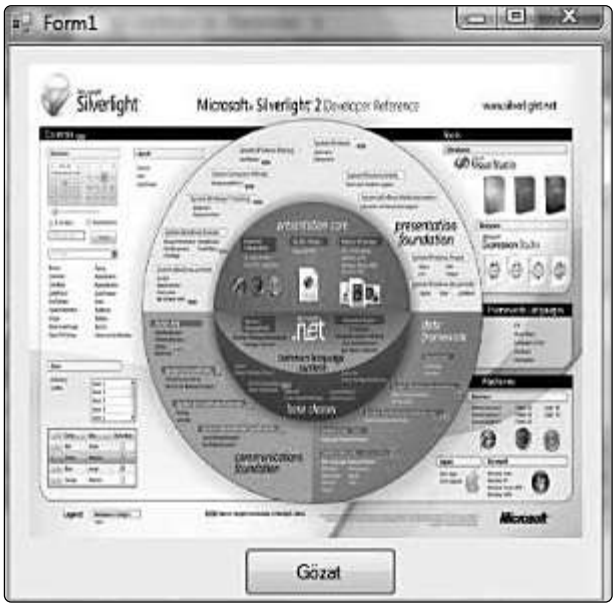
                    System.Drawing.Bitmap resim =
                        new System.Drawing.Bitmap
                        (fs, true);

                    pictureBox1.Image = resim;
                }
            }
            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";
            }
        }
}
```

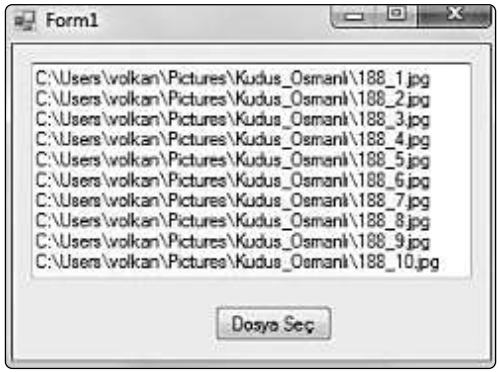
Kodumuzu çalıştırıp butonu tıkladığımızda dosya seçme penceresi ekrana gelecektir.



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 =
        "C:\\Users\\volkan\\Pictures";
    openFileDialog1.ShowDialog();
    for (i=0;i<openFileDialog1.FileNames.Length-1;i++)
    {
        textBox1.Text = textBox1.Text +
            openFileDialog1.FileNames[i].ToString() +
            Environment.NewLine;
    }
}
```

`Openfiledialog` nesnesi içerisinde birden fazla dosyayı seçmek için öncelikle `MultiSelect` özelliğine `true` değerini atıyoruz. Seçilen dosyaları textbox içerisine aktarmak için 0'dan openfiledialog nesnesi içerisindeki eleman sayısına (`openFileDialog1.FileNames.Length`) ka-

dar bir döngü kurarak `FilenNames` koleksiyonu içerisindeki elemanları textbox içerisine ekleriz. `Environment.NewLine` textbox içerisine yeni bir satır eklemek amacı ile kullanılır.



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 iki adet buton, bir adet picturebox nesnesi, bir adet `OpenFileDialog` nesnesi ve bir adet de `SaveFileDialog` nesnesi ekleyelim. Kodumuz aşağıdaki gibi olacak:

```
using System;
using System.IO;
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 =
          "C:\\Users\\volkan\\Pictures";

     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 =
          "C:\\Users\\volkan\\Pictures";
     saveFileDialog1.DefaultExt = "jpeg";
}
private void btnResimSec_Click_1(object sender, ↓
EventArgs e)
{
     DialogResult basilan_tus;
     basilan_tus = openFileDialog1.ShowDialog();
     if (basilan_tus == DialogResult.OK)
     {
          string yol = System.IO.Path.GetFullPath
                    (openFileDialog1.FileName);
          System.IO.FileStream fs =
                    new System.IO.FileStream
                    (yol, FileMode.Open,
                    FileAccess.Read);
```

```
                System.Drawing.Bitmap resim =
                        new System.Drawing.Bitmap
                        (fs, true);
                pictureBox1.Image = resim;
            }
        }
        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;
                }
            }
        }
}
}
```

8

ŞARTLI DALLANMA YAPILARI

IF ELSE 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.

Bir örnek verelim. Örneğimiz 45'den küçük değer girilirse geçtiniz, büyük değer girilirse kaldınız şeklinde mesaj görüntülüyor. Ayrıca 100'den büyük değer girişi de kontrol ediliyor. 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 if_not_gecti_kaldı
{
    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 > 44)
                {
                    MessageBox.Show("Geçtiniz");
                }
            }
            else
            {
                MessageBox.Show("Not 100'den küçük olmalı");
            }
        }
    }
}
```

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.

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 kontrol edilmekte, 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 programın çalışması sona erdirilmektedir.

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 ya da değer aralığı:
     {
           Komutlar;
           break;
     }
   case değer ya da değer aralığı:
     {
           Komutlar;
           break;
     }
   default
     {
                komutlar;
          break;
     }
}
```

Örnek olarak girilen puana göre geçti, 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 örneğimizi `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;
          }
     }
}
```

9 DÖNGÜLER

FOR DÖNGÜSÜ

Kullanımı şu şekildedir:

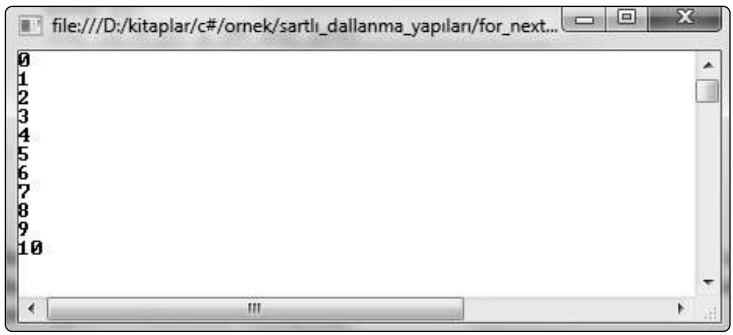
```
for (sayaç değişkeni ; şart ; artış değeri)
{
  Komutlar;
}
```

Şart sağlandığı sürece `for` döngüsü içerisindeki komut satırları artış değeri adedince tekrarlanır. Örneğin; `0` ile `10` arasındaki sayıları ekranda listeleyen bir konsol uygulaması yazalım:

```
static void Main(string[] args)
{
    byte i;
    for (i = 0; i <= 10; i++)
    {
        Console.WriteLine(i);
    }
```

```
        Console.ReadLine();
}
```

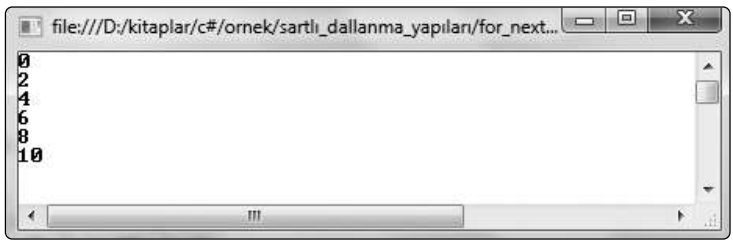
Döngümüz 0'dan 10'a kadar kurulduğu için i değişkeninin değeri ekrana 10 defa artan şekilde yazılacaktır. `Console.Readline` ifadesi ile de ekran çıktısını enter tuşuna basılana kadar bekletiyoruz.



Aynı uygulamamızı 0'dan 10'a kadar olan çift sayıları görüntüleyecek şekilde değiştirelim.

```
static void Main(string[] args)
{
    byte i;
    for (i = 0; i <= 10; i+=2)
    {
        Console.WriteLine(i);
    }
        Console.ReadLine();
}
```

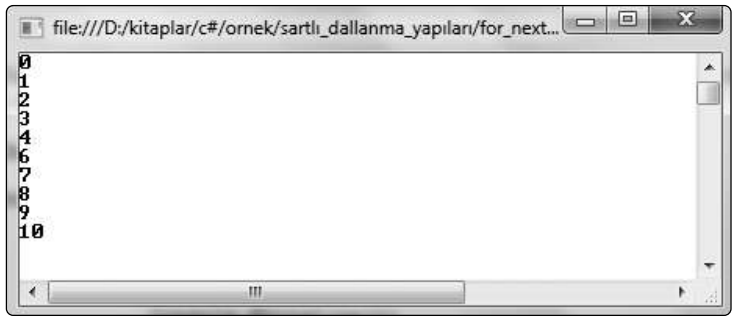
Görüldüğü gibi i++ şeklindeki artış değerimizi i+=2 yaptık. Ekran çıktımız aşağıdaki gibi oldu:



Yine 1 ile 10 arasındaki sayıları yazacak olan ama 5 sayısını atlayarak yazmayacak bir kod yazalım. Bunu iki şekilde yapabiliriz.

İlkinde...

```
static void Main(string[] args)
{
     byte i;
     for (i = 0; i <= 10; i++)
     {
          if (i == 5 ) i ++;
          Console.WriteLine(i);
     }
          Console.ReadLine();
}
```



i değişkeninin değeri 5 olduğunda i değişkeninin değerini bir arttırarak değişkenin içeriğinin yazılmadan bir arttırılmasını sağladık.

Diğer yöntemde ise continue komutunu kullanarak döngünün bir sonraki adımdan ilerletilmesini sağlarız.

```
static void Main(string[] args)
{
     byte i;
     for (i = 0; i <= 10; i++)
     {
          if (i == 5 ) continue ;
          Console.WriteLine(i);
     }
```

```
        Console.ReadLine();
}
```

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 btnSayıGirisi_Click(object sender, EventArgs e)
{
    int i,sayi,fak;
    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 `for` döngüsünden döngü sonuna gelmeden çıkmak için `break` komutu 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` komutu ile döngüden çıkılmasını sağladık.

FOREACH DÖNGÜSÜ

Listeler ya da diziler üzerinde işlem yapmak için kullanılan döngü yapısıdır. Özellikle eleman sayısının bilinmediği durumlarda kullanılabilmesi özelliği 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;

```
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. Döngü içindeki komutlar mutlaka bir defa icra edilir.

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 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. `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.



10

STRING İŞLEMLERİ

Stringler tanımlanırken;

```
String Değişken adı = İlk Değer;
```

şeklinde tanımlanırlar.

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.

Stringler üzerinde işlem gerçekleştirilirken `String` sınıfını, `StringBuilder` sınıfını ya da `Microsoft.VisualBasic` 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 stringin içeriğini başka bir stringe kopyalamak amacı ile kullanılır.

```
static void Main(string[] args)
{
    string a;
    string b;
    a = "Bilgisayar";
    b = string.Copy(a);
    Console.WriteLine(b);
    Console.ReadKey();
}
```

a değişkenin içeriği b değişkenine kopyalanarak "Bilgisayar" değerini alacaktır.

CONCAT Metodu

İki stringi birleştirmek amacı ile kullanılır.

Kullanımı şu şekildedir:

```
String.Concat(String1 , String2);
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.

EMPTY METODU

Bir stringin içeriğini 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 stringi karşılaştırmak amacı ile kullanılır.

Kullanımı şu şekildedir:

String.Compare(String1 , String2, Karşılaştırma Seçenekleri)

Karşılaştırma seçenekleri aşağıdaki değerleri alabilir.

CurrentCulture: Stringler karşılaştırılırken geçerli kültür ayarları kullanılır. Karşılaştırma esnasında büyük küçük harf ayrımı göz önünde bulundurulur.

CurrentCultureIgnoreCase: Stringler 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 stringler 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 btnTamam_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.");
    }
}
```

`textBox1` içerisine yazdığımız yazıyı `textBox2` içerisine büyük harfle yazarak karşılaştırma işlemi yaptığımızda yanlış sonuç üretecektir.



Kodumuzu değiştirerek `String.Compare` metodunu `StringComparison.CurrentCultureIgnoreCase` parametresi ile kullanalım.

```
private void btnTamam_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 doğru sonuç üretecektir.



`String.Compare` metodunu `StringComparison.InvariantCulture-IgnoreCase` 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 btnTamam_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.

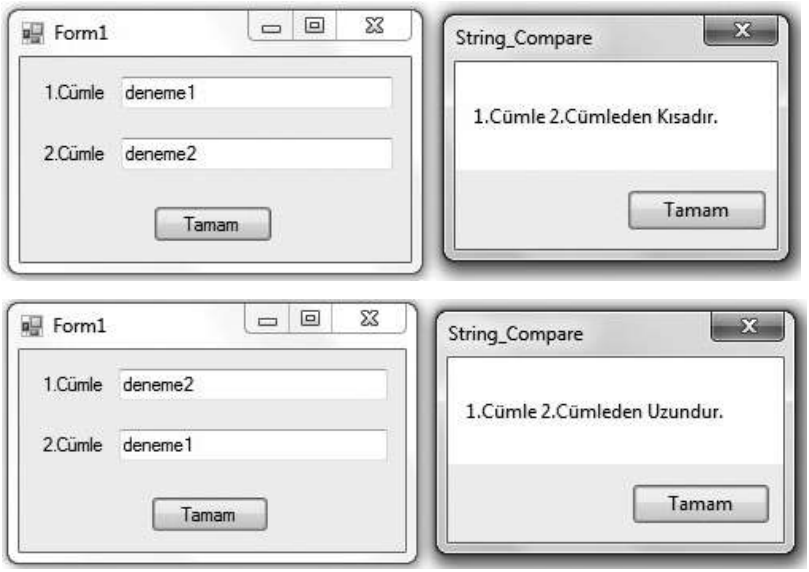


`String.Compare` metodunu `StringComparison.Ordinal` parametresi ile kullanalım.

```
private void btnTamam_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 stringler içerisinde bulunan rakamlar karşılaştırılarak bu rakamların büyüklük ve küçüklük durumuna göre stringlerin uzunluk ve kısalığı değerlendirilir.



SPLIT METODU

Bir metni ya da metin dizisini belirtilen bir karaktere göre parçalara bölmek amacı ile kullanılır.

Kullanımı şu şekildedir:

`String Değişken Adı.Split(Karakter)`

Bir text kutusuna girilen yazının kelime sayısını bulan programı `Split` fonksiyonu 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

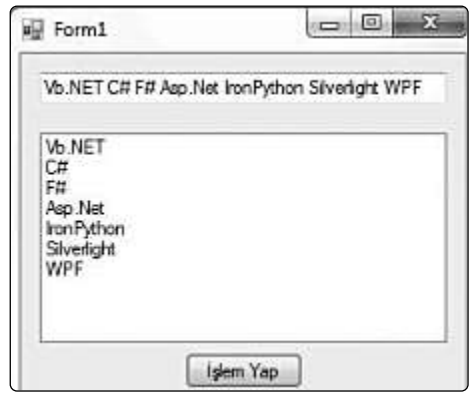
```
Split(' ')
```

metodunda cümlemizi boşluk karakterine göre parçalara ayırarak `Length` metodu ile ayrılan parçaların sayısını buluyoruz.

Diğer bir örnekle de bir text kutusu içerisindeki yazıyı kelimelere ayırıp bir listbox içerisine ekleyen bir örnek yapalım.

```
private void btnislem_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 bir döngü içerisinde 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 btnislem_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 btnislem_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 iki 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 belirtilen ayırıcı karakter kullanılarak aktaran metoddur.

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 btnislem_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 da-

ha sonra bu dizinin elemanları textbox içerisine join metodu yardımıyla "," karakteri kullanılarak aktarıldı.



INDEXOF, INDEXOFANY, LASTINDEX VE LASTINDEXOFANY METODLARI

IndexOf Metodu: Belirtilen bir karakter ya da stringin başka bir string içerisinde olup olmadığını, eğer mevcutsa hangi konumda olduğu bilgisini verir.

IndexOfAny Metodu: Belirtilen bir dizi içerisindeki karakterlerin başka bir string içerisinde olup olmadığını, eğer mevcutsa hangi konumda olduğu bilgisini verir.

LastIndexOf Metodu: `IndexOf` metodu gibi belirtilen bir karakter ya da stringin başka bir string içerisinde olup olmadığını, eğer mevcutsa hangi konumda olduğu bilgisini verir. Ancak arama işlemini sondan itibaren yapar.

LastIndexOfAny Metodu: `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 metod da geriye 3 değer döndürür.

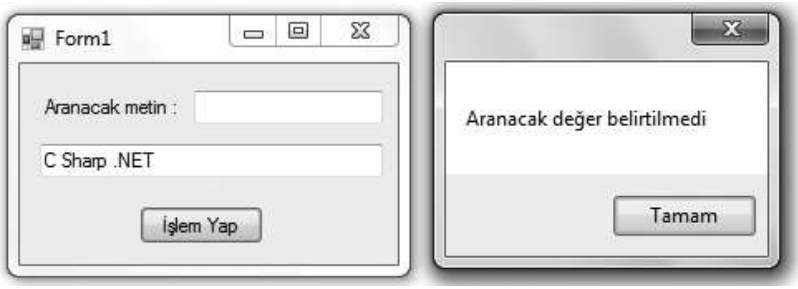
Eğer dönen değer `0` ise aranacak karakter ya da string belirtilmemiştir.

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;

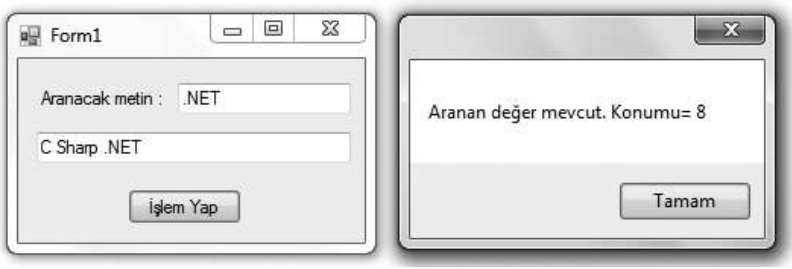
```
private void btnislem_Click(object sender, EventArgs e)
{
     if ((textBox1.Text.IndexOf(txtAranacakMetin.Text) == 0)) {
          MessageBox.Show("Text kutusu boş"); }
     else if (textBox1.Text.IndexOf(txtAranacakMetin.Text) == -1) {
          MessageBox.Show("Aranan değer mevcut değil"); }
     else {
          MessageBox.Show("Aranan değer mevcut. Konumu= "
               +(textBox1.Text.IndexOf(txtAranacakMetin.Text))); }
}
```



Aranacak metni belirtmeyip işlem yaptığımızda dönen değer sıfır olacağından dolayı *"Aranacak değer belirtilmedi"* mesajı görüntülenecektir.



Aranan metin bulunamadığında dönen değer `-1` olacağından dolayı *"Aranan değer mevcut değil"* mesajı görüntülenecektir.

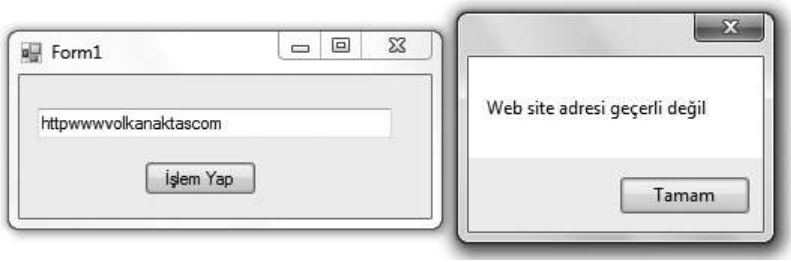


Aranan metin bulunduğunda *"Aranan değer mevcut"* mesajı ile birlikte bulunan metnin başlangıç konumu görüntülenecektir.

`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 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));
}
```

Kodumuzu çalıştırıp örnek bir adres girelim;



Görüldüğü gibi site adresinin ilk noktadan sonrasını keserek bir mesaj penceresi içerisinde görüntüledi.

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 başlıca 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.

• **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.

`String.Format` metodunu en basit hali ile kullanarak herhangi bir format biçimlendirmesi yapmadan sadece index özelliğini kullanan bir uygulama yapalım.

```
static void Main(string[] args)
{
    string numara1;
    string numara2;
    string sonuc;
    Console.Write("1. Numarayı Girin: ");
    numara1 = Console.ReadLine();
    Console.Write("2. Numarayı Girin: ");
```

```
        numara2 = Console.ReadLine();
        Console.Clear();
        sonuc = string.Format("Girilen Numaralar {0} ve {1}",
                              numara1, numara2);
        Console.WriteLine(sonuc);
        Console.ReadKey();
}
```

Kodumuzu çalıştırdığımızda örnek olarak `5` ve `10` sayılarını girdiğimizde bu girilen değerler `numara1` ve `numara2` değişkenlerine aktarılır. Bu değişkenlerin içeriğini String.Format metodu ile `{0} {1}` şeklinde `sonuc` değişkenine aktararak `WriteLine` komutu ile yazdırıyoruz. Konsol ekranında *"Girilen Numaralar 5 ve 10"* mesajı görüntülenecektir.

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();
}
```



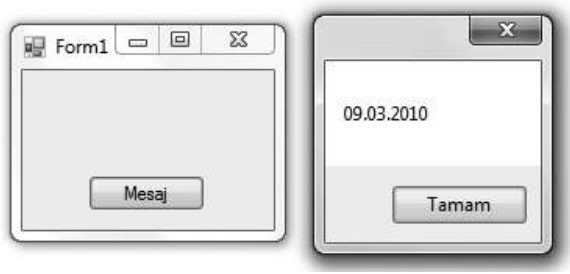
STANDART DATE AND TIME FORMAT STRINGS

Karakter	Tanımı	Örnek
"d"	Kısa tarih	25.11.2009 20:54:39 -> 25.11.2009
"D"	Uzun tarih	25.11.2009 20:54:39 -> 25, Kasım, Çarşamba
"f"	Uzun tarih/zaman (kısa zaman)	25.11.2009 20:54:39 -> 25, Kasım, Çarşamba 20:54
"F"	Uzun tarih/zaman (uzun zaman)	25.11.2009 20:54:39 -> 25, Kasım, Çarşamba 20:54:39

"g"	Genel tarih/zaman (kısa zaman)	25.11.2009 20:54:39 -> 25/11/2009 20:54
"G"	Genel tarih/zaman (uzun zaman)	25.11.2009 20:54:39 -> 25/11/2009 20:54:39
"M", "m"	Ay/Gün	25.11.2009 20:54:39 -> 25 Kasım
"t"	Kısa zaman	25.11.2009 20:54:39 -> 20:54
"T"	Uzun zaman	25.11.2009 20:54:39 -> 20:54:39
"Y", "y"	Yıl/Ay	25.11.2009 20:54:39 -> Kasım, 2009

`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);
}
```



CUSTOM DATE AND TIME FORMAT STRINGS

Karakter	Tanımı
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 şeklinde görüntülenir.
yyyy	Yıl bilgisini tam hali ile görüntüler. Yıl bilgisi 2009 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;
```

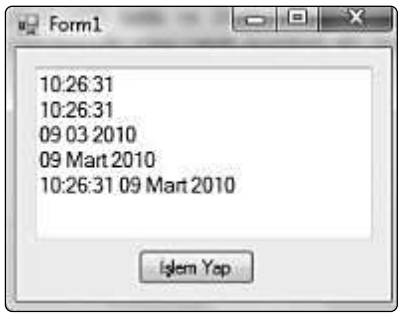
```
    sonuc = string.Format("{0:hh:mm:ss}", DateTime.Now);
    textBox1.AppendText(sonuc + "\n");

    sonuc = string.Format("{0:HH:mm:ss}", DateTime.Now);
    textBox1.AppendText(sonuc + "\n");

    sonuc = string.Format("{0:dd MM yyyy}", DateTime.Now);
    textBox1.AppendText(sonuc + "\n");

    sonuc = string.Format("{0:dd MMMM yyyy}", DateTime.Now);
    textBox1.AppendText(sonuc + "\n");

    sonuc = string.Format("{0:HH:mm:ss dd MMMM yyyy}",
      DateTime.Now);
    textBox1.AppendText(sonuc + "\n");
}
```



`09.03.2010` tarihi ve `10:26:31` zamanı için işlemler yaptığımızda sonuç yanda görüldüğü gibi olacaktır.

STRINGBUILDER SINIFI

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` isim uzayını dahil etmemiz gerekir.

```
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.

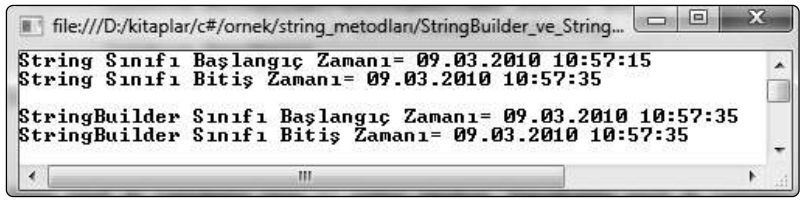
`StringBuilder` sınıfı ile `String` sınıfının performansını karşılaştıran örnek bir uygulama yapalım:

```
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();
        }
    }
}
```

String sınıfı ile yaptığımız işlem 20 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 SINIFI ÖZELLİKLERİ

LENGTH ÖZELLİĞİ

Stringbuilder içerisinde mevcut olan string değerlerin uzunluğunu verir.

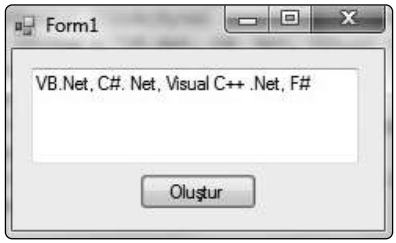
CAPACITY ÖZELLİĞİ

Stringbuilder içerisinde saklanabilecek string değer kapasitesini verir.

ENSURECAPACITY ÖZELLİĞİ

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.

MAXCAPACITY Özelliği

Stringbuilder içerisinde saklanabilecek maksimum veri kapasitesini belirtir. Bu değer varsayılan olarak `2147483647` değeridir.

STRINGBUILDER Sınıfı Metodları

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 `Sting.Format` konusunda ayrıntılı olarak belirtildiğinden dolayı burada yazılmamıştır.

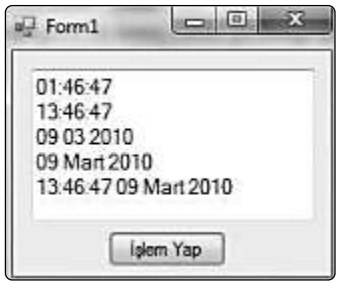
Length: Biçimlendirme işleminin uygulanacağı karakter uzunluğunu belirtir.

```
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());
    }
```



`09.03.2010` tarihi ve `13:46:47` zamanı için işlemler yaptığımızda sonuç yanda görüldüğü gibi olacaktır.

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)`

İndex numarası: StringBuilder içerisine eklenecek olan verinin index numarasını int32 türünden belirtir. StringBuilder içerisinde index numaraları 0'dan itibaren başlar.

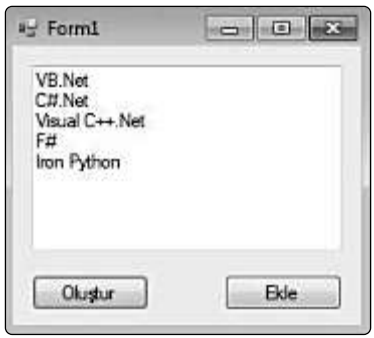
Veri: Stringbuilder içerisine eklenecek olan veridir.

Adet: Eklenecek olan verinin adetini belirtir.

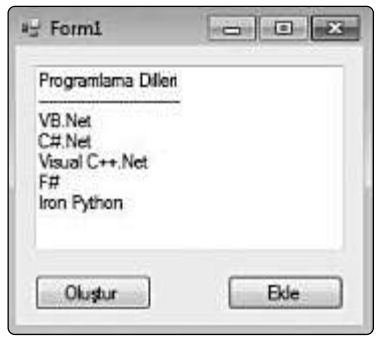
```
using System;
using System.Text;
using System.Windows.Forms;
namespace StringBuilder_Insert
{
```

```
    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();
        }
    }
}
```



Ekleme öncesi



Ekleme sonrası

REMOVE Metodu

Stringbuilder içerisinde istediğimiz bir veriyi silmek için Remove metodunu kullanabiliriz.

Kullanımı şu şekildedir.

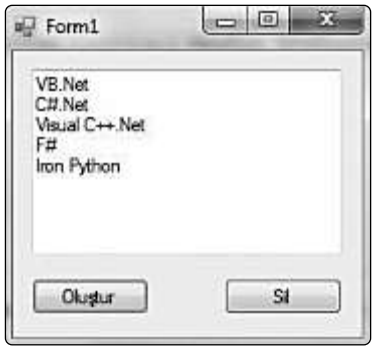
StringBuilder.Remove(başlangıç, uzunluk)

Başlangıç: StringBuilder içerisinden silinecek olan verinin index numarasını belirtir.

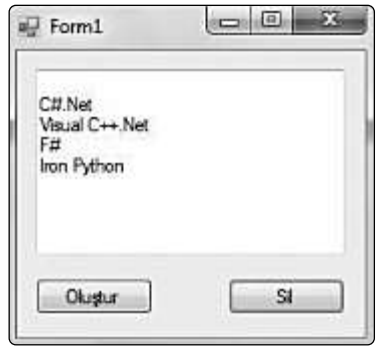
Uzunluk: StringBuilder içerisinden silinecek olan verinin uzunluğunu belirtir.

```
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 btnSil_Click(object sender, EventArgs e)
        {
            sb.Remove(0, 6);
            textBox1.Text = sb.ToString();
        }
    }
}
```



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.

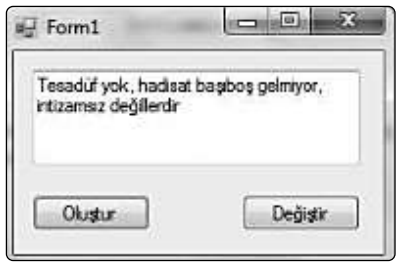
Yeni değer: Değiştirilecek olan verinin yerine konulacak veriyi belirtir.

Başlangıç: StringBuilder içerisinde değiştirme işleminin başlayacağı index numarasını belirtir.

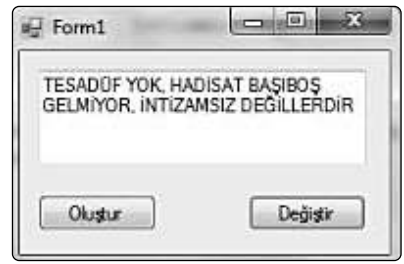
Adet: Değiştirme işleminin kaç adet tekrarlanacağını belirtir.

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



Değiştirme öncesi



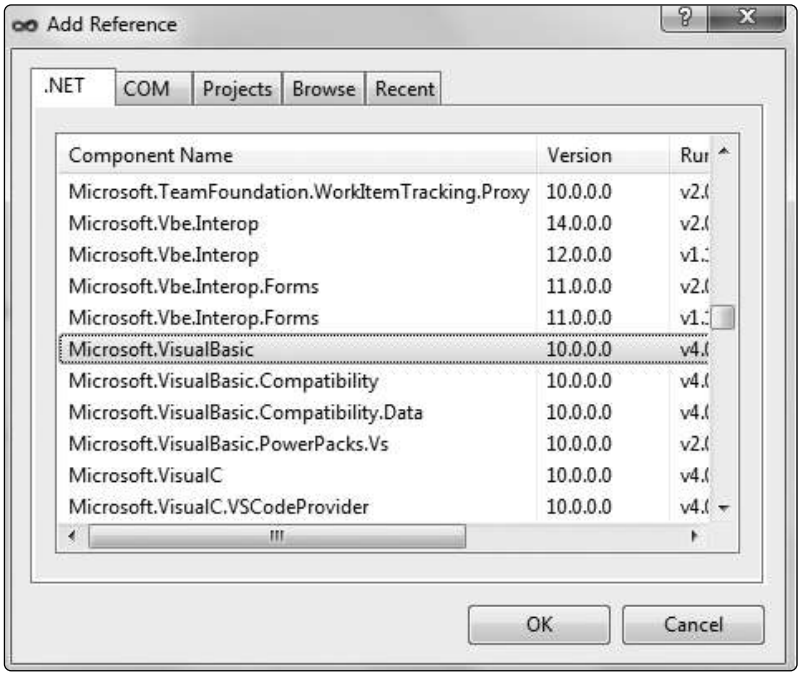
Değiştirme sonrası

CLEAR METODU

StringBuilder içerisindeki tüm karakterleri silmek amacı ile kullanılır.

MICROSOFT.VISUALBASIC SINIFI

Microsoft.VisualBasic sınıfı içerisinde Visual Basic 6 versiyonundan bu yana kullanılan string metodları mevcuttur. C# içerisinde bu metodları kullanabilmek için öncelikle Microsoft.VisualBasic sınıfını **Project/Add Reference** seçeneği ile ekleyip daha sonra **using Microsoft.Visual Basic** şeklinde projemize dahil etmeliyiz.



Bu metodların başlıcaları şunlardır:

CHR Metodu

ASCII kodu verilen sayının karakter karşılığını verir.



Kodu 32'den küçük olan karakterler yazım karakteri değil özel kodlardır. Örneğin 13 enter, 27 Esc, 32'de boşluk tuşunun karakter kodlarıdır.

Kullanımı şu şekildedir:

```
Chr(ASCII Kodu)
```

ASCII karakter tablosunu görüntüleyecek bir program yazalım. Örneğimiz için form üzerine bir adet textbox ekleyeceğiz. Texbox nesnesi içerisine birden fazla satır ekleyeceğimiz için, multiline özelliği true olmalı.

Kodumuz aşağıdaki gibi olacak:

```
using System;
using Microsoft.VisualBasic;
```

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

        private void Form1_Load(object sender, EventArgs e)
        {
           int i;
           for(i = 32;i<=255;i++)
           {
                textBox1.Text = textBox1.Text + "  " + i + ":" +
                                Strings.Chr(i)+ Strings.Space(3);
           }
        }
    }
}
```

`Chr` metodunu kullanmak için;

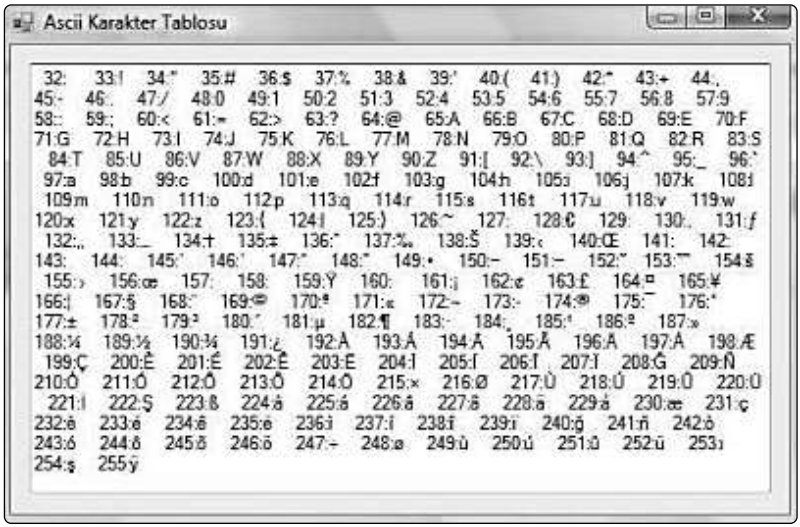
```
using Microsoft.VisualBasic;
```

ifadesi ile `Microsoft.VisualBasic` sınıfını projemize eklemek yerine

```
Microsoft.VisualBasic.Strings.Chr(i)
```

şeklinde de `Chr` metodunu kullanabiliriz.

`Space` metodu belirtilen miktar kadar boşluk bırakmak amacı ile kullanılır. Kodumuzu çalıştırdığımızda textbox içerisine ASCII karakter tablosuna ait karakterlerin eklendiğini görürüz:



LEN METODU

Bir metnin içerisindeki karakter sayısını geriye döndürür.

Kullanımı şu şekildedir:

```
Len(string ifade veya string değişken)
```

String ifade: Üzerinde işlem yapılacak olan metin.

String değişken: Üzerinde işlem yapılacak olan string değişken.

MID METODU

Bir metnin içerisinde başlangıçtan belirtilen uzunluğa kadar olan karakteri verir. Geriye string türünden bir değer döndürür.

Kullanımı şu şekildedir:

```
Mid(string , başlangıç ,uzunluk)
```

String: Üzerinde işlem yapılacak olan metin.

Başlangıç: String içerisindeki başlangıç karakterini belirtir.

Uzunluk: Başlangıç değerinden itibaren kaç karakter üzerinde işlem yapılacağını belirtir.

Bir cümledeki kelime sayısını bulan örnek bir uygulama yapalım. Örneğimiz için form üzerine bir adet textbox eklememiz yeterlidir.

```
using System;
using Microsoft.VisualBasic;
using System;
using Microsoft.VisualBasic;
using System.Windows.Forms;
namespace Mid_Metodu
{
  public partial class Form1 : Form
  {
       public Form1()
       {
            InitializeComponent();
       }
       private void textBox1_KeyPress(object sender,
       KeyPressEventArgs e)
       {
            int bosluksay=0;
            int i;
            for(i = 1;i<= Strings.Len(textBox1.Text);i++)
            {
               if (Strings.Mid(textBox1.Text, i, 1) ==
                                  Strings.Chr(32).ToString())
               {
                    bosluksay++;
               }
            }
            if (e.KeyChar == Strings.Chr(13))
            {
              MessageBox.Show("Cümlenin Kelime Sayısı= " +
               (bosluksay + 1));
            }
       }
  }
}
```



Kodumuzu textbox'ın keypress olayına yazdığımız için textbox içerisinde herhangi bir tuşa bastığımız anda kodumuz çalışmaktadır.

```
for(i = 1;i<= Strings.Len(textBox1.Text);i++)
```

satırı ile döngümüzü 1'den text kutusuna girilen yazının uzunluğuna kadar kuruyoruz.

```
if (Strings.Mid(textBox1.Text, i, 1) ==
  Strings.Chr(32).ToString())
```

satırı ile döngü içerisinde sırası ile karakterleri kontrol ediyoruz. Eğer basılan tuşun karakter kodu 32 ise (yani basılan tuş boşluk tuşu ise) boşluk sayısını bir arttırıyoruz.

```
if (e.KeyChar == Strings.Chr(13))
{
     MessageBox.Show("Cümlenin Kelime Sayısı= " + (bosluksay + 1));
}
```

satırı ile de basılan tuşun kodu 13 ise (yani basılan tuş enter ise) kelime sayısını boşluk sayısına bir ekleyerek bir mesaj penceresi içerisinde yazdırıyoruz. Boşluk sayısına bir eklememizin nedeni kelime sayısının boşluk sayısından bir fazla olmasıdır.

LEFT VE RIGHT METODLARI

`Left` fonksiyonu, bir metnin sol tarafından itibaren belirtilen miktarda karakterini verir. `Right` fonksiyonu ise, bir metnin sağ tarafından itibaren belirtilen miktarda karakterini verir.

Kullanımları şu şekildedir:

```
Left (string, uzunluk)
Right (string, uzunluk)
```

Eğer uzunluk parametresinde girilen değer, üzerinde işlem yapılacak olan metnin uzunluğundan büyükse geriye metnin tamamı döndürülür.

Örnek olarak:

```
String testcumlesi = "KodLab Yayınevi";

MessageBox.Show(Microsoft.VisualBasic.Strings.Left(testcumlesi, 6));
```

Komut satırları *"Kodlab"* sonucunu üretirken

```
String testcumlesi = "KodLab Yayınevi";

MessageBox.Show(Microsoft.VisualBasic.Strings.Right(testcumlesi, 8));
```

Komut satırları *"Yayınevi"* sonucunu üretecektir.

TRIM, LTRIM VE RTRIM METODLARI

`Trim` fonksiyonu belirtilen metnin her iki tarafındaki boşlukları kaldırır. `LTrim` fonksiyonu metnin başındaki boşlukları kaldırırken, `RTrim` fonksiyonu ise metnin sonundaki boşlukları kaldırır.

Kullanımları şu şekildedir:

`Trim( string ), LTrim( string ), RTrim( string )`

Örneğin;

```
string mesaj = "  Bugün Hava Yağmurlu   ";
mesaj = Strings.Trim(mesaj);
```

şeklindeki bir işlem bize mesajın sağındaki ve solundaki boşlukları kırparak *"Bugün Hava Yağmurlu"* sonucunu verecektir.

```
string mesaj = "  Bugün Hava Yağmurlu   ";
mesaj = Strings.LTrim(mesaj);
```

Şeklindeki bir işlem bize mesajın solundaki boşlukları kırparak *" Bugün Hava Yağmurlu"* sonucunu verecektir.

```
string mesaj = "  Bugün Hava Yağmurlu   ";
mesaj = Strings.RTrim(mesaj);
```

şeklindeki bir işlem bize mesajın sağındaki boşlukları kırparak *"Bugün Hava Yağmurlu "* sonucunu verecektir.

INSTR Metodu

Bir metnin içerisinde belirtilen bir karakter ya da karakter dizisini arama amacıyla kullanılır.

Kullanımı şu şekildedir:

InStr([başlangıç,]arama yapılacak metin, aranacak metin[, karşılaştırma şekli])

Karşılaştırma şekli iki değer alır:

CompareMethod.Text: Text karşılaştırma yapar.

CompareMethod.Binary: Binary karşılaştırma yapar.

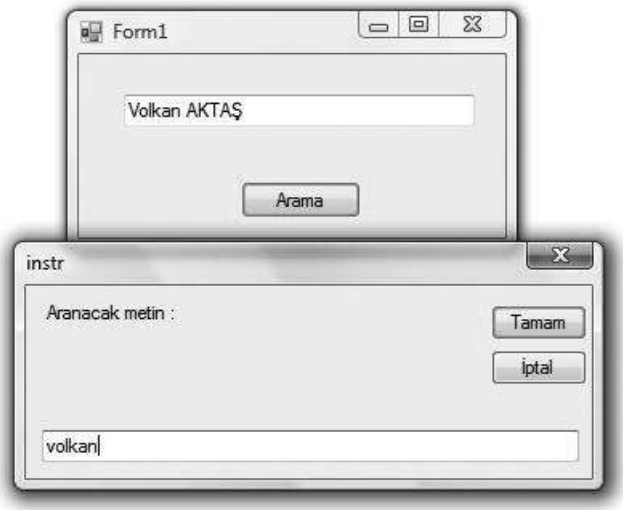
Örnek olarak bir text kutusu içerisinde arama yaparak bulunan kelime ya da kelimeleri işaretleyen bir kod yazalım. Form üzerine bir adet textbox, bir adet de command button nesnesi ekliyoruz.

Kodumuz şu şekilde olacak:

```
using System;
using Microsoft.VisualBasic;
using System.Windows.Forms;
namespace instr_metodu
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void btnArama_Click(object sender, EventArgs e)
        {
          String aranan;
          int yer=0;
          aranan = ↓
Strings.LCase(Interaction.InputBox("Aranacak metin :"));
          if (Strings.Len(aranan) < Strings.Len(textBox1.Text))
          {
            yer = Strings.InStr(1, textBox1.Text,
                                aranan,CompareMethod.Text);
```

```
            if (yer != 0)
            {
                textBox1.Focus();
                textBox1.Select(yer - 1, Strings.Len(aranan));
            }
            else
            {
              MessageBox.Show("String bulunamadı.");
            }
          }
        }
    }
}
```

Kodumuzu çalıştırarak text kutusu içerisine örnek bir ifade girelim ve arama yapalım:





Tamam düğmesine bastığımızda aradığımız kelime bulunarak işaretlenecektir.

Burada instr metodu içerisinde `CompareMethod.Text` parametresini kullanarak karşılaştırma işlemi yaptığımız için text kutusu içerisinde *"Volkan"* kelimesini baş harfi büyük olarak yazıp küçük harfle *"volkan"* şeklinde arama yapmamıza rağmen kelime bulunarak işaretlenmiştir.



Karşılaştırma parametresini `CompareMethod.Binary` şekline çevirerek aynı işlemi bir daha yapalım. Görüldüğü gibi karşılaştırma işlemi binary metoda göre yapıldığı için büyük küçük harf ayrımı yapılmış ve aradığımız kelime küçük harfle yazıldığı için *"string bulunamadı"* mesajı görüntülenmiştir.

INSTRREV METODU

`INSTRREV` metodu da `instr` metodu gibi bir metnin içerisinde belirtilen bir karakter ya da karakter dizisini arama amacıyla kullanılır. Ancak `instr` fonksiyonuna göre tam ters bir şekilde arama yapar. Arama işlemi `instr` fonksiyonundaki gibi soldan sağa değil de, sağdan sola olacak şekilde yapılır. Aranan bilgi 2. parametrede verilmelidir.

Kullanımı şu şekildedir:

```
InStrRev(arama yapılacak metin, aranacak metin [,başlangıç ]
[, karşılaştırma şekli])
```

Karşılaştırma şekli iki değer alır:

CompareMethod.Text: Text karşılaştırma yapar.

CompareMethod.Binary: Binary karşılaştırma yapar.



Eğer başlangıç değeri olarak `-1` verilirse arama işlemi en son karakterden başlanarak yapılır.

`InStr` fonksiyonu ile yaptığımız örnek uygulamayı şimdi de `InstrRev` fonksiyonu ile yapalım. Kodumuz aşağıdaki gibi olacak:

```
using System;
using Microsoft.VisualBasic;
using System.Windows.Forms;
namespace instr_metodu
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void btnArama_Click(object sender, EventArgs e)
        {
          String aranan;
          int yer=0;
          aranan = ↓
Strings.LCase(Interaction.InputBox("Aranacak metin :"));
          if (Strings.Len(aranan) < Strings.Len(textBox1.Text))
          {
            yer = Strings.InStr(textBox1.Text, aranan,
                                -1,CompareMethod.Text);
            if (yer != 0)
            {
                textBox1.Focus();
                textBox1.Select(yer - 1, Strings.Len(aranan));
            }
            else
            {
              MessageBox.Show("String bulunamadı.");
            }
          }
        }
    }
}
```

`InStrRev(textBox1.Text, aranan, -1, CompareMethod.Text)` şeklindeki kullanım yerine `InStrRev(textBox1.Text, aranan)` şeklinde-

ki kullanımı da tercih edebiliriz. Çünkü arama işleminde varsayılan başlangıç karakteri en son karakterdir. Ayrıca karşılaştırma metodu olarak da `CompareMethod.Text` metodu varsayılan karşılaştırma metodudur.

STRCOMP METODU

İki string ifade ya da değişkeni karşılaştırarak geriye integer bir değer döndürür.

Kullanımı şu şekildedir:

`StrComp(string1, string2[, karşılaştırma şekli])`

Karşılaştırma şekli iki değer alır:

CompareMethod.Text: Text karşılaştırma yapar.

CompareMethod.Binary: Binary karşılaştırma yapar.

Geriye dönen integer değerler ve anlamları şu şekildedir:

string1, string2'den küçükse	-1
string1,string2'ye eşitse	0
string1,string2'den daha büyükse	1

Örnek olarak bir iki adet text kutusu içerisine girilen metinleri karşılaştıracak uygulama yapalım.

Kodumuz şu şekilde olacak:

```
using System;
using Microsoft.VisualBasic;
using System.Windows.Forms;
namespace strcomp_metodu
{
  public partial class Form1 : Form
  {
    public Form1()
    {
        InitializeComponent();
    }
    private void btnKarsilastir_Click(object sender, EventArgs e)
    {
       int karsilastirma;
```

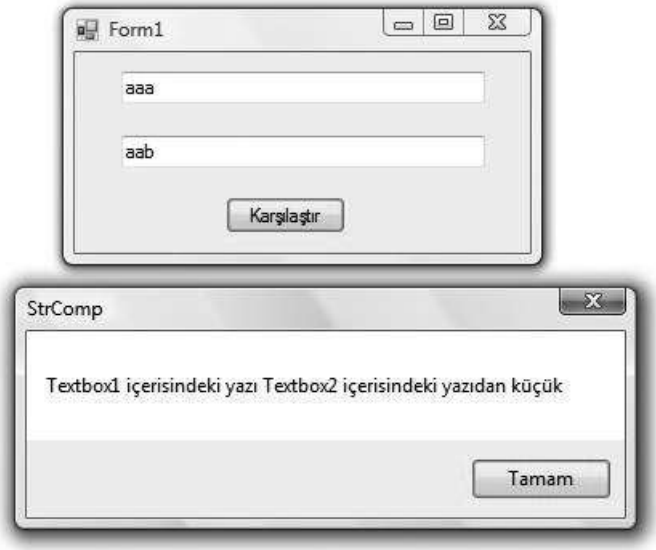
```
        karsilastirma = Strings.StrComp(textBox1.Text,
            textBox2.Text, CompareMethod.Text);
        if (karsilastirma == -1)
        {
            MessageBox.Show
             ("Textbox1 içerisindeki yazı Textbox2 içindeki ↓
             yazıdan küçük");
        }
        else if (karsilastirma == 1)
        {
        MessageBox.Show
         ("Textbox1 içerisindeki yazı Textbox2 içindeki ↓
         yazıdan büyük");
        }
        else if (karsilastirma == 0)
        {
            MessageBox.Show
             ("Textbox1 içerisindeki yazı Textbox2 içindeki ↓
             yazıya eşit");
        }
    }
  }
}
```

Uygulamamızı çalıştırarak örnek değerler girelim:



Yukarıda görüldüğü gibi `textbox1` ve `textbox2` içerisindeki değerler birbirine eşit olduğu için *"Textbox1 içerisindeki yazı Textbox2 içerisindeki yazıya eşit"* şeklinde bir mesaj görüntülendi.

Şimdi ise text kutusu içerisindeki değerlere birer harf ilave ederek karşılaştırma işlemini yeniden yapalım.



Yukarıda görüldüğü gibi farklı olan karakterler bulunduğu anda farklı olan karakterler karşılaştırılmış ve **"a"** harfi **"b"** harfinden önce olduğu için *"Textbox1 içerisindeki yazı Textbox2 içerisindeki yazıdan küçük"* şeklindeki mesaj görüntülenmektedir.

UCASE VE LCASE METODLARI

`UCase` bir metni büyük harfe çevirmek için kullanılırken, `LCase` ise küçük harfe çevirmek için kullanılır.

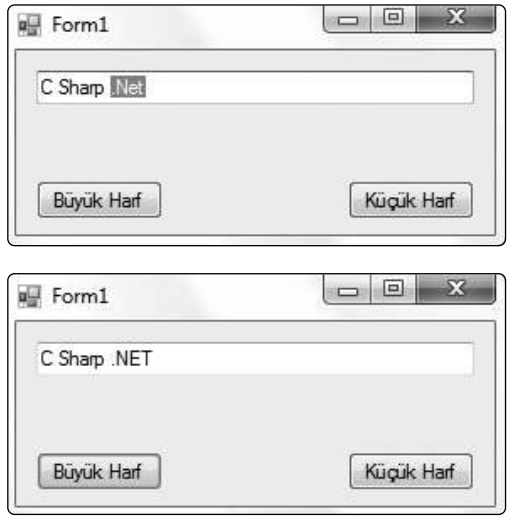
Kullanımları şu şekildedir:

```
UCase (String), LCase (String)
```

Örnek olarak bir text kutusunda seçili bir kısım varsa seçili kısmı, seçili kısım yoksa tümünü büyük ya da küçük harfe çevirecek bir uygulama yapalım.

Kodumuz şu şekilde olacak:

```
using System;
using Microsoft.VisualBasic;
using System.Windows.Forms;
namespace ucase_lcase_metodları
{
  public partial class Form1 : Form
  {
    public Form1()
    {
       InitializeComponent();
    }
    private void btnBuyuk_Click(object sender, EventArgs e)
    {
       if (textBox1.SelectionLength > 0)
       {
           textBox1.SelectedText =
             Strings.UCase(textBox1.SelectedText);
       }
       else
       {
           textBox1.Text = Strings.UCase(textBox1.Text);
       }
    }
    private void btnBuyuk_Click(object sender, EventArgs e)
    {
       if (textBox1.SelectionLength > 0)
       {
           textBox1.SelectedText =
              Strings.LCase(textBox1.SelectedText);
       }
       else
       {
           textBox1.Text = Strings.LCase(textBox1.Text);
       }
    }
  }
}
```



Kodumuzu çalıştırarak örnek bir ifade girelim. Text kutusu içerisinde bir bölümü seçerek "Büyük Harf" düğmesine bastığımızda seçili olan kısmın büyük harfe çevrildiğini görürüz.

STRCONV METODU

Bir metni belirtilen özel bir biçime dönüştürmek için kullanılır.

Kullanımı şu şekildedir:

```
StrConv(string, biçim, [LocaleID])
```

`LocaleID` seçeneği isteğe bağlı olarak kullanılabilir. Eğer yapılan uygulamanın `LocaleID` değeri, sistem `LocaleID`'sinden farklıysa kullanılabilir.

Kullanılabilecek dönüştürme biçimleri şunlardır;

VbStrConv.None: Herhangi bir dönüştürme işlemi yapılmaz.

VbStrConv.LinguisticCasing: Bu seçeneğin tercih edilerek kullanılması durumunda, yalnızca `VbStrConv.UpperCase` ve `VbStrConv.LowerCase` seçenekleri kullanılabilir. Bu seçenekleri kullanabilmek için `VbStrConv.LinguisticCasing` seçeneğinden sonra "+" işareti ekleyerek `VbStrConv.UpperCase` ya da `VbStrConv.LowerCase` seçeneği yazılır.

VbStrConv.UpperCase: Büyük harfe dönüştürme işlemi yapar.

VbStrConv.LowerCase: Küçük harfe dönüştürme işlemi yapar.

VbStrConv.ProperCase: Kelimelerin ilk harflerini büyük harfe dönüştürür.

VbStrConv.Wide: Dar karakterleri, geniş karakterlere çevirir.

VbStrConv.Narrow: Geniş karakterleri, dar karakterlere çevirir.

VbStrConv.Katakana: Hiragana karakterlerini Katakana karakterlerine çevirir.

VbStrConv.Hiragana: Katakana karakterlerini Hiragana karakterlerine çevirir.

VbStrConv.SimplifiedChinese: Geleneksel Çince karakterlerini basitleştirilmiş Çince karakterlere çevirir.

VbStrConv.TraditionalChinese: Basitleştirilmiş Çince karakterlerini geleneksel Çince karakterlere çevirir.

Örneğin;

```
string TestCumlesi  = "sümeyye aktaş";
TestCumlesi = Strings.StrConv(TestCumlesi, VbStrConv.Uppercase);
```

Komut satırları *"SÜMEYYE AKTAŞ"* sonucunu üretir.

11

REGULAR EXPRESSIONS (DÜZENLİ İFADELER)

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:

1. **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.

2. 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 metodlara 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.

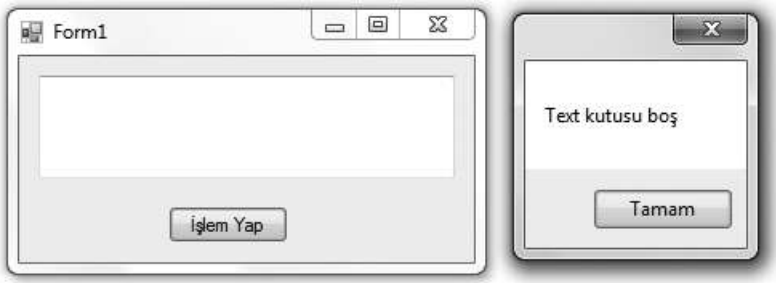
• **ECMAScript:** ECMAScript uyumluluğunu sağlar. `IgnoreCase`, `Multiline` ve `Compiled` değerleri ile birlikte kullanılır.

• **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());
    }
}
```

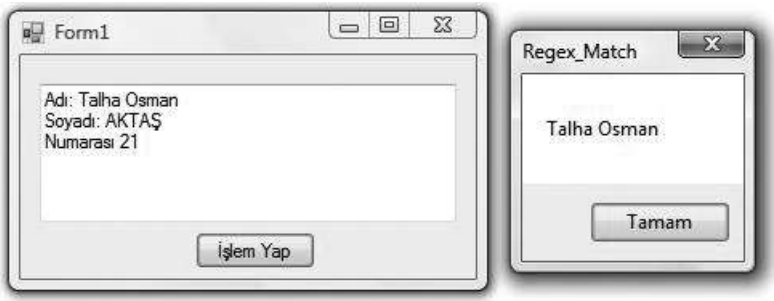
Örneğimizde

^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, tckimlik, telefon numaralarını kontrol ederek yanlış bir değer girişini önleyen bir uygulama yapalım.

```
private void txtAdSoyad_Leave(object sender, EventArgs e)
{
     Regex regex = new Regex("^([\\s\\S]){1,30}([\\s\\.])");
     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

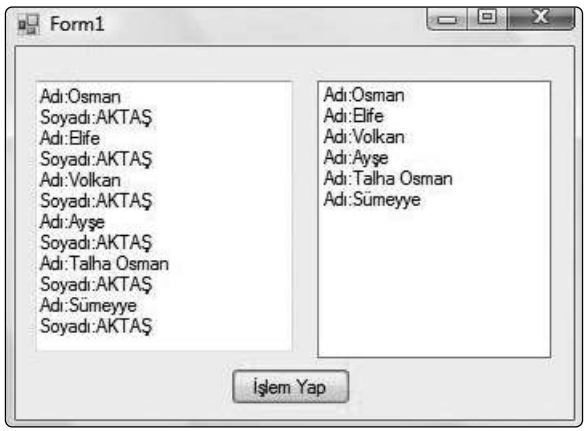
Match metodunda olduğu gibi bir metni başka bir metin içerisinde aramak amacı ile kullanılır. Ancak bulunan tek bir sonucu değil tüm sonuçları içerir.

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}");
```

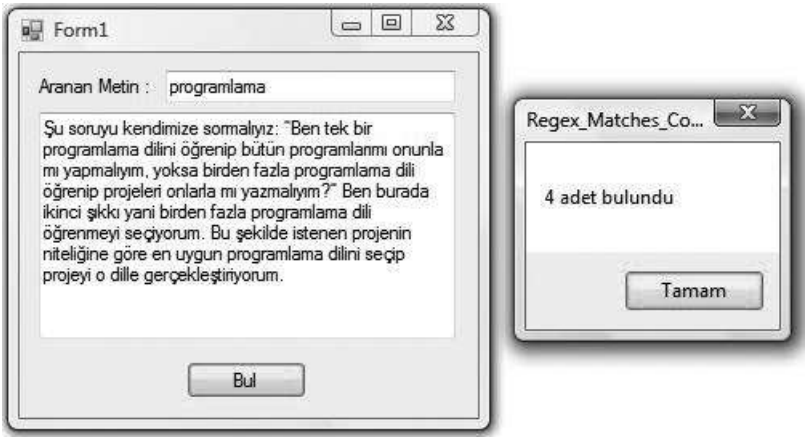
satırında mail deseni adlı yeni bir Regex nesnesi oluşturuyoruz.

```
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.

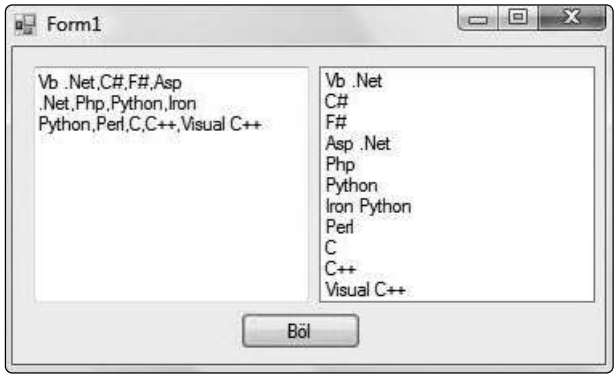
Split Metodu

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 `","` 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.

12

HATALARI BULMAK VE KONTROL ETMEK

Hatalar, programın çalışması esnasında oluşurlar ve programın kendi çalışmasının kesilmesine neden olur. 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 yapılarak gerekirse kullanıcıyı bilgilendirerek ondan alınan geri dönüşlerle, gerekmiyorsa programın kendi içerisinde yanlış sonuçlar üretmesine neden olmayacak şekilde programın akışını yönlendirerek oluşabilecek hataları kontrol etmektir.

Try Catch Finally Yapisi

Kodlamada oluşan hataları yakalamak için try catch metodu 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. Kodumuzda tek bir hata durumu oluştuğu için herhangi bir sorun oluşmamaktadır. Ancak 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.

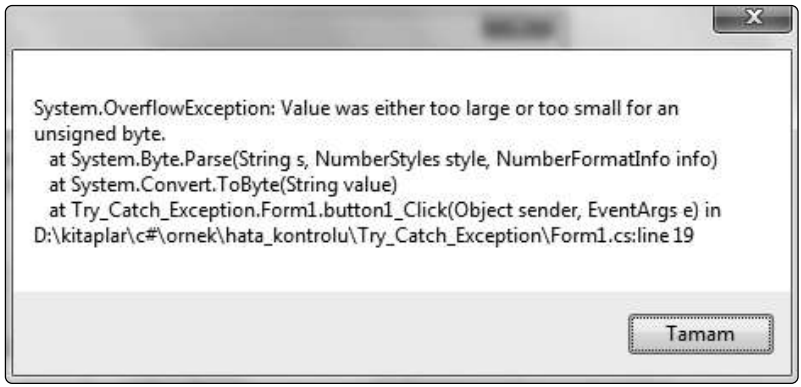
```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        byte notu;
        notu = Convert.ToByte(Interaction.InputBox("Notu Girin",
                                "Not Girişi"));
        if (notu < 45)
        {
            MessageBox.Show("Kaldınız");
        }
        else 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.

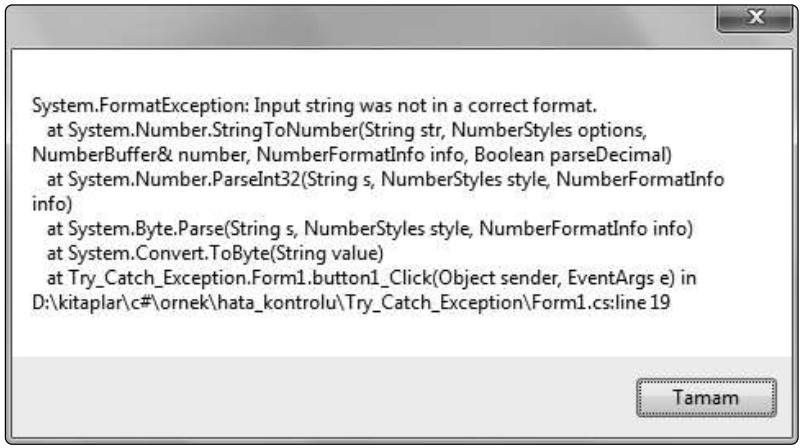
```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        byte notu;
        notu = Convert.ToByte(Interaction.InputBox("Notu Girin",
                                "Not Girişi"));
        if (notu < 45)
        {
            MessageBox.Show("Kaldınız");
        }
        else 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 **overflow** (*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.

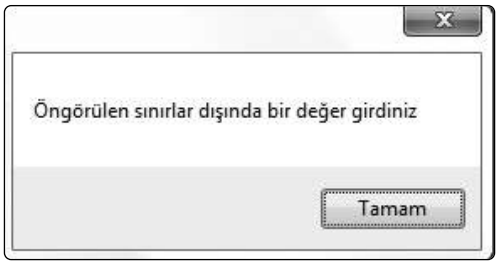


`Exception` nesnesinin `Source` metodunu kullanarak hatanın oluştuğu sınıf hakkında bilgi alınabilir. Yukarıdaki örneğimizdeki `MessageBox.Show(ex.ToString());` ifadesi yerine `MessageBox.Show(ex.Source);` yazıp çalıştırdığımızda herhangi bir hata durumunda mscorlib mesajını görüntüleyecektir. Çünkü 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 ofw)
    {
        MessageBox.Show("Öngörülen sınırlar dışında bir
        değer girdiniz");
    }
    catch (FormatException fe)
    {
        MessageBox.Show("Rakam girmelisiniz");
    }
}
```



Örneğimizi çalıştırıp byte değişkeninin sınırları dışında bir rakam girdiğimizde bizim belirlediğimiz hata mesajı görüntülenecektir. Görüldüğü gibi oluşan hatalara sabit ifadeler şeklinde .Net içerisinden erişmek mümkündür. Bu sabitleri kullanarak oluşan hata mesajının türü belirlenmektedir.

FINALLY DEYİMİ

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( )"` 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 DEYİMİ

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 `overflow exception` hatası 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 sınıfından 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 ofw)
    {
        MessageBox.Show("Öngörülen sınırlar dışında bir ↓
        değer girdiniz");
    }
    catch (FormatException fe)
    {
        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.

Kullanici Tanimli Exception Nesnesi Oluşturmak

`Application.Exception Class`'ını 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 Try_Catch_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 metodları 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.

13

FORMLAR VE KONTROLLER

Şimdiye kadar yaptığımız uygulamalarda kısmen de olsa formların ve kontrollerin bazı özelliklerini ve olaylarını kullandık. Bu bölümde ayrıntılı olarak inceleyeceğiz.

Bütün kontrollerde aynı olan özellikler:

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.

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. Form uygulamalarımızı **SDI** (**S***ingle* **D***ocument* **I***nterface*) ve **MDI** (**M***ultiple* **D***ocument* **I***nterface*) olmak üzere iki ana biçimde gerçekleştirebiliriz.

SDI formlar, şu ana kadar kullandığımız, içerisinde bir adet form penceresi olan form yapısıdır.

MDI formlar ise içerisinde birden fazla form penceresi kullanılabilen form yapısıdır.

Biz öncelikle SDI formlar'a değineceğiz.

Standart bir formun 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 iki adet form olsun. Her ikisine de birer adet buton ekleyerek formlar 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.



`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 formu 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;
}
```

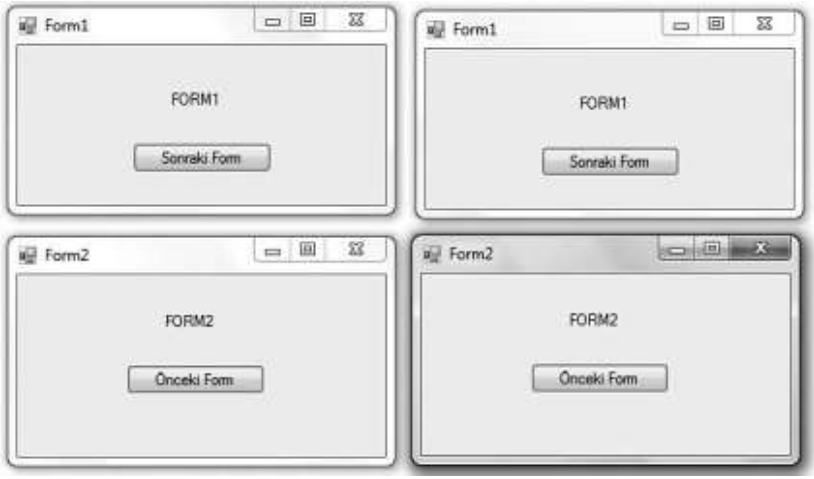
Ancak formları görüntülerken dikkat edilirse aynı form birden fazla açılabilmektedir. Kodumuzu aşağıdaki gibi düzenleyip çalıştıralım;

`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.

- `Close` metodunu kullanarak formun tamamen kapatılmasını ve hafızadan silenebilmesini sağlayabiliriz. Ancak bu işlemi ana formda yapamayız. Ana formdaki form kapatma olayını C#.NET programı sonlandırma olayı gibi algılamaktadır. Dolayısı ile ana formda 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();
}
```

- Formu görüntülemede kullanılan bir diğer metod ise `ShowDialog` metodudur. `Show` metodunda form açık iken diğer formlar aktifleştirilebilir ve geçiş yapılabilir. `ShowDialog` metodu ile görüntülenen formlar kapatılmadığı sürece diğer formlara geçiş yapılamaz ve aktifleştirilemez.

  Yani kodumuzu;

```
private void btnSonrakiForm_Click(object sender, EventArgs e)
{
    this.Hide();
    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 formun nasıl kapatıldığı hakkında bilgi sahibi olabiliriz.

  `ShowDialog` ile ilgili bir şifre giriş formu örneği yapalım. Yeni bir winform proje açıp yeni bir form ekleyerek üzerine tamam ve iptal butonlarını ekliyoruz. Eklediğimiz formun ismini `Dialog` olarak verelim.

  `Dialog` formumuzun 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.OK;
    this.Close();
}
```

  Ayrıca `Form1` içerisinden dialog formu içerisindeki label nesnesine erişim için `frmDialog.Designer.cs` isimli formun 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 şekilde:

```
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` formunda basılan butonların bilgisi formun `DialogResult` bilgisine aktarılmakta ve `Form1` içerisinde `Dialog` formunun `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 ile görüntülenen form dispose işlemi gerçekleştirilene kadar gizlenmiş olarak çalışmaya devam eder. Bu nedenle `Dispose` metodunu kullanmak gerekmektedir.

Örneğimizde `Dialog` formu için yapılacak `Dispose` işlemini `Form1` içerisinde yapmak daha uygun olacaktır. Çünkü aksi takdirde hafıza içeriği temizleneceğinden dolayı `Dialog` formunun 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 formlardan 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.

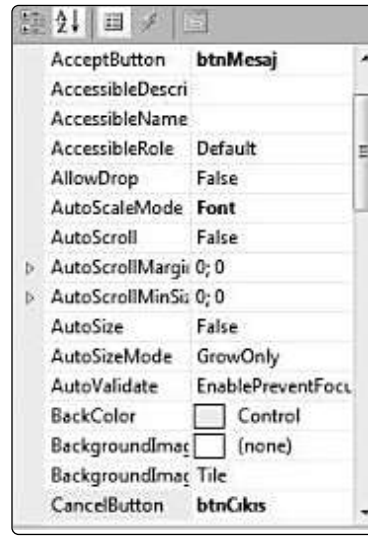
FORM NESNESİNİN ÖZELLİKLERİ

Icon: Form'un simgesini belirler.

AcceptButton: Form üzerinde **Enter** tuşuna basıldığında çalıştırılacak olan butonu belirler.

CancelButton: Form üzerinde **ESC** tuşuna basıldığında çalıştırılacak olan butonu belirler.

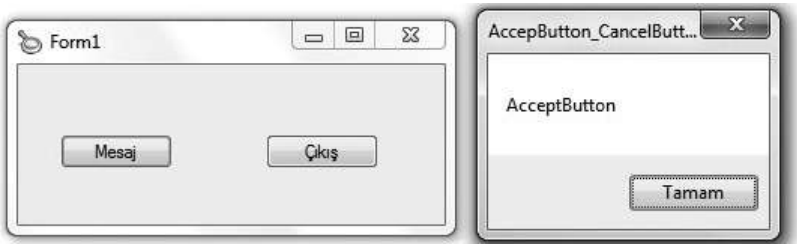
Örnek olarak bir form üzerine iki adet buton ekleyelim, butonlardan birisi ekranda mesaj penceresini görüntületsin (btnMesaj), diğeri ise programdan çıkışı sağlasın (btnCıkıs).



Form üzerinde **Enter** tuşuna basıldığında btnMesaj butonunun çalışması için AcceptButton özelliğine btnMesaj, **ESC** tuşuna basıldığında btnCıkıs butonunun çalışması için 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 üç değer alır. **Normal**, **Minimized**, **Maximized**.

Normal seçeneği, formu hangi boyutlarda ayarladı isek formun o boyutlarda ekrana gelmesini sağlar.

Minimized seçeneği, formun simge durumuna küçültülmüş olarak ekrana gelmesini sağlar.

Maximized seçeneği, formun tam ekran boyutunda ekranı kaplamış olarak ekrana gelmesini sağlar.

FormBorderStyle özelliği ile formun 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: Formun boyutunun değiştirilmesini engeller. Bu seçenek aktif iken `MaximizeBox` özelliğini de `false` yaparak tam ekran butonunu pasif yapıp, formun ekranı kaplamasını da engelleyerek formun boyutlarının değiştirilmesini tamamen engelleyebiliriz. İsteğe bağlı olarak `MinimizeBox=false` ile simge duruma küçültme düğmesini, `ControlBox=false` ile de kapatma düğmesini pasif yapabiliriz.

Fixed3D: Forma 3 boyutlu görünüm vererek aynı zamanda boyutlarının değiştirilmesini engeller.

FixedDialog: Formun boyutları değiştirilemez. Aynı zamanda formun simgesi de görünmez. Form simgesi, `Icon` özelliği ile değiştirilebilmektedir.

Sizable: Varsayılan görüntüleme ayarıdır. Formun 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 formun boyutları değiştirilebilir.

`Size` başlığı altındaki `Width` ve `Height` özellikleri ile formun boyutları belirlenir. `Width` özelliği ile genişlik belirlenirken, `Height` özelliği ile yükseklik belirlenir.

Örnek olarak formun boyutlarını yatay ve dikey iki adet kaydırma çubuğu ile değiştirelim. Form'un `FormBorderStyle` özelliğini `FixedSingle` yaparak formun boyutunun fare ile değiştirilmesini engelleyelim. `MaximizeBox` özelliğini `false` yaparak da tam ekran boyutuna getirilmesini engelleyelim. Form üzerine birer adet `HScrollBar` ve `VScrollBar` (*yatay ve dikey kaydırma çubukları*) ekleyelim.

```
private void HScrollBar1_Scroll(object sender, ScrollEventArgs e)
{
     this.Width = this.Width + HScrollBar1.Value;
     HScrollBar1.Width = HScrollBar1.Width + HScrollBar1.Value;
}
private void VScrollBar1_Scroll(object sender, ScrollEventArgs e)
{
     this.Height = this.Height + VScrollBar1.Value;
     VScrollBar1.Height = VScrollBar1.Height + VScrollBar1.Value;
}
```

Kodumuz içerisinde geçen `this`, formumuzu temsil etmektedir. Projemizde sadece bir form kullandığımız için this kullanımını tercih ettim. this, yerine `ActiveForm` şeklinde kullanmak da mümkündür. `ActiveForm`, o an aktif olan formu temsil eder. Formun genişliği ve yüksekliği, kaydırma çubukları arttırıldıkça kaydırma çubuğunun değişim miktarı kadar formun boyutu değişmektedir. Ayrıca kaydırma çubuklarının boyutu da bu değişim miktarına bağlı olarak artmaktadır.

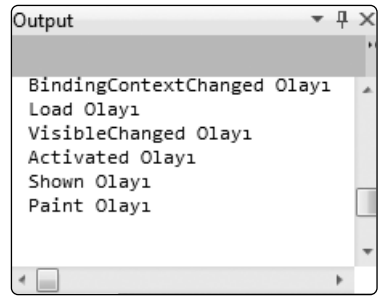
Form Olayları

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.

Formun 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.

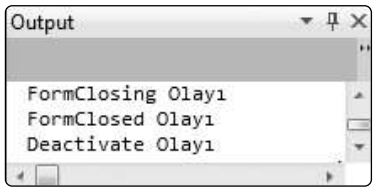
Shown: Formun görüntülenmesi bittiği zaman gerçekleşen olaydır.

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.

Formun 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.

Formun kapanış olayları ile ilgili birkaç örnek uygulama yapalım. İlk örneğimizde formun kapanış olayında art arda mesaj pencereleri çıkartalım ve kullanıcıyı biraz bıktıralım.

```
private void btnCikis_Click(object sender, EventArgs e)
{
    Application.Exit();
}
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
```

```
byte i;
DialogResult  basilan_tus;

for (i = 1; i <= 5; i++)
{
  if (i == 1)
  {
    basilan_tus = MessageBox.Show("Çıkmak ↓
      istediğinize Emin misiniz?", "Soru", ↓
      MessageBoxButtons.YesNo);
    if (basilan_tus == DialogResult.No)
    {
        MessageBox.Show("Çıkmak istemediğini ↓
         biliyordum zaten");
        e.Cancel = true;
        break;
    }
  }
  else if (i == 2)
  {
    basilan_tus = MessageBox.Show("Gerçekten Çıkmak ↓
      istiyor musunuz?", "Soru", MessageBoxButtons.YesNo);
    if (basilan_tus == DialogResult.No)
    {
        MessageBox.Show("Çıkmak istemediğini biliyordum ↓
          zaten");
        e.Cancel = true;
        break;
    }
  }
  else if (i == 3)
  {
    basilan_tus = MessageBox.Show("Emin misiniz?",
                          "Soru", MessageBoxButtons.YesNo);
    if (basilan_tus == DialogResult.No)
    {
        MessageBox.Show("Çıkmak istemediğini biliyordum ↓
           zaten");
        e.Cancel = true;
        break;
    }
  }
```

```
        else if (i == 4)
        {
          basilan_tus = MessageBox.Show("Hımm. Düşünmem lazım",
                                "Soru", MessageBoxButtons.YesNo);
          if (basilan_tus == DialogResult.No)
          {
              MessageBox.Show("Çıkmak istemediğini biliyordum ↓
                 zaten");
              e.Cancel = true;
              break;
          }
        }
        else if (i == 5)
        {
          basilan_tus = MessageBox.Show("Müsait bir zamanımda ↓
            düşünürüm", "Soru", MessageBoxButtons.YesNo);
          if (basilan_tus == DialogResult.No)
          {
              MessageBox.Show("Çıkmak istemediğini biliyordum ↓
                 zaten");
              e.Cancel = true;
              break;
          }
          i = 0;
        }
      }
}
```

Mesaj pencereleri `Form_Closing` olayına yazıldığı için formun kapatılması esnasında ekrana gelmektedir. 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 formu kapatılmaya başlanıldığında gerçekleşen olaydır.

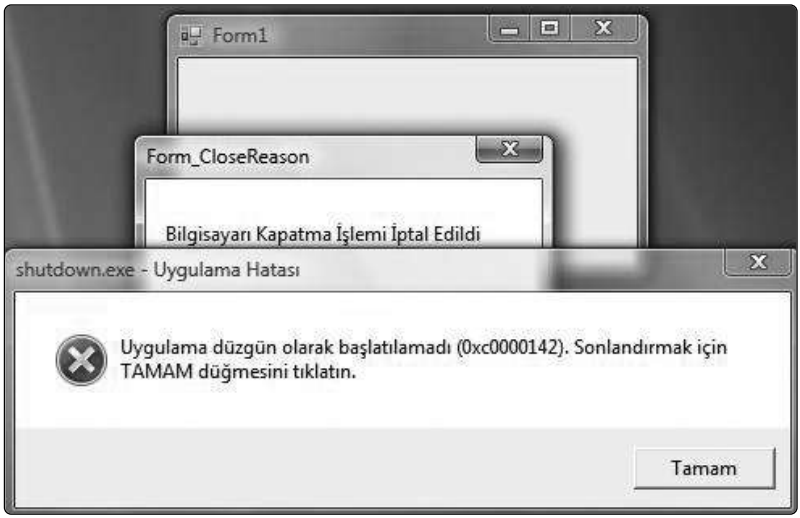
TaskManagerClosing: Form, görev yöneticisi tarafından sonlandırmak istenildiğinde gerçekleşen olaydır.

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 sistemi 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 kapatma 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.

```
byte i;
private void Form1_KeyPress(object sender, KeyPressEventArgs e)
{
    i ++;
    if (e.KeyChar == 27) MessageBox.Show(i - 1 + " tuş tıklandı");
}
```

`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.

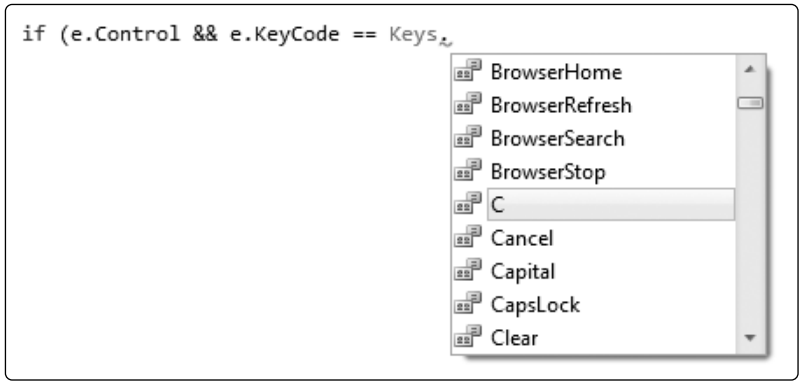
Ancak `Keypress` olayında **`Alt`**, **`AltGr`**, **`Ctrl`** gibi kontrol tuşlarına basıldığını algılayamayız.

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.




```
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 formdan çıkış işlemi yapılmaktadır.

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*) formlar oluşturmak mümkündür. Bir owner form simge durumuna küçültüğünde owned form da simge durumuna küçülür, kapatıldığında o da kapatılır.

Owned formlar, `AddOwnedForm` metoduyla bir forma eklenir ve `RemoveOwnedForm` metoduyla çıkarılır.

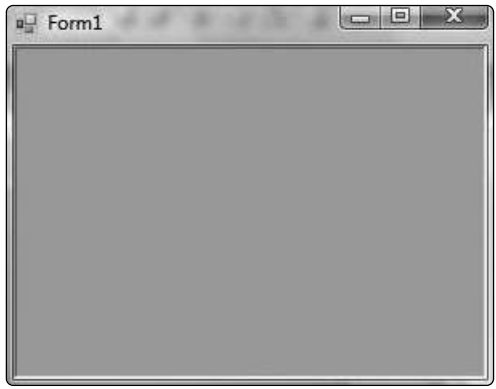
```
Form ownedForm1 = new Form();
private void btnOwnedFormEkle_Click(object sender, EventArgs e)
{
    this.AddOwnedForm(ownedForm1);
    ownedForm1.Show();
}
private void btnOwnedFormSil_Click(object sender, EventArgs e)
{
    this.RemoveOwnedForm(ownedForm1);
}
```

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 `owned form`'un simge durumuna küçülmediğini görürüz. Çünkü `RemoveOwnedForm` metodu ile owned form olma özelliğini sildik.



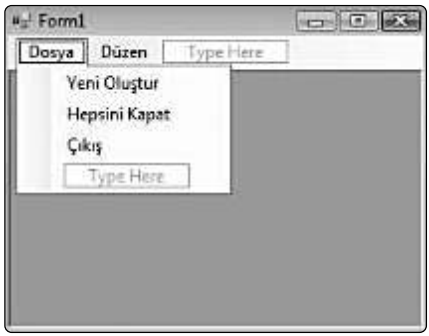
MDI (Multiple Document Interface) Formlar

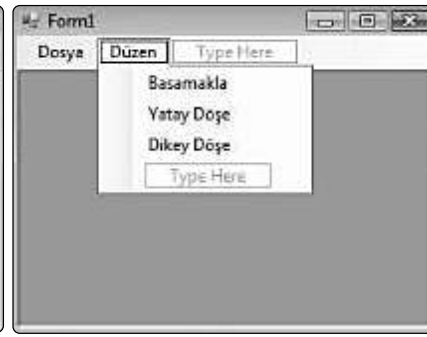
MDI Formlar içerisinde birden fazla form penceresi kullanılabilen form yapısıdır. MDI formuna **parent** (*ebeveyn*) **form**, içerisindeki formlara ise **child** (*çocuk*) **form** adı verilir.



Projemiz içerisindeki formlardan birini MDI Parent Form yapmak için `IsMdiContainer` özelliğini true yapmalıyız. Bunu yaptığımız anda formumuz yandaki gibi bir görünüm alacaktır.

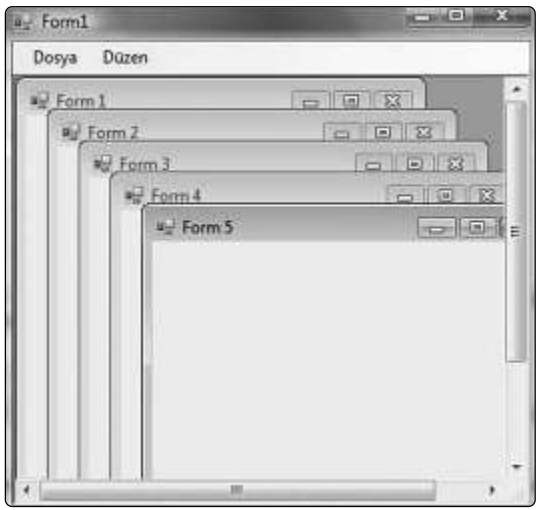
Form üzerine bir `MenuStrip` kontrolü ekleyerek menülerimi oluşturuyorum. Menüler konusuna daha sonra ayrıntılı değineceğimiz için ayrıntılarına girmiyorum. Burada önemli olan sadece mdi form olayları.





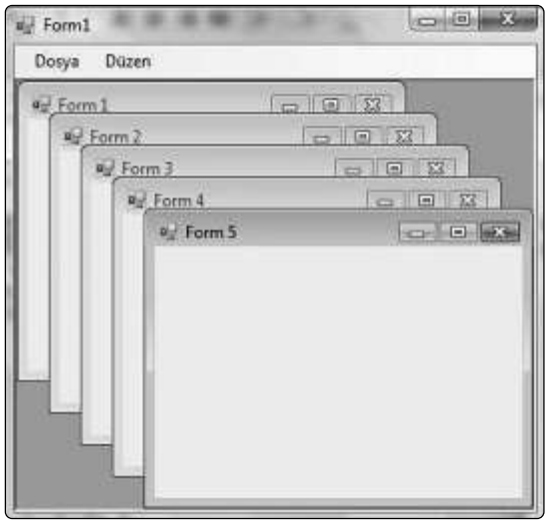

```
byte sayi;
private void mnuYeni_Click(object sender, EventArgs e)
{
     Form yeniform = new Form();
     yeniform.MdiParent = this;
     sayi++;
```

```
        yeniform.Text = "Form " + sayi;
        yeniform.Show();
    }
    private void mnuKapat_Click(object sender, EventArgs e)
    {
        foreach (Form eleman in this.MdiChildren)
        {
            eleman.Close();
        }
    }
    private void mnuCikis_Click(object sender, EventArgs e)
    {
        Application.Exit();
    }
    private void mnuBasamakla_Click(object sender, EventArgs e)
    {
        this.LayoutMdi(MdiLayout.Cascade);
    }
    private void mnuYatayDose_Click(object sender, EventArgs e)
    {
        this.LayoutMdi(MdiLayout.TileHorizontal);
    }
    private void mnuDikeyDose_Click(object sender, EventArgs e)
    {
        this.LayoutMdi(MdiLayout.TileVertical);
    }
```

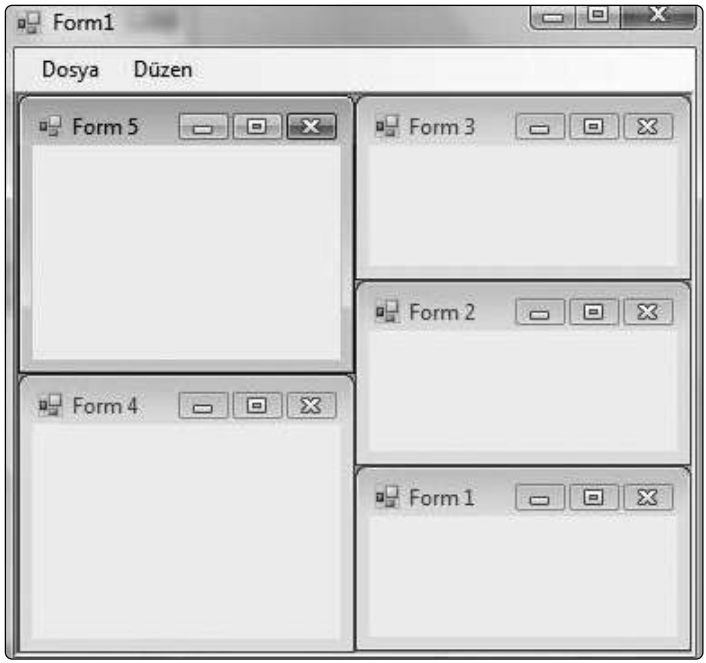


Kodumuzu çalıştırarak 5 adet child form oluşturalım.

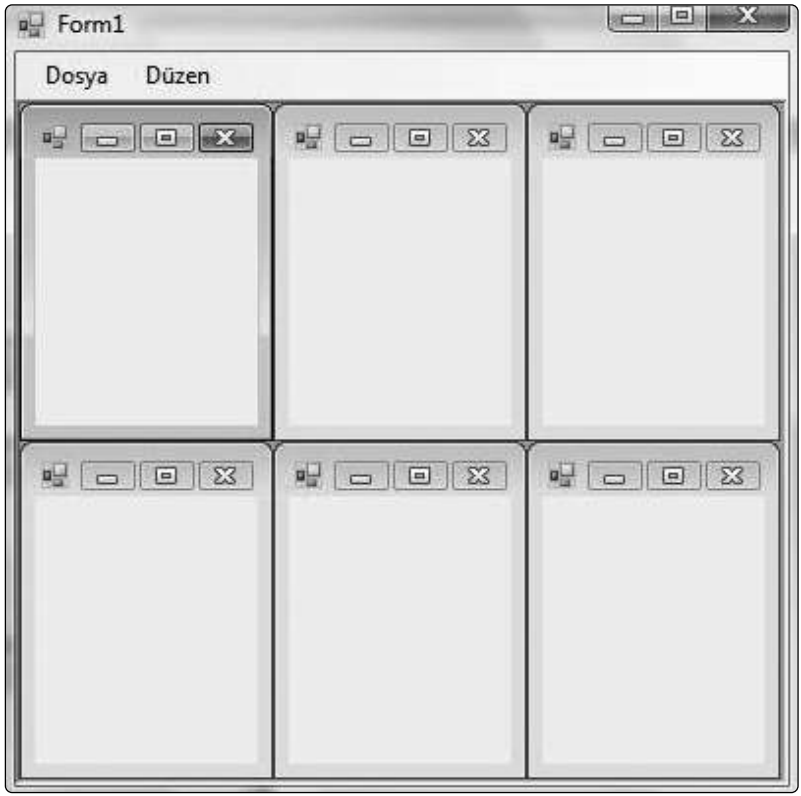
Basamaklama işlemi yapalım.



Yatay döşeme işlemi yapalım.



Bir tane daha child formu ekleyerek dikey döşeme yapalım.



Kodumuzda yeni bir child formu oluştururken

```
yeniform.MdiParent = this;
```

şeklinde oluşturulan formun `MdiParent` özelliğine `MdiParentForm` olarak belirlediğimiz formu atıyoruz.

Oluşturduğumuz child formları kapatmak için `ChildFormİsmi.Close` metodunu kullanıyoruz.

Açık olan child formları basamaklamak için MDI formun yerleşim biçimini `LayoutMdi(MdiLayout.Cascade)` metodu ile değiştiriyoruz.

Açık olan child formları yatay döşemek için MDI formun yerleşim biçimini `LayoutMdi(MdiLayout.TileHorizontal)` metodu ile değiştiriyoruz.

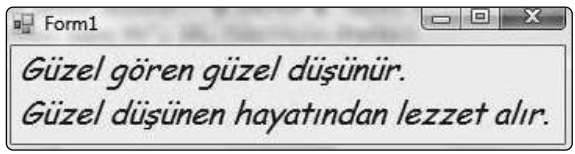
Açık olan child formları dikey döşemek için ise MDI formun yerleşim biçimini `LayoutMdi(MdiLayout.TileVertical)` metodu ile değiştiriyoruz.

Diğer Kontroller

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;
        }
    }
}
```



Yazıtipleri üzerinde işlem yapabilmek için `System.Drawing` sınıfını projemize dahil ettik. Ayrıca kodumuz içerisindeki `AutoSize` metodu ile label nesnesinin boyutunun içerisindeki yazıya göre boyutlandırılmasını 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 formu 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");
```

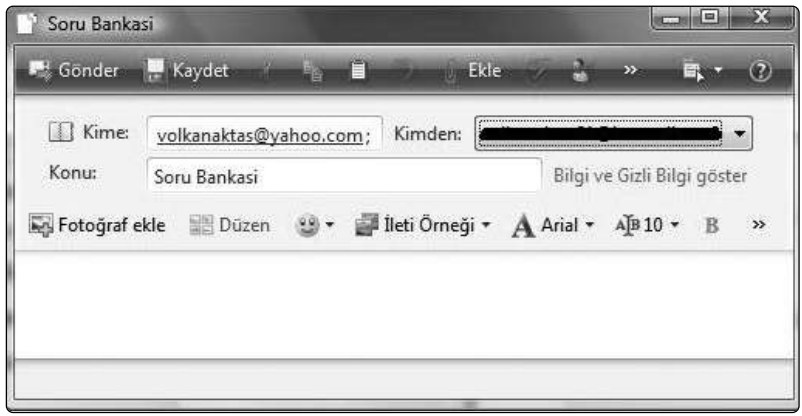
Komut satırı ile varsayılan mail adresi açılmaktadır.

Mailto: Mailin gönderileceği adresi temsil eder.

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 ile bir satırdan daha uzun olan metinlerin görüntülenmesinde **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.

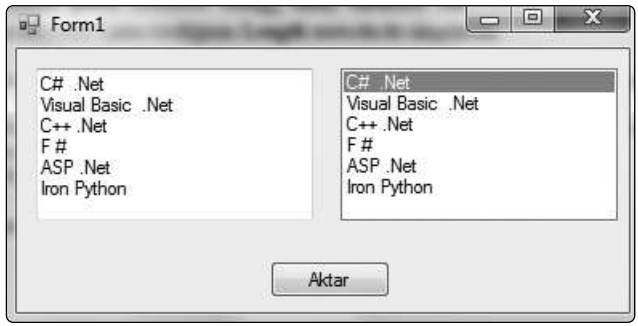
Font özelliği yazı tipini değiştirmek amacı ile kullanılır.

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]);
      }
}
```



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.

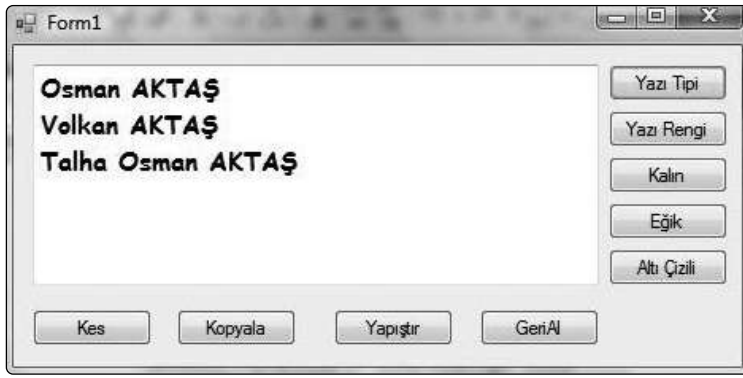
Örnek olarak bir textbox içerisinde temel işlemleri gerçekleştirelim. Projemize yazı rengini değiştirmek için bir adet color dialog nesnesi, yazı tipini değiştirmek için de bir adet fontdialog nesnesi ekliyoruz.

```
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.

TextBox Kontrolünün Olayları

TextChanged: Textbox içerisine yazı girişi yapılırken gerçekleşen olaydır.

```
using System;
using Microsoft.VisualBasic;
using System.Windows.Forms;
namespace textbox_textchanged
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void textBox1_TextChanged(object sender,
        EventArgs e)
        {
            if (textBox1.TextLength > 1)
            {
                if ((textBox1.Text.IndexOf(".") > 0))
                {
                    if (Strings.Mid(textBox1.Text, Strings.Len(
                            textBox1.Text) - 1, 1) == ".")
                    {
                        textBox1.Text = textBox1.Text.Replace(
                            textBox1.Text. ↓
                            Substring(Strings.Len(
                            textBox1.Text) - 1, 1),
                            textBox1.Text. ↓
                            Substring(Strings.Len(
                            textBox1.Text) - 1, 1).ToUpper());
                    }
                    textBox1.SelectionStart =
                      textBox1.TextLength;
                }
            }
        }
    }
}
```

Örneğimizde textbox içerisine metin girişi yapılırken

```
Strings.Mid(textBox1.Text, Strings.Len(
                          textBox1.Text) - 1, 1) == ".")
```

satırı ile girilen sondan bir önceki karakterin "**.**" olup olmadığı kontrol edilmekte, eğer nokta ise noktadan sonraki karakter büyük harfe çevrilerek küçük harfli olan karakterler değiştirilmektedir.

Button Kontrolü



Bir işlemi, olayı gerçekleştirmek amacı ile kullanılan nesnesidir.

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.

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";
}
```

Örneğimizde 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.

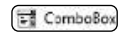
```
System.Drawing.Color renk;
private void button1_MouseLeave(object sender, EventArgs e)
{
    button1.BackColor = renk;
}
private void button1_MouseMove(object sender, MouseEventArgs e)
{
    button1.BackColor = System.Drawing.Color.DarkGray;
}
private void Form1_Load(object sender, EventArgs e)
{
    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 arka plan 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 içerisinde değer listesini eklemek amacı ile kullanılır.

Text: Combox'a tıklanmadan önce 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 değer sayısını verir.

Insert: Combobox içerisine belirtilen değeri listenin istenilen bir yerine eklemek amacı ile kullanılır.

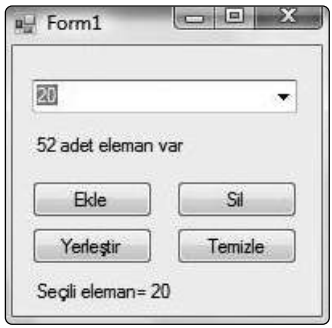
Clear: Combobox içerisindeki değerlerin hepsini siler.

Contains: Eğer belirtilen değer mevcutsa `true` sonucunu, 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 olan sayıların kontrolünü yaptırarak

```
comboBox1.Items.Add(i);
```

komut satırı ile combobox içerisine sıra ile elemanlarımızı ekliyoruz.

```
comboBox1.Items.Insert(0, -1);
```

`Insert` metodu ile `0` nolu index sırasına `-1` adlı elemanımızı ekliyoruz.

`label2` adlı label içerisinde seçili olan elemanı 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, metod ve olayları aynı olduğu için burada farklı olanlara değinelim.

Listbox içerisinde sıralama işlemi yapan bir örnek kod yazalım:

```
using System;
using System.Windows.Forms;
using Microsoft.VisualBasic;
namespace listbox_siralama
{
    public partial class Form1 : Form
    {
        public Form1()
        {
```

```
            InitializeComponent();
        }
        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;

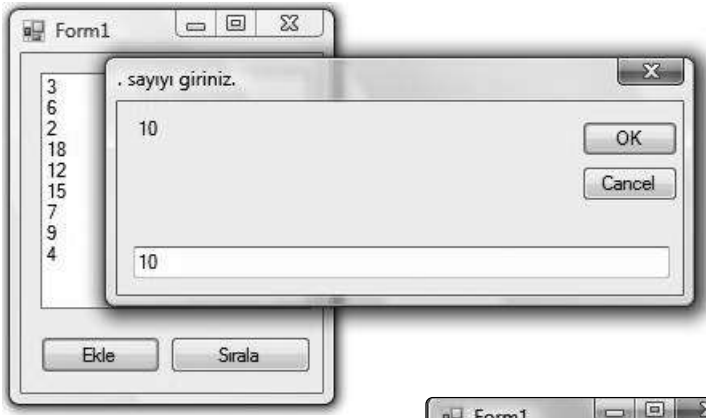
            for (l = 0; l <= 9; l++)
            {
                for (m = l + 1; m <= 9; m++)
                {
                    if (Int32.Parse(listBox1.Items[l]. ↓
                      ToString()) >

        Int32.Parse(listBox1.Items[m].ToString()))
                    {
                        int aradeger =
        Int32.Parse(listBox1.Items[l].ToString());
                        listBox1.Items[l] =
        listBox1.Items[m];
                        listBox1.Items[m] = aradeger;
                    }
                }
            }

        }
    }
}
```

Komut satırlarında öncelikle iç içe 2 döngü kurulmakta. Bu döngülerden ikincisi birincisinden bir fazla değer almakta. Bunun nedeni önceki değerle sonraki değeri karşılaştırmak için. Burada önceki değer sonraki de-

ğerden 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 yukarıdaki resimde 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");
    }
}
```

Komutu çalıştırıp listbox içerisinde herhangi bir değer aradığımızda değer bulunduğunda;

```
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.



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 mouse-down 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.

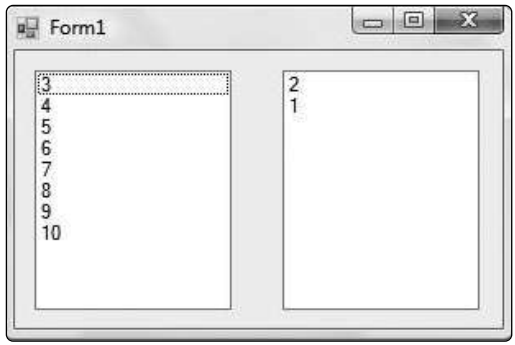
Son aşamada ise taşınan değeri `listBox2` içerisine bırakacağız. Bunun için `listBox2`'nin **DragDrop** olayını kullanıyoruz.

```
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 **ana düğüm** (*parent node*) ve **çocuk düğüm** (*child node*) 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

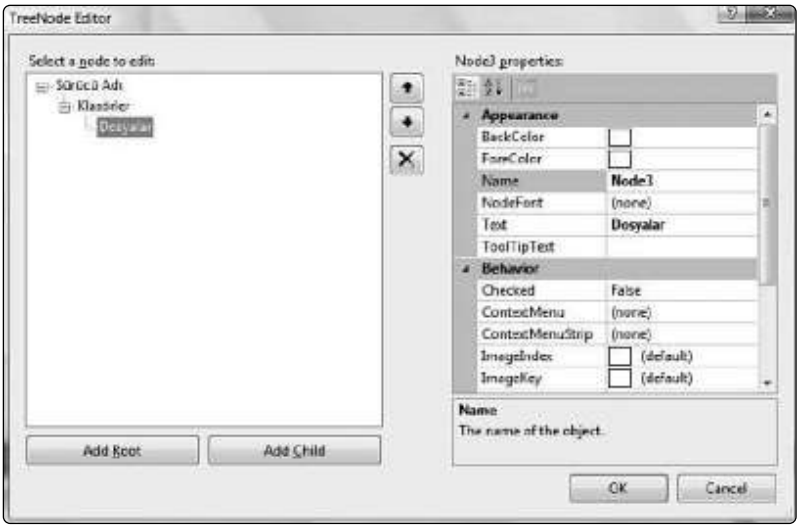
CheckBoxes: `True` değerini aldığında her bir düğümün yanında checkbox görüntülenmesini sağlar.

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 **child node** elemanlarının eklenip düzenlenebildiği **TreeNode Editör** penceresini açar.



SelectedNode: Seçili olan bir düğümü görüntülemek ya da bir düğümü 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.

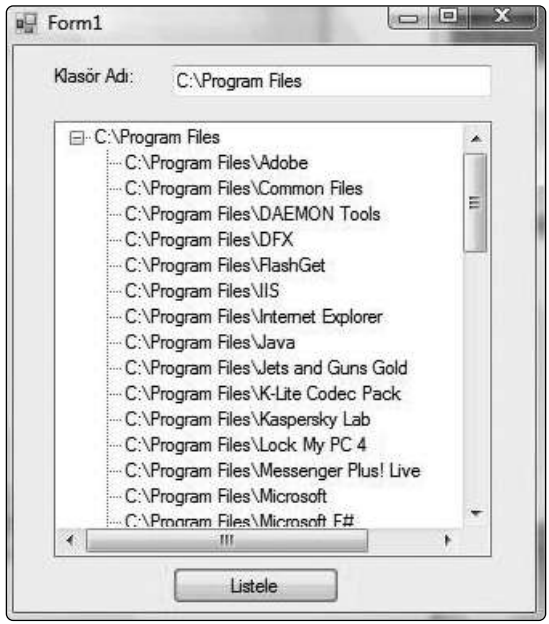
NodeMouseDoubleClick: Düğümlerden biri çift tıklandığında gerçekleşen olaydır.

NodeMouseHover: Düğümlerden birinin üzerinden fare ile geçildiğinde gerçekleşen olaydır.

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.

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.

```
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 formun 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 dugum adında yeni bir TreeNode nesnesi oluşturarak aktifklasor değişkeni içerisinde saklanmakta olan klasor 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.

GroupBox Nesnesi

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 aşağıdaki gibi bir görünüm alacaktır.



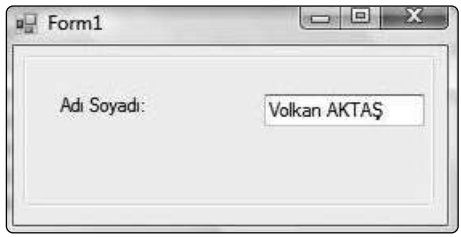
Kodla GroupBox nesnesi oluşturarak içerisinde bir adet Label bir adet de TexBox nesnesi ekleyelim.

```
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.

PANEL KONTROLÜ 

GroupBox nesnesinde olduğu gibi nesneleri gruplamak için kullanılır. Örnek olarak otomatik olarak gizlenebilen bir panel tasarlayalım. 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.

Formun 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.

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.

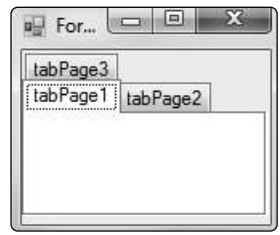
ItemSize: Sekme boyutlarını belirler.

Multiline: TabControl nesnesindeki sekmelerin birden fazla satırda görüntülenmesini sağlar.

Aşağıda şekilde MultiLine özelliği **`false`** yapılmıştır.



Aşağıda şekilde MultiLine özelliği **`true`** yapılmıştır.



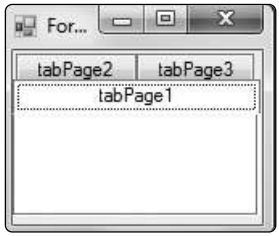
TabCount: Sekme sayısını belirler.

SizeMode: Sekmelerin boyutlandırma biçimini belirler.

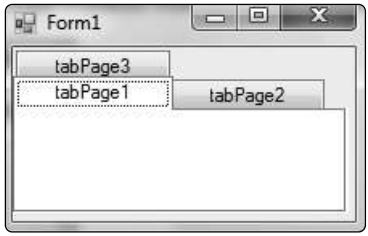
Şu değerleri alır:

• **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.

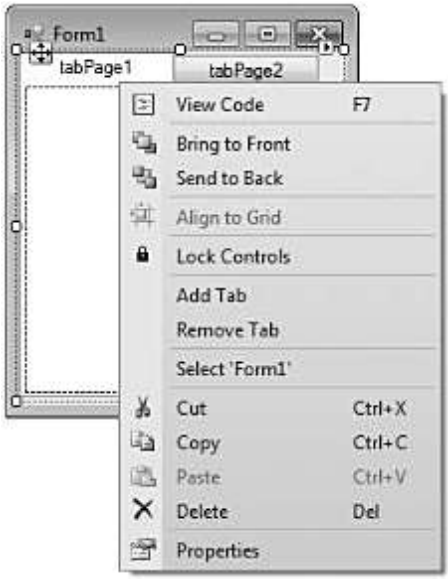
```
tabControl1.SelectedIndex = 1;
```

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

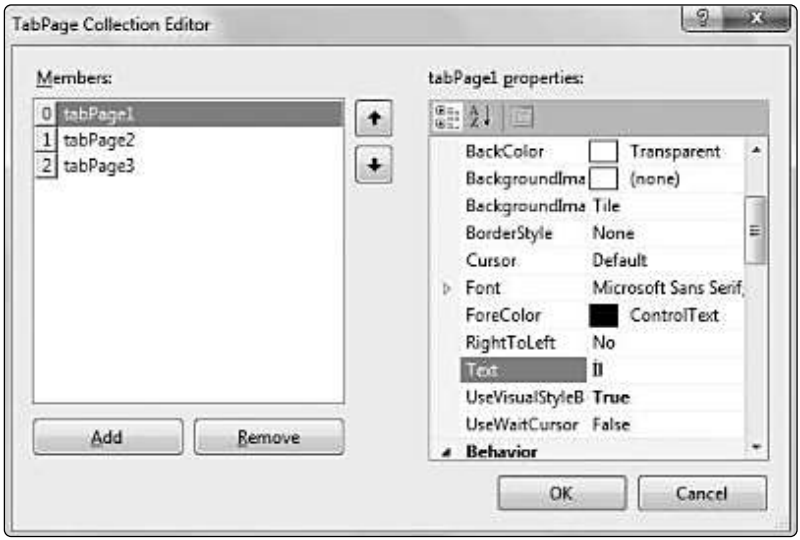


TabControl nesnesine farenin sağ tuşu ile tıklanarak gelen menüden **Add Tab** ve **Remove Tab** seçenekleri ile sekme eklenip silme işlemi gerçekleştirilebilir.



Ya da TabControl nesnesinin sağ üst köşesinde bulunan kulakçığa tıklanarak gelen menüden **Add Tab** ve **Remove Tab** seçeneklerinden sekmeler eklenebilir ya da silinebilir.

Bir diğer yöntem ise TabControl nesnesi seçili iken **Properties** penceresinden **TabPages** özelliğinden **Collection** seçeneğine tıklayarak **PageCollectionEditor** penceresini kullanmaktır.



Kod yolu ile yeni bir sekme eklemek için `TabPages.Add` metodu kullanılabilir.

```
tabControl1.TabPages.Add("Yeni Sekme");
```

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);
```

CheckBox Kontrolü

Genellikle 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: Üç adet değer alır.

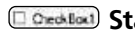
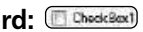
- **Unchecked:** İşaretsiz.

• **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 checkboxlar işaretli iken yazı stilinin uygulanmasını, işaretli değilken yazı stilinin uygulanmamasını sağladık. ^ (XOR - Özel veya) mantıksal operatörü ile 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.

CopyTo: Belirtilen elemanı bir dizi içerisine kopyalayarak aktarmak amacı ile kullanılır.

`CheckedItems` özelliği ise `Count` ve `CopyTo` metodlarına sahiptir.

SetItemChecked Metodu: CheckedListBox içerisindeki bir elemanı işaretli duruma getirmek ya da işaretini kaldırmak için kullanılır.

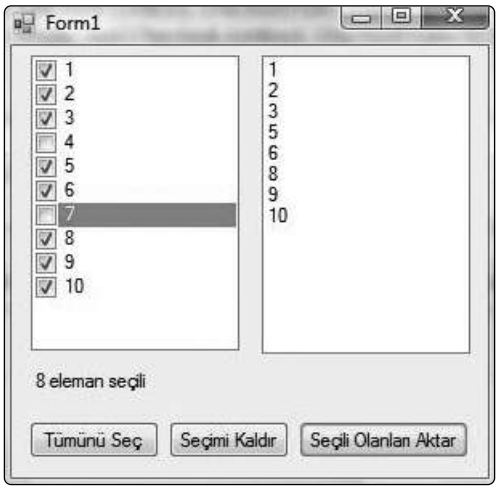
Kullanımı şu şekildedir:

`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";
}
```



Örneğimizde CheckedListBox içerisinde seçili olan elemanların sayısı label nesnesi içerisinde görüntülenmekte ve seçili olan elemanlar ListBox nesnesi içerisinde aktarılmaktadır.

RadioButton Nesnesi

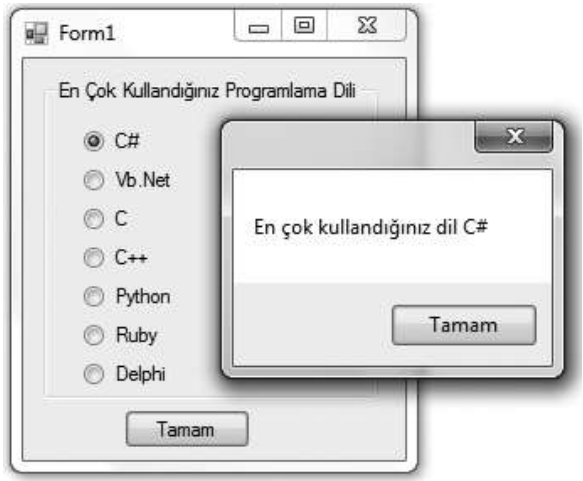


Birden fazla seçenekten yalnızca birini seçme durumlarında kullanılan nesnedir.

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 işaretli olan radioButton nesnesi kontrol edilerek en çok kullanılan dile değeri 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. Ancak önce eriştiğimiz elemanları radiobutton nesnesine dönüştürmemiz gerekiyor. Bu işlemin ardından radiobutton nesnelerinin `Checked` özelliğini kontrol ederek işaretli ise bir mesaj penceresi içerisinde işaretli olan radiobutton nesnesinin metninin görüntülenmesini sağlıyoruz.

```
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);
    }
}
```

Timer Kontrolü



Zamanlama işlemleri için kullanılır.

.Net içerisinde üç türlü 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` saniyeye eşit gelir. 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 METODLARI

Enabled: `True` değerini aldığında Timer nesnesini aktif yapar. `False` değerini aldığında ise pasif yapar

Start: Timer nesnesini başlatır.

Stop: Timer nesnesini durdurur.



Timer nesnesi **enabled=false** seçeneği ile pasif yapıldığında, **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 OLAYLARI

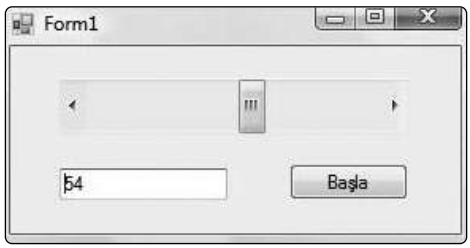
Tick: Interval özelliğinde belirlenen zaman aralığına göre zamanlananan olaylar Tick olayı içerisinde gerçekleşir.

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 `true` olarak ayarlayalım.

```
Boolean son;
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 = "Bitir";
        timer1.Enabled = false;
    }
    else
    {
        button1.Text = "Başla";
        timer1.Enabled = true;
    }
    }
}
```



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 yaparak

```
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

```
hScrollBar1.Value = hScrollBar1.Value + 1;
```

Kaydırma çubuğunun değerini `1` arttırıyoruz.

`Son` değişkenin değeri `true` olduğunda ise kaydırma çubuğunu geriye doğru döndürmek için ise

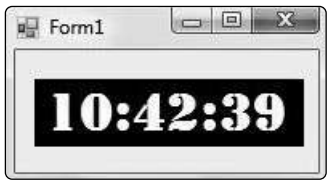
```
hScrollBar1.Value = hScrollBar1.Value - 1;
```

satırını kullanıyoruz.

Basit bir saat uygulaması yapalım.

Form üzerine bir adet label nesnesi ile bir 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.

Bir kronometre örneği için form üzerine 2 adet button, 2 adet label, bir adet textbox, bir adet de timer nesnesi ekleyelim. Timer nesnesinin interval değeri `1000` 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 `timer1`'in `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 saymayı bitirdi. Şimdi sıra Timer2 de."* şeklinde mesaj görüntülenecek. 3 saniye sonra da *"Timer2 de saymayı bitirdi."* şeklinde bir mesaj daha görüntülenecektir.

ProgressBar Kontrolü



Gerçekleştirilen bir işlem sırasında işlemin ilerleme durumunu göstermek amacıyla kullanılır.

ProgressBar Kontrolünün Özellikleri

Maximum: ProgressBar nesnesinin alabileceği en fazla değeri belirtir.

Minimum: ProgressBar nesnesinin alabileceği en az 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 bir 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);
        }
    }
}
```

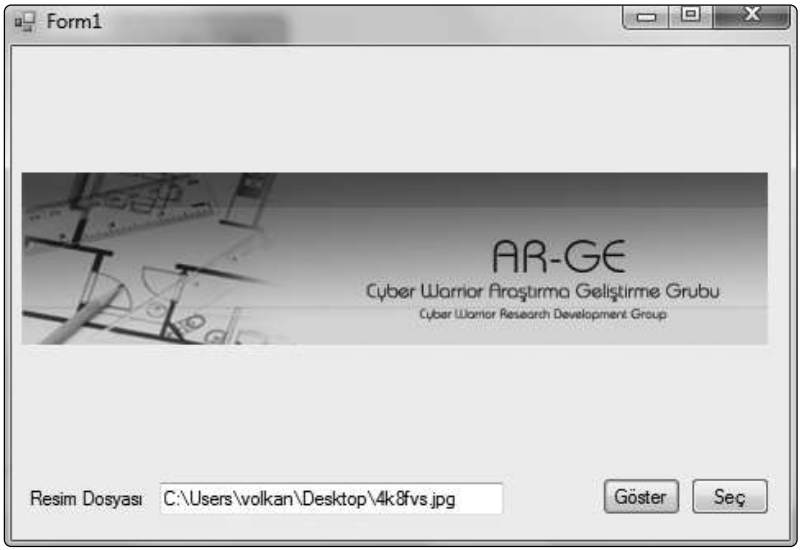
Örneğimizde;

```
pictureBox1.Image = new Bitmap(openFileDialog1.OpenFile());
```

satırı ile hem `Image` özelliğini kullanarak `OpenFileDialog` nesnesi yardımıyla seçilen resim dosyasını picturebox nesnesi içerisine ekledik. Hem de `Load` özelliğini kullanarak

```
pictureBox1.Load(textBox1.Text);
```

satırı ile TextBox içerisine yazılan resim dosyasını picturebox nesnesi içerisine ekledik.



Örneğimizde PictureBox'ın `SizeMode` özelliği `Zoom` olarak seçilmiştir.

NOT

Uygulamalarınızda kullanabilmeniz için Visual Studio ile birlikte gelen **Visual Studio 2010 Image Library** isimli resim kütüphanesini kullanabilirsiniz. Bu kütüphane içerisinde binden fazla imaj bulunmaktadır. Ancak bu kütüphane sıkıştırılmış biçimde *C:\Program Files\Microsoft Visual Studio 10.0\Common7\VS2010ImageLibrary\1033* klasörü içerisinde **VS2010ImageLibrary.zip** ismi ile yer almaktadır. Bu zip dosyasını arşivden çıkartarak içerisindeki imaj dosyalarını kullanabilirsiniz.

MenuStrip Kontrolü

MenuStrip

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.

DefaultItem: Varsayılan değer olarak `false` seçilidir. `True` olarak değiştirildiğinde ilgili menü öğesinin varsayılan menü öğesi olması sağlanır.

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.



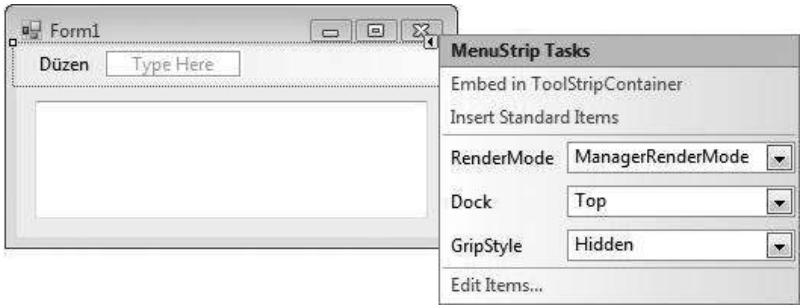
ShortCutKeyDisplayString: Kısayol tuşu ile açıklayıcı bilgi görüntülenmek istenirse bu özellik yardımıyla belirtilir.

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 **Check** işareti görüntülenemez.

Items: Ana menü altındaki alt menü öğeleri listesine erişimi sağlar.

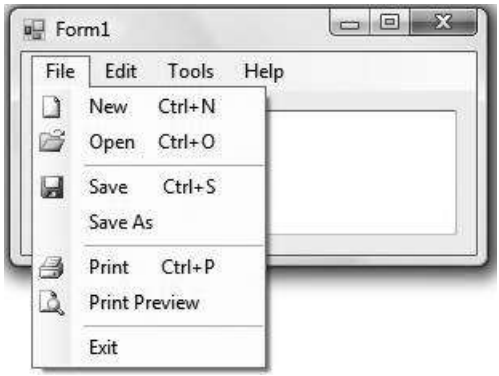
MenuStripTasks: MenuStrip nesnesi seçili iken sağ üst köşede bulunan kulakçığa tıklayarak **MenuStripTasks** menüsünü görüntüleriz.



Bu menüde bulunan seçenekleri inceleyelim:

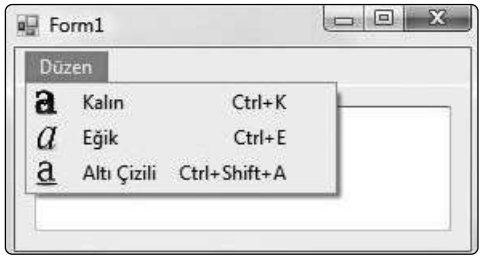
Embed in ToolStripContainer: Menüyü `ToolStripContainer` içerisine yerleştirir. (`ToolStripContainer` nesnesi daha sonra ayrıntılı olarak anlatılacaktır)

Insert Standart Items: Alt menüleri ile birlikte hazır standart bir menü ekler.

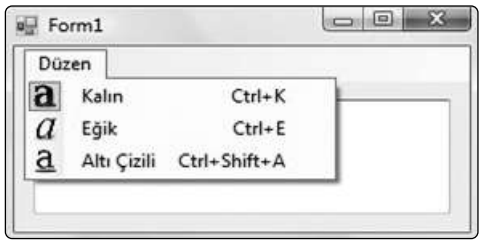


RenderMode: Menü çubuğunun görünümü ile ilgili ayarlar yapılır. Şu değerleri alır.

- **System:** Menü çubuğunun varsayılan Windows görünüm ayarları ile görüntülenmesini sağlar.



- **Professional:** Menü çubuğunun Office uygulamaları görünümü şeklinde görüntülenmesini sağlar.

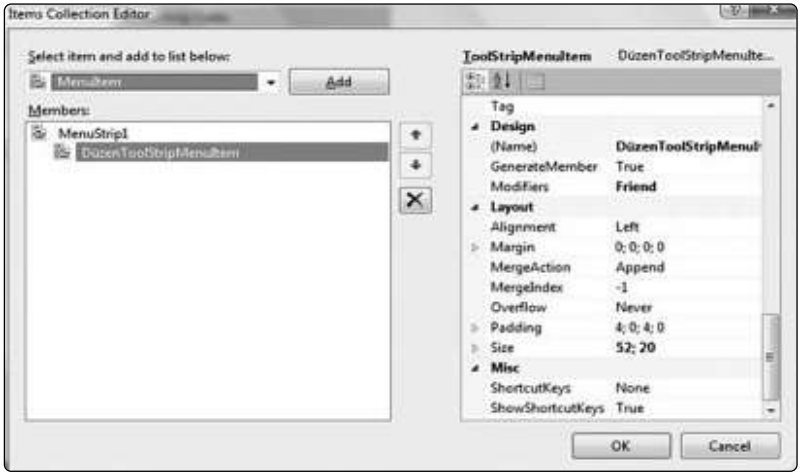


- **ManagerRenderMode:** Özel menü stilleri tanımlamamıza olanak sağlar.

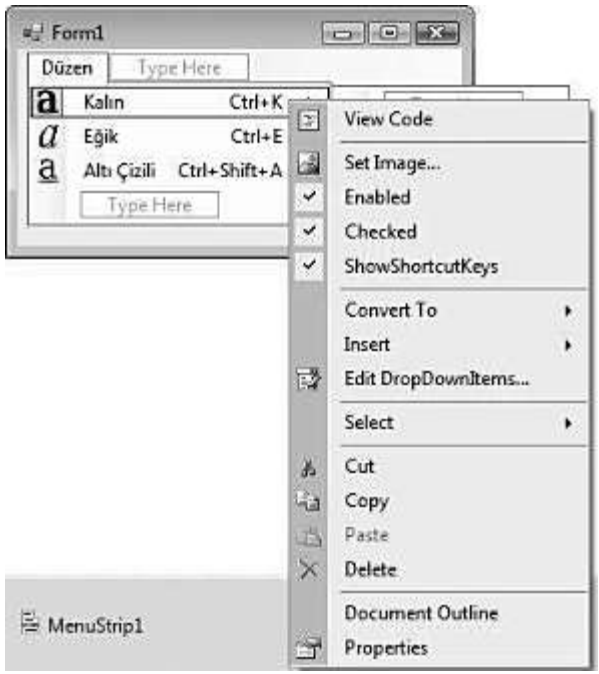
Dock: Menünün form üzerindeki yerleşim yerini belirler.

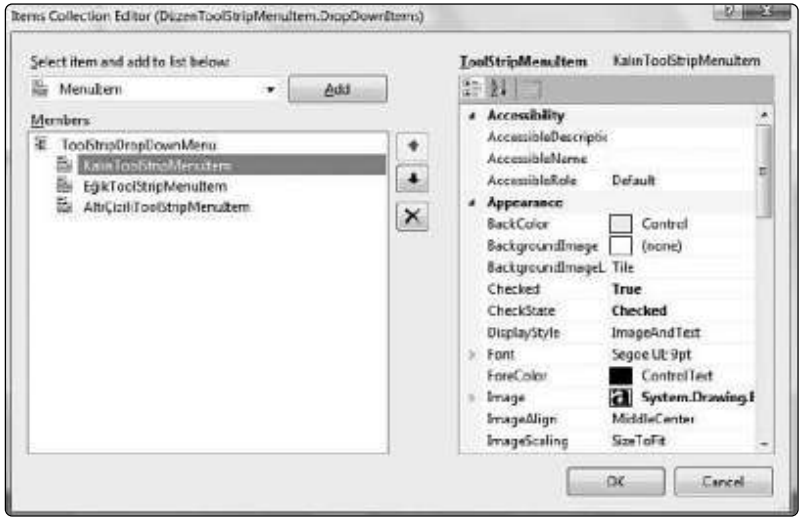


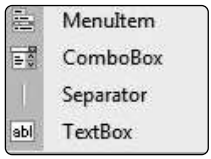
EditItems: Bu seçenek tıklanarak ana menü nesneleri üzerinde her türlü ekleme, silme, sıralama ve düzenleme işlemleri gerçekleştirilebilir. Ancak burada yalnızca ana menü nesneleri eklenebilir ve üzerlerinde işlemler gerçekleştirilebilir.



Alt menü seçenekleri üzerinde işlem yapmak için ise herhangi bir alt menü elemanı farenin sağ tuşu ile tıklanarak **EditDropDownItems** seçeneği tıklanır.







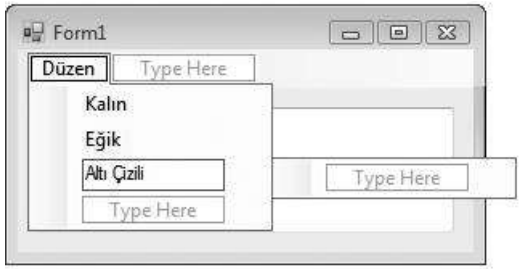
Menü öğeleri olarak **Insert** menüsü ile combobox, TextBox ya da menü elemanları arasına **ayırıcı çizgi** (*separator*) ekleyebileceğimiz gibi **ConvertTo** seçeneği ile de daha önceden oluşturduğumuz menü elemanlarını da dönüştürebiliriz.

Örnek bir menü tasarımı yapalım...

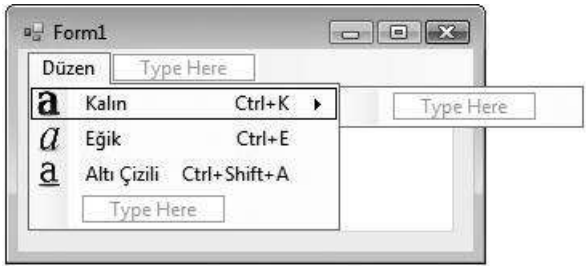
Öncelikle form üzerine bir adet textbox nesnesi ekleyerek Multiline özelliğini true yaparak textbox nesnesini çok satırlı hale getirelim.

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.

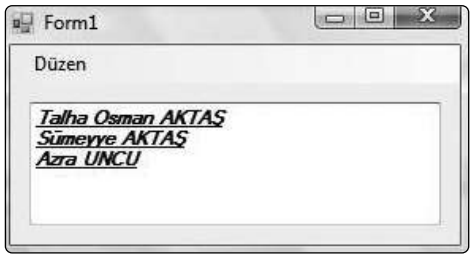
Örnek olarak **Düzen** adlı bir ana menü, bu ana menü altında da **Kalın**, **Eğik** ve **Altı Çizili** şeklinde alt menüleri oluşturduk.



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);
}
```



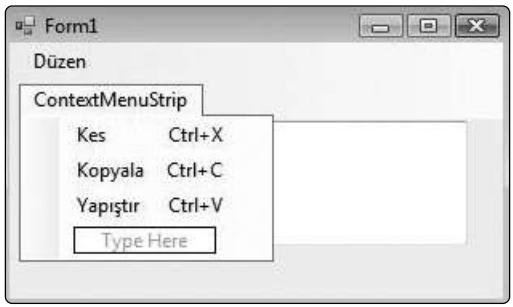
Kalın menüsü tıklandığında text box içerisindeki yazı kalın yapılır. Bir defa daha tıklanırsa kalınlık özelliği iptal edilir. Diğer menü öğeleri de aynı şekildedir. Aynı zamanda kısayol tuşları da kullanılabilir.

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 değinmeyeceğim. MenuStrip nesnesi konusunda yaptığımız örneği geliştirerek TextBox nesnesi içerisinde farenin sağ tuşu ile tıklandığında **Kes**, **Kopyala**, **Yapıştır** seçeneklerinden oluşan sağ menü çıkmasını sağlayalım.

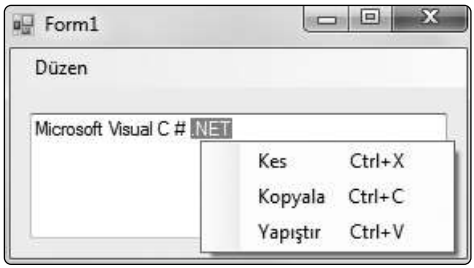


Formumuzun üzerine ek olarak `ContextMenuStrip` nesnesini ekliyoruz. `ContextMenuStrip` tasarımını yukarıdaki gibi yapıyoruz:

TextBox'ın ContextMenuStrip özelliğinden `ContextMenuStrip1` değerini seçerek ContextMenuStrip nesnemizi TextBox nesnesine bağlıyoruz.

```
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();
}
```



Kodumuzu çalıştıralım. Textbox içerisinde farenin sağ tuşu ile tıkladığımızda sağ menü görüntülenir.

NOTIFYICON KONTROLÜ

Formu taskbar içerisine simge durumuna küçültmek amacı ile kullanılan menü nesnesidir. Form üzerine eklenmeyip Visual Studio IDE penceresinin alt tarafındaki panele eklenir.

NOTIFYICON KONTROLÜNÜN ÖZELLİKLERİ

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 METODLARI

ShowBalloonTip: Belirtilen milisaniye cinsinden değer kadar süre balon yardımını görüntüler. Örneğin; `ShowBalloonTip(1000)` balon yardımını 1 saniye boyunca görüntüler.

Örnek olarak form kapatıldığında simge durumuna küçülten bir uygulama yapalım. Form üzerine bir adet notifyicon kontrolu ve form simge

durumuna küçültüldüğünde görüntülenecek menüler için bir adet de `contextMenuStrip` kontrolü ekliyoruz.

ContextMenuStrip içerisine Önceki `Boyut` ve `Çıkış` adı ile iki 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();
}
```

Formu simge durumunda görüntülemek için öncelikle formun `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);
```

Formun gizlenmesini, taskbar içerisinde simgenin görüntülenmesini ve balon yardımının bir saniye süre ile görüntülenmesini sağlıyoruz.



Oluşturduğumuz context menü 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 formun görüntülenerek simgenin gizlenmesini sağlıyoruz.

ImageList Kontrolü

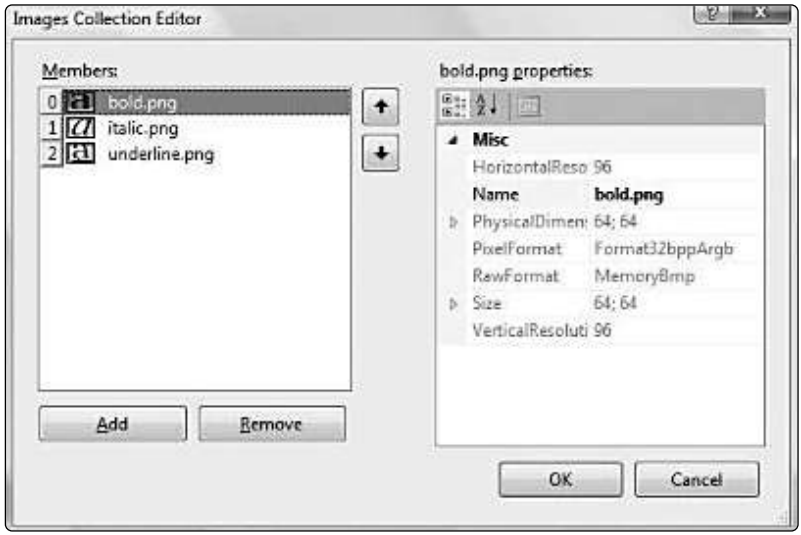
Resim dosyalarının bir arada saklanıp herhangi bir nesne içerisinden resim dosyalarına erişilmesini sağlamak amacı ile kullanılır. Form üzerine eklenmeyip Visual Studio IDE penceresinin alt tarafındaki panele eklenir.

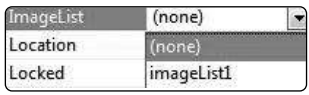
ImageList Kontrolünün Özellikleri

ImageSize: `ImageList` içerisine eklenecek resimlerin genişlik ve yükseklik değerlerini belirler.

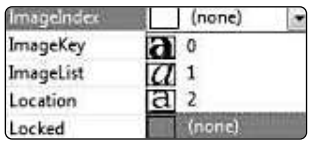
ColorDepth: `ImageList` nesnesi içerisindeki resimlerin renk derinliğini belirler.

Images: Images Collection Editor penceresini görüntüler. Resim ekleme, silme, düzenleme işlemleri yapılabilir.

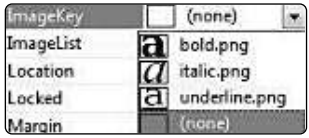




Bir imagelist oluşturulduktan sonra herhangi bir nesneye ImageList nesnesini bağlamak için ImageList özelliğinden ImageList nesnesinin adını seçmek yeterlidir.



Daha sonra bağlantı kurduğumuz nesnenin ImageIndex özelliği ile resim dosyalarının index numaralarını seçerebiliriz.



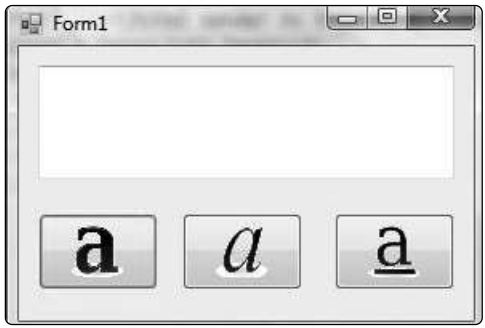
Aynı işlemi ImageKey özelliği ile de resim dosyalarının isimlerini seçerek gerçekleştirebiliriz.

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);
```



ToolStrip Kontrolü

ToolStrip

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.

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.

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.

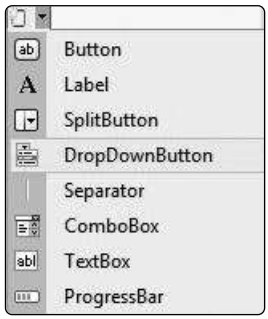

ToolTipText: Araç çubuğu elemanı üzerinde fare ile beklenildiğinde görüntülenecek olan açıklama yazısını belirler.

Örnek bir web browser uygulaması yaparak örneğimiz içerisinde araç çubuğu tasarımı yapalım. Form üzerine bir adet `MenuStrip`, bir adet `ContextMenuStrip`, bir adet `ToolStrip` nesnesi, bir 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 ise **DropDownButton** nesnesi eklemeliyiz.



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 yandaki şekildeki 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 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 web browser 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 isminin 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://");
```

Adresin baş kısmına `"http://"` ifadesini ekliyoruz.

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.GeriTo-
olStripMenuItem_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

StatusStrip

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.



Örneğin toolStrip konusunda yapmış olduğumuz WebBrowser örneğini StatusStrip ekleyerek geliştirelim. Ayrıca bir adet de timer nesnesi ekleyeceğiz. Timer nesnesinin `Interval` özelliği `1000`, `enabled=false` olacak.

Status strip içerisine ise ziyaret edilen sayfaların isimlerinin görüntülenmesi için bir adet `StatusStripLabel`, sayfanın görüntülenme hızını göstermesi için bir adet `ToolStripProgressBar`, ziyaret edilen sayfaların listesinin görüntülenmesi için de bir adet `ToolStripDropDownButton` nesnesi ekliyoruz. `ToolStripDropDownButton` nesnesinin `DisplayStyle` özelliğini `Text` olarak ayarlayarak yalnızca yazı görüntülenmesini sağlıyoruz.

```
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 başlanıldığında 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 da 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.

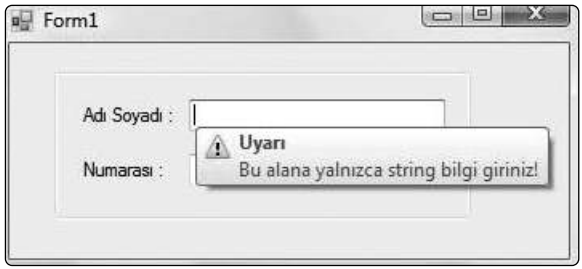
ToolTipTitle: Açıklama penceresinin başlığında görüntülenecek metni belirler.

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.

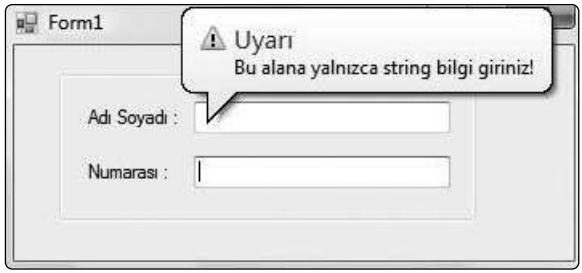
UseFading: Açıklama penceresinin görünürken saydam görünümden normal görünüme doğru geçiş yaparak, kaybolurken de normal görünümden saydam görünüme geçiş yaparak kapanmasını sağlar.

IsBallon: True değerini aldığında açıklama penceresi balon şeklinde görüntülenir.

IsBallon özelliği **false** değerini aldığında:



IsBallon özelliği **true** değerini aldığında:



TOOLTIP KONTROLÜNÜN METODLARI

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) ;
```

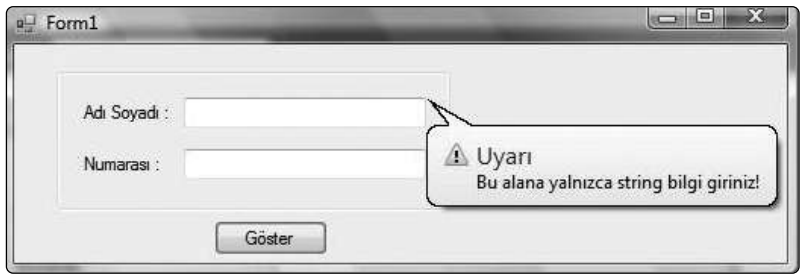
`txtadSoyad` kontrolü üzerinde açıklama penceresini görüntüler.

```
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);
```

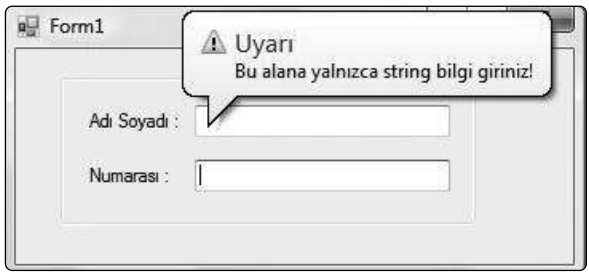
`txtAdSoyad` kontrolünün üzerinde x koordinatı `txtAdSoyad` kontrolünün genişliği, y koordinatı ise `txtAdSoyad` kontrolünün yüksekliğinin `20` `piksel` aşağısı olan noktada açıklama penceresini `5` saniye görüntüler.



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!");
}
```



`ToolTipIcon` özelliği daha önce belirttiğimiz gibi açıklama üzerinde yer alacak simgeyi belirler. Bu özelliğe kod ile `ToolTipIcon.Warning`, `ToolTipIcon.Error`, `ToolTipIcon.Info`, `ToolTipIcon.None` şeklinde açıklama penceresi üzerinde görüntülenecek olan simge tiplerinin atamasını yapabiliriz.

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 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.

```
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 (txtAdSoyad.Text == "")
    {
        errorProvider1.SetError(txtNumara, "Ad Soyad 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 var ise `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

HelpProvider

Herhangi bir uygulama ya da kontrolle ilgili yardımı 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.

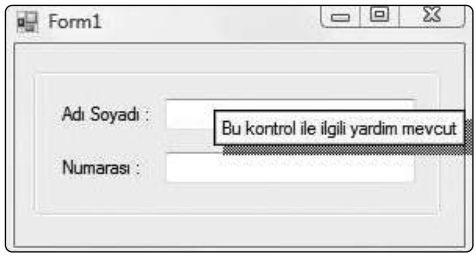
```
helpProvider1.HelpNamespace = "http://www.volkanaktas.com";
```

şeklinde kullanıldığında ise **F1** tuşuna basıldığında belirtilen internet sayfası görüntülenir.

HelpProvider Kontrolünün Metodları

SetHelpString: Belirtilen kontrol aktifleştirildiğinde görüntülenecek olan açıklama mesajını belirler. Görüntülenebilmesi için `HelpNameSpace` özelliğinin boş bırakılması gerekmektedir.

```
private void Form1_Load(object sender, EventArgs e)
{
    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. Keywordlar yardım dosyası içerisinde arama esnasında ulaşımı kolaylaştırmak için tanımlanan anahtar kelimelerdir.

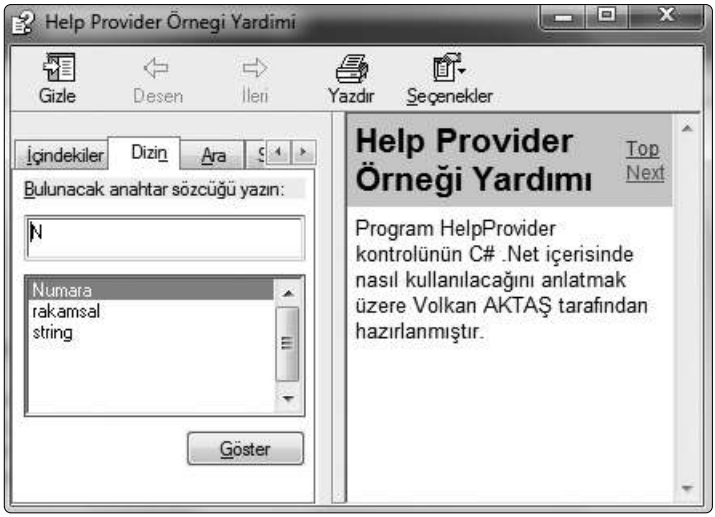
GetHelpKeyword: Belirtilen bir konrol için `SetHelpKeyword` metodu ile atanmış olan anahtar kelimeyi görüntüler.

SetHelpNavigator: Açılan yardım dosyasının açılış şeklini belirler.

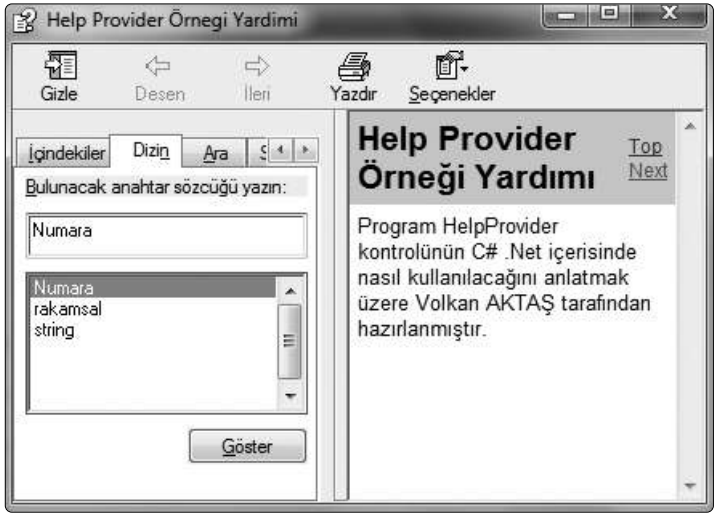
Şu değerleri alır;

- **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.

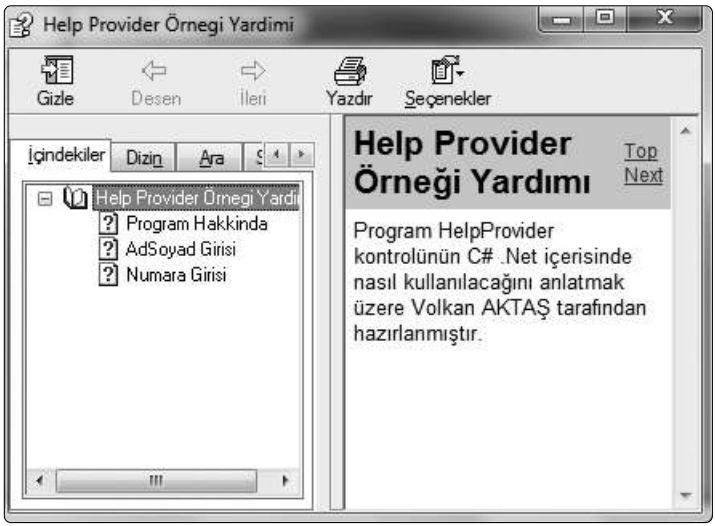


- **KeywordIndex:** Yardım dosyası açılırken `SetHelpKeyword` metodu ile belirtilmiş keyword'un bulunduğu ilgili yardım bölümünün görüntülenmesini sağlar.

```
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);
}
```

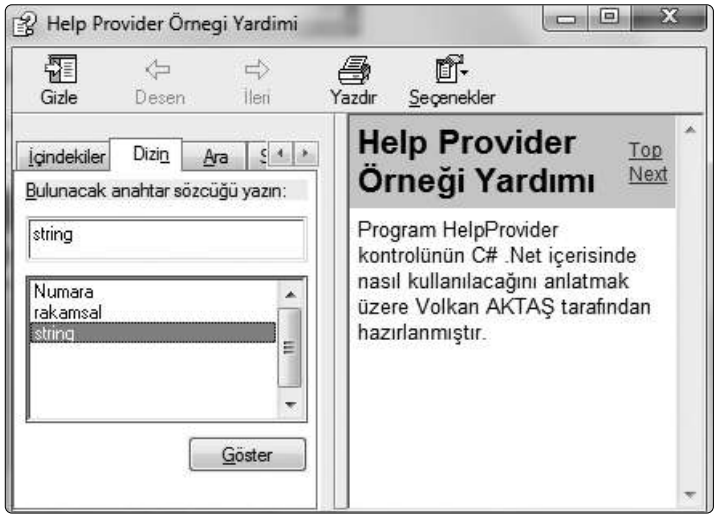


- **TableOfContents:** Yardım dosyası açılırken **İçindekiler** sekmesinin aktif olarak açılmasını sağlar.



- **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 `DateTime Picker` kontrolü eklendiğinde varsayılan olarak o anki tarih ve zamanı gösterir.

DATETIMEPICKER KONTROLÜNÜN ÖZELLİKLERİ

Format: Görüntülenecek olan tarih veya zamanın gösterim biçimini belirler. Aşağıdaki değerleri alır:

- **Long:** Tarih ve zamanı birlikte görüntüler.



- **Short:** Yalnızca tarihi görüntüler.



- **Time:** Yalnızca zamanı görüntüler.



- **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.

Karakter	Anlamı
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 şeklinde görüntülenir.
yyyy	Yıl bilgisini tam hali ile görüntüler. Yıl bilgisi 2009 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 kullanarak örnek formatlar tanımlayalım:

`18/09/2009` tarihi ve `17:32:10` zamanı için işlemler yapalım.

`hh:mm:ss:` `05:32:10` değerini görüntüler.

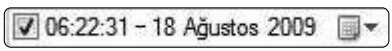
`HH:mm:ss:` `17:32:10` değerini görüntüler.

`dd MM yyyy:` `18 09 2009` değerini görüntüler.

`dd MMMM yyyy:` `18 Ağustos 2009` değerini görüntüler.

`HH:mm:ss – dd MMMM yyyy:` `18 Ağustos 2009 – 17:32:10` değerini görüntüler.

ShowCheckBox: `True` değerini aldığında `DateTimePicker` kontrolünün sol tarafında checkbox 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.



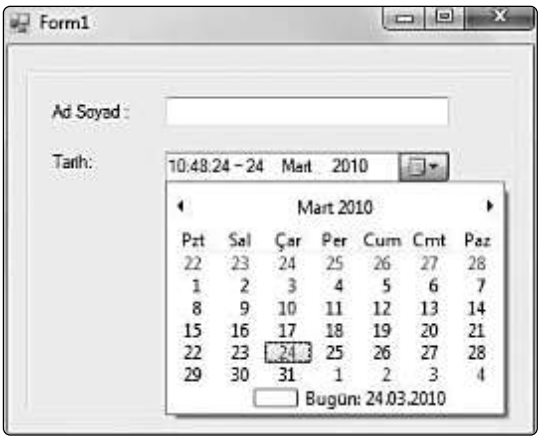
Value: DateTimePicker kontrolünün aldığı tarih ve zaman değerini verir. Görüntülenen tarih ve zaman formatı değiştirilebilir. Şu format değerlerini alabilir:

- **ToLongDateString:** `21 Ağustos 2009 Cuma` örneğinde olduğu gibi tarihi görüntülerken hem ay hem de gün ismi ile beraber görüntüler.
- **ToShortDateString:** `21.08.2009` örneğinde olduğu gibi tarihi rakamlar şeklinde görüntüler.
- **ToLongTimeString:** `13:27:08` örneğinde olduğu gibi zamanı saat: dakika:saniye şeklinde görüntüler.
- **ToShortTimeString:** `13:27` 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.

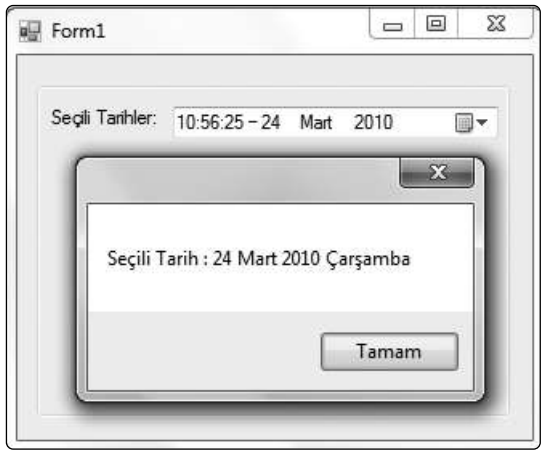
```
private void Form1_Load(object sender, EventArgs e)
{
    dateTimePicker1.Format = DateTimePickerFormat.Custom;
    dateTimePicker1.CustomFormat = "hh:mm:ss - dd MMMM yyyy";
}
```



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());
}
```



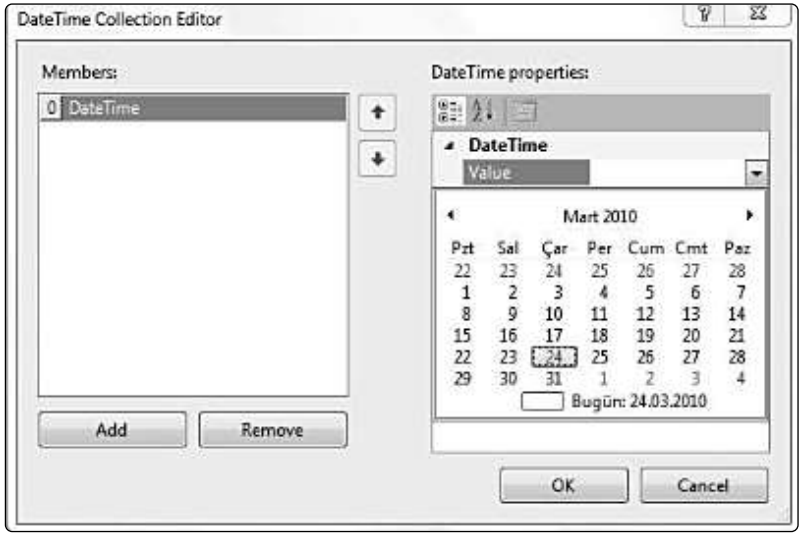
MonthCalendar Kontrolü



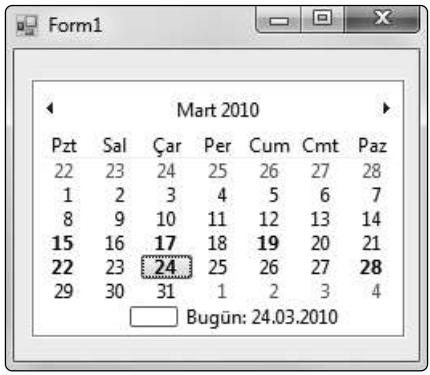
Bir tarihi grafiksel olarak göstererek bu tarihler arasında gezinmeyi sağlayan bir kontroldür.

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.

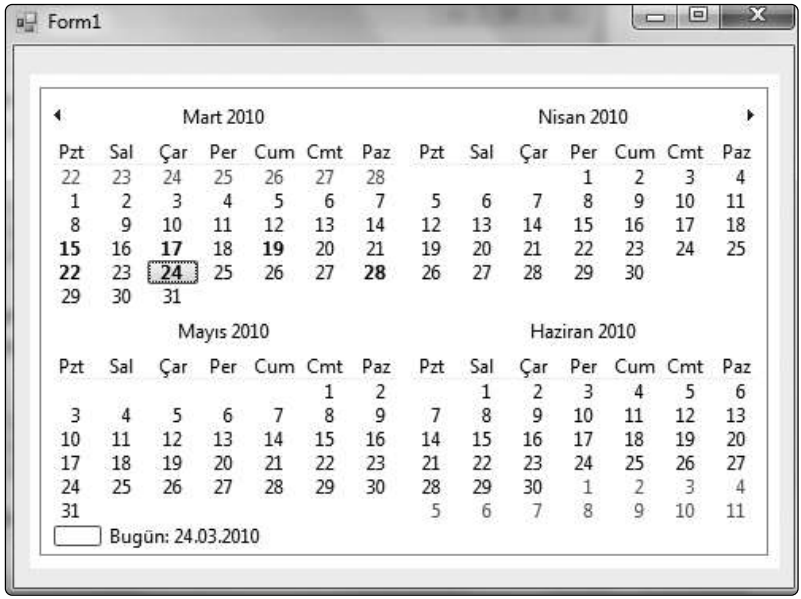


Örneğin takvimimizde `BoldedDates` özelliğine değer olarak `15, 17, 19 Mart 2010` tarihlerini ekleyelim. `AnnuallyBoldedDates` özelliğine ise değer olarak `22, 24, 28 Mart 2010` tarihlerini ekleyelim.



Tarihi değiştirip `Mart 2011` yaptığımızda `AnnuallyBoldDates` özelliği ile eklediğimiz `22`, `24` ve `28 Mart` günlerinin koyu renkli olarak gösterildiğini 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.

MonthlyBoldedDates: Her ayın belli bir gün ya da günlerini koyu renkli olarak görüntülemek amacı ile kullanılır.

SelectionRange: Seçili olarak görüntülenecek tarih aralığını belirler.



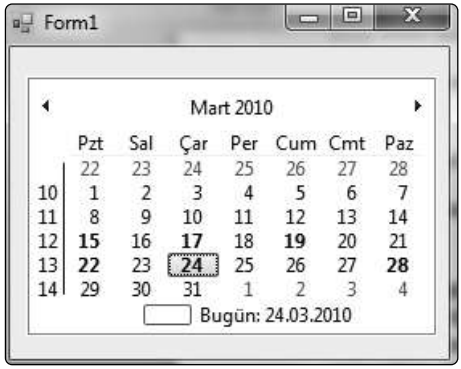
Örneğin bu özelliğe `24.04.2010; 27.04.2010` değerini girdiğimizde `24-27 Nisan 2010` günleri seçili olarak görüntülenir. `Start` metodu ile seçimin başlangıç noktası, end metodu ile de bitiş noktası belirlenebilir ya da belirlenmişse görüntülenebilir.

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 aşağıdaki 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.

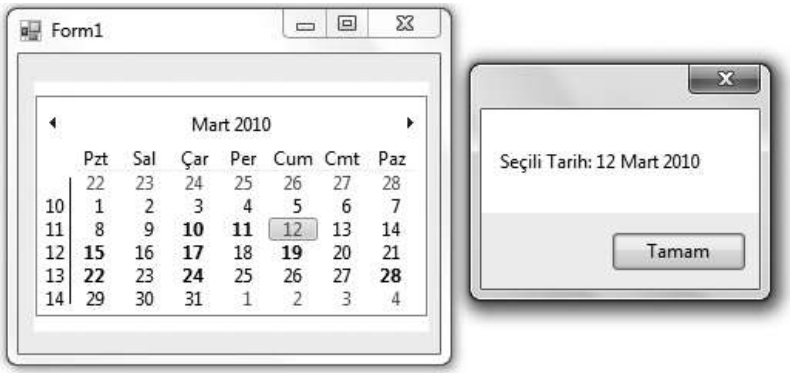
UpdateBoldedDates: Koyu renkli olarak görüntülenmek üzere eklenmiş olan günlerin görüntülenmesi amacı ile kullanılır.

MonthCalendar Kontrolünün Olayları

DateSelected: MonthCalendar kontrolünün üzerinde bir tarih seçildiğinde görüntülediği tarih değiştirildiğinde gerçekleşen olaydır.

```
private void Form1_Load(object sender, EventArgs e)
{
    monthCalendar1.ShowToday = false;
    monthCalendar1.ShowTodayCircle = false;
    monthCalendar1.AddBoldedDate(DateTime.Parse("2010/3/10"));
    monthCalendar1.AddBoldedDate(DateTime.Parse("2010/3/11"));
    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"* gibi.

Archive: Arşiv dosyalarını listelemek amacı ile kullanılır.

Hidden: Gizli dosyaları 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 dosyayı çift tıklanıldığında çalıştırıyoruz.

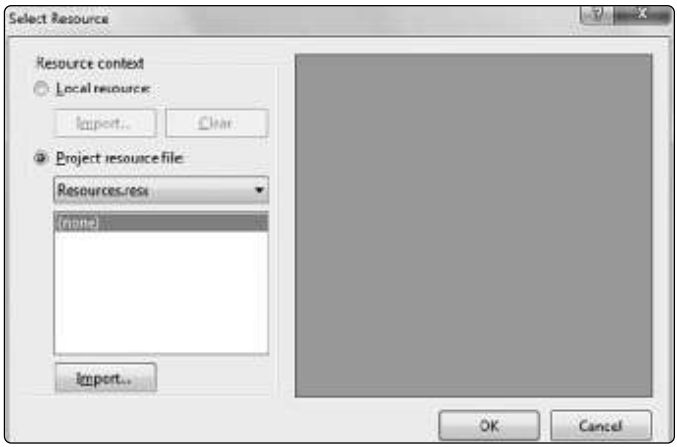


14

RESOURCE DOSYALARI

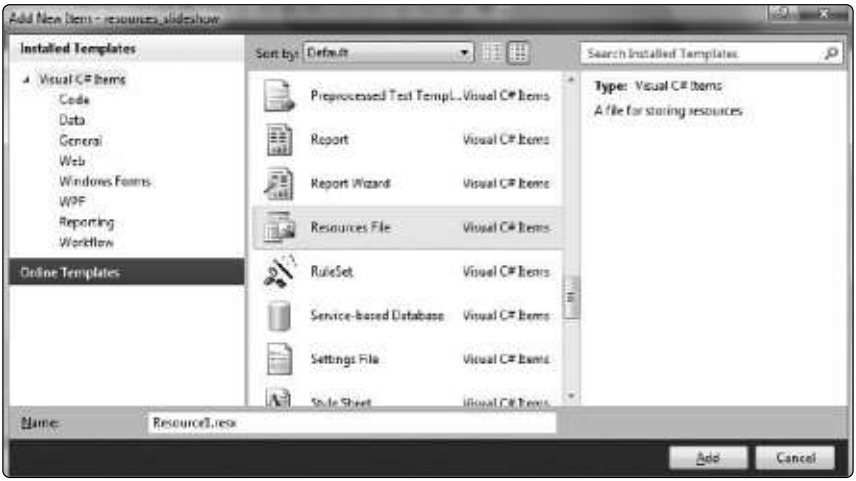
Resource Dosyaları

.Net içerisinde resim, müzik, video gibi dosyaları projelerimiz içerisine eklerken iki yöntemle ekleyebiliriz. Birinci yöntem, **local resource** olarak eklemektir. Bu yöntemde proje içerisinde kullanacağımız dosyaları proje başka bilgisayarlarda çalıştığında oraya taşımamız gerekir.

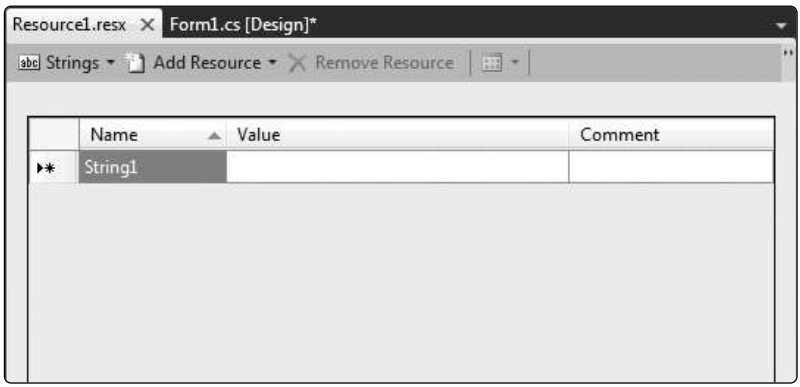


İkinci yöntem ise, **resource dosyası** oluşturarak içerisine eklemektir. Bu yöntemde dosyalar proje içerisine eklenmiş olur. Resource dosyaları resx uzantısı ile kaydedilirler. Resource dosyası içerisine eklenen tüm resim, yazı, ses, video gibi dosyalara tek bir resource dosyası içerisinden erişilebilinir.

Örnek bir resource dosyası oluşturarak içerisine resim dosyaları ekleyelim. **Project / Add New Item** seçeneği ile yeni bir **Resources File** oluşturuyoruz.



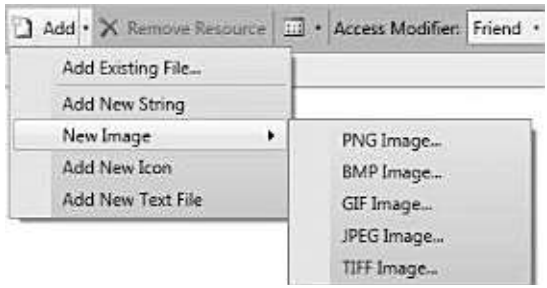
Gelen **Resources** ekranında Resources dosyası içerisine ekleme ve silme işlemleri gerçekleştirebiliriz.



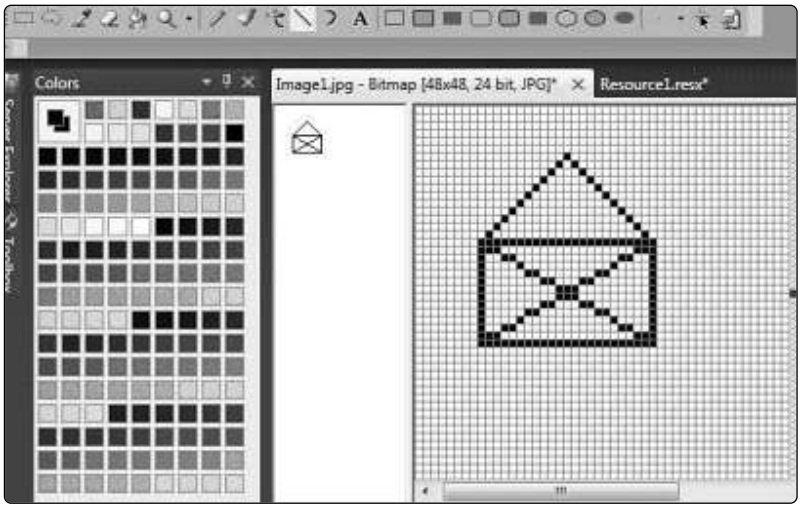


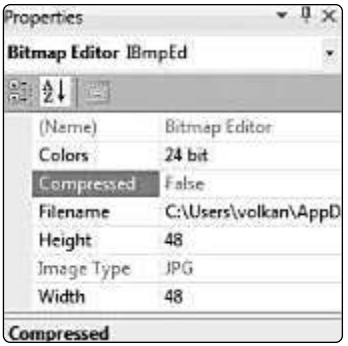
Strings yazısının yan tarafında bulunan kulakçığa tıklayarak Resources dosyası içerisine eklenebilecek dosya türünü değiştirebiliriz. Biz resim dosyası ekleyeceğimiz için **Images** seçeneğini işaretliyoruz.

Add New Image seçeneği ile yeni bir resim dosyası oluşturulur.



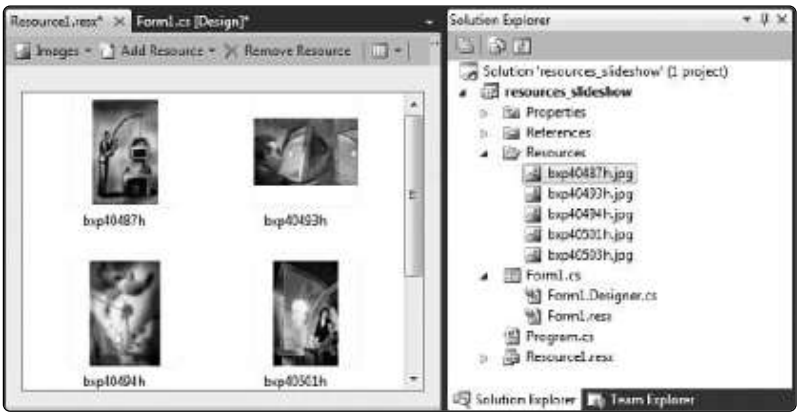
Yeni bir resim dosyası oluşturmak için karşımıza bir tasarım ekranı gelecektir. Bu ekran vasıtası ile yeni bir resim dosyası oluşturabiliriz. Ayrıca gerekli olan temel araçlar Visual Studio IDE içerisine eklenen bir araç çubuğu vasıtası ile sağlanır. Burada temel çizim ve boyama işlemleri, şekiller oluşturma gibi işlemleri gerçekleştirebiliriz.





Properties penceresi yardımı ile resim boyutu, renk düzeni, sıkıştırma ile ilgili 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** penceresinden 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 oluştulmuş bir resource dosyasına eklenmiş olan resimleri kullanarak slayt gösterisi yapalım.

Form üzerine bir adet `timer` nesnesi (`interval` değeri `1` olacak), bir adet de `updown` nesnesi (`minimum` değeri `1`, `maksimum` değeri `100` olacak) ekliyoruz.

```
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.bxp40487h;    }
      else if (i == 2)  {
            pictureBox1.Image = Resource1.bxp40493h;    }
      else if (i == 3)  {
            pictureBox1.Image = Resource1.bxp40494h;    }
      else if (i == 4)  {
            pictureBox1.Image = Resource1.bxp40501h;    }
      else if (i == 5)  {
            pictureBox1.Image = Resource1.bxp40503h;
            i = 0;
      }
}
private void numericUpDown1_ValueChanged(object sender,
EventArgs e)
```

```
{
     timer1.Interval = Convert.ToInt32(numericUpDown1.Value) * 1000;
}
```



Properties penceresinde `pictureBox` nesnesinin `SizeMode` özelliğini `true` olarak ayarladık.

Slaytlar arası geçiş süresini saniye cinsinden bir `UpDown` nesnesi yardımı ile ayarlabiliyoruz.

Resource Dosyası İşlemleri

Resources dosyaları üzerinde işlem yapabilmek için öncelikle `System.Resources` sınıfını projenize dahil etmeniz gerekmektedir. Çünkü Resources dosyaları bu sınıftan türetilmektedir. Daha sonra ise Assembly nesnesine erişebilmemiz için `System.Reflection` sınıfını projenize dahil etmeniz gerekmektedir.

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( )`, bize 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.

```
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!");
     }
  }
}
```

Formun 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 elemanları `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();
```

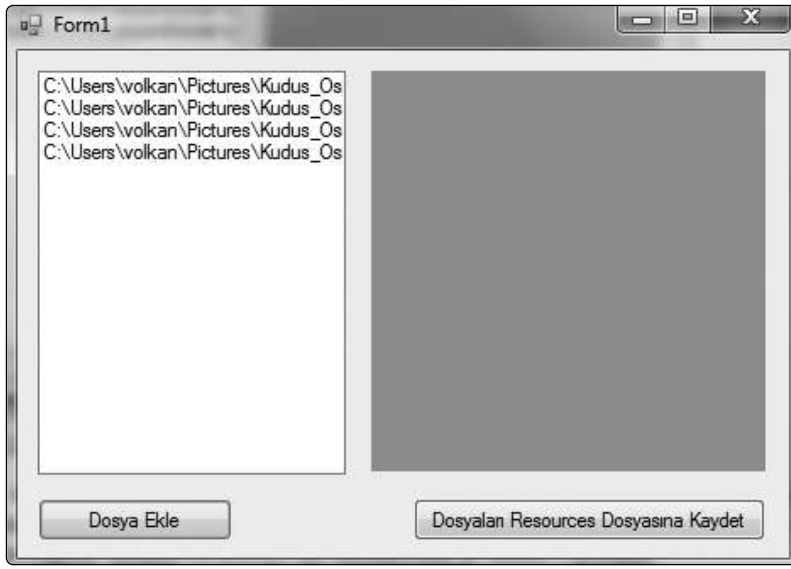
Yazma işleminin bitiminde `ResxResourceWriter` nesnemizi kapatıyoruz.

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());
```

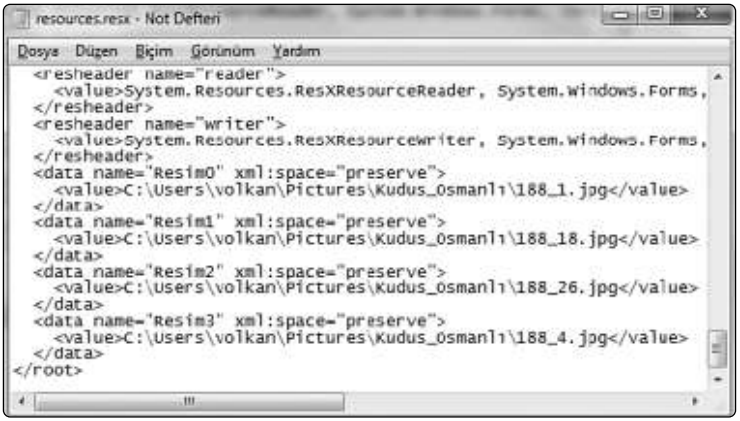
Eğer Resource dosyası içerisinde okunan isim ile listBox nesnesi içerisinde okunan isim birbirine eşit ise picturebox içerisinde resource dosyası içerisinde okunan değer ismi ile yeni bir Bitmap nesnesi oluşturularak görüntülenir.

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 tagında dosya ismini, <value> tagında 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.



15

Çizim İşlemleri

Form üzerine yazı ya da grafikleri çizdirmek amacıyla `Graphics` sınıfı kullanılır. `Graphics` sınıfı `System.Drawing` sınıfı altında yer almaktadır.

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 stringin 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.

NOT

Bundan sonraki form üzerine çizdirme örneklerimizdeki

```
using System.Drawing;
```

satırını gereksiz tekrarlamamak için yazmayacağım. Ancak okuyucularım her form üzerine çizim uygulamasına bunları yazmayı unutmasınlar.

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 btnCiz_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);
        }
}
```

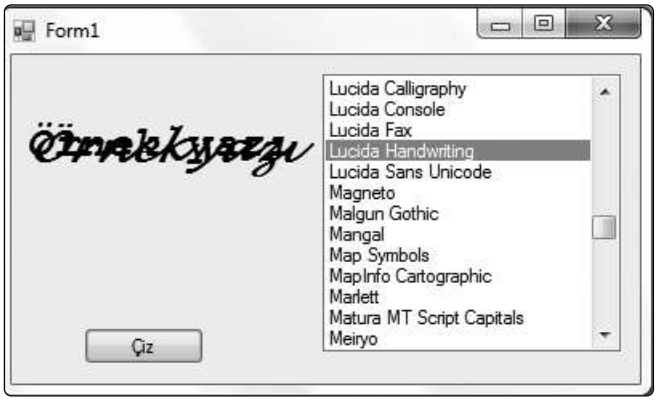
Örneğimizde

```
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 bir sonraki sayfada 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 metodlarını 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. Kullanımı aşağıdaki gibidir.

```
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);

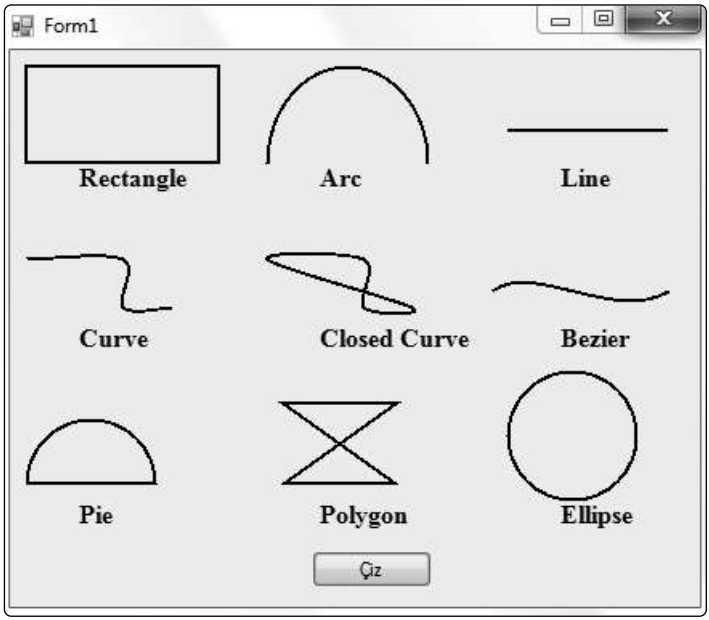
        formGraphics.DrawBezier(myPen, new Point(300, y + 140),
            new Point(325,y+120), new Point(375, y+160),
            new Point(x + 400, y + 140));
        formGraphics.DrawString("Bezier", drawFont, drawBrush,
            x + 330, y + 160);

        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

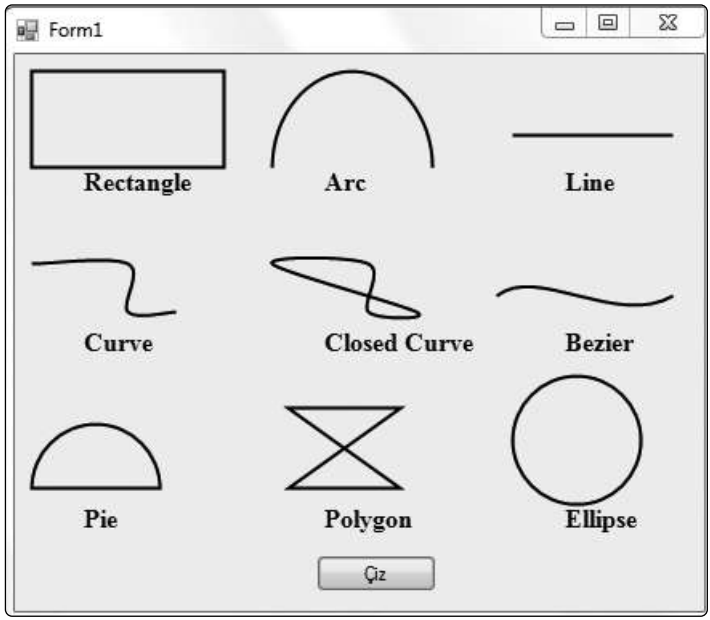
```
Pen myPen = new Pen(Color.Black,2);
```

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 kodumuza

```
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` metodları 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();
        }

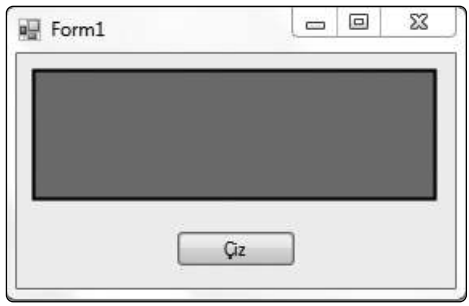
        private void btnCiz_Click(object sender, EventArgs e)
        {
```

```
            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:

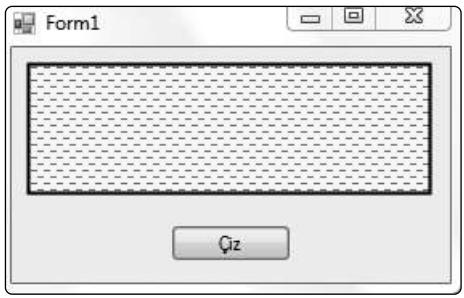
```
HatchBrush(Brush biçimi, Çizim rengi, Zemin Rengi)
```

```
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);

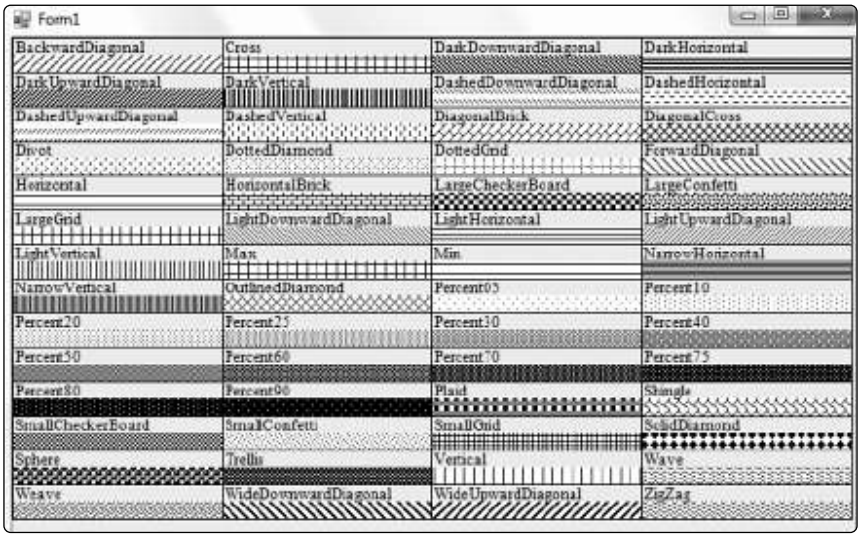
     formGraphics.SmoothingMode = SmoothingMode.AntiAlias;

     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. Kullanım biçimi aşağıdaki gibidir:

```
LinearGradientBrush (Point1 , Point 2, Renk 1, Renk 2,
LinearGradientMode)
```

Renk tonlamasında brush seçeneği olarak iki 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;

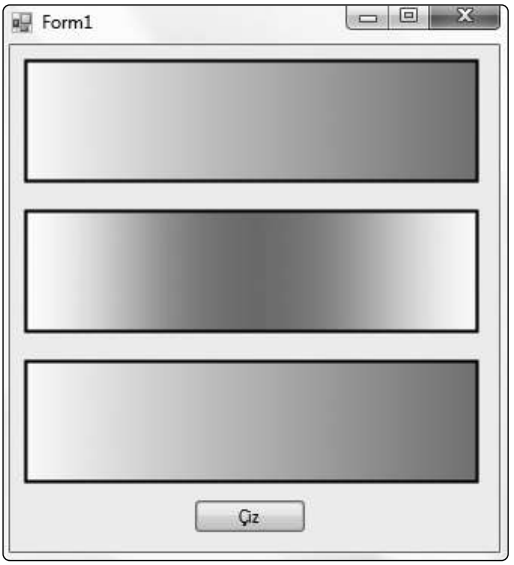
          LinearGradientBrush LGBrush = new LinearGradientBrush
              (ClientRectangle, Color.White, Color.Gray,
              LinearGradientMode.Horizontal);

          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.

```
private void btnCiz_Click(object sender, EventArgs e)
{
     Pen myPen = new Pen(Color.Black, 2);
     Graphics formGraphics = null;
     formGraphics = this.CreateGraphics();

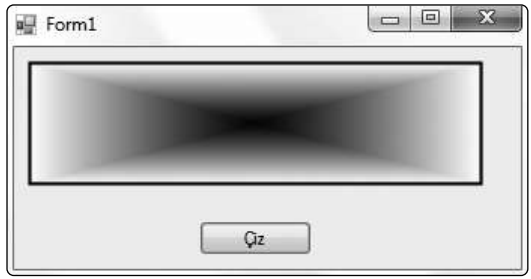
     formGraphics.SmoothingMode = SmoothingMode.AntiAlias;

     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, y, 300, 80));
     formGraphics.DrawRectangle(myPen, new Rectangle
              (10, y, 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);
```

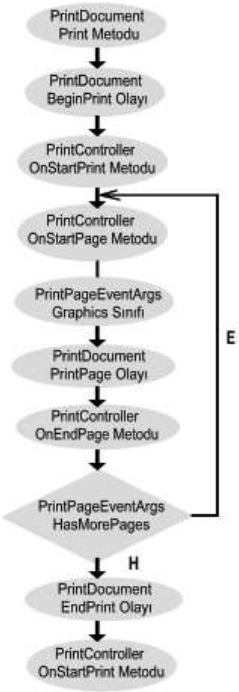
satırlarında ise çizdirdiğimiz dikdörtgen içerisine eklenen satırların, dikdörtgenin çizgileri üzerine denk gelmeyecek biçimde kaydırılmasını sağlıyoruz.



16

YAZDIRMA İŞLEMLERİ

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 Metodları

Print: Yazdırma işlemini başlatmak amacı ile kullanılan metoddur. Print olayı gerçekleşirken `PrintController` sınıfının `OnStartPrint`, `OnEndPrint`, `OnStartPage` ve `OnEndPage` olayları gerçekleşir. `PrintController` sınıfı bir dökümanın nasıl yazdırılacağını belirler. `PrintController` sınıfı `Print-`

`Document` sınıfı tarafından kullanılıp, tek başına kullanılamaz. `PrintController` ve `PrintDocument` sınıfları arasındaki olaylar yandaki resimde 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 inceleyelim.

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.Text;
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.

```
pd.PrintPage += new PrintPageEventHandler(pd_PrintPage);
```

`PrintPage` olayı içerisine event handler ile `pd_PrintPage` adındaki metodumuzu bağladık ve bu metod içerisinde yazdıracağımız belgenin içeriği 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);
```

`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 stringin 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 yazırılacak 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.

`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 dökümanın 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.

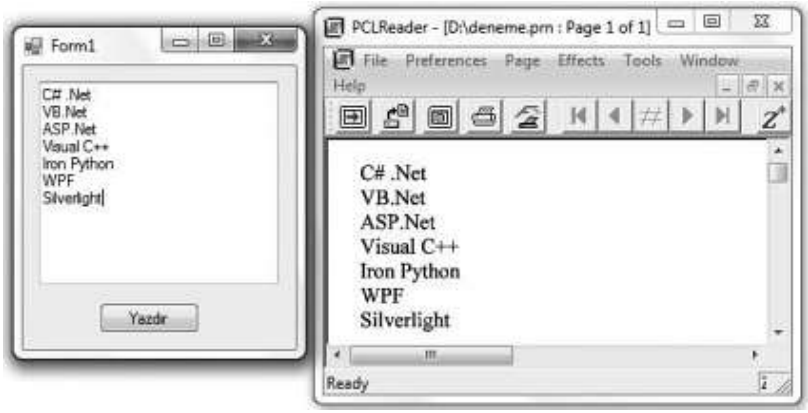
ToPage: Yazdırılan belgenin son sayfasının numarasını bulmak yada belirlemek amacıyla kullanılır.

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";
      openFileDialog1.InitialDirectory =
          "C:\\Users\\volkan\\Documents";
      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);

      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.



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 `OnPrint-Page` 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 üç farklı print diyalog nesnesi bulunmaktadır. Bunlar; `PrintDialog`, `PrintPreviewDialog` ve `PageSetupDialog` nesneleridir.

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.



`PrintDialog` yerine `PrintPreviewDialog` penceresini görüntülemek için yukarıdaki kod satırlarımızı 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);

     sr = new StreamReader(txtDosyaAdı.Text);

     DialogResult basilan_tus;
     basilan_tus = openFileDialog1.ShowDialog();
     if (basilan_tus == DialogResult.OK)
          txtDosyaAdı.Text = openFileDialog1.FileName;
}
```

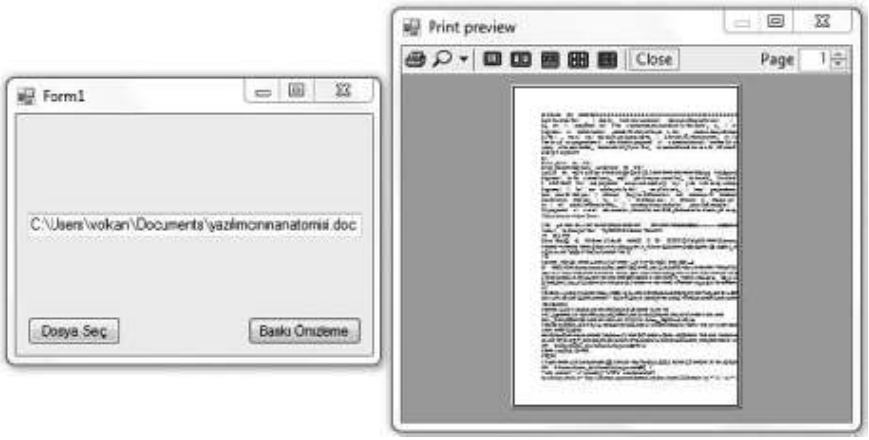
Görüldüğü gibi `PrintDialog` nesnesi yerine `PrintPreview` nesnesini oluşturmak için;

```
PrintDialog dlgPrint = new PrintDialog();
```

tanımlamasını

```
PrintPreviewDialog dlgPrintPreview = new PrintPreviewDialog();
```

biçiminde tanımlamak yeterlidir.



`PageSetupDialog` penceresini görüntülemek için 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);

     sr = new StreamReader(txtDosyaAdı.Text);

     PageSetupDialog dlgPageSetup = new PageSetupDialog();
     dlgPageSetup.Document = pd;
     dlgPageSetup.PageSettings.Landscape = true;
     dlgPageSetup.ShowDialog();
}
```

`PageSetupDialog` nesnesini

```
PageSetupDialog dlgPageSetup = new PageSetupDialog();
```

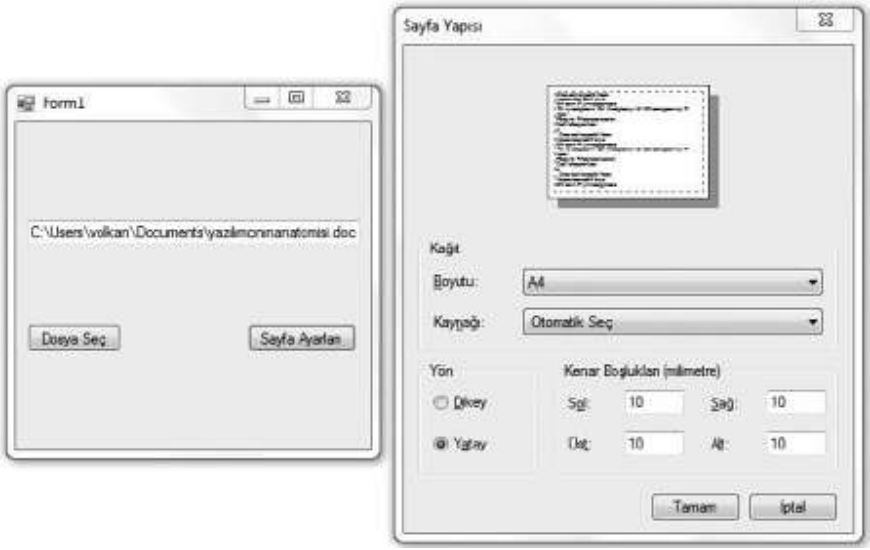
biçiminde tanımlıyoruz.

`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.



17

METODLAR

Program yazarken bazı durumlarda bir işi gerçekleştiren kod yapılarını tekrar tekrar yazmak durumunda kaldığımız olur. Bu kodları yeniden yazmak yerine metodları kullanabiliriz. C# içerisinde geriye değer döndüren ve değer döndürmeyen metodlar olmak üzere iki çeşit metod yapısı kullanılır.

Değer Döndürmeyen Metodlar

Kendi ismi ile çağırılarak içerisinde yazılmış olan kod satırlarını çalıştıran ve geriye herhangi bir değer döndürmeyen kod bloğudur. Değer döndürmeyen metodlar `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 metod tanımlanarak, butona tıklandığı anda bu metod ismi ile çağırılmış ve form üzerindeki textbox nesnelerinin renginin değiştirilmesi sağlanmıştır.

Metod İçerisinde Parametre Kullanımı

Metod 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 metod içerisinde farklı işlemler yapılıyor.

Yukarıdaki örneğimizde nesne parametresinin kullanımı zorunludur. Kullanımı zorunlu olmayan parametreleri vb.net içerisinde bulunan `optional` ifadesi ile tanımlayabiliriz. `Optional` ifadesini kullanabilmek için `System.Runtime.InteropServices` sınıfını projemize dahil etmemiz gerekmektedir.

Optional parametreleri tanımlarken ilk değer vermek zorunludur. DefaultParameterValue parametresi ile ilk değer atama işlemi gerçekleştirilir.

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Runtime.InteropServices;
namespace Renk_Degistir_Metod_Optional
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        void Renk_Degistir(Control nesne, [Optional,
        DefaultParameterValue("")]  string yazi)
        {
            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;
            }
            nesne.Text = yazi;
        }
        private void Button1_Click(object sender, EventArgs e)
        {
            Renk_Degistir(TextBox1);
            Renk_Degistir(Label1, "AdSoyad");
        }
    }
}
```

Optional olarak tanımladığımız yazi değişkenine ilk değer olarak boş string ifade aktardık. Button1 içerisinden ise "AdSoyad" değerini gön-

dererek Label1 nesnesinin text değerinin değiştirilmesini sağladık. Görüldüğü gibi;

```
Renk_Degistir(TextBox1);
```

şeklinde yazi parametresini kullanmadan Renk_Degistir alt programını çağırabildiğimiz gibi

```
Renk_Degistir(Label1, "AdSoyad");
```

şeklinde yazi parametresini kullanarak da Renk_Degistir alt programını çağırabiliriz.

Eğer optional parametreye değer ataması yapılmazsa varsayılan değer kullanılır.

Örneğimiz üzerinde değişiklik yapıp bu defa yazi değişkenine "Örnek Değer" varsayılan değerini aktaralım.

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Runtime.InteropServices;
namespace Renk_Degistir_Metod_Optional
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        void Renk_Degistir(Control nesne, [Optional,
        DefaultParameterValue("Örnek Değer")] ↓
        string yazi)
        {
            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;
            }
            nesne.Text = yazi;
        }
        private void Button1_Click(object sender, EventArgs e)
        {
            Renk_Degistir(TextBox1);
            Renk_Degistir(Label1, "AdSoyad");
        }
    }
}
```

`Renk_Degistir(TextBox1)` şeklinde `yazi` değişkenine değer ataması yapmadan kullandığımızda `TextBox1` nesnesinin text özelliğine `yazi` değişkeninin varsayılan değeri olan `"Örnek Değer"` string ifadesi aktarılacaktır.

Bir `optional` parametreden sonra başka bir `optional` parametre kullanılmalıdır. Dolayısı ile `optional` parametreler diğer parametrelerden sonra kullanılmalıdır.

Parametre sayısının belli olmadığı durumlarda `params` deyimi ile `sub` metodu 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 tanımladım.

```
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.

C# 4 ile birlikte metodlarda parametre kullanımı daha fonksiyonel hale gelmiştir. Örneğin daha önce vb.net'e ait olan `optional` ifadesini kullanarak yazdığımız kontrollerin renklerini değiştiren metodu yeniden yazalım.

```
void Renk_Degistir(Control nesne, string yazi="", 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 parametrelere değer aktarmak için ise

```
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.

Değer Döndüren Metodlar

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 metod 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));
        }
    }
}
```

`NotHesapla` metodumuz `notu` parametresi yardımı ile kendisine gönderilen değeri `5`'lik not sistemine çevirip geriye göndermektedir.

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 imkan 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.

Kontrollere Metod 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 textbox içerisinde enter tuşuna basıldığında `sorgula` isimli metodu çağırarak çalıştırılmasını sağlıyoruz.



Metodların Aşırı Yüklenmesi (Overloading)

Bir metodla birden fazla işlemi gerçekleştirmek için o metodla aynı isme sahip yeni metodlar tanımlanabilir. Ancak tanımlanan yeni metodlar farklı parametrelere sahip olmalıdır ya da tanımlanan parametrelerin isimleri aynı ise türleri farklı olmalıdır. Bu işleme metodların **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 metod 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)));
        }
}
```

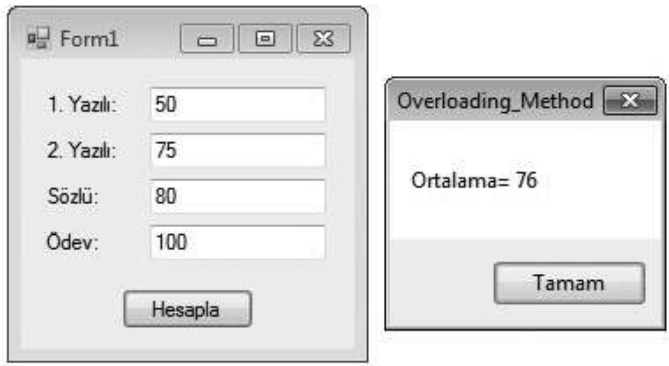
`Overloading` metodlar arasında yukarı aşağı ok tuşları ile dolaşabiliriz. Metodumuzun ilk kullanım şeklinde `yazılı1`, `yazılı2` ve `sozlu` adlı parametrelerin tanımlandığı ilk metodumuz görüntülenir.



Diğer kullanım şeklinde ise `odev` parametresinin de tanımlandığı ikinci metodumuz görüntülenir.



`odev` değerinin text kutusuna girilip girilmediğine göre ortalamayı hesaplatıp sonucu mesaj penceresi içerisinde görüntülüyoruz.



TEMSİLCİLER (DELEGATES)

Temsilciler en genel ifadesi ile metodların hafızadaki adreslerini tutarlar. Bu yönü ile C++ içerisinde kullanılan pointer'lara benzerler. Ancak pointer'lara göre daha güvenlidirler. Temsilciler metodların çağırılmasında aracılık ederler.

Örnek olarak kendisine gönderilen mesaj içeriği ve mesaj penceresi başlığı parametrelerini alarak bir mesaj penceresi içerisinde görüntüleyen bir metod hazırlayalım.

```
public void DelegateMetod(string mesaj, string baslık)
{
    MessageBox.Show(baslık,mesaj);
}
```

Bu metodu çağırmak için bir delagates 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 metod 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 metod 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 metodları bir structure, modül ya da class içerisinde tanımlayarak çağırabiliriz.

Örneğin Metodlar isminde bir `structure` tanımlayarak içerisinde `Mesaj`, `Topla` ve `Cıkar` adı ile 3 adet metod 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 metodları 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 structure tipinden veriyoruz.

Son olarak delegemize metodlarımızı 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 metod ekleme işlemi yapabiliriz. -= operatörü ile de eklenmiş olan metodlar çıkartılabilir. Yukarıda yaptığımız delegate içerisine metod 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 metodları kullanarak bu iki adımı bir tek adıma düşürebiliriz. Yani aynı anda hem delegate tanımlayıp hem de delegate ile çağıracağımız metodu tanımlayabiliriz.

Örnek olarak daha önce delegate konusunda yaptığımız mesaj penceresi görüntüleyen metodu anonim metod ile yeniden yazalım.

Metodun ilk hali 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 metod biçimde 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");
}
```

18

GENİŞLETME METODLARI (EXTENSION METHODS)

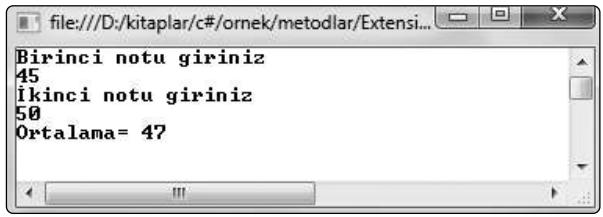
Genişletme metodları (*extension method*) kullanmakta olduğumuz metodlara yeni özellikler eklemek amacı ile kullanılırlar. Genişletme metodu tanımlarken aşağıdaki işlemleri yapmamız gerekmektedir:

1. Genişletme metodu bir class içerisinde tanımlanırlar.
2. Genişletme metodlarının içerisinde yer aldığı sınıf `static` olarak tanımlanmalıdır.
3. Genişletme metodunun ilk parametresinin değişken tipi metodun bağlandığı değişken tipi ile aynı olmalıdır.
4. Genişletme metodunun ilk parametresi `this` ifadesi ile tanımlanmalıdır.

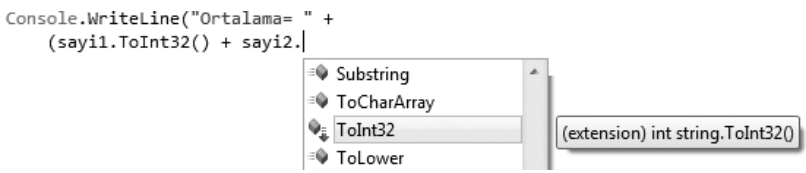
Örnek olarak bir `string` değeri `Int32` türüne dönüştürmek için bir genişletme metodu yazalım.

```
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);
     }
}
```

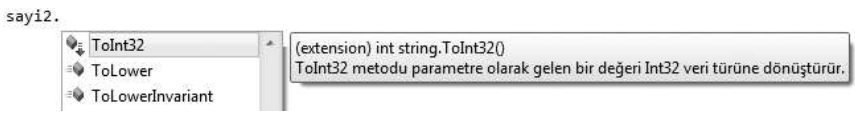
satırlarında genişletme metodu içerisinde kullanacağımız fonksiyon ve parametreleri tanımlıyoruz. Burada tanımladığımız `deger` parametresinin tipi ile genişletme metodunun çağırıldığı `sayi1` ve `sayi2` değişkenlerinin aynı tipte olduğuna dikkat edelim.

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 metodlarını `namespace` içerisinde tanımlayıp `using` komutu ile `namespace` adı ve genişletme metodu isimlerini proje içerisine dâhil ederek kullanabiliriz.

Bir console 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 metodlarını çağıracağımız kod satırlarını 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.



GENİŞLETME METODLARININ AŞIRI YÜKLENMESİ (OVERLOADING)

Genişletme metodları **aşırı yüklenme** (*overloading*) işlemi ile metodlar 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 metod 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 metodumuzu ilk tanımlamamızda

```
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 metodlarımızı 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 metodların kullanılabilmesini sağlıyoruz.

C#.Net içerisinde mevcut olan metodlardan birinin ismi ile aynı isme sahip bir genişletme metodu yazdığımızda yazdığımız genişletme metodu yerine var olan metod görüntülenecektir.

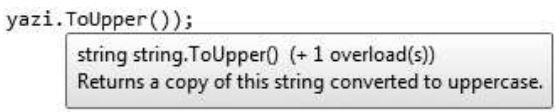
Örneğin bir string'i büyük harfe çeviren ToUpper metodu ile aynı isimde bir genişletme metodu yazalım.

```
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 metodlarının isimlerini genişletme metodu olarak kullanamayacağımız anlamına gelmiyor. Overloading işlemi ile var olan C#.Net metodlarını genişletme metodu olarak kullanabiliriz.

`ToUpper` metodu üzerinde overloading işlemi uygulayalım.

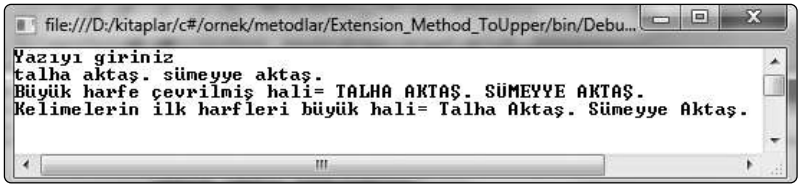
```
using System;
using System.Text;
namespace Extension
{
    static class Extensions
    {
        /// <summary>
        /// ToUpper metodu parametre olarak gelen bir değeri
```

```
        /// 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) (+ 2 overload(s))
ToUpper metodu parametre olarak gelen bir değerin kelimelerinin ilk harflerini büyük harfe dönüştürür.

19

CLASS (SINIF) YAPISI

Programlarımız içerisinde sıklıkla kullandığımız metodları 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 haline 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 proje şablonları içerisinden **Class Library** şablonunu seçmeliyiz.



Class yapısı içerisinde belli bir işlemi gerçekleştiren kod yapılarına **metod** adı verilir. Örnek olarak kendisine parametre olarak 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 metod 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 metodlar sınıflardan türetilen nesnelerden bağımsız hareket ederler. Do-

layısı ile static metodları 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 metodlara **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 metod otomatik olarak oluşturulur. Yapıcı metodlar 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();
      }

    }
}
```

Yapıcı metod 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 metod içerisine yazdığımız mesaj penceresi görüntüleme kodu sınıf çağrıldığında öncelikli olarak çalışmıştır.

Constructor metodların yanında sınıftan türettiğimiz nesne hafızadan silinirken çalışan **destructor** (*yıkıcı*) metodlar da mevcuttur. Yıkıcı metodlar metod 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.

Class yapısı içerisindeki metodlarımızda da class dışarısında tanımladığımız metodlarda 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. 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 metod tanımlayarak bu metodlara 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);
```

```
        }
    }
}
```

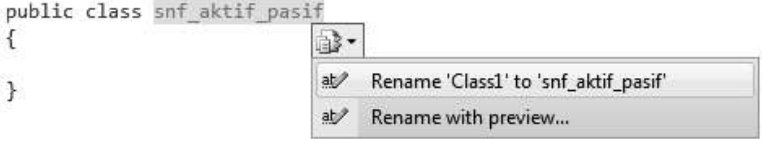
Hazırlamış olduğumuz bu class'ı dll dosyası şeklinde oluşturacak bir proje yazalım.

`aktif_pasif_dll` adı ile yeni bir `Class Library` proje oluşturuyoruz. Projemiz içerisine `System.Windows.Forms` dll dosyasını referans olarak ekliyoruz.

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

satırı ile bu dll dosyasını projemize dahil ediyoruz.

Class ismini daha anlaşılır olması için `snf_aktif_pasif` şeklinde değiştiriyoruz.



Class kodumuzu 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 haline 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** proje 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 pencereden **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 metodlarına erişimi sağladık.

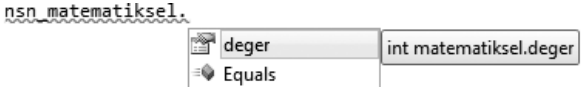
Property Kullanimi

Hazırladığımız sınıflar içerisinde **property** (*özellik*) kullanarak sınıflar içerisine değer gönderip, 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. `matematiksel` isimli class kodumuz:

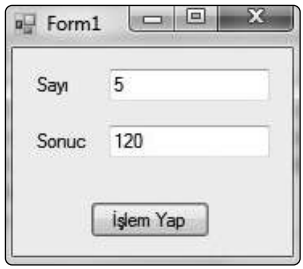
```
class matematiksel
{
    int i,sayi,faktoriyel;
    public int deger
    {
        get
        {
            faktoriyel = 1;
            for (i = 1; i <= sayi; i++)
            {
                faktoriyel = faktoriyel * i;
            }
            return faktoriyel;
        }
        set
        { 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)
{
    matematiksel nsn_matematiksel = new matematiksel();
    nsn_matematiksel.deger = Convert.ToInt32(txtSayi.Text);
    txtSonuc.Text = nsn_matematiksel.deger.ToString();
}
```



Sınıfımıza faktöriyeli alınacak sayıyı `Set` ile gönderirken, hesaplanan işlem sonucunu `Get` ile alıyoruz.

Property tanımlaması yapılırken `Read Only` ve `Write Only` olmak üzere iki biçimde tanımlanabilir.

`Read Only` property sadece değer gönderir. `Read Only` property içerisinde yalnızca `Get` bloğu kullanılabilir.

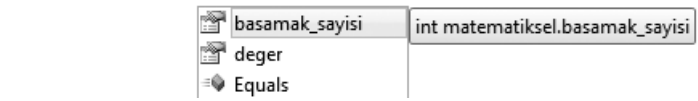
`Write Only` property sadece değer alır. `Write Only` property içerisinde yalnızca `Set` bloğu kullanılabilir.

`Matematiksel` 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 matematiksel
{
    int i,sayi,faktoriyel;
    public int deger
    {
```

```
        get
        {
            faktoriyel = 1;
            for (i = 1; i <= sayi; i++)
            {
                faktoriyel = faktoriyel * i;
            }
            return faktoriyel;
        }
        set
        { sayi = value; }
    }
    public int basamak_sayisi
    {
        get
        {
            return (faktoriyel.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)
{
    matematiksel nsn_matematiksel = new matematiksel();
    nsn_matematiksel.deger = Convert.ToInt32(txtSayi.Text);
    txtSonuc.Text = nsn_matematiksel.deger.ToString();
    lblBasamak.Text = "Basamak Sayısı="
      + nsn_matematiksel.basamak_sayisi;
}
```



Sınıfımıza faktöriyeli alınacak sayıyı `Set` ile gönderirken, hesaplanan işlem sonucunu `Get` ile alıp, faktöriyel sonucunu ve faktöriyel sonucunun basamak sayısını görüntülüyoruz.

C# 3.0 ile birlikte kullanılmaya başlanan `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 epey kısalttık.

Inheritence (Miras)

OOP (**O***bject* **O***riented* **P***rogramming, Nesne Yönelimli Programlama*)'in temel unsurlarından biri olan inheritence ile bir sınıftan başka bir sınıf türeterek bu sınıfa ait metodları kullanabiliriz. Miras alınan sınıfın özellik ve metodlar üzerinde değişiklikler yapılarak geliştirilebilir, yeni özellik ve metodları elde edilebilir.

```
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());
            }
        }

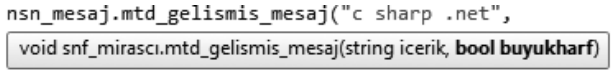
    }
}
```

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
{
    namespace Rastgele_Sayi
    {
        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;
            }
        }
        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(); ;
}
```



XML Yorum Satırı Eklemek

Oluşturduğumuz bir dll 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 metodlarla ilgili açıklama satırlarını içermektedir. Eğer bu dosya otomatik olarak oluşturulmamış ise **Project** menüsünden **Project Properties** seçeneği ile gelen pencereden **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: Metodlar 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 harici 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 metodlarda 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 sözdizimi 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;
            }
        }
    }
}
```

`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.

Metodlarımızdan 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.

```
nsn_akt_pas.mtd_pasif(
```

void snf_aktif_pasif.mtd_pasif(**params Button[] btn**)
Parametre olarak gelen buton dizisini (btn) pasif yapmak için kullanılır.
btn: *Pasif yapılacak buton dizisi. Örneğin "Button1,Button2" şeklinde*

Access Modifiers (Erişim Belirleyiciler)

Hazırladığımız class yapıları içerisindeki değişkenlere ve metodlara 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 metod ya da parametreye erişim tamamen serbesttir. Gerek tanımlandığı sınıf içerisinden gerekse tanımlandığı sınıf dışından erişmek mümkündür.

Internal: Internal belirleyicisinin kullanıldığı metod ya da parametreye sadece tanımlandığı sınıf içerisinden erişilinebilir. Hazırladığınız dll dosyasını aynı proje ya da program içerisinde kullanabilir, ancak başka proje ya da programdan erişemezsiniz.

Protected: Protected belirleyicisinin kullanıldığı metod ya da parametreye sadece tanımlandığı sınıf içerisinden veya bu sınıftan türetilmiş sınıflardan erişilinebilir.

Static: Eğer metoda direkt olarak sınıfın ismini kullanarak erişmek istersek metodu static olarak tanımlamamız gerekir. `static` olarak tanımlanan metodlar sınıflardan türetilen nesnelerden bağımsız hareket ederler. Dolayısı ile `static` metodları içerisinde tanımlandıkları class'dan yeni bir nesne oluşturmadan direkt olarak "sınıf ismi.metod ismi" şeklinde kullanabiliriz.

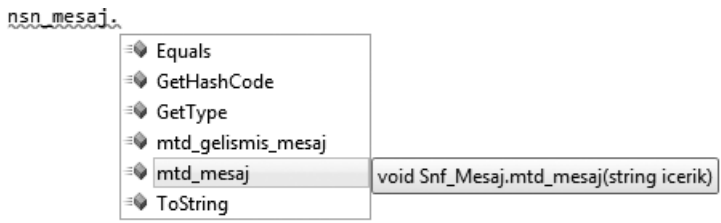
Protected internal: Protected internal belirleyicisinin kullanıldığı metod ya da parametreye tanımlandığı sınıf içerisinden, bu sınıfdan türetilmiş başka bir sınıf içerisinden ya da aynı assembly içerisinden erişilinebilir.

Private: Private belirleyicisinin kullanıldığı metod ya da parametreye sadece tanımlandığı sınıf içerisinden erişilinebilir. Sınıflar varsayılan olarak private olarak tanımlanırlar.

```
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 metodlar 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_mirascı` adlı mirasçı sınıfımızdan türettiğimiz nesneye ait metodlar arasında listelenmeyip yalnızca mirasçı sınıfımıza ait `mtd_gelismis_mesaj` isimli metodumuz listelenmektedir.



ENCAPSULATION

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.

`snf_encaps` isimli sınıfımızın kodu:

```
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.

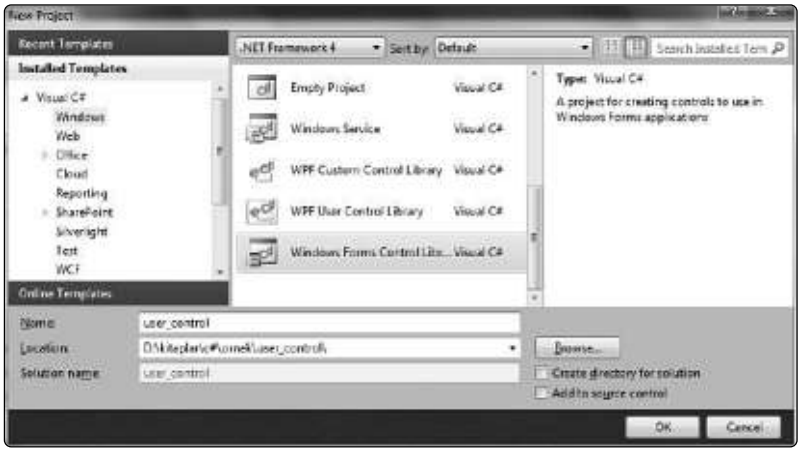
- *"Ben şu an B metodundayım"* şeklinde mesaj penceresi görüntülenerek `B` metoduna erişim sağlandığı belirtilir. Ancak `B` metodu protected olarak tanımlandığından dolayı erişim hatası meydana gelmektedir. Bu nedenle `public` olarak tanımladığımız `mtd_B_ye_KendiSinifindanErisim` isimli metod ile `B` metodunu çağırmaktayız.
- `Pi` değeri mesaj penceresi içerisinde görüntülenir. `Pi`, Read Only tanımlandığından dolayı değeri görüntülenebilir, ama değiştirilemez.

20

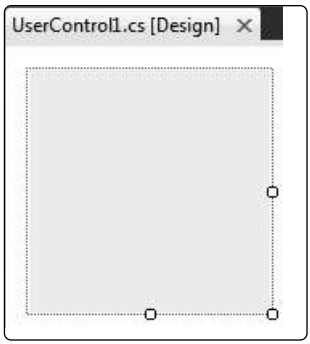
USER CONTROL HAZIRLAMA

C#, kullanıcıların hemen hemen bütün ihtiyaçlarını karşılayabilecek kontrollere sahiptir. Ancak bazen kontrollerin özellik ya da metodları 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.

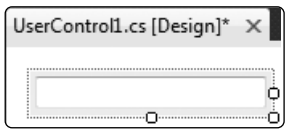
Yeni bir user control hazırlamak için **File \ New Project \ Windows Forms Control Library** seçenekleri ile yeni bir user control projesi açılır.



Karşımıza boş bir tasarım ekranı gelecektir. Bunun içerisine istediğimiz kontrolleri ekleyebiliriz.



Biz örnek olarak bir text kutusu ekleyerek özelliklerini biraz geliştireceğiz. Text kutusu ile formu aynı boyutlara gelecek biçimde ayarlıyoruz. TextBox nesnesinin ismini `metin_kutusu` biçiminde belirliyoruz.



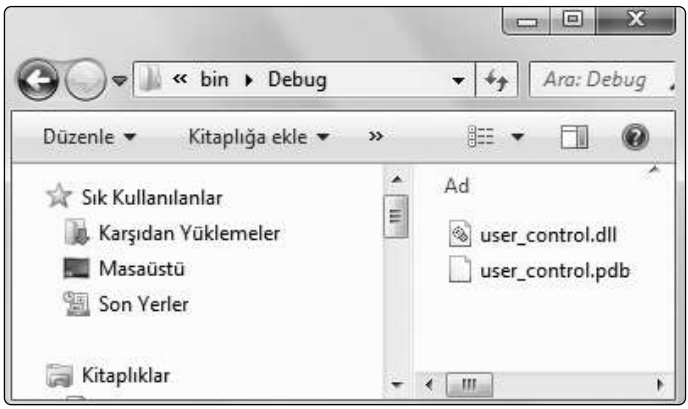
Textbox'ımızı ekledikten sonra user control'un kod görünümüne geçip aşağıdaki kod satırlarını yazıyoruz:

```
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"; }
}
```

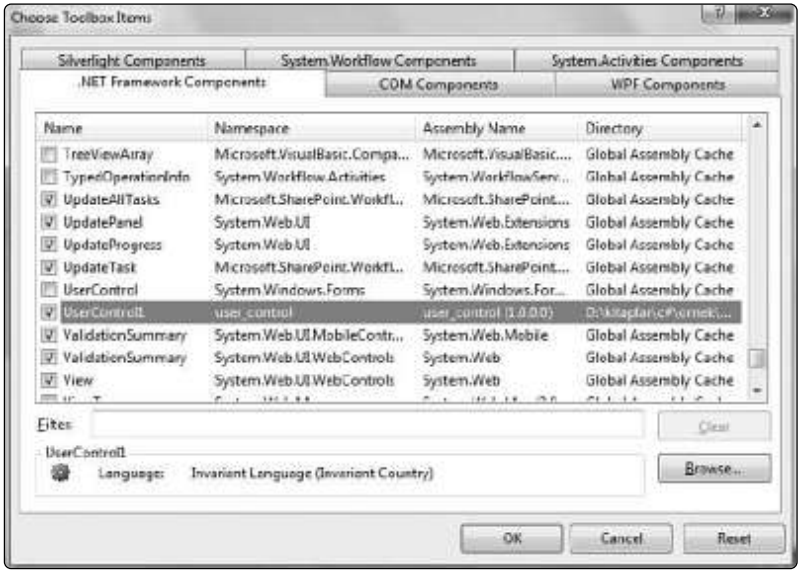
User control'a üç adet yalnız okunabilir özellik ekledik. Yalnız okunabilir özellik olduğundan dolayı `Get` bloğunu kullandık. Eğer dışarıya değer göndermek istese idik `Set` bloğunu da kullanmamız gerekirdi.

`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.

Yeni bir windows form projesi açarak user control'umuzu Toolbox içerisine ekleyeceğiz. Toolbox'a farenin sağ tuşu ile tıklayarak **Choose Items...** seçeneğini tıklıyoruz.





User control'a ait dll dosyasını seçerek toolbox içerisine eklenmesini sağlıyoruz.

Toolbox içerisine eklenen user control'u form üzerine ekleyerek aşağıdaki kod satırlarını yazıyoruz.

```
private void btnHarfSayisi_Click(object sender, EventArgs e)
{
     label1.Text = "Harf Sayısı= " + txt_metin.uzunluk.ToString();
     label2.Text = "Cümle Sayısı= " +
        txt_metin.cumle_sayısı.ToString();
}
```



User control'u yazan kişi özelliğini test etmek için ise user control seçili iken **Properties** penceresinden **yazan** isimli özelliğe bakıyoruz.

uzunluk	0
Visible	True
yazan	Volkan AKTAŞ

21

REGISTRY İŞLEMLERİ

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 konfigurasyon 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

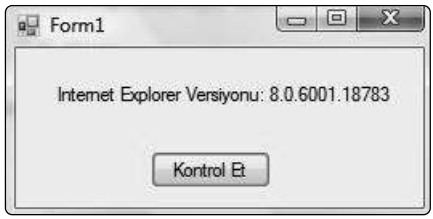
Registry'deki anahtarlara erişmek için `OpenSubKey` metodu, değerlere erişmek için ise `GetValue` metodu kullanılır.

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.

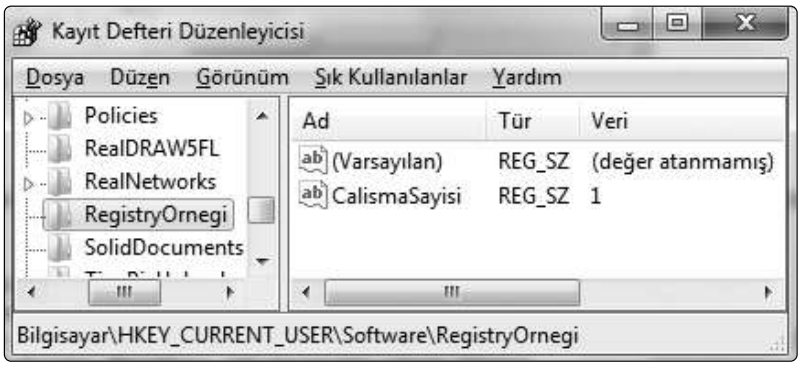
Unknown: Genişletilebilir dize değerine karşılık gelir.

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();
}
```

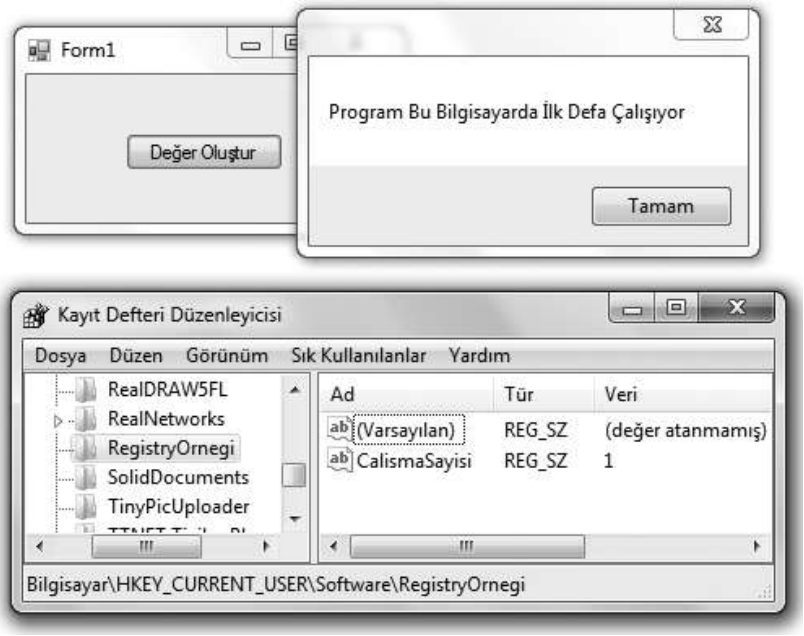
Kodumuzu ilk çalıştırdığımızda

```
    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");
```

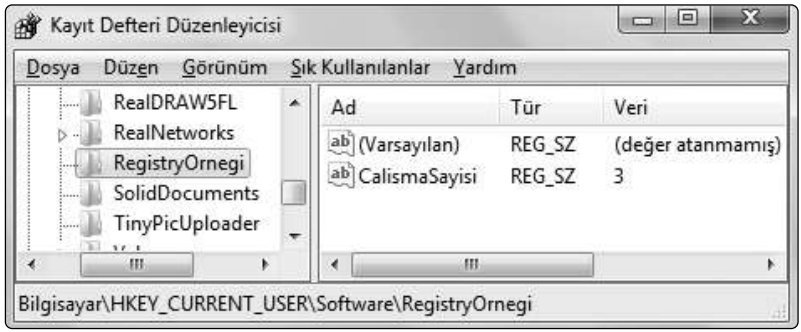
satırları ile `CreateSubKey` metodu kullanılarak aşağıda görüldüğü gibi `RegistryOrnegi` adlı anahtar oluşturulur ve `SetValue` metodu ile bu anahtar altında ilk değer olarak `1` değeri girilmekte, *"Program Bu Bilgisayarda İlk Defa Çalışıyor"* mesajı görüntülenmektedir.



Bundan sonraki çalıştırmalarımızda ise

```
byte tekrar=Convert.ToByte(anahtar.
            OpenSubKey("RegistryOrnegi", true).GetValue
            ("CalismaSayisi").ToString());
anahtar.OpenSubKey("RegistryOrnegi", true).SetValue
      ("CalismaSayisi", tekrar+1,RegistryValueKind.String);
```

`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` metodu kullanılırken bir anahtar içerisinde mevcut olan değerlerin sayısını bulmak için ise `ValueCount` metodu kullanılır.

Örneğin bir önceki uygulamamızda yaptığımız, bir değerin var olup olmadığını `GetValue` metodu ile kontrol etmek yerine `ValueCount` metodundan faydalanarak `RegistryOrnegi` adlı anahtarın altındaki değerlerin sayısını da kontrol edebiliriz. Örneğimizi `ValueCount` metodunu 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();
}
```

Anahtar ve Değer Listesini Alma

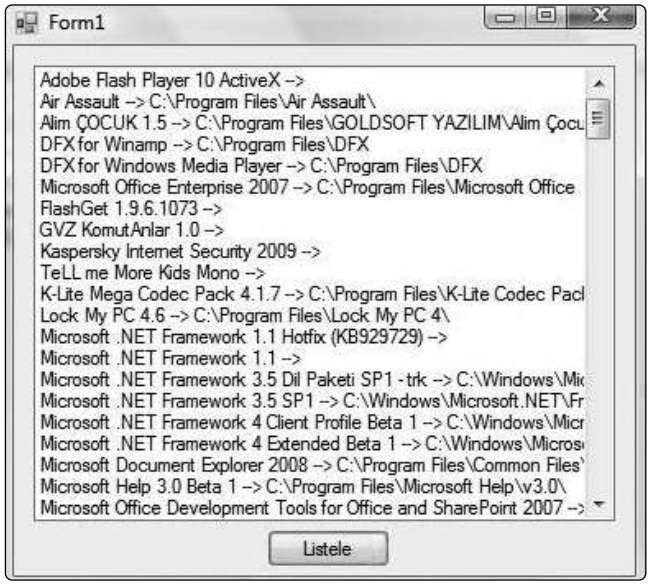
`GetSubKeyNames` metodu ile alt anahtarların listesini dizi şeklinde elde ederken, `GetValueNames` metodu ile de bir anahtarın altındaki değerlerin listesini bir dizi şeklinde elde edebiliriz.

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 makinasını 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 etmemiz 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` ya da `DeleteSubKey`, `DeleteSubKeyTree` metodlarının 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( )
```

satırında ise `Close` metodu yardımı ile kullandığımız registry anahtarını kapattık.

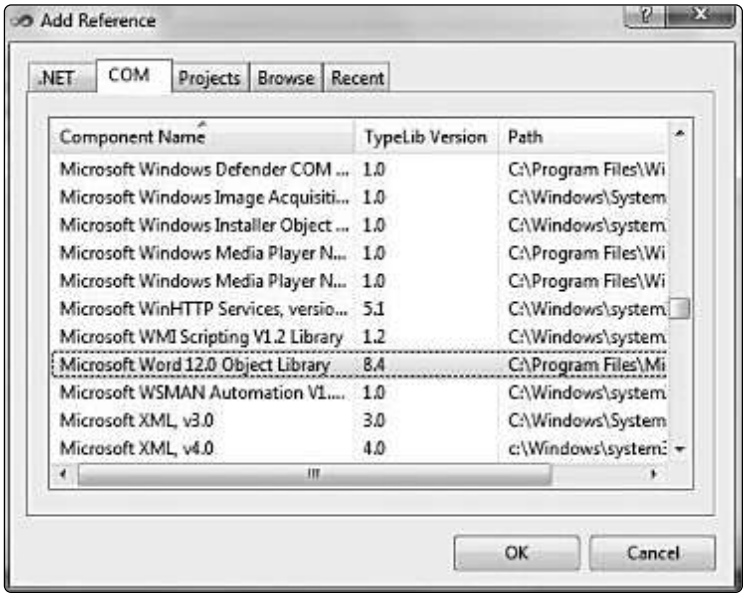
22

WORD VE EXCEL PROGRAMLARINI KULLANMA

C# .Net veya diğer .NET ailesi ile ofis uygulamaları geliştirmenin başlıca iki yöntemi vardır. Birincisi **VBA** (**V***isual* **B***asic* f*or* **A***pplication*) yapısını kullanmak, ikinci yöntem ise **VSTO** (**V***isual* **S***tudio* **T***ools for* **O***ffice*) isimli ofis yazılımı geliştirme ortamını kullanmaktır.

VBA (**V***isual* **B***asic* f*or* **A***pplication*) 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.

VBA (**V***isual* **B***asic* f*or* **A***pplication*) yapısını kullanarak program yazarken Framework 4.0 öncesinde ofis uygulamaları yazarken kullanacağımız metodlara ait tüm parametreleri yazmak zorunda iken, framework 4.0 sonrasında optional parameters ile yalnızca kullanacağımız parametreleri yazmamız yeterli olmaktadır. Aradaki farkı görmek için bir uygulama yapalım:



Bir **Windows Application** açıyoruz. Daha sonra **Project** menüsünden **Add Reference** seçeneği ile gelen pencereden **COM** başlığı altından **Microsoft Word 12.0 Object Library** seçeneği seçilip **OK** tuşuna basıyoruz. Elbette kullandığınız Word programının sürümüne göre buradaki 12 rakamı değişecektir.

```
using Word = Microsoft.Office.Interop.Word;
```

satırı ile referansımızı projemize dahil ediyoruz.

Daha sonra form üzerine bir komut butonu ekleyerek text özelliğini *"yeni belge oluştur"* şeklinde değiştiriyoruz.

Formumuzun tanımlama kısmında `WordApp` ve `WordDoc` nesnelerimizi tanımlıyoruz. `WordApp` word uygulaması, `WordDoc` ise word belgesidir.

Sonuç olarak kodumuz aşağıdaki gibi olacak:

```
using System;
using System.Windows.Forms;
using Word = Microsoft.Office.Interop.Word;
namespace word_framework2
```

```
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void btnYeniBelge_Click(object sender,
        EventArgs e)
        {
            object missing = System.Reflection.Missing.Value;
            Word.Application WordApp = new Word.Application();
            Word.Document WordDoc = new Word.Document();
            WordDoc = WordApp.Documents.Add(ref missing,
                    ref missing,ref missing, ref missing);
            WordApp.Visible = true;
        }
    }
}
```

Örneğimizde görüldüğü gibi `Documents.Add` metodunun kulanmadığımız parametreleri için bu parametrelere `System.Reflection.Missing.Value` değerini gönderiyoruz.

Aynı kodu **Framework 4.0** ile yazacak olursak;

```
using System;
using System.Windows.Forms;
using Word = Microsoft.Office.Interop.Word;
namespace word_framework4
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void btnYeniBelge_Click(object sender,
        EventArgs e)
        {
            Word.Application WordApp = new Word.Application();
            Word.Document WordDoc = new Word.Document();
            WordDoc = WordApp.Documents.Add();
```

```
                WordApp.Visible = true;
            }
        }
    }
```

Görüldüğü gibi kodumuzda optional parameters kullanımı ile kullanmadığımız parametreleri yazmadık. Bu da bizim yükümüzü çok azalttı.

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.



Bundan sonraki Word ve Excel uygulamalarımızı **Framework 4.0** versiyonuna göre yazacağız.

Word Programını 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 command buton ekleyelim. Birinci buton word belgemizi açmak ikincisi ise kapatmak için olsun ve şu kodları formumuza yazalım:

```
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.

Kodumuz içerisinde

`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` metodları kullanılır.

Ayrıca `Close` metodunun parametreleri kullanılarak kapatmadan önce belgede değişiklik yapıldığında ekrana çıkan farklı kaydet penceresi ekrana gelmeden kayıt işlemi yapılmadan belgenin kapatılması sağlanabilir ya da kayıt işleminin yapılması için farklı kaydet penceresi görüntülenerek kapatma işlemi yapılır.

`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.

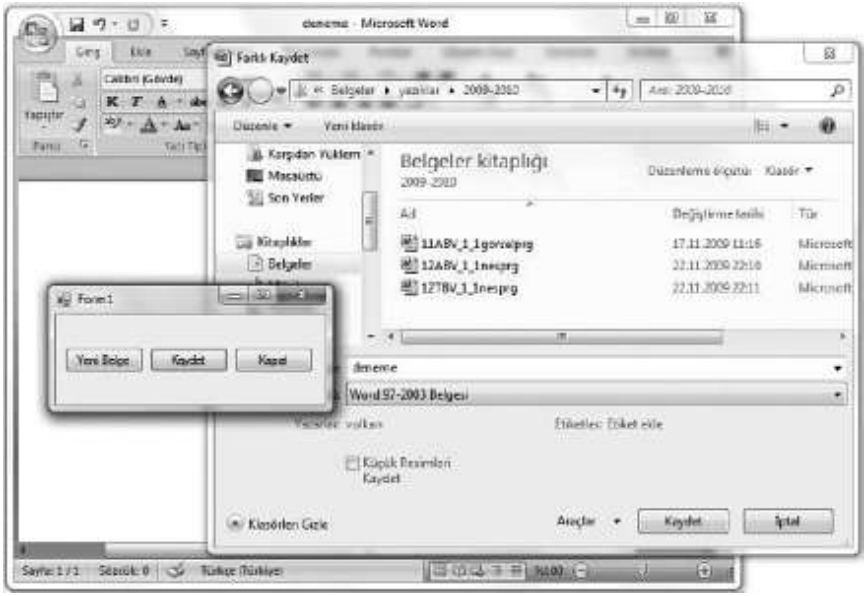
Form üzerine `btnYeniBelge`, `btnKaydet`, `btnKapat` adı ile 3 adet buton ekliyoruz. Kodumuz aşağıdaki gibi olacak:

```
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 =
"Word Belgeleri (*.doc)|*.doc|Zengin Metin Biçimi (*.rtf)|*.rtf";
            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;
```

```
            }
        }
    }
}
```

Yazı Ekleme ve Düzenleme İşlemleri

Yazı ekleme işlemi birden fazla şekilde yapılabilir.

Range Nesnesi Kullanarak

`Range` nesnesi belirtilen aralıklarda işlem yapmak amacı ile kullanılır. `Range` nesnesi ile iki metod kullanılabilir. Bunlar: `InsertBefore`, `InsertAfter` metodlarıdır.

`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 text 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.

NOT

Bundan sonraki Word örneklerimizdeki

```
using System;
using System.Windows.Forms;
using Word = Microsoft.Office.Interop.Word;
```

satırlarını gereksiz tekrarlamamak için yazmayacağım. Ancak okuyucularım her word uygulamasına bunları yazmayı unutmasınlar.

SENTENCES NESNESİ KULLANARAK

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

```
int cumlesayisi = WordDoc.Sentences.Count;
```

komut satırı ile cümle sayısını tespit ederek bunu cumlesayisi adlı değişkene aktardık.

Eğer cümle sayısı `0`'dan büyük ise yani belge içerisinde mevcut cümle varsa;

```
WordDoc.Sentences[cumlesayisi].Text = textBox1.Text;
```

komut satırı çalıştırılarak belge içerisindeki son cümle `textbox1` içerisine yazılan cümle ile değiştirilmektedir.

Eğer belge içerisinde mevcut cümle yoksa;

```
WordDoc.Sentences[1].Text = textBox1.Text;
```

komut satırı ile belge içerisine ilk cümle olarak textbox1 içerisindeki cümleyi aktardık.

Paragraph Nesnesi Kullanarak

`Paragraph` nesnesi belge içerisindeki paragrafları temsil etmektedir. Range metodu yardımı ile kullanılabilir.

`Paragraph` 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)
{
    openFileDialog1.Title = "Belge seçiniz...";
    openFileDialog1.Filter =
```

```
        "Word Belgeleri (*.doc)|*.doc|Zengin Metin Biçimi ↓
        (*.rtf)|*.rtf";
    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

```
int paragrafsayisi = WordDoc.Paragraphs.Count;
```

komut satırı ile paragraf sayısını tespit ederek bunu paragrafsayisi adlı değişkene aktardık.

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;

```
WordDoc.Paragraphs[1].Range.Text = textBox1.Text;
```

komut satırı ile belge içerisine ilk paragraf olarak `textbox1` içerisindeki cümleyi aktardık.

Selection Nesnesi Kullanarak

`Selection` nesnesi belgedeki yazının seçili olan kısmını temsil eder. `TypeText` metodu ile birlikte kullanılarak yazı ekleme işlemi yapılabilir. `Selection` 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)
{
    openFileDialog1.Title = "Belge seçiniz...";
    openFileDialog1.Filter =
       "Word Belgeleri (*.doc)|*.doc|Zengin Metin Biçimi ↓
       (*.rtf)|*.rtf";
    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);
    }
}
```

`Selction` metodunun `TypeText` nesnesi ile metin üzerinde işlem yapabilirken `TypeParagraph` nesnesi ile parağraf 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:

```
using System;
using System.Windows.Forms;
using Word = Microsoft.Office.Interop.Word;
namespace word_onizleme_yazdirma
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        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;

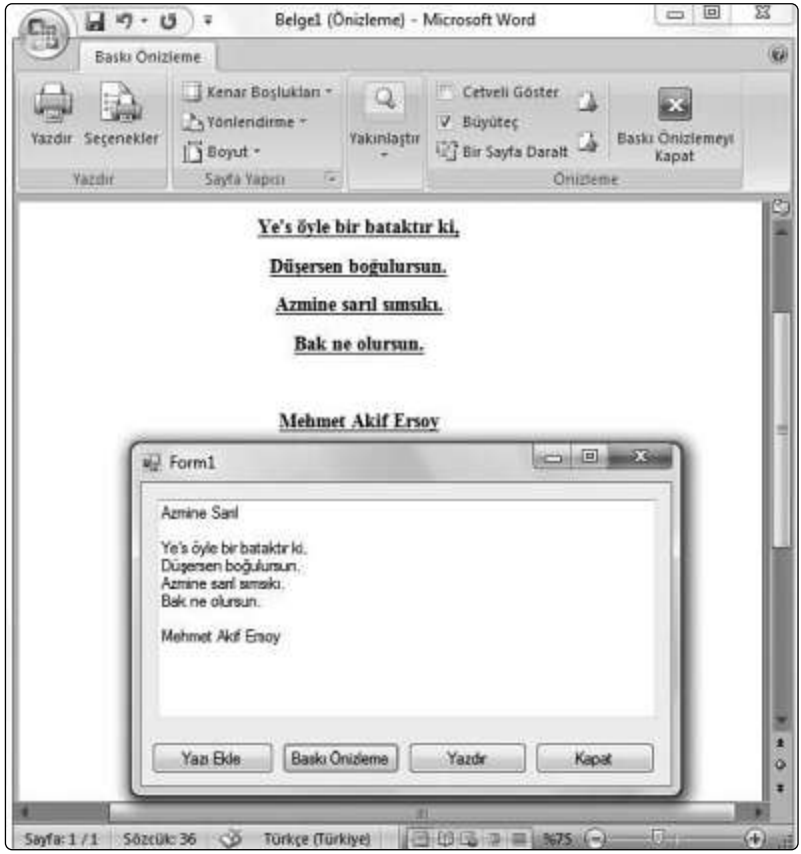
                WordApp.Selection.Font.Name =
                  "Times New Roman";
                WordApp.Selection.Font.Size = 14;
                WordApp.Selection.Font.Underline =
                    Word.WdUnderline.wdUnderlineSingle;
                WordApp.Selection.BoldRun();

                //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:



Örneğimizde

```
WordDoc.Paragraphs[1].Alignment =
                Word.WdParagraphAlignment.wdAlignParagraphCenter;
```

satırında `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.
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.
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

`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.

`Dökümannesnesi.Range.Tables(tablo numarası).Columns(sütun numarası)` ile tablo içerisindeki sütunlara erişilir.

`Dökümannesnesi.Range.Tables(tablo numarası).Rows(satır numarası)` ile tablo içerisindeki satırlara erişilir.

Dökümannesnesi.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;
  //6 satır 2 sütun genişliğinde tablo oluşturuyoruz
  WordDoc.Tables.Add(WordApp.Selection.Range, 6, 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("İl");
  WordDoc.Range().Tables[1].Cell(1, 2).Range.InsertAfter("İlçe");
  WordDoc.Range().Tables[1].Cell(2, 1).Range.InsertAfter("Diyarbakır");
  WordDoc.Range().Tables[1].Cell(2, 2).Range.InsertAfter("Ergani");
  WordDoc.Range().Tables[1].Cell(3, 1).Range.InsertAfter("Mersin");
  WordDoc.Range().Tables[1].Cell(3, 2).Range.InsertAfter("Merkez");
  WordDoc.Range().Tables[1].Cell(4, 1).Range.InsertAfter("Ankara");
  WordDoc.Range().Tables[1].Cell(4, 2).Range.InsertAfter("Merkez");
```

```
   WordDoc.Range().Tables[1].Cell(5, 1).Range.InsertAfter("Şanlıurfa");
   WordDoc.Range().Tables[1].Cell(5, 2).Range.InsertAfter("Merkez");
   WordDoc.Range().Tables[1].Cell(6, 1).Range.InsertAfter("Van");
   WordDoc.Range().Tables[1].Cell(6, 2).Range.InsertAfter("Özalp");
}
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 yandaki gibi bir tablo oluşturulacaktır.

İl	İlçe
Diyarbakır	Ergani
Mersin	Merkez
Ankara	Merkez
Şanlıurfa	Merkez
Van	Özalp

`SetWidth` özelliği bir tablodaki sütun genişliklerini ayarlamak için kullanılan metoddur ve kullanımı şu şekildedir:

```
ifade.SetWidth(Sütun genişliği, stil)
```

ifade 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              wdGreen
wdBlack             wdNoHighlight
wdBlue              wdPink
wdBrightGreen       wdRed
wdByAuthor          wdTeal
wdDarkBlue          wdTurquoise
wdDarkRed           wdViolet
wdDarkYellow        wdWhite
wdGray25            wdYellow
wdGray50
```

```
.Range().Tables[1].Cell(1, 1).Range.Paragraphs.Alignment
```

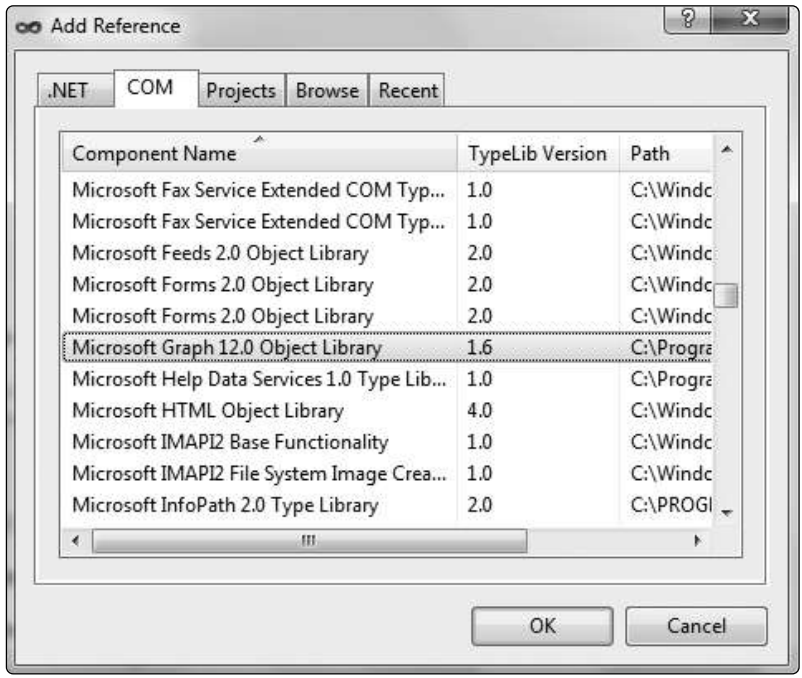
satırı ile tablomuzdaki hücrelerin hizalama şeklini belirledik. Herhangi bir yazı ya da paraqrafı hizalamak için de `Alignment` ifadesi kullanılabilir. Hizalama biçimi olarak şu seçenekler kullanılabilir:

```
wdAlignParagraphCenter
wdAlignParagraphDistribute
```

```
wdAlignParagraphJustify
wdAlignParagraphJustifyHi
wdAlignParagraphJustifyLow
wdAlignParagraphJustifyMed
wdAlignParagraphLeft
wdAlignParagraphRight
```

Grafik İşlemleri

Genellikle veritabanı 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 12.0 Object Library"* referansına ek olarak *"Microsoft Graph 12.0 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 buton 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();

            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();
            }

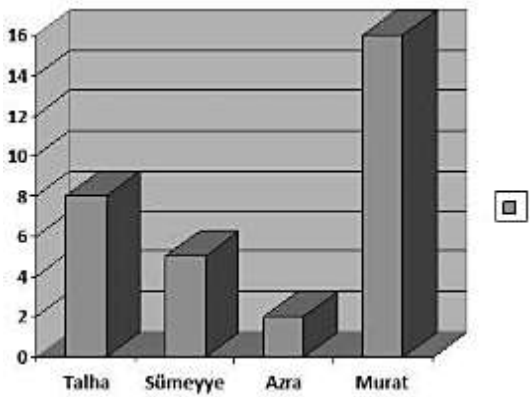
            Shape.Width = WordApp.InchesToPoints(4.25f);
            Shape.Height = WordApp.InchesToPoints(3.25f);

            objChart.Application.Update();
            oChartApp.GetType().InvokeMember("Update",
                BindingFlags.InvokeMethod, null,
                oChartApp, null);
            oChartApp.GetType().InvokeMember("Quit",
                BindingFlags.InvokeMethod, null,
                oChartApp, null);
        }
    }
}
```

Programımızın ekran görüntüleri yandaki gibi:



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");
```

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 metodlarını 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 metod 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;
```

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. Listboxların eleman sayıları eşit olmadığı takdirde iki 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.

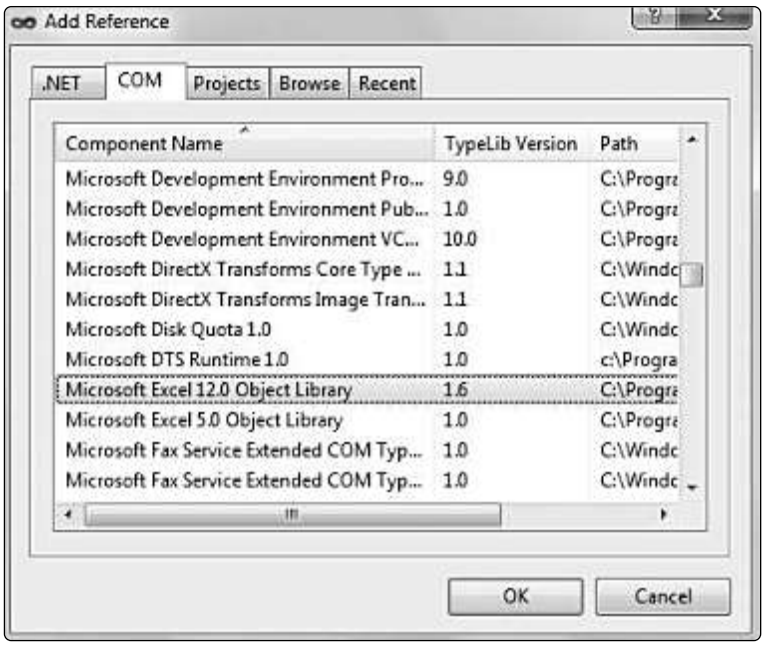
Eğer daha önceden mevcut olan bir Excel çalışma sayfasını açacaksak `Workbooks.Open(dosya adı)` metodu kullanılır.

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 12.0 Object Library**

seçeneği seçilip **OK** tuşuna basıyoruz. Elbette kullandığınız Excel programının sürümüne göre buradaki 12 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.

Kodumuzda;

```
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 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

```
ExcelApp.Workbooks.Open(openFileDialog1.FileName);
```

satırında `Workbooks` nesnesinin `Open` metodu ile dosya açma diyalog penceresinde seçilen Excel dosyası açılıyor.

Kaydetme olayı için `Save`, farklı biçimde kaydetmek için `GetSaveAsFilename` metodları kullanılır.

`Save`, `GetSaveAsFilename` metodları ile belge kaydetme ve farklı kaydetme işlemleri ve çalışma sayfasını ve Excel uygulamasını kapatma ile ilgili örnek yapalım. Bu olay ve metodlar Word konusunda detaylıca anlatıldığı için ayrıntılara girmiyorum.

Form üzerinde `btnYeni`, `btnKapat`, `btnKaydet`, `btnFarkliKaydet` adları ile 4 adet buton olacak.

Kodumuz ise aşağıdaki şekilde olacak:

```
using System;
using System.Windows.Forms;
using Excel = Microsoft.Office.Interop.Excel;
namespace excel_kayıt
{
    public partial class Form1 : Form
    {
        public Form1()
        {
```

```
        InitializeComponent();
    }
    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.

NOT

Bundan sonraki Excel örneklerimizdeki

```
using System;
using System.Windows.Forms;
using Excel = Microsoft.Office.Interop.Excel;
```

satırlarını gereksiz tekrarlamamak için yazmayacağım. Ancak okuyucularım her excel uygulamasına bunları yazmayı unutmasınlar.

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 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 komut butonu ekliyoruz ve kod olarak da 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;
```

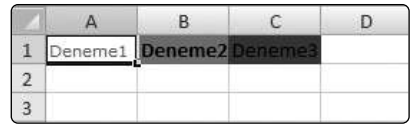
satırında geçen `Interior.ColorIndex` ifadesi ile ise `B1` hücresinin zemin rengini kırmızı renk olarak değiştiriyoruz. Bunu ister o renge karşılık gelen rakam ile istersek `RGB` değerleri ile yapabiliriz.

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 metodlarını kullanmamız da mümkündür.

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

Önizleme için `PrintPreview` metodu kullanılırken, yazdırma işlemi için `PrintOut` metodu kullanılır.

Bir çalışma kitabını içerisindeki çalışma sayfalarını yazdırmak ve baskı ön izleme yapmak için şu metodlar kullanılır:

```
Excelnesnesi. Worksheets["Çalışma Sayfası Adı "]. printpreview()
Excelnesnesi.ActiveSheet. printout()
```

Bu metodlar ile baskı ön izleme ve yazdırma işlemleri yapabileceğimiz gibi, sayfa içerisinde belli bir bölümü

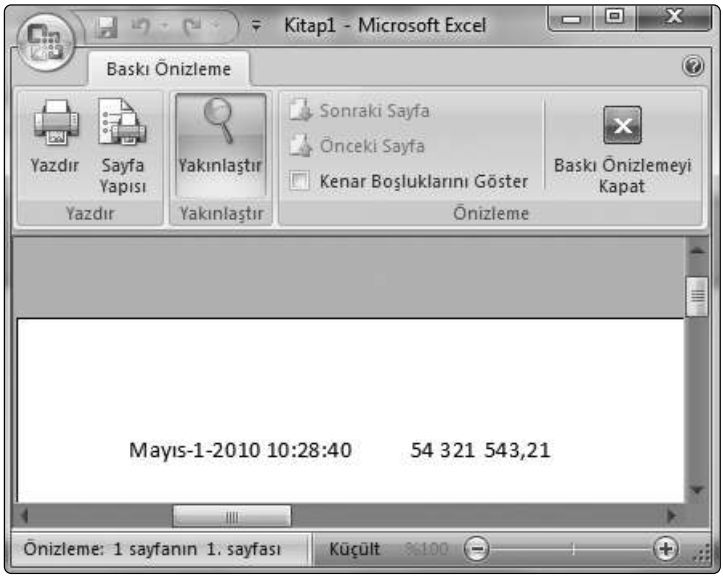
```
Excelnesnesi.Worksheets[Çalışma Sayfası Numarası].Range[“sütun
satır”].PrintPreview
Excelnesnesi.Worksheets[Çalışma Sayfası Numarası].Range[“sütun
satır”].PrintOut
```

metodları 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 2 adet `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.

NOT

Excel uygulamalarını yazarken C# 4 ile gelen özelliklerden olan **optional parameters** yanında **named parameters** (*isimlendirilmiş parametreler*) özelliğinden de faydalanacağız.

Örneğin;

```
ExcelApp.Sheets.PrintOut(1, 2, 3);
```

satırında kullanılan parametreler sırası ile olduğundan dolayı `optional parameters`'i kullanıyoruz. Ancak en sondaki parametre olan `Collate` parametresini kullanmak için kullanılmayan diğer parametreleri tek tek yazmayıp, yalnızca kullanacağımız parametreyi yazmak için **named parameters** özelliğini kullanmamız gerekmektedir.

`Collate` parametresini kullanmak için;

C# 4 öncesi versiyonlarda;

```
ExcelApp.Sheets.PrintOut(1, 2, 3, System.Reflection.Missing.Value,
System.Reflection.Missing.Value, System.Reflection.Missing.Value,
Collate: true);
```

biçiminde,

C# 4 named parameters ile;

```
ExcelApp.Sheets.PrintOut(1, 2, 3, Collate:true);
```

biçiminde yazabiliriz.

Bu satır ile komut satırı ile 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. Ben bir excel tablosu hazırladım. Bu tablo içerisinde öğrencilerin isim numara ve 3 adet notları yer alıyor. 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 komut butonu 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;

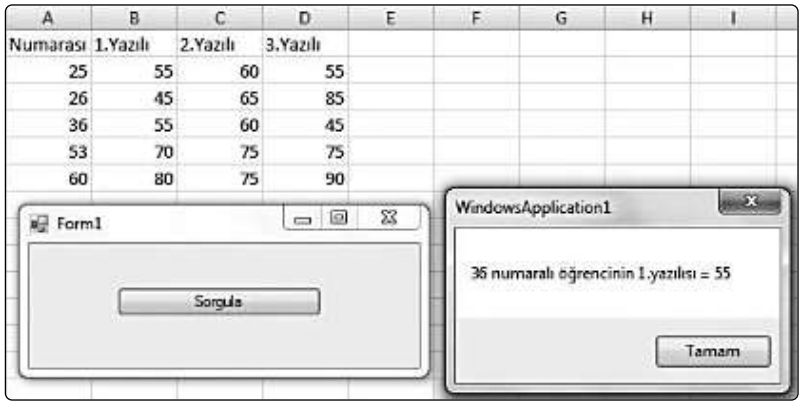
         sınıf = Interaction.InputBox("Lütfen sınıfı giriniz");
         aranan_dosya = File.Exists(Application.StartupPath +
               "\\" + sınıf + ".xls");

         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("1.yazılı = " +
                            ExcelApp.Cells
                            [bulunan.Row, bulunan.Column +
                            1].value);
                          break;
                        }
                        case 2:
                        {
                          MessageBox.Show("2.yazılı = " +
                            ExcelApp.Cells
                            [bulunan.Row, bulunan.Column +
                            2].value);
                          break;
                        }
                        case 3:
                        {
                          MessageBox.Show("3.yazılı = " +
                            ExcelApp.Cells
                            [bulunan.Row, bulunan.Column +
                            3].value);
                          break;
                        }
                    }
                }
                ExcelApp.Application.Quit();
            }
        }
    }
}
```



Örneğimizde D sürücüsünde sınıf isimleri ile yer alan Excel dosyaları aranarak ilgili sınıfın Excel dosyası bulunup, Excel dosyası içerisinde belirtilen numara ve kaçıncı yazılı üzerinde arama yapılacağı bilgilerine göre yazılı sonucu mesaj penceresi içerisinde görüntülenmiştir.

Grafik İşlemleri

Grafik işlemleri için Word'de olduğu gibi **Chart** adı verilen grafik öğeleri kullanılabilir. Grafik öğelerini kullanabilmek için MSGraph nesnesini projemize ekleyeceğiz.

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 buton ekleyelim.

Kodumuz aşağıdaki gibi olacak:

```
using System;
using System.Windows.Forms;
using Excel = Microsoft.Office.Interop.Excel;
using Microsoft.Office.Interop.Excel;
namespace excel_grafik
{
  public partial class Form1 : Form
  {
      public Form1()
      {
          InitializeComponent();
      }
```

```
        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;

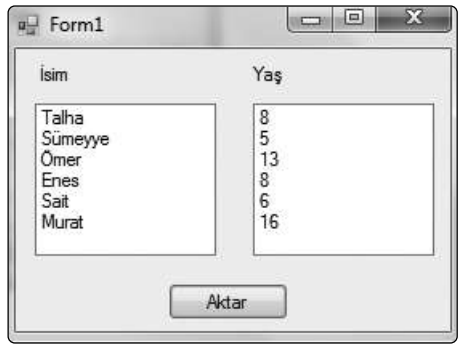
            ExcelApp.Worksheets[1].Activate();

            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];
            }

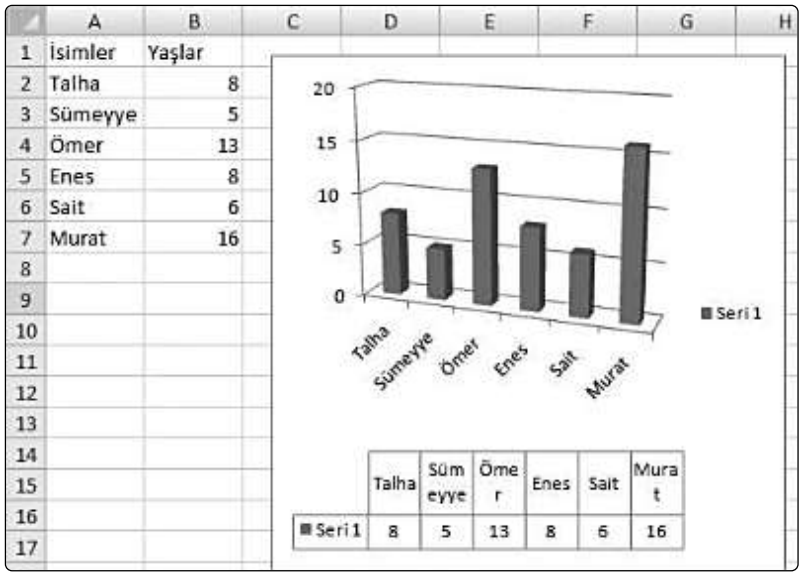
            Excel.Worksheet xlWorkSheet=
                (Excel.Worksheet)ExcelApp.ActiveSheet;
            Excel.ChartObjects xlCharts =
                (Excel.ChartObjects)xlWorkSheet.ChartObjects();
            Excel.ChartObject Chart =
                (Excel.ChartObject)xlCharts.Add(110, 10, 250, 250);
            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 yandaki gibi:



Formumuzu çalıştırıp aktar butonuna bastığımızda listbox içerisinde bulunan değerler 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);
```

cümlesi 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);
```

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;
```

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 listboxlarda mevcut olan değerleri Excel içerisindeki `1` ve `2` nolu sütunlara sırası ile yazdırıyoruz.

Ayrıca;

```
ExcelApp.Cells[1, 1].value = "İsimler";
ExcelApp.Cells[1, 2].value = "Yaşlar";
```

satırlarında `1` ve `2` nolu 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.

23

WINDOWS API'LERİNİ KULLANMA

Windows API'leri Windows işletim sisteminin parçası olan Dll kütüphaneleridir. .NET yardımı ile yapmak istediğimiz işlemlere ait api'leri kullanabiliriz. .NET altındaki apilerin tüm listesini elde edebilmek için herhangi bir program yok. Ancak, *http://www.pinvoke.net* adresinde hemen hemen bütün apiler ve tanımlamaları hatta kullanımları bile mevcut.

Örneğin ses çalma ile ilgili apiyi kullanmak için;

http://www.pinvoke.net/winmm.PlaySound adresinden

winmm.dll dosyasını seçip onunla ilgili tanımlamaları kullanabiliriz.

Kodlama aşamasında ise C# .NET içerisinde bir apiyi kullanmak için `DllImport` fonksiyonundan yararlanırız.

Bu fonksiyonu kullanmak için;

```
using System.Runtime.InteropServices;
```

şeklinde `System.Runtime.InteropServices` sınıfını projemize dahil etmeliyiz.

Disk Toplam Alanini ve Boş Alanini 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);

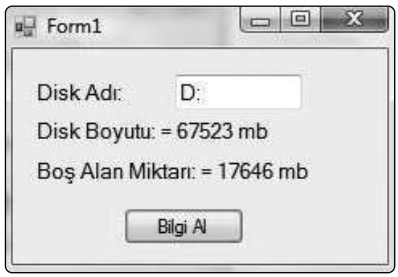
        private void btnBilgiAl_Click(object sender, EventArgs e)
        {
            ulong toplamalan;
            ulong ulasilanbosalan;
            ulong bosalan;
            double alan;

            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 sonuc değişkeni false değerinden farklı bir değer içeriyorsa işlem başarı ile gerçekleştirilmiş demektir.

NOT

Bundan sonraki Api örneklerimizdeki

```
using System;
using System.Runtime.InteropServices;
using System.Windows.Forms;
```

satırlarını gereksiz tekrarlamamak için yazmayacağım. Ancak okuyucularım her api uygulamasına bunları yazmayı unutmasınlar.

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 metod 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 metod 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();
    }
```



RAM Bellek Hakkında Bilgi Almak

Ram bellek hakkında bilgi almak için **kernel32.dll** dosyasını kullanabiliriz. Bu işlem için bu dll dosyasının `GlobalMemoryStatusEx` metodunu kullanacağız.

NOT

Bellek miktarı **4 gb**'dan büyük olan sistemlerde **`GlobalMemoryStatusEx`** metodu **4 gb**'dan küçük olan sistemlerde ise GlobalMemoryStatus metodu kullanılabilir. Ayrıca **`GlobalMemoryStatusEx`** metodu Windows 2000 ve sonrası işletim sistemlerince desteklenirken, **`GlobalMemoryStatus`** metodu ise Windows 2000 öncesi işletim sistemlerince desteklenir.

`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** (*Takas bellek dosyası*) miktarını belirtir.
- **ullTotalVirtual:** Kullanılan toplam sanal bellek miktarını byte cinsinden belirtir.
- **ullAvailVirtual:** Boş sanal bellek miktarını byte cinsinden belirtir.
- **ullAvailExtendedVirtual:** Daima `0` değerini alır.

Tanımlama satırlarını ise aşağıdaki gibi yazmalıyız.

```
MEMORYSTATUSEX hafıza = new MEMORYSTATUSEX();
hafıza.dwLength = 64;
GlobalMemoryStatusEx(hafıza);
```

NOT

Yukarıda belirtilen api dosyasını çağırma ve tanımlama satırları Windows 7 öncesi işletim sistemi versiyonlarına göredir. Windows 7'de ise bu satırları aşağıdaki gibi değiştirmeliyiz.

Api dosyasını çağırma satırı aşağıdaki gibi değişecek:

```
[DllImport("kernel32.dll")]
public static extern void GlobalMemoryStatusEx(ref
MEMORYSTATUSEX hafıza);
```

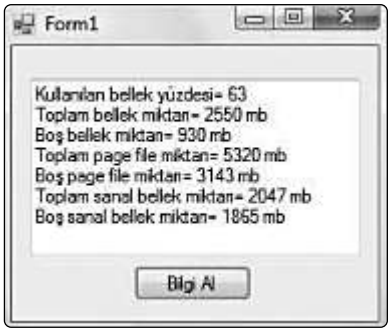
Tanımlama satırları ise aşağıdaki gibi değişecek:

```
MEMORYSTATUSEX hafıza = new MEMORYSTATUSEX();
hafıza.dwLength = 64;
GlobalMemoryStatusEx(ref hafıza);
```

```
//Windows 7 öncesi versiyonlar için
//[return: MarshalAs(UnmanagedType.Bool)]
//[DllImport("kernel32.dll", CharSet = CharSet.Auto,
//         SetLastError = true)]
//static extern bool GlobalMemoryStatusEx([In, Out]
//         MEMORYSTATUSEX lpBuffer);
[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;
     //Windows 7 öncesi versiyonlar için
     //GlobalMemoryStatusEx(hafıza);
     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 RESMİNİ DEĞİŞTİRMEK

Masaüstü resmini değiştirmek için **user32.dll** dosyasını kullanabiliriz. Bu dll dosyasının `SystemParametersInfo` metodundan faydalanacağız.

`SystemParametersInfo` metodunun kullanımı şu şekildedir:

```
[DllImport("user32.dll", CharSet = CharSet.Auto)]
private static extern Int32 SystemParametersInfo(UInt32 ↓
  uiAction, UInt32 uiParam, String pvParam, UInt32 fWinIni);
```

NOT

Vista öncesi versiyonlarda masaüstü resmi olarak yalnızca **BMP** formatı kullanılmalıdır. **Vista ve sonrası** versiyonlarda ise **JPEG** ve **GIF** gibi resim formatları da kullanılabilir.

```
[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:\\Lighthouse.jpg";
```

```
private void btnMasaustunuDegistir_Click(object sender,
EventArgs e)
{
    SystemParametersInfo(SPI_SETDESKWALLPAPER, 0,
        dosyaadi, SPIF_UPDATEINIFILE);
}
```

Ekran Çözünürlüğünü Bulma ve Değiştirmek

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.

`EnumDisplaySettings` metodunun 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,
```

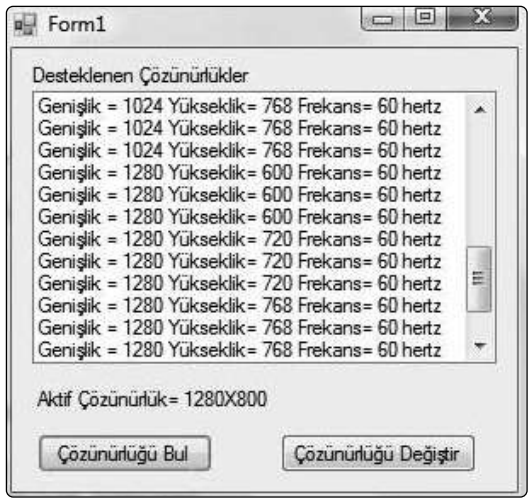
```
    ICMIntent = 0x1000000,
    MediaType = 0x2000000,
    DitherType = 0x4000000,
    PanningWidth = 0x8000000,
    PanningHeight = 0x10000000,
    DisplayFixedOutput = 0x20000000
}
public enum DISP_CHANGE : int
{
    Successful = 0,
    Restart = 1,
    Failed = -1,
    BadMode = -2,
    NotUpdated = -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` değerleri ile frekans değerlerini ise `dmDisplayFrequency` değeri ile elde edip listbox içerisine ekliyoruz.

Aktif ekran çözünürlük değerlerini `Screen` nesnesinin `Bounds.Width` ve `Bounds.Height` metodları ile elde edip görüntülüyoruz.

Ekran çözünürlüğünü değiştirmek için ise `ChangeDisplaySettings` metodunu;

```
ChangeDisplaySettings(ref vDevMode, 0)
```

biçiminde kullanarak seçilen grafik modu dinamik olarak değiştirilir.

`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. Çünki sistemde çift ekran etkin.

BadFlags: Geçersiz çözünürlük değiştirme biçimi.

BadMode: Seçilen grafik modu desteklenmiyor. (The graphics mode is not supported.)

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 Kullanmak

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 metod 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.

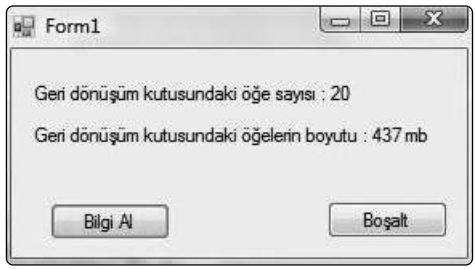
- **SHERB_NOCONFIRMATION:** Geri dönüşüm kutusunun boşaltılması sırasında herhangi bir diyalog penceresi görüntülenmez.
- **SHERB_NOPROGRESSUI:** Geri dönüşüm kutusunun boşaltılması sırasında **ilerleme çubuğu** (*progress bar*) görüntülenmez.
- **SHERB_NOSOUND:** İşlem tamamlandığında herhangi bir ses 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;
}

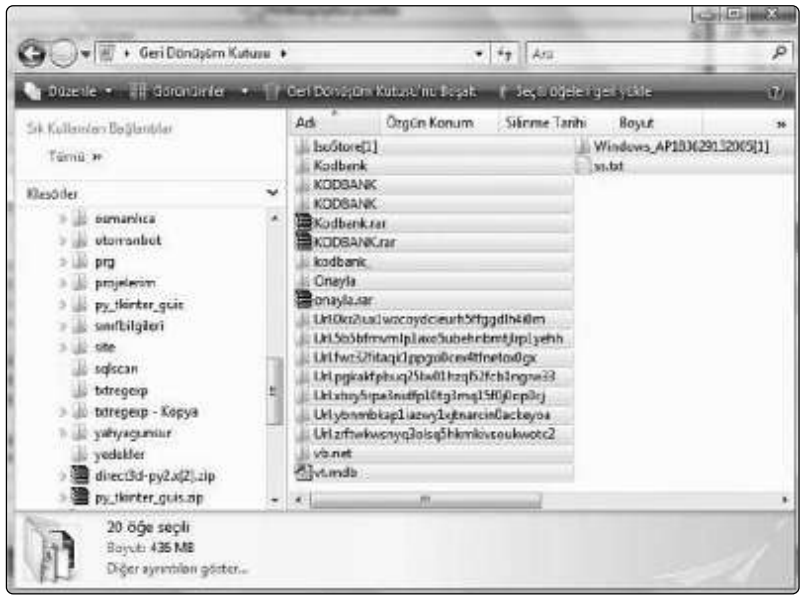
[DllImport("shell32.dll")]
static extern int SHEmptyRecycleBin(IntPtr hWnd, string
pszRootPath, uint dwFlags);
const int SHERB_NOCONFIRMATION = 0x00000001;
const int SHERB_NOPROGRESSUI = 0x00000002;
const int SHERB_NOSOUND = 0x00000004;

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 bilgiler ile Windows gezgini yardımı ile aldığımız bilgileri karşılaştırarak programımızın doğru çalıştığını kontrol ettik.



İNTERNET BAĞLANTISINI TEST ETMEK

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

```
if (flags== InternetGetConnectedStateFlags.INTERNET_CONNECTION_LAN)
{
    MessageBox.Show("Bağlantı türü LAN.");
}
```

biçiminde `InternetGetConnectedStateFlags` parametresine dönen değer okunarak, buna göre bağlantı türü belirlenebilir.

Ses Dosyalarını Çalmak

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 birlike 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();

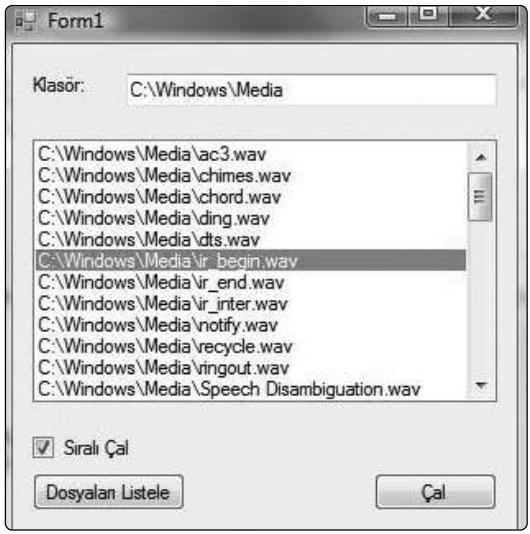
        [Flags]
        public enum SoundFlags
        {
```

```
        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.



Örneğimizde `waveOutGetNumDevs` metodu ile öncelikle 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 Değiştirmek

Ses ayarlarını değiştirmek için `winmm.dll` dosyasını kullanabiliriz. Bilgisayarın ses seviyesini öğrenmek için `waveOutGetVolume` metodunu kullanırken, ses seviyesini değiştirmek için `waveOutGetVolume` metodunu kullanacağız.

`waveOutGetVolume` metodunun kullanımı şu şekildedir:

```
waveOutGetVolume(IntPtr hwo, uint dwVolume)
```

`waveOutSetVolume` metodunun kullanımı ise şu şekildedir:

```
waveOutSetVolume(IntPtr uDeviceID, uint dwVolume)
```

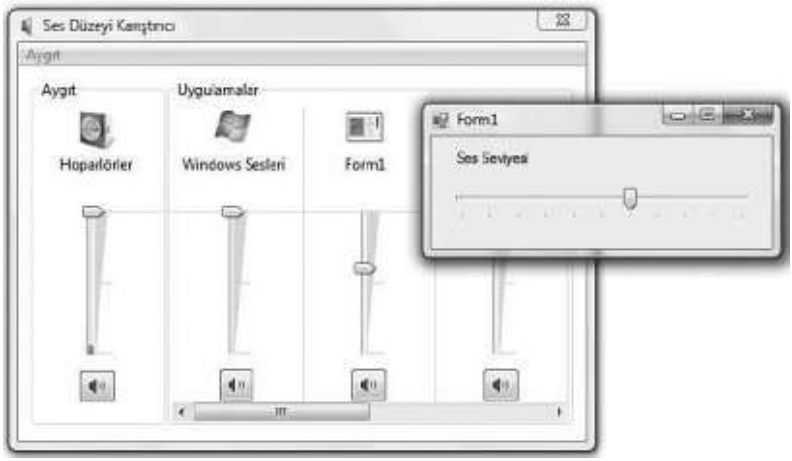
hwo: Ses çıkış aygıtının handle numarasını belirtir.

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 Yürütmek

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	stop
capability	load	record	sysinfo
capture	open	resume	undo
close	paste	save	where
copy	pause	seek	
cut	play	setaudio	
delete	put	spin	

```
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);
        }
    }
}
```

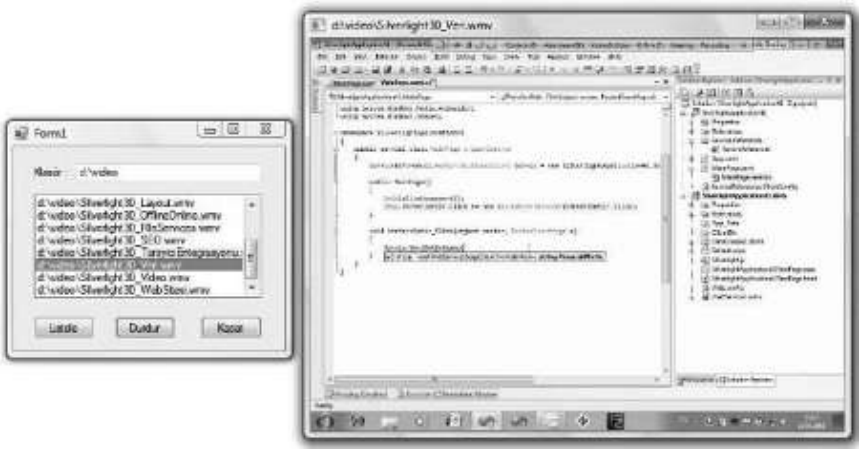
Kodumuzda

```
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.



Windows 7 API'lerini Kullanmak

Windows 7 işletim sisteminde mevcut olan özellikleri .net uygulamalarında kullanabilmek için **WindowsAPICodePack** isimli paketi kullanabiliriz. Bu paketi *http://code.msdn.microsoft.com/WindowsAPICodePack* adresinden indirebilirsiniz.

Bu pakette mevcut olan uygulamalar şunlardır:

Windows 7 Taskbar

- Jump Lists, Icon Overlay, Progress Bar, Tabbed Thumbnails, and Thumbnail Toolbars

Windows Shell

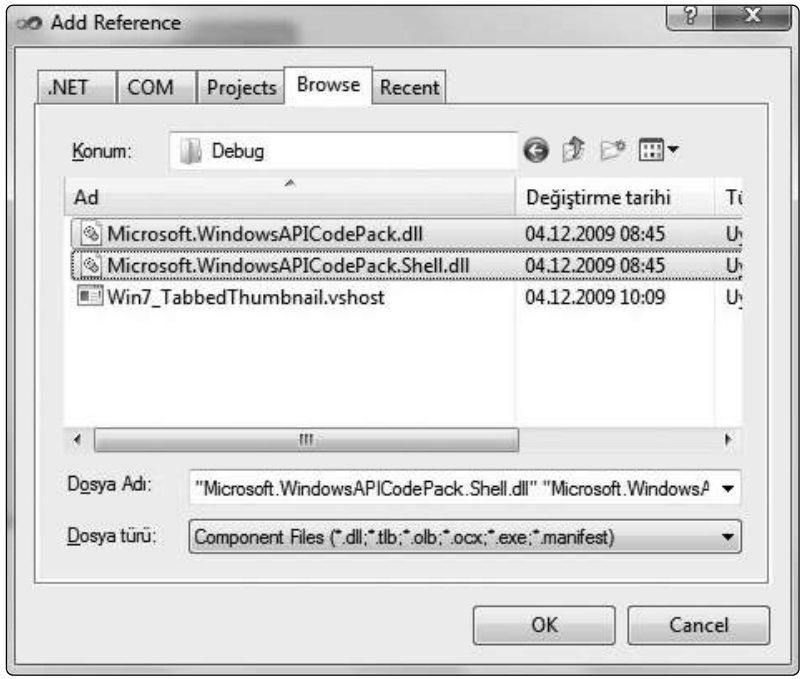
- Windows 7 Libraries
- Windows Shell Search API support
- Explorer Browser Control
- A hierarchy of Shell Namespace entities
- Windows Shell property system
- Drag and Drop for Shell Objects
- Windows Vista and Windows 7 Common File Dialogs, including custom controls
- Known Folders and non-file system containers

DirectX

- Direct3D 11.0, Direct3D 10.1/10.0, DXGI 1.0/1.1, Direct2D 1.0, DirectWrite, Windows Imaging Component (WIC) APIs
- Windows Vista and Windows 7 Task Dialogs
- Sensor Platform APIs
- Extended Linguistic Services APIs
- Power Management APIs
- Application Restart and Recovery APIs
- Network List Manager APIs
- Command Link control and System defined Shell icons

DİYALOG PENCERELERİ

`Windows7ApiCodePack` içerisinde mevcut olan diyalog pencerelerini kullanabilmek için öncelikle `Microsoft.WindowsAPICodePack.dll` ve `Microsoft.WindowsAPICodePack.Shell.dll` dosyalarını projemize referans olarak eklememiz gerekmektedir. `Microsoft.WindowsAPICodePack.dll` dosyasını `Windows7ApiCodePack` paketi içerisindeki `WindowsAPICodePack\Core\bin\Debug` klasörü içerisinde, `Microsoft.WindowsAPICodePack.Shell.dll` dosyasını ise `WindowsAPICodePack\Shell\bin\Debug` klasörü içerisinde bulabilirsiniz.



Yeni bir diyalog penceresi tanımlarken

```
CommonOpenFileDialog DiyalogPenceresiAdı = new
CommonOpenFileDialog( );
```

şeklinde tanımlarız.

Dosya Açma

CommonOpenFileDialog nesnesi varsayılan olarak dosya açma diyalog kutusu şeklinde açılmaktadır. Örnek olarak seçilen bir resim dosyasını bir picturebox içerisinde görüntületelim.

```
using System;
using System.Windows.Forms;
using System.IO;
using Microsoft.WindowsAPICodePack.Dialogs;
using Microsoft.WindowsAPICodePack.Shell;
namespace Win7_FileDialog
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void btnGozat_Click(object sender, EventArgs e)
        {
            CommonOpenFileDialog d = new CommonOpenFileDialog();
            pictureBox1.Image = null;

            d.AllowNonFileSystemItems = true;
            d.EnsureReadOnly = true;
            d.Title = "Lütfen Bir Resim Dosyası Seçiniz";
            d.InitialDirectoryShellContainer =
                (ShellContainer)KnownFolders.PicturesLibrary;

            d.Filters.Add(new CommonFileDialogFilter
                ("jpeg dosyaları", "*.jpg"));
            d.Filters.Add(new CommonFileDialogFilter
                ("bmp dosyaları", "*.bmp"));
            d.Filters.Add(new CommonFileDialogFilter
                ("gif dosyaları", "*.gif"));
            d.Filters.Add(new CommonFileDialogFilter
                ("png dosyaları", "*.png"));

            if (d.ShowDialog() == CommonFileDialogResult.OK)
            {
                string yol = System.IO.Path.GetFullPath
                    (d.FileAsShellObject.ParsingName);
```

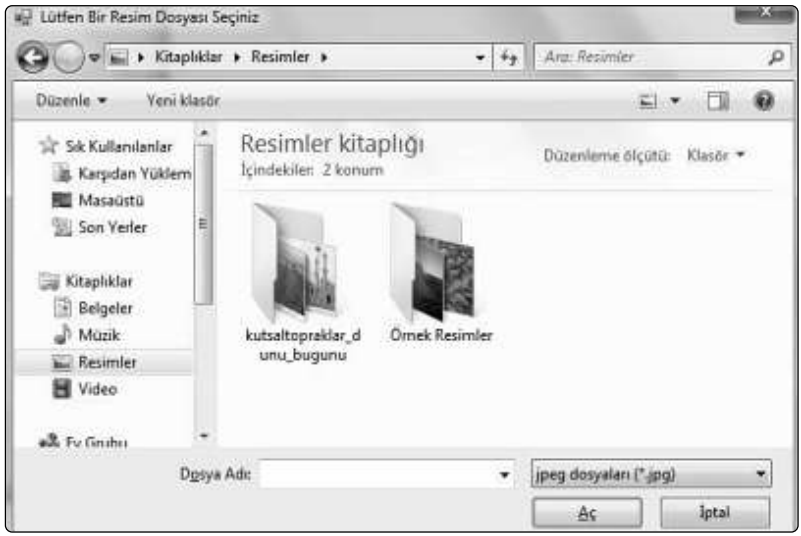
```
                    System.IO.FileStream fs = new
                        System.IO.FileStream
                        (yol, FileMode.Open, FileAccess.Read);
                    System.Drawing.Bitmap resim = new
                        System.Drawing.Bitmap(fs, true);
                    pictureBox1.Image = resim;
                }
            }
        }
}
```

Diyalog penceresinde dosya uzantılarına göre filtreleme yapmak için `Filters.Add` metodu kullanılır. Dosya uzantısına erişmek için `FileAsShellObject` nesnesinin `ParsingName` özelliği kullanılır.

Kodumuzda

```
d.InitialDirectoryShellContainer =
(ShellContainer)KnownFolders.PicturesLibrary;
```

satırı ile diyalog penceresinin varsayılan olarak *"Resimlerim"* klasörü ile açılmasını sağlıyoruz. `KnownFolder` özelliği ile bilgisayarda mevcut olan *Resimlerim*, *Belgelerim*, *Müziklerim*, *Windows*, *Program Files* gibi özel klasörler görüntülenir. Kodumuzu çalıştırıp butonu tıkladığımızda dosya seçme penceresi ekrana gelecektir.



Resmi seçip **Aç** düğmesine bastığımızda resim picturebox içerisinde görüntülenecektir.



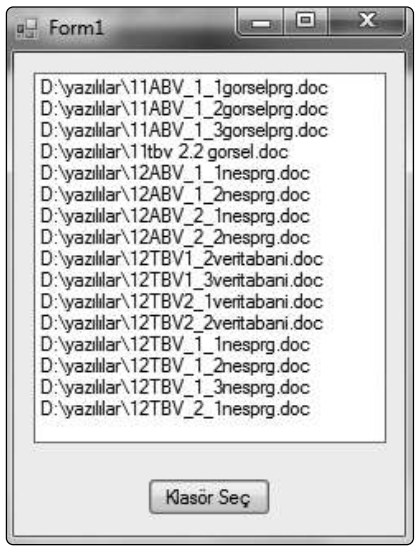
Klasör Seçme

`CommonOpenFileDialog` nesnesini klasör seçme modunda görüntülemek için `IsFolderPicker` özelliğine `true` değerini atamamız gerekmektedir. Örnek olarak seçilen bir klasör içerisinde bulunan word belgelerini bir listbox içerisinde listeletelim.

```
using System;
using System.Windows.Forms;
using Microsoft.WindowsAPICodePack.Dialogs;
using Microsoft.WindowsAPICodePack.Shell;
using System.IO;
namespace Win7_FolderDialog
{
    public partial class Form1 : Form
    {
        public Form1()
        {
```

```
            InitializeComponent();
        }
        private void btnKlasorSec_Click(object sender,
        EventArgs e)
        {
            CommonOpenFileDialog d = new CommonOpenFileDialog();
            d.IsFolderPicker = true;
            d.Title = "Lütfen Bir Klasör Seçiniz";
            d.InitialDirectory = "D:\\";
            if (d.ShowDialog() == CommonFileDialogResult.OK)
            {
                foreach (string bulunandosya in ↓
                    Directory.GetFiles
                    (d.FileAsShellObject.ParsingName,
                    "*.doc", SearchOption.TopDirectoryOnly))
                {

                  listBox1.Items.Add(bulunandosya.ToString());
                }
            }
        }
    }
}
```



Örnek olarak `WindowsApiCodePack` içerisinde yer alan diyalog penceresi örneğini sadeleştirerek yapalım. Örneğimizde seçili olan dosya ya da klasörün özelliklerini bir `DetailsListView` nesnesi içerisinde listeliyoruz. Ayrıca seçili olan dosya ya da klasörün simgesini bir picturebox nesnesi içerisinde önizleme biçiminde görüntülüyoruz.

Projemize yukarıdaki diyalog penceresi örneklerinden farklı olarak

```
Microsoft.WindowsAPICodePack.Dialogs
Microsoft.WindowsAPICodePack.Shell
```

sınıflarını eklemenin yanında dosya ve klasör özelliklerine erişim için

```
Microsoft.WindowsAPICodePack.Shell.PropertySystem
```

sınıfını da dahil ediyoruz.

```
using System;
using System.Windows.Forms;
using Microsoft.WindowsAPICodePack.Dialogs;
using Microsoft.WindowsAPICodePack.Shell;
using Microsoft.WindowsAPICodePack.Shell.PropertySystem;
namespace win7_Dialogs
{
  public partial class Form1 : Form
  {
      public Form1()
      {
          InitializeComponent();
      }
      private ShellObject currentlySelected = null;
      private void btnKlasorSec_Click(object sender, EventArgs e)
      {
        detailsListView.Items.Clear();
        pictureBox1.Image = null;

        CommonOpenFileDialog d = new CommonOpenFileDialog();
        d.EnsureReadOnly = true;
        d.IsFolderPicker = true;
        d.AllowNonFileSystemItems = true;

        if (d.ShowDialog() == CommonFileDialogResult.OK)
        {
          ShellContainer selectedSO = null;
          try
          {
            selectedSO = d.FileAsShellObject as ShellContainer;
          }
          catch
```

```
            {
              MessageBox.Show("ShellObject nesnesi ↓
               oluşturulamadı.");
            }

            currentlySelected = selectedSO;
            txtKlasorAdı.Text = selectedSO.ParsingName;
            DisplayProperties(selectedSO);
          }
        }

        private void btnDosyaSec_Click(object sender, EventArgs e)
        {
          CommonOpenFileDialog d = new CommonOpenFileDialog();
          detailsListView.Items.Clear();
          pictureBox1.Image = null;

          d.AllowNonFileSystemItems = true;
          d.EnsureReadOnly = true;

          if (d.ShowDialog() == CommonFileDialogResult.OK)
          {
            ShellObject selectedSO = null;
            try
            {
              selectedSO = d.FileAsShellObject;
            }
            catch
            {
              MessageBox.Show("ShellObject nesnesi ↓
               oluşturulamadı");
            }
            currentlySelected = selectedSO;
            txtDosyaAdı.Text = selectedSO.ParsingName;
            DisplayProperties(selectedSO);
          }
        }
        private void DisplayProperties(ShellObject selectedSO)
        {
        if (selectedSO != null)
          {
       pictureBox1.Image = selectedSO.Thumbnail.Bitmap;
```

```
        AddProperty("İsim", selectedSO.Name);
        AddProperty("Yol", selectedSO.ParsingName);
        AddProperty("Tip", selectedSO.GetType().Name);

        foreach (IShellProperty prop in selectedSO.Properties.
                 DefaultPropertyCollection)
            {
          if (prop.ValueAsObject != null)
                {
            try
                {
              if (object.ReferenceEquals(prop.ValueType,
                       typeof(string[])))
                 {
                   string[] arr = (string[])prop.ValueAsObject;
                string value = "";
                if (arr != null && arr.Length > 0)
                    {
                  foreach (string s in arr)
                      {
                    value = value + s + "; ";
                  }
                  if (value.EndsWith("; "))
                      {
                    value = value.Remove(value.Length - 2);
                  }
                }
                AddProperty(prop.CanonicalName, value);
             }
                  else
                  {
                AddProperty(prop.CanonicalName,
                        prop.ValueAsObject.ToString());
              }
            }
                catch
                {
            }
           }
       }
      }
```

```
        }
        private void AddProperty(string property, string value)
        {
            if (!string.IsNullOrEmpty(property))
            {
              ListViewItem lvi = new ListViewItem(property);

              ListViewItem.ListViewSubItem subItemValue = new
                  ListViewItem.ListViewSubItem(lvi, value);
              lvi.SubItems.Add(subItemValue);
              detailsListView.Items.Add(lvi);
            }
        }
    }
}
```

Kodumuzda `DisplayProperties` metodu ile seçili olan dosya ya da klasörün seçtiğimiz özelliklerini listeliyoruz. `AddProperty` metodu ile belirttiğimiz herhangi bir özelliği ekleyebiliriz.

`AddProperty` özelliğinin kullanımı şu şekildedir:

```
AddProperty(özellik adı, değeri)
```

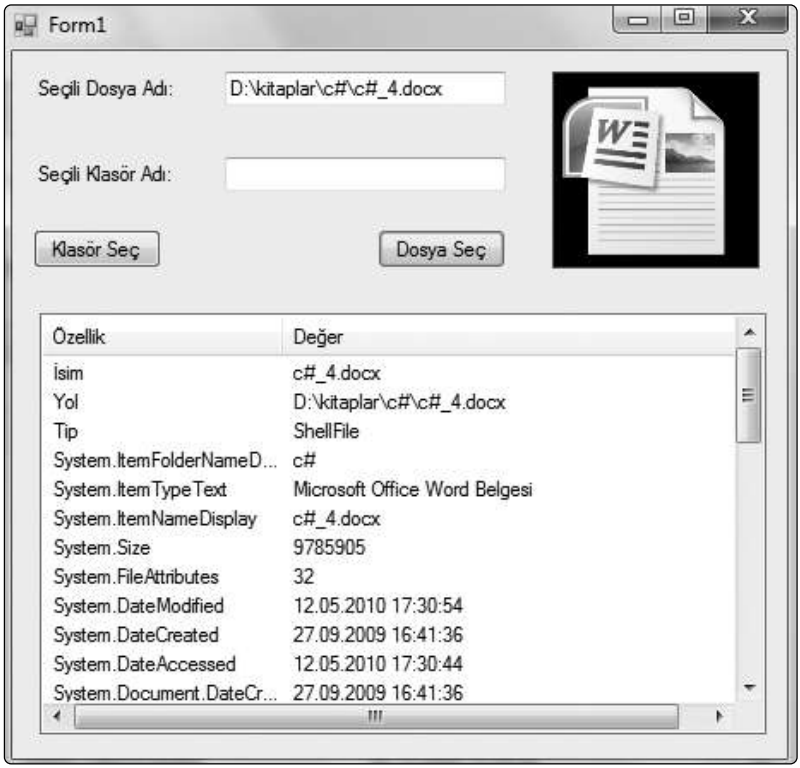
Örneğin;

```
AddProperty("İsim", selectedSO.Name);
```

ile seçili olan nesnenin isim özelliğini eklerken,

```
AddProperty("Yol", selectedSO.ParsingName);
```

ile seçili olan nesnenin yolunu ekleriz.



ExplorerBrowser Control

Dosya ve klasörler arasında gezinmek amacıyla kullanılan bir kontroldür. Formumuza dâhil etmek için toolbox'a `ExplorerBrowserControl` nesnesinin eklenmesi gerekmektedir. Bu nesneyi eklemek için toolbox'a farenin sağ tuşu ile tıklayarak **Choose Items...** seçeneğini tıklarız. Gelen pencereden **.Net Framework Components** sekmesinde **Browse** butonuna tıklayarak `Microsoft.WindowsAPICodePack.Shell.dll` dosyasını seçeriz.

`ExplorerBrowser` nesnesi ile işlem yapabilmek için ayrıca

```
Microsoft.WindowsAPICodePack.Shell
Microsoft.WindowsAPICodePack.Controls
```

sınıflarını projemize dahil etmemiz gerekmektedir.

`ExplorerBrowser` nesnesi ile dosya ve klasörler arasında gezinmek için `Navigate` metodu kullanılır.

Bir önceki ziyaret edilen yere gitmek için

```
NavigateLogLocation(NavigationLogDirection.Backward)
```

metodu kullanılırken, bir sonraki ziyaret edilen yere gitmek için

```
NavigateLogLocation(NavigationLogDirection.Forward)
```

metodu kullanılır.

Gezinilen klasörlerin listesi `NavigationLog.Locations` nesnesi içerisinde saklanır. Gezinilen klasör listesini temizlemek için `NavigationLog.ClearLog( )` metodu kullanılır.

```
using System;
using System.Windows.Forms;
using Microsoft.WindowsAPICodePack.Shell;
using Microsoft.WindowsAPICodePack.Controls;
using System.Runtime.InteropServices;
using System.Collections.Generic;
namespace Win7_ExplorerBrowserControl
{
  public partial class Form1 : Form
  {
      public Form1()
      {
          InitializeComponent();
          List<string> klasor_listesi = new List<string>();
          foreach (IKnownFolder klasor in KnownFolders.All)
          {
              klasor_listesi.Add(klasor.CanonicalName);
          }
          klasor_listesi.Sort();
          lstKlasorler.Items.AddRange(klasor_listesi.ToArray());

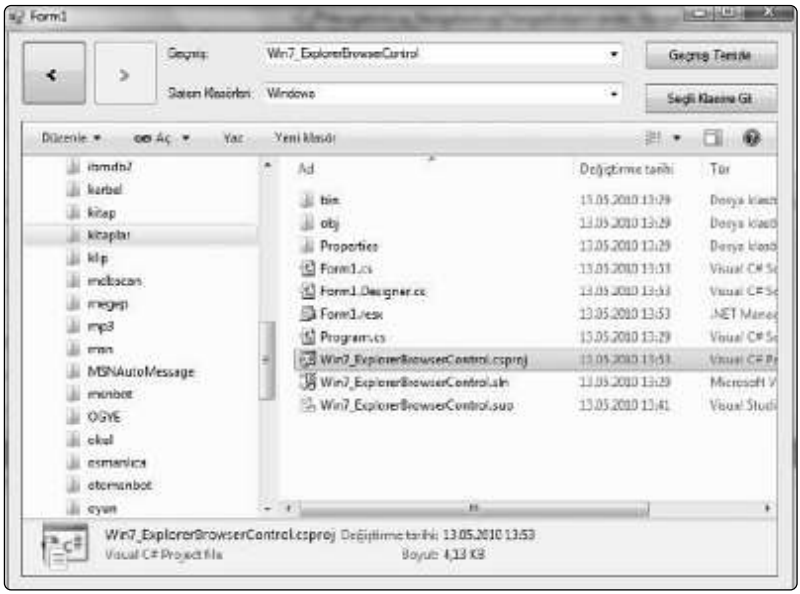
          ExplorerBrowser1.NavigationComplete +=
              ExplorerBrowser1_NavigationComplete;
          ExplorerBrowser1.NavigationLog.NavigationLogChanged +=
              NavigationLog_NavigationLogChanged;
      }
      private void ExplorerBrowser1_Load(object sender,
      EventArgs e)
      {
          btnileri.Enabled = false;
          btngeri.Enabled = false;
```

```
            ExplorerBrowser1.ContentOptions.SingleClickActivate =
              true;
            ExplorerBrowser1.Navigate((ShellObject) ↓
             KnownFolders.Documents);
        }
        private void btngeri_Click(object sender, EventArgs e)
        {
            ExplorerBrowser1.NavigateLogLocation
              (NavigationLogDirection.Backward);
        }
        private void btnileri_Click(object sender, EventArgs e)
        {
            ExplorerBrowser1.NavigateLogLocation
              (NavigationLogDirection.Forward);
        }

        private void ExplorerBrowser1_NavigationComplete(object ↓
        sender,
            NavigationCompleteEventArgs e)
        {
            btngeri.Enabled = ExplorerBrowser1.NavigationLog.
              CanNavigateBackward;
            btnileri.Enabled = ExplorerBrowser1.NavigationLog.
              CanNavigateForward;
        }
        private void lstGecmis_SelectedIndexChanged(object ↓
        sender, EventArgs e)
        {
            ExplorerBrowser1.NavigateLogLocation(this.
              lstGecmis.SelectedIndex);
        }
        private void btnSistemKlasoruneGit_Click(object ↓
        sender, EventArgs e)
        {
            try
            {
                IKnownFolder kf = KnownFolderHelper.FromCanonicalName
                   (this.lstKlasorler.Items[lstKlasorler.
                   SelectedIndex].ToString());
                this.ExplorerBrowser1.Navigate((ShellObject)kf);
            }
```

```
            catch (COMException e1)
            {
                MessageBox.Show("Seçili klasöre erişilemedi!");
            }
        }
        private void NavigationLog_NavigationLogChanged(object ↓
        sender, NavigationLogEventArgs args)
        {
            BeginInvoke(new MethodInvoker(() => ↓
              Gezinme_Logu(args)));
        }
        private object Gezinme_Logu(NavigationLogEventArgs args)
        {
            if (args.CanNavigateBackwardChanged)
            {
                this.btngeri.Enabled = ExplorerBrowser1.
                    NavigationLog.CanNavigateBackward;
            }
            if (args.CanNavigateForwardChanged)
            {
                this.btnileri.Enabled = ExplorerBrowser1.
                    NavigationLog.CanNavigateForward;
            }
            if (args.LocationsChanged)
            {
                this.lstGecmis.Items.Clear();
                foreach (ShellObject shobj in this.
                    ExplorerBrowser1.NavigationLog.Locations)
                {
                    this.lstGecmis.Items.Add(shobj.Name);
                }
            }
            if (this.ExplorerBrowser1.NavigationLog.
                CurrentLocationIndex == -1)
            {
                this.lstGecmis.Text = "";
            }
            else
            {
                this.lstGecmis.SelectedIndex = this.
                    ExplorerBrowser1.NavigationLog. ↓
                      CurrentLocationIndex;
```

```
            }
            return null;
        }
        private void btnGecmisiTemizle_Click(object
        sender, EventArgs e)
        {
            ExplorerBrowser1.NavigationLog.ClearLog();
        }
    }
}
```



Örneğimizde açılan klasörlerin listesi **Geçmiş** bölümünde saklanmakta ve geri ve ileri butonları yardımı ile bu liste arasında gezinilebilmektedir.

Belgelerim, *Program Files*, *Resimlerim*, *Müziklerim*, *Windows* gibi klasörler ise sistem klasörleri bölümünde listelenmekte ve bu klasörler **Seçili Klasöre Git** butonuna tıklanarak açılabilmektedir.

Seçili olan dosya ve klasörlerle ilgili bilgiler **ExplorerBrowser** penceresinin alt tarafında görüntülenmektedir.

Power Manager

Blgisayarın güç yönetimi ile ilgili özellikler hakkında bilgi almak için `PowerManager` nesnesi kullanılır.

Bilgisayarın o anki güç besleme türü hakkında bilgi almak için `PowerSource` özelliği kullanılır.

Bu özellik şu değerleri alır:

```
PowerSource.AC
PowerSource.Battery
PowerSource.UPS
```

Bilgisayar bataryasının o anki şarj seviyesi hakkında bilgi almak için `PowerManager` nesnesinin `BatteryLifePercent` özelliği kullanılır. Bu özellik şarj durumunu yüzde cinsinden geriye döndürür.

Bilgisayar bataryasınının bağlı olup olmadığını `PowerManager` nesnesinin `IsBatteryPresent` özelliği ile öğrenebiliriz. Bu değer geriye boolean değer döndürür. Eğer `true` ise batarya bağlıdır, `false` ise bağlı değildir.

Batarya hakkındaki diğer bilgiler için `PowerManager` nesnesinin `GetCurrentBatteryState()` özelliği kullanılır. Bu özellik ile alınabilecek bilgilerden `batteryState.MaxCharge` özelliği ile bataryanın maksimum şarj akım seviyesi `mWh` cinsinden öğrenilebilirken, `batteryState.CurrentCharge` özelliği ile de bataryanın aktif şarj akım seviyesi `mWh` cinsinden öğrenilebilir.

`Power Manager` nesnesi ile işlem yapabilmek için ayrıca

```
Microsoft.WindowsAPICodePack.ApplicationServices
Microsoft.WindowsAPICodePack.Shell
```

sınıflarını projemize dahil etmemiz gerekmektedir.

```
private void Form1_Load(object sender, EventArgs e)
{
  lblBesleme.Text = PowerManager.PowerSource.ToString();

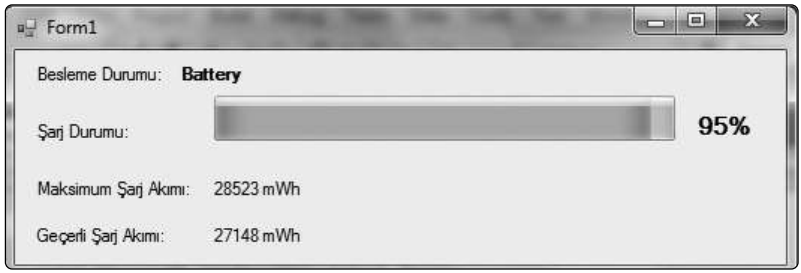
  BatteryState batteryState =
    PowerManager.GetCurrentBatteryState();
  progressBar1.Value = PowerManager.BatteryLifePercent;
  lblSarjYuzde.Text = PowerManager.BatteryLifePercent.ToString() + "%";
```

```
    lblMaksimumSarj.Text = batteryState.MaxCharge + " mWh";
    lblGecerliSarj.Text = batteryState.CurrentCharge + " mWh";
}
```

Prizden besleme durumu



Bataryadan besleme durumu



JUMP LIST

Windows 7 ile gelen yeni özelliklerden birisi olan jump list ile bir pencere içerisine sık kullandığınız uygulama kısayollarını ekleyerek hızlı bir şekilde bu programlara erişim sağlayabilirsiniz.

Jump List'i uygulamamızda kullanabilmek için öncelikle

```
Microsoft.WindowsAPICodePack.Taskbar
```

sınıfını projemize dahil etmemiz gerekmektedir.

Jump list'i oluşturmak için `JumpList.CreateJumpList()` metodu kullanılır. Görevler kategorisine bir eleman eklemek için `JumpList.AddUserTasks( )` metodu kullanılır.

Bu metodun kullanımı şu şekildedir:

```
JumpList.AddUserTasks(new JumpListLink(pathValue, titleValue))
```

pathValue: Eklenecek olan program kısayolunun dosya yolunu belirtir.

titleValue: Eklenecek olan program kısayolunun ismini belirtir.

JumpList içerisinde yeni bir kategori oluşturmak için şu işlemler yapılmalıdır.

- `JumpListCustomCategory("Kategori Adı")` metodu ile yeni bir kategori oluşturulur.
- `JumpList.AddCustomCategories(Kategori Adı)` metodu ile oluşturulan kategori jump list içerisine eklenir.
- `KategoriAdı.AddJumpListItems(new JumpListLink(pathValue, titleValue))` metodu ile oluşturulmuş olan kategori içerisine yolu ve ismi belirtilen program kısayolu eklenir.

Görevler kategorisi ya da oluşturulan özel kategorilere eklenen elamanlar arasında ayırıcı çizgiler eklemek için `JumpListSeparator()` nesnesi kullanılır. Örneğin görevler kategorisine ayırıcı çizgi eklemek için

```
JumpList.AddUserTasks(new JumpListSeparator( ))
```

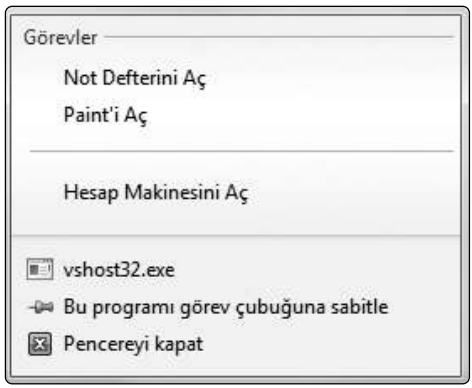
şeklinde kullanılır.

Jump list içerisinde yapılan değişikliklerin güncellenmesi için `JumpList.Refresh( )` metodu kullanılır.

```
using System;
using System.Windows.Forms;
using Microsoft.WindowsAPICodePack.Taskbar;
using Microsoft.WindowsAPICodePack.Shell;
using System.IO;
namespace Win7_JumpList
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
```

```
        }
        private JumpList jl;
        string sistem_klasoru= Environment.GetFolderPath
            (Environment.SpecialFolder.System);
        private void Form1_Shown(object sender, EventArgs e)
        {
          jl = JumpList.CreateJumpList();

          jl.AddUserTasks(new JumpListLink(Path.Combine ↓
              (sistem_klasoru, "notepad.exe"),
              "Not Defterini Aç"));
          jl.AddUserTasks(new JumpListLink(Path.Combine ↓
              (sistem_klasoru, "mspaint.exe"), "Paint'i Aç"));
          jl.AddUserTasks(new JumpListSeparator());
          jl.AddUserTasks(new JumpListLink(Path.Combine ↓
              (sistem_klasoru, "calc.exe"),
              "Hesap Makinesini Aç"));
          jl.Refresh();
        }
    }
}
```



NetworkListManager

Bilgisayarda mevcut olan ağ donanımı ve bağlantıları hakkında bilgi almak amacı ile kullanılır. Ağ donamımlarının listesi `NetworksCollection` adlı bir dizi değişken içerisinde tutulur. Bu dizi değişken içerisine

```
NetworkListManager.GetNetworks(NetworkConnectivityLevels.All)
```

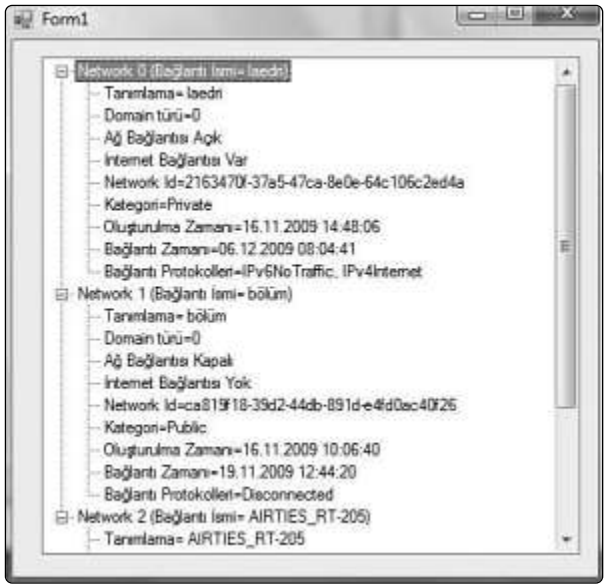
metodu ile mevcut ağ bağlantıları aktarılır.

Bilgisayarda mevcut olan ağ bağlantılarını ve bu bağlantıların özelliklerini bir treeview içerisinde listeleyen örnek bir uygulama yapalım.

```
using System;
using System.Windows.Forms;
using Microsoft.WindowsAPICodePack.Net;
namespace Win7_NetworkListManager
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            NetworkCollection networks = NetworkListManager.
                GetNetworks(NetworkConnectivityLevels.All);
            foreach (Network n in networks)
            {
                TreeNode ana_dugum = new TreeNode();
                ana_dugum.Text = string.Format("Network ↓
                    {0} ({1})",
                    TreeView1.Nodes.Count, "Bağlantı İsmi= ↓
                    " + n.Name);
                TreeView1.Nodes.Add(ana_dugum);

                ana_dugum.Nodes.Add("Tanımlama= " +
                  n.Description);
                ana_dugum.Nodes.Add("Domain türü=" +
                  n.DomainType);
                if (n.IsConnected == true)
                {
                    ana_dugum.Nodes.Add("Ağ Bağlantısı Açık");
                }
                else
                {
                    ana_dugum.Nodes.Add("Ağ Bağlantısı ↓
                      Kapalı");
                }
                if (n.IsConnectedToInternet == true)
                {
```

```
                    ana_dugum.Nodes.Add("İnternet ↓
                      Bağlantısı Var");
                }
                else
                {
                    ana_dugum.Nodes.Add("İnternet ↓
                      Bağlantısı Yok");
                }
                ana_dugum.Nodes.Add("Network Id=" +
                    n.NetworkId.ToString());
                ana_dugum.Nodes.Add("Kategori=" +
                    n.Category.ToString());
                ana_dugum.Nodes.Add("Oluşturulma Zamanı=" +
                    n.CreatedTime.ToString());
                ana_dugum.Nodes.Add("Bağlantı Zamanı=" +
                    n.ConnectedTime.ToString());
                ana_dugum.Nodes.Add("Bağlantı Protokolleri=" +
                    n.Connectivity.ToString());
            }
            TreeView1.ExpandAll();
        }
    }
}
```



24

DOSYA VE KLASÖR İŞLEMLERİ

Sınırlı sayıdaki bilgilerin disk ortamında saklanması, bu bilgilere erişilmesi, üzerinde değişiklik yapılması gibi işlemler için veritabanı oluşturmak yerine dosyalama işlemleri kullanılabilir. .Net klasörler ve dosyalar üzerinde işlemler yapılması için geniş bir komut kümesine sahiptir. Bu bölümde kullanılacak olan yöntemler şunlardır:

1. `System.IO` sınıfı kullanılarak,
2. `Isolated Storage File` sınıfı kullanılarak dosya ve klasör işlemleri 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 Metodları

Dosyalar üzerinde işlem yapmak için `File` nesnesi kullanılabilir. Aşağıdaki metodlara 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** (*Coordinated Universal Time*), yani eşgüdümlü **evrensel zaman** (yahut saat) 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 iki 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 metodları 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 arabellek 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.

```
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.volkanaktas.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.volkanaktas.com");` text adı ile bir byte dizisi oluşturarak `UTF8` 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)
```



Bundan sonraki `System.IO` sınıfı ile yapacağımız örneklerimizde;

```
using System.IO;
```

satırını gereksiz tekrarlamamak için yazmayacağım. Ancak okuyucularım her uygulamaya bu satırı yazmayı unutmasınlar.

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 metdoddur. Örnek olarak formun yüklenişi sırasında bir dosya içerisindeki değerleri okuyarak listbox'a aktaran ve listbox içerisine değer ekleyen ve bu 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();
        }
    }
}
```

Formun `Load` olayı içerisinde 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.

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.

Bu metodun kullanımı şu şekildedir:

```
File.GetLastWriteTime(Dosya yolu)
```

Diğer iki metodun da kullanımı bu 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.

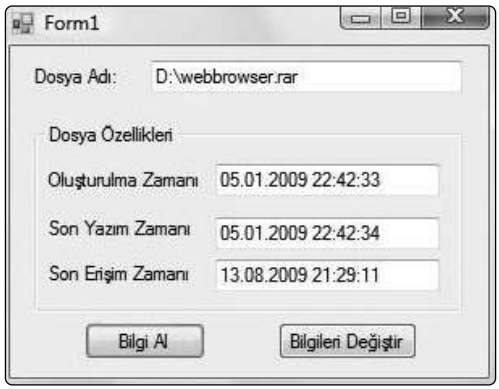
Bu metodun kullanımı şu şekildedir:

```
File.SetLastWriteTime(Dosya yolu, tarih ve zaman)
```

Diğer iki metodun da kullanımı bu 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` metodlarının 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(txtKaynak.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** (*eşzamansız*) 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");
}
```

StreamWriter nesnesi kullanılırken yazma işleminde Write ve WriteLine metodları 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 metodları 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 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.

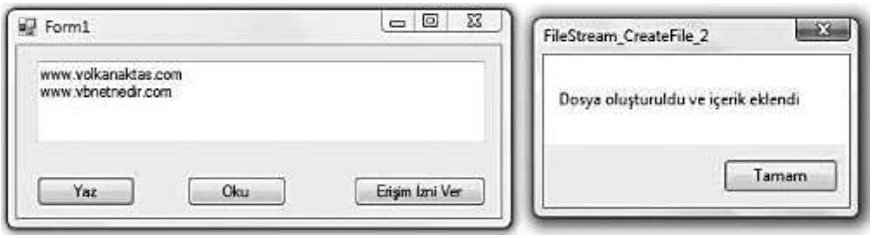
`Environment.NewLine` ile yeni bir satır başlangıcı yapılıp yapılmadığını kontrol ediyoruz.

Dosyalar üzerinde erişim hakları ile ilgili örnek bir uygulama yapalım.

```
using System;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Security.AccessControl;
namespace FileStream_AddAccessRule
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        FileSecurity fs = new FileSecurity();
        private void btnYaz_Click(object sender, EventArgs e)
        {
          fs.AddAccessRule(new FileSystemAccessRule ↓
              (“volkan-PC\\volkan”, ↓
              FileSystemRights.ReadData, ↓
              AccessControlType.Allow));
          byte[] icerik = Encoding.ASCII.GetBytes(textBox1.Text);
          FileStream fw = new FileStream
              (“d:\\sifrelenmis1.txt”, ↓
              FileMode.Create, FileSystemRights.Modify, ↓
              FileShare.None, 8, FileOptions.None, fs);
          fw.WriteByte(System.Convert.ToByte(icerik.Length));
          fw.Write(icerik, 0, icerik.Length);
          fw.Close();
```

```
            MessageBox.Show("Dosya oluşturuldu ve içerik ↓
              eklendi");
          }
          private void btnOku_Click(object sender, EventArgs e)
          {
            FileStream fr = new FileStream("d:\\sifrelenmis1.txt",
                FileMode.Open, FileSystemRights.ReadData,
                FileShare.None, 8, FileOptions.None, fs);
            int length = fr.ReadByte();
            byte[] readBytes = new byte[length];
            fr.Read(readBytes, 0, readBytes.Length);
            fr.Close();
            textBox1.Text = (Encoding.ASCII.GetString(readBytes));
          }
          private void btnizinVer_Click(object sender,
          EventArgs e)
          {
            fs.AddAccessRule(new FileSystemAccessRule
              ("volkan-PC\\volkan", FileSystemRights.FullControl, ↓
              AccessControlType.Allow));
            MessageBox.Show("Dosyaya erişim izni verildi");
          }
      }
}
```

Örneğimizi çalıştırıp text kutusu içerisine değerler girelim ve yaz butonuna basarak içerikleri dosya içerisine ekleyelim ve erişim iznini ayarlayalım.



Disk üzerinde oluşturmuş olduğumuz dosyaya erişmek istediğimizde dosyanın boş bir şekilde açılıp erişim izni verilmediğini görürüz. Dosyaya programımız içerisinden erişim sağlayıp bilgileri okuyabilirken,

haricen disk üzerinden erişilmeye kalkıldığında uyarı verip erişim engellenecektir.



Haricen erişim iznini yeniden vermek için **Erişim İzni Ver** butonunu tıklamalıyız.

Klasörler Üzerinde Yapılan İşlemler

`Directory` nesnesi klasörler üzerinde işlem yapmak için kullanılır. Directory nesnesinin başlıca özellik ve metodları 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)
```

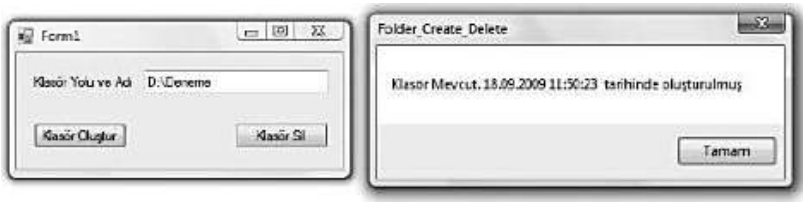
`Delete` metodunun kullanımı şu şekildedir:

```
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 klasörü oluşturuyoruz. Aynı şekilde `Exist` metodu ile klasörün mevcut olup olmadığı bilgisini alıp eğer klasör mevcut ise 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)
```

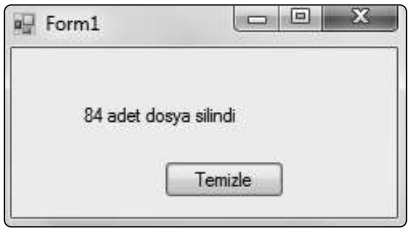
Örnek olarak bilgisayarın cookies klasöründe bulunan dosyaları temizleyen bir uygulama yapalım.

```
private void btnTemizle_Click(object sender, EventArgs e)
{
    string kullanıcı = System.Environment.UserName;
    string yol =
System.Environment.GetEnvironmentVariable("HOMEDRIVE") +
        System.Environment.GetEnvironmentVariable("HOMEPATH");

    string[] cookies = System.IO.Directory.GetFiles(yol +
        @"\AppData\Roaming\Microsoft\Windows\cookies\", "*.*");
    label1.Text = cookies.Length + " adet dosya silindi";

    foreach (string dosya in cookies)
    {
        try
        {
            System.IO.File.Delete(dosya);
```

```
            }
            catch (Exception ex)
            { }
        }
}
```



KLASÖR ÖZELLİKLERİNİ ELDE ETME VE DEĞİŞTİRME

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)
```

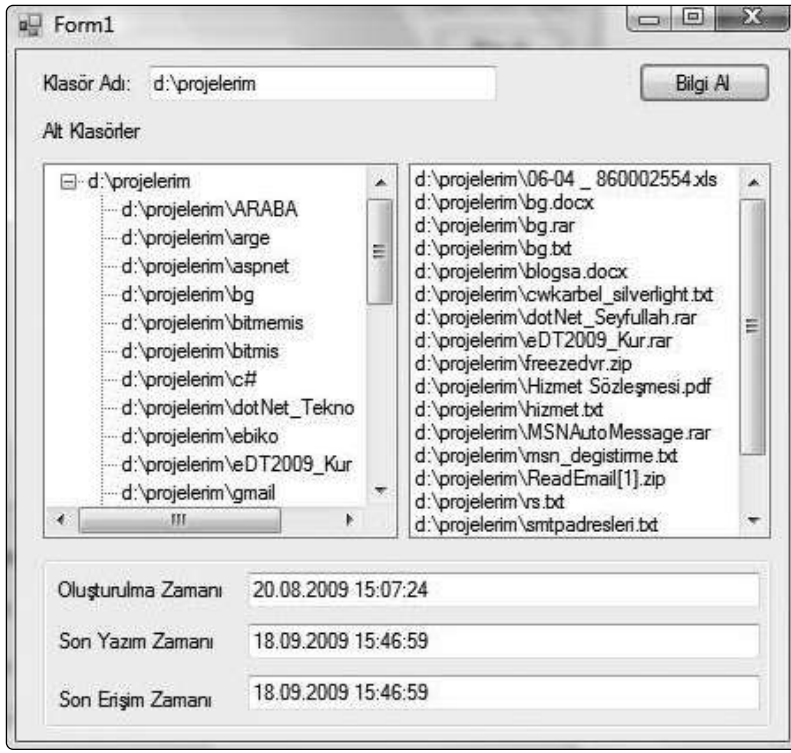
`GetDirectories` metodunun da kullanımı aynı şekildedir.

```
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 formun `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 değişkenimizin 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.

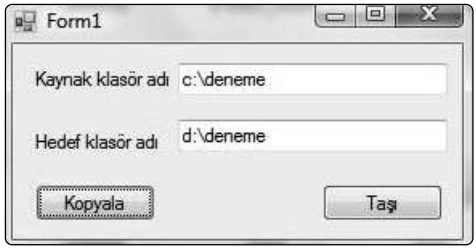
`FileInfo` sınıfının `CopyTo` metodunun kullanımı şu şekildedir:

```
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);
```

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 Metodları

Close: Daha önceden `GetStore`, `GetUserStoreForAssembly` veya `GetUserStoreForDomain` metodları 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 metod 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 metod 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
```

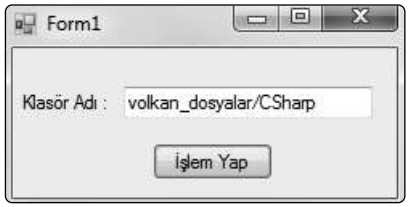
sınıfını dahil etmeliyiz.

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 isolated storage file alanı içerisinde bir klasör oluşturalım.

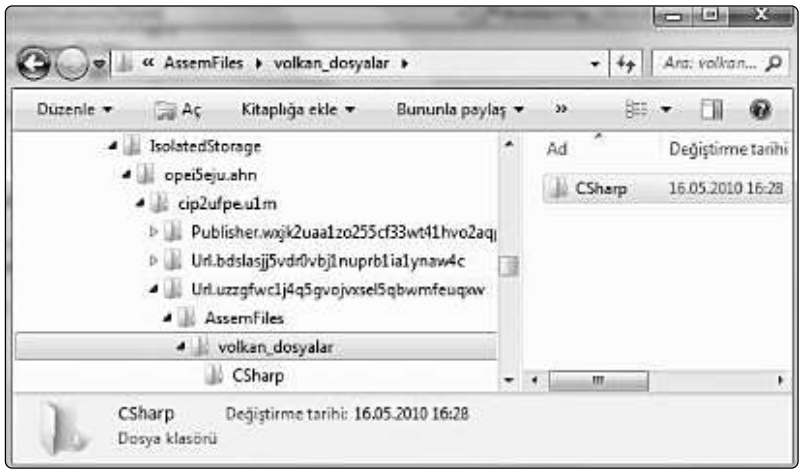
```
using System;
using System.IO.IsolatedStorage;
using System.Windows.Forms;
namespace IsolatedStorageFile_KlasorOlusturma
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        IsolatedStorageFile isf = IsolatedStorageFile.
            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 *"\opei5eju.ahn\cip2ufpe.u1m\Url.uzzgfwc1j4q5gvojvxsel5qbwmfeuqxw"* 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 de yeni bir dosya oluşturuyoruz.

`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 sıfır 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 isolated storage file 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);

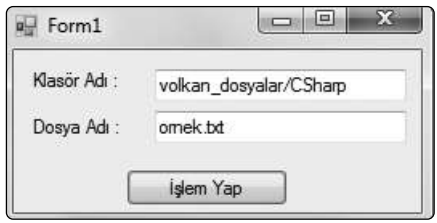
            if (txtDosya.Text != "")
            {
                IsolatedStorageFileStream isf_fst = new
                   IsolatedStorageFileStream(txtKlasor.Text + ↓
                     "\\" + txtDosya.Text, ↓
                     FileMode.Create, isf);
            }
        }
    }
}
```

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.



NOT

Bundan sonraki Isolated Storage File ile yapacağımız örneklerimizde;

```
using System.IO;
using System.IO.IsolatedStorage;
```

satırlarını gereksiz tekrarlamamak için yazmayacağım. Ancak okuyucularım her uygulamaya bu satırları yazmayı unutmasınlar.

Dosyaya Yazma

İzole edilmiş depolama alanı içerisinde oluşturulan dosyaya yazmak için StreamWriter nesnesini kullanabiliriz. Örneğin; isolated storage file 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);

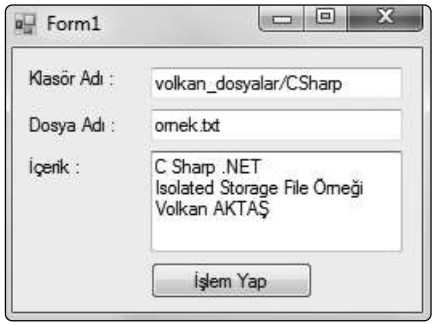
    if (txtDosya.Text != "")
    {
        IsolatedStorageFileStream isf_fst = new
            IsolatedStorageFileStream(txtKlasor.Text + "\\" +
            txtDosya.Text, FileMode.OpenOrCreate, isf);
        StreamWriter str_wrt = new StreamWriter(isf_fst);
        str_wrt.Write(txticerik.Text);
        str_wrt.Flush();
        str_wrt.Close();
    }
}
```

Örneğimizde

```
IsolatedStorageFileStream isf_fst = new
            IsolatedStorageFileStream(txtKlasor.Text + "\\" +
            txtDosya.Text, FileMode.OpenOrCreate, isf);
```

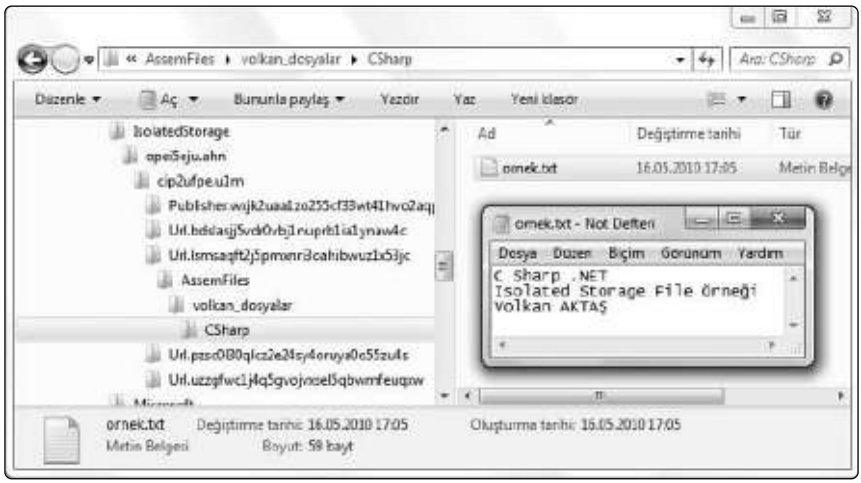
komut satırları ile `OpenOrCreate` modunu kullanarak yolu ve dosya adı textbox nesnelerine girilen `IsolatedStorageFileStream` nesnesini eğer mevcutsa açıyoruz. Mevcut değilse yeni bir tane oluşturuyoruz.

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 metod oluşturduk. Bu metod içerisinde `GetDirectoryNames` ve `GetFileNames` metodları ile klasör ve dosya isimlerini elde ettik. Bu metodlar kullanılırken örneğimizde olduğu gibi

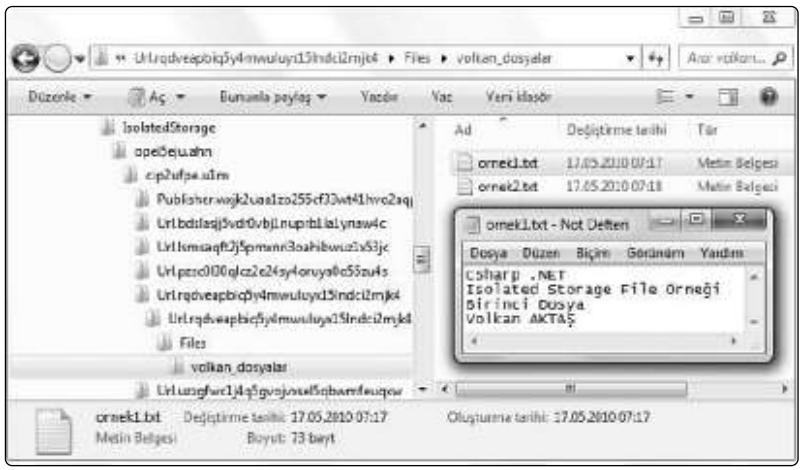
```
GetDirectoryNames("*")
GetFileNames("*")
```

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` metodları ile de listelenen klasör ve dosya isimlerinden seçili olanların izole edilmiş depolama alanı içerisinden silinmesini sağlıyoruz.

25

VERİTABANI İŞLEMLERİ

C# veritabanları üzerinde işlemler gerçekleştirmek üzere **DAO**, **ADO**, **RDO**, **ODBC** gibi teknolojilere sahiptir. Biz bu bölümde bu teknolojilerden en çok kullanılanı olan **ADO.NET**'e değineceğiz. ADO .NET ile **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 veritabanı 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 veritabanı içerisindeki verileri okuma ve yazma amacı ile kullanılır.
- **DataAdapter:** Veritabanı bağlantısı olmasa bile veriler üzerinde işlemler gerçekleştirilebilir.

Access Veritabanı İşlemleri

Access veritabanlarına erişim için kullanılabilecek veritabanı nesneleri şunlardır;

- `OleDBConnection` nesnesi
- `OleDBCommand` nesnesi
- `OleDbDataAdapter`
- `DataReader`
- `DataSet`

Öncelikle kod yazmadan bir veritabanına erişimi nasıl sağlarız, onunla ilgili bir uygulama yapalım.

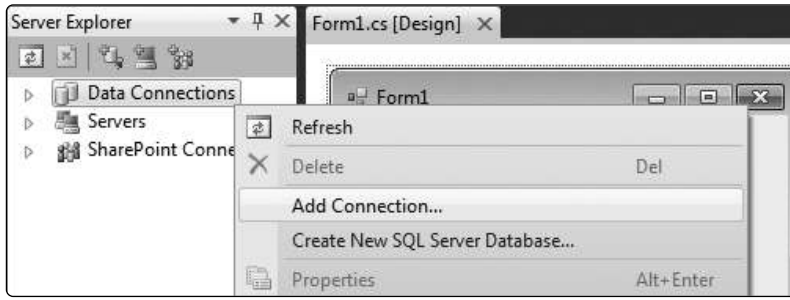
Örnek bir Access veritabanı oluşturalım.

Veritabanı adı: `sozluk`
Tablo adı: `sozcukler`
Alan adları: `id, ingilizce, turkce`

şeklinde olsun.



C# içerisine veritabanımızı ekleyerek üzerinde görsel olarak işlemler yapabilmek için **Server Explorer**'ı kullanabiliriz. Server Explorer içerisinde **Data Connections** seçeneğine farenin sağ tuşu ile tıklayarak **Add Connection** seçeneğine tıklıyoruz.

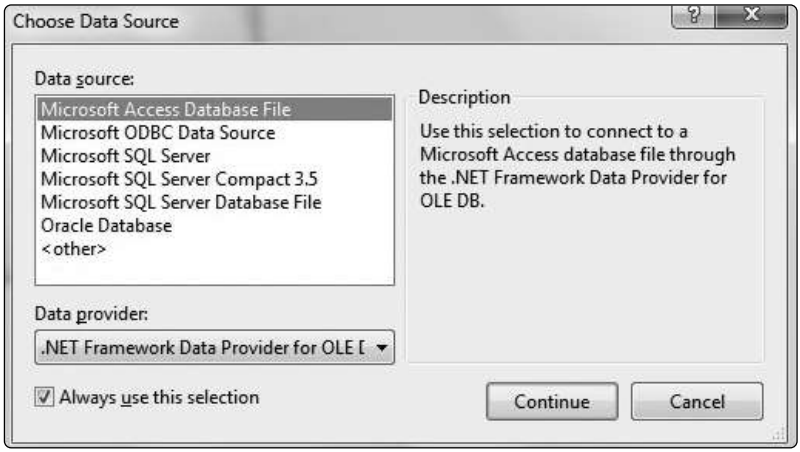


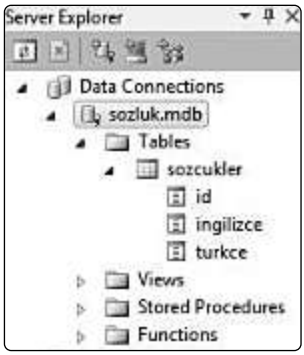
Gelen **Add Connection** penceresinde daha önce oluşturmuş olduğumuz `sozluk` isimli Access veritabanımızı seçiyoruz.



Eğer veritabanı için kullanıcı adı ve şifre belirlemişsek bu bölümde gireriz.

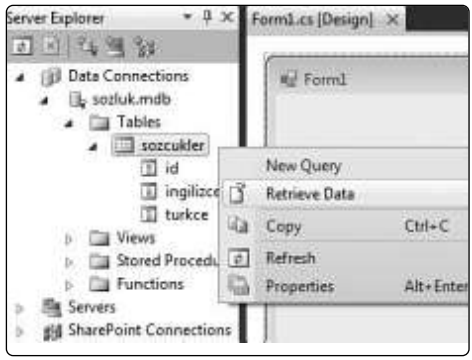
Bu pencerede varsayılan olarak **Microsoft Access Database File** (*OLE DB*) seçili durumdadır. Eğer değiştirmek istersek **Change** butonuna tıklayarak karşımıza gelen **Choose Data Source** penceresinden istediğimiz veritabanı türünü seçerek **Continue** butonuna tıklarız.

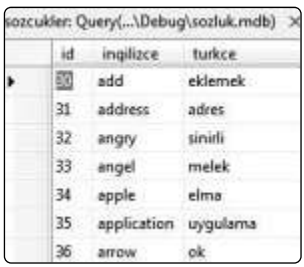




Seçtiğimiz veritabanı Server Explorer içerisine eklenecektir. Veritabanına ait tablo ve alanları listeleyerek görebilir ve alanlara veri girebiliriz.

Alanlara veri girişi yapmak için tablo ismi üzerine farenin sağ tuşu ile tıklayarak **Retrieve Data** seçeneğine tıklıyoruz.

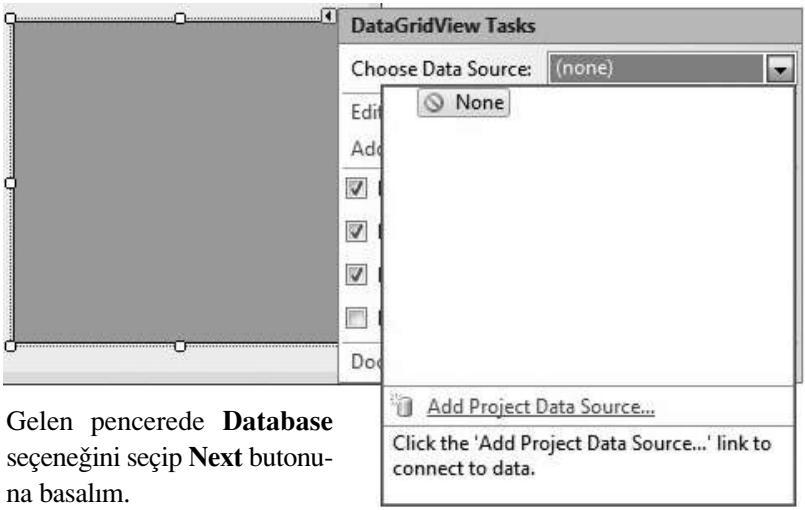




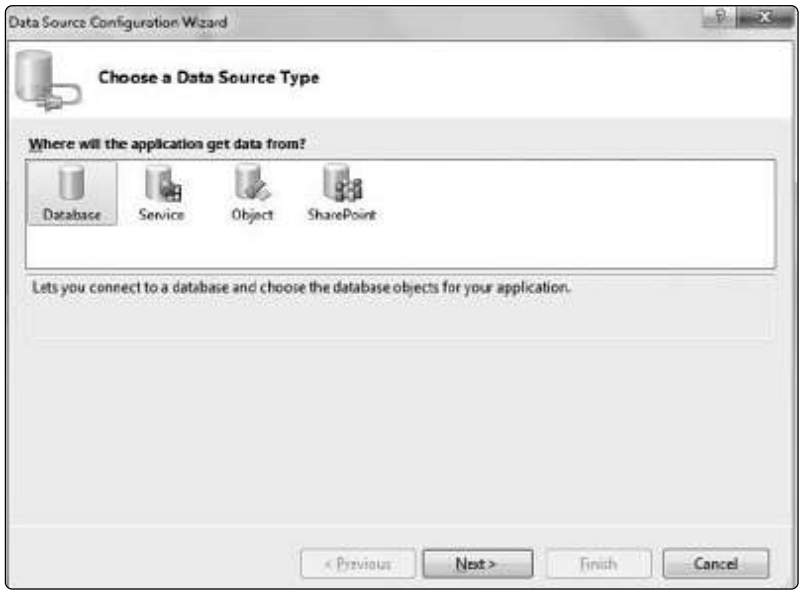
Ekrana seçili tablo içerisindeki kayıtların listesi gelmektedir. Burada yeni eleman ekleyebilir, kayıtlar üzerinde değişiklik yapabilir ya da kayıtları silebiliriz.

Kod yazmadan veritabanındaki nesnelere erişim için **Data Source Configuration**

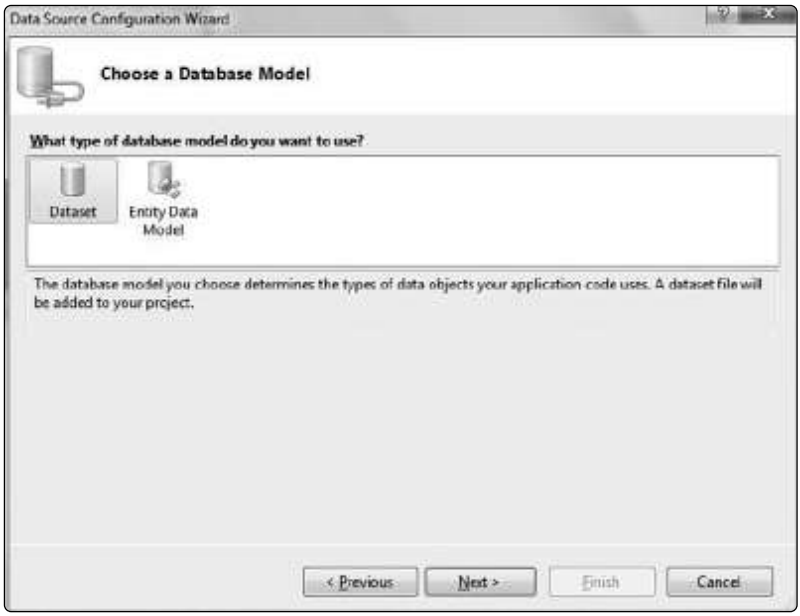
Wizard isimli aracı kullanacağız. Öncelikle veritabanımızdaki değerleri görüntülemek için form üzerine bir adet **DataGridView** nesnesi ekleyelim. DataGridView nesnesinin sağ üst tarafındaki kulakçığa tıklayarak **Choose Data Source** seçeneğine tıklayalım. Açılan pencereden **Add Project Data Source** yazısına tıklayalım.



Gelen pencerede **Database** seçeneğini seçip **Next** butonuna basalım.



DataSet simgesini tıklayalım.



Gelen pencerede daha önce oluşturmuş olduğumuz Access veritabanımızı Server Explorer içerisine eklediğimizden dolayı veritabanı ismi hazır olarak eklenmiş olacaktır. Bu pencerede **Connection String**'in (*veritabanı bağlantı cümlesi*) otomatik olarak oluşturulduğunu görebilirsiniz.

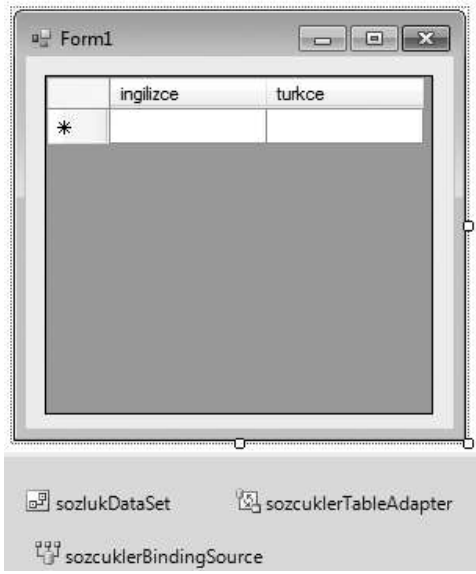


Gelen pencerede **connection string ismini** belirliyoruz.



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.



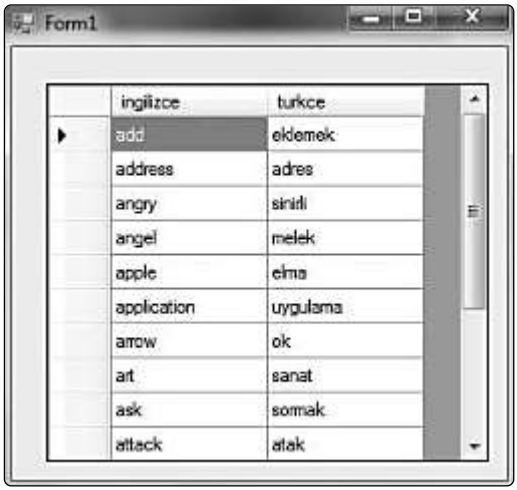


Veritabanı bağlantı işleminin bitmesinin ardından proje içerisine `Dataset`, `TableAdapter`, `BindingSource` nesnelerinin eklendiğini görürüz.

`Dataset` nesnesi ile veritabanı bağlantısının yapılacağı veritabanı seçilir.

`TableAdapter` nesnesi ile verilerin çekileceği tablo belirtilir.

`BindingSource` nesnesi ile veritabanındaki 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 dört 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. Veritabanımızın 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** (*bağlantı cümlesi*) oluşturmamız gerekmektedir. Access 97-2003 veritabanları için bağlantı cümlesi aşağıdaki biçimde yazılır:

```
OleDbConnection baglantı = new OleDbConnection
("Provider=Microsoft.Jet.Oledb.4.0;Data Source=" +
Application.StartupPath + "/sozluk.mdb")
```

Access 2007 veritabanları için ise bağlantı cümlesi aşağıdaki biçimde yazılır:

```
OleDbConnection baglantı = new
OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0;Data Source="
+ Application.StartupPath + "/sozluk.accdb")
```

Access 2007 veritabanına erişim sağlayabilmek için bilgisayarda Ofis 2007'nin ya da **2007 Office System Driver: Data Connectivity Components** isimli bileşenin yüklü olması gerekiyor.

KAYIT 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.Jet.Oledb.4.0;Data Source=" +
        Application.StartupPath + "/sozluk.mdb");
  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();
      }
  }
}
```

KAYIT 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.Jet.Oledb.4.0;Data Source=" +
        Application.StartupPath + "/sozluk.mdb");
  OleDbCommand komut = new OleDbCommand();
```

```
  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();
       }
  }
}
```

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.Jet.Oledb.4.0;Data Source=" +
```

```
        Application.StartupPath + "/sozluk.mdb");
  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
            ("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();
      }
  }
}
```

Kayıt ekleme, **güncelleme** ve **silme** işlemlerine dair kodlarımızı birleştirerek bir uygulama yapalım. Projemiz iki formdan oluşacak. İlk form ile şifreli giriş işlemi yapılacak. Eğer şifre doğru girilmiş ise Sözlük kodlarının yer aldığı form ekrana gelecek.

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 frmSifre()
    {
        InitializeComponent();
    }
    OleDbConnection baglantı = new OleDbConnection
        ("Provider=Microsoft.Jet.Oledb.4.0;Data Source=" +
        Application.StartupPath + "/sozluk.mdb");
    OleDbCommand komut = new OleDbCommand();
    string deger;

    private void btnGiris_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();
                Form1 frm1 = new Form1();
                frm1.Show();
            }
            else
            {
                MessageBox.Show("Şifre yanlış");
            }
            baglantı.Close();
        }
        catch
        {
        }

    }
    private void btnCikis_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.

Sözlük formumuzun kodu:

```
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.Jet.Oledb.4.0;Data Source=" +
            Application.StartupPath + "/sozluk.mdb");
        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();
            }
        }
    }
}
```



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. Ayrıca formun load olayı esnasında veritabanında bulunan kayıt sayısını bularak görüntülüyoruz.

Bu işlem için

```
"SELECT COUNT(*) FROM sozcukler"
```

sorgusunu kullanıyoruz. 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.

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 ingilizce='" + 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.

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 projemizde dahil etmemiz gerekmektedir.

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 datasete 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.Jet.Oledb.4.0;Data Source=" +
      Application.StartupPath + "/sozluk.mdb");
  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 datasete 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.

KAYIT 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 datasete aktarılmaktadır.

```
using System.Data;
using System.Data.OleDb;
public partial class Form1 : Form
{
  OleDbConnection baglantı = new OleDbConnection
      ("Provider=Microsoft.Jet.Oledb.4.0;Data Source=" +
      Application.StartupPath + "/sozluk.mdb");
  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 datasete aktarılmaktadır. Daha sonra veritabanı ile bağlantı kurulduğunda değişmiş olan veriler, veritabanındaki eski değerleri ile değiştirilir.

```
using System.Data;
using System.Data.OleDb;
public partial class Form1 : Form
{
  OleDbConnection baglantı = new OleDbConnection
     ("Provider=Microsoft.Jet.Oledb.4.0;Data Source=" +
     Application.StartupPath + "/sozluk.mdb");
  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.Jet.Oledb.4.0;Data Source=" +
      Application.StartupPath + "/sozluk.mdb");
  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();
    }
  }
}
```

KAYIT 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.

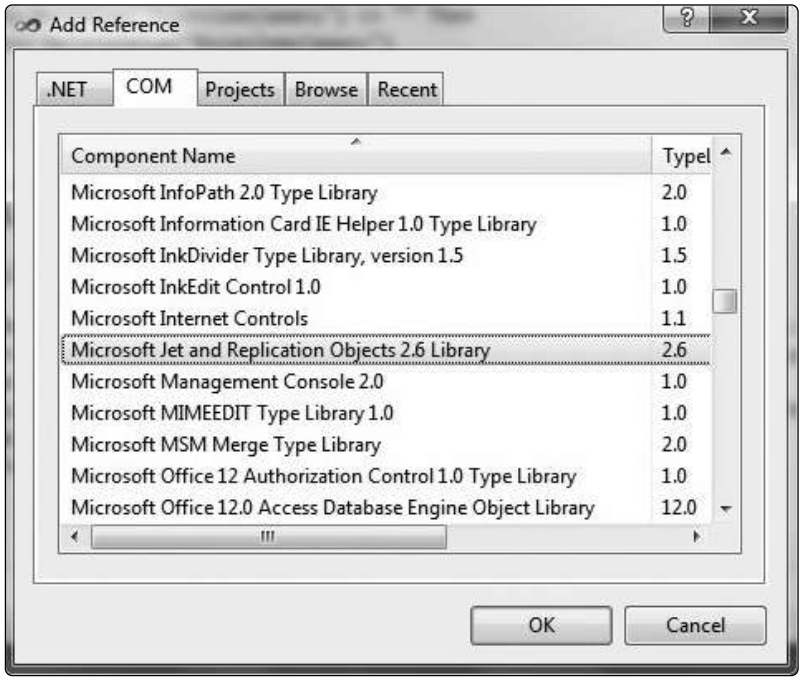
```
using System.Data;
using System.Data.OleDb;
public partial class Form1 : Form
{
  OleDbConnection baglantı = new OleDbConnection
    ("Provider=Microsoft.Jet.Oledb.4.0;Data Source=" +
    Application.StartupPath + "/sozluk.mdb");
```

```
  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();
    }
  }
}
```

VERİTABANINI DÜZENLEME

Access veritabanları kullanıldıkça kayıt ekleme, silme işlemleri sonucu kayıt sayısına bağlı olmayan boyut artması meydana gelmektedir. Bu da zamanla ver itabanına 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** sekmesinden *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. ↓
      Jet.OLEDB.4.0;Data Source=" +
         Application.StartupPath + "/sozluk.mdb",
         "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
         Application.StartupPath + "/yenisozluk.mdb;" +
         "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");
  }
}
```

Kayıt ekleme, güncelleme ve silme işlemlerine dair kodlarımızı birleştirerek bir uygulama yapalım.

Projemiz iki formdan oluşacak. İlk formda veritabanındaki kayıtlar üzerinde arama işlemi gerçekleştirilerek ingilizce sözcükler ve buna karşılık gelen Türkçe sözcükler listelenecek. Ayrıca ilk formda veritabanı üzerinde düzenleme işlemi gerçekleştirilecek. Ancak bu işlem her ayın ilk günü bir defa gerçekleştirilecek. İkinci formda ise kayıt ekleme, güncelleme, silme işlemleri yapıalcak.

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.Jet.Oledb.4.0;Data Source=" +
       Application.StartupPath + "/sozluk.mdb");

    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.Jet.OLEDB.4.0;Data Source=" +
         Application.StartupPath + "/sozluk.mdb",
         "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
         Application.StartupPath + "/yenisozluk.mdb;" +
```

```
            "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");
      }
    }
}
```

Kayıt ekleme, güncelleme, silme gibi işlemleri yaptığımız `frmAyar` isimli ikinci 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.Jet.Oledb.4.0;Data Source=" +
        Application.StartupPath + "/sozluk.mdb");

    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);

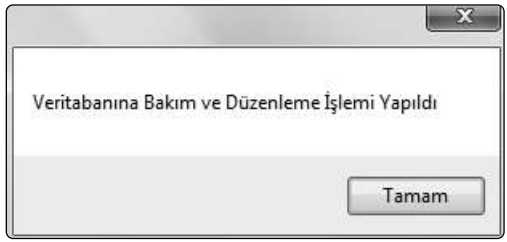
            baglantı.Close();
            MessageBox.Show("Kayıt Silindi","Bilgi");
          }
          catch (Exception ex)
          {
            MessageBox.Show(ex.Message);
            baglantı.Close();
          }
        }

        private void btnAnasayfa_Click(object sender, EventArgs e)
        {
          this.Close();
        }
      }
    }
```

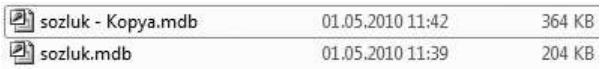
Programın ilk çalıştırılışı esnasında ekrana *"Program Bu Bilgisayarda İlk Defa Çalışıyor"* şeklinde bir mesaj penceresi gelecektir.



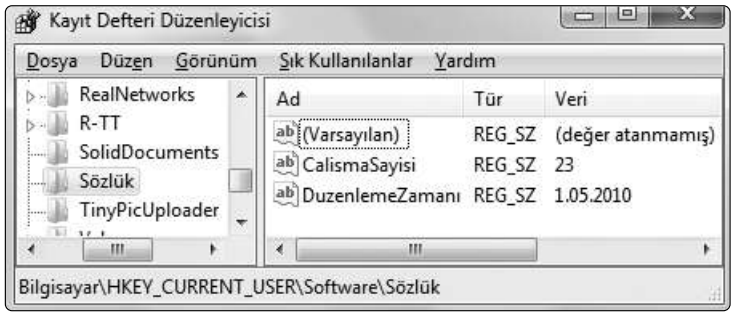
Ayrıca her ayın ilk gününde veritabanı düzenlenip ekrana *"Veritabanına Bakım ve Düzenleme İşlemi Yapıldı"* şeklinde bir mesaj penceresi gelecektir.



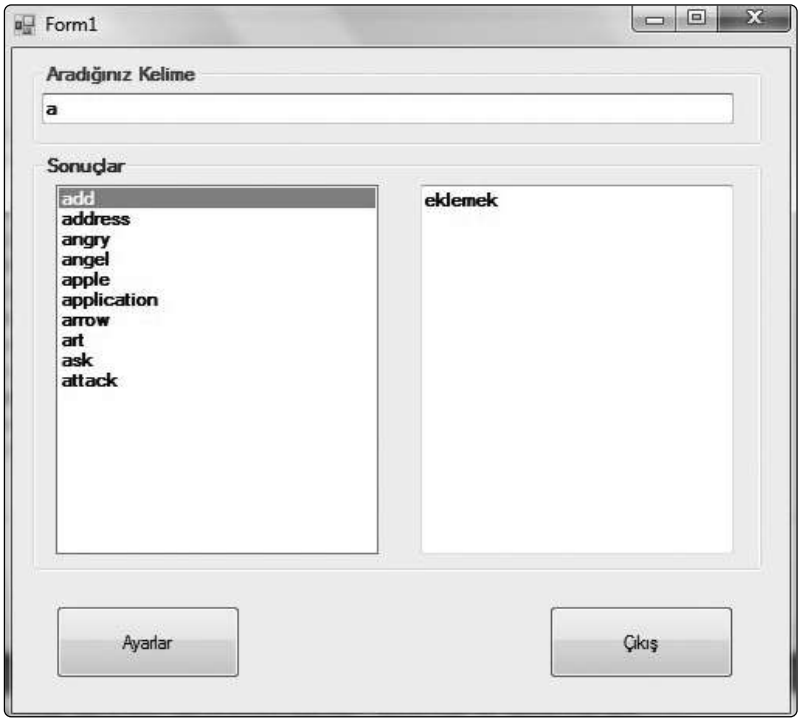
Veritabanının düzenleme öncesi boyutu `364 kb` iken, düzenleme işlemi sonrası `204 kb`'a inmiştir.



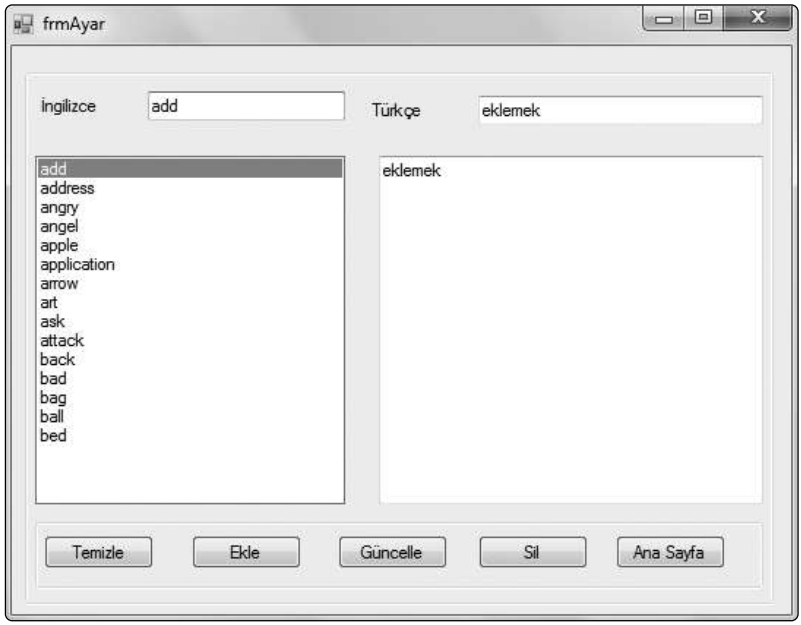
Registry içerisine baktığımızda düzenleme zamanı, çalışma sayısı bilgilerinin kaydedildiğini görürüz.



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.

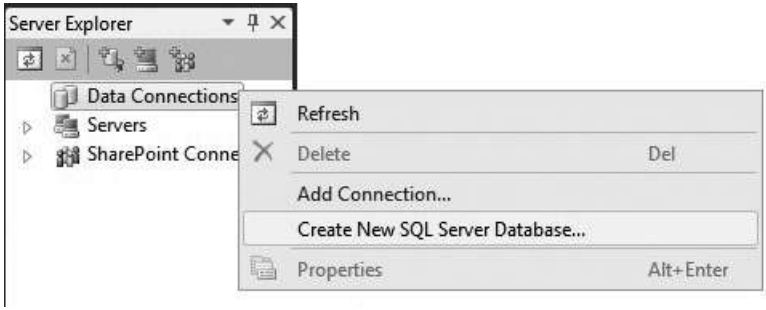


SQL VERİTABANI İŞLEMLERİ

SQL veritabanlarına erişim için kullanılabilecek veritabanı nesneleri şunlardır:

- `SqlConnection` nesnesi
- `SqlCommand` nesnesi
- `SqlDataAdapter`
- `DataReader`
- `DataSet`

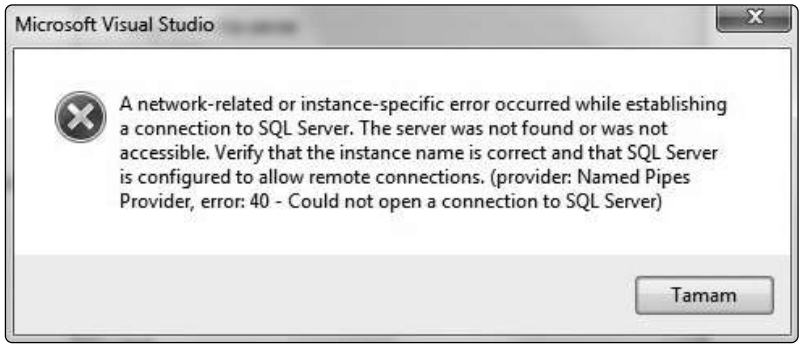
Veritabanı işlemlerimizde kullanacağımız örnek bir SQL veritabanı tasarlayalım. Veritabanımız bir stok takip uygulamasının veritabanı olacak. **Server Explorer** içerisinde **Data Connections** seçeneğine farenin sağ tuşu ile tıklayarak **Create New SQL Server Database** seçeneğine tıklıyoruz.



Server adını seçip, veritabanımızın adını stok olarak belirliyoruz.

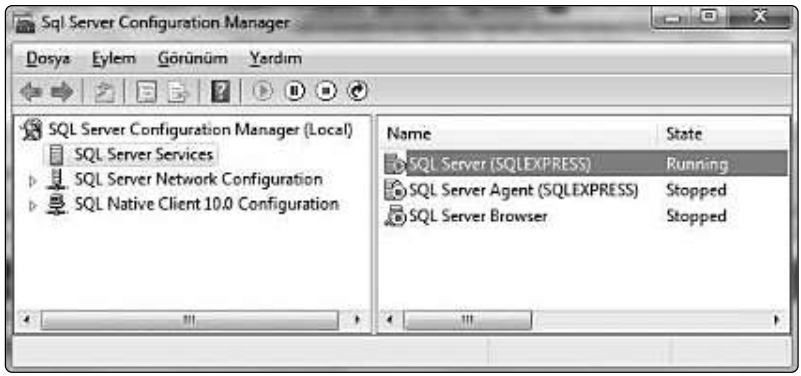


Bir sonraki adımda SQL veritabanı ile ilgili **remote connection hatası** meydana gelebilir. Bunun nedeni SQL Server içerisinde remote connection izni verilmediğinden kaynaklanabilir.



Bu izni vermek için **Başlat / Programlar / Microsoft SQL Server 2008 / Configuration Tools / SQL Server Configuration Manager** aracını açıyoruz.

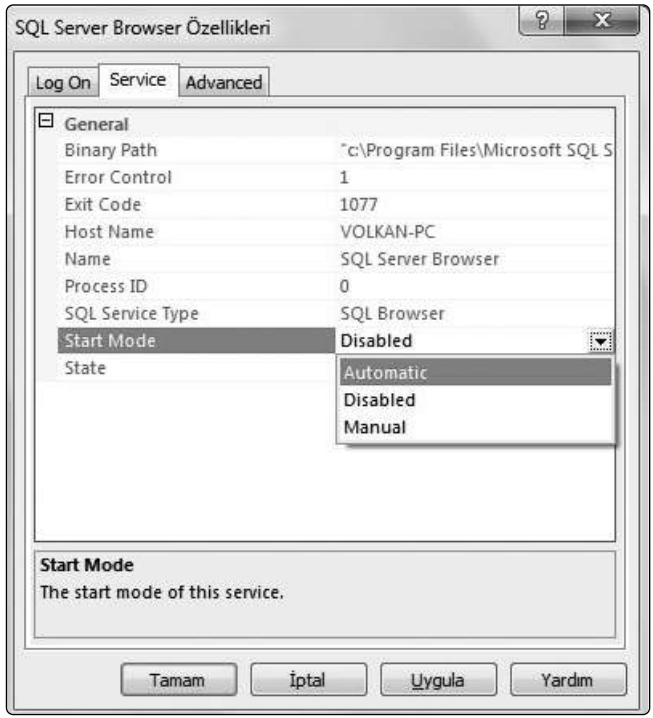
Görüldüğü gibi SQL Server Browser, çalışma modunda (*Running*) değildir. Bunu çalıştıracağız.



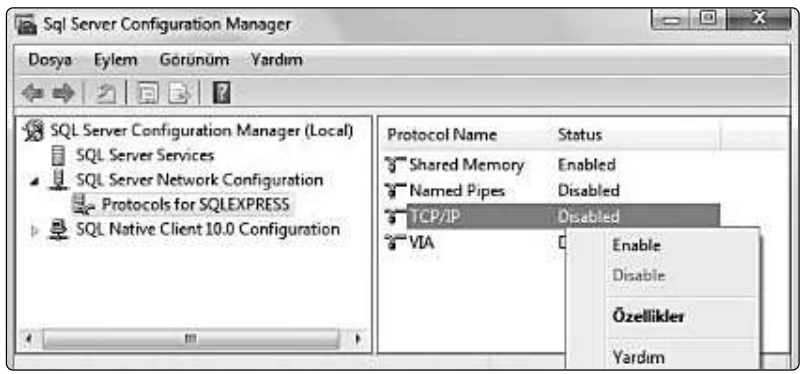
SQL Server Browser ismine çift tıklayarak özellikler penceresini görüntülüyoruz. **Log On** sekmesinde **Built-in account** seçeneğinden **Local Service** seçeneğini seçiyoruz.



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.

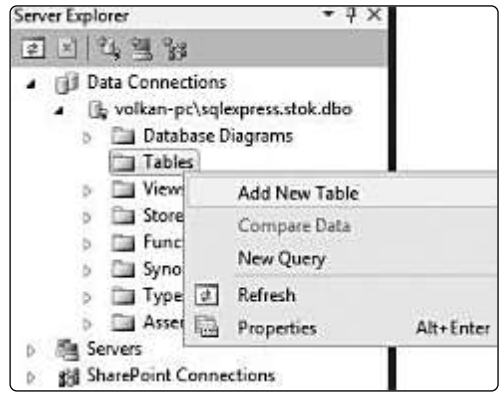


SQL Server Browser'a farenin sağ tuşu ile tıklayarak **Start** seçeneğini seçip SQL Server Browser'ı başlatıyoruz.



Bu işlemden sonra **SQL Server Configuration Manager** penceresini kapatarak SQL veritabanı oluşturma işlemini yeniden başlattığımızda hata meydana gelmeden boş bir SQL veritabanı oluşturulacaktır.

Server Explorer içerisinde **Data Connections** bölümünde oluşturmuş olduğumuz veritabanı ismine tıklıyoruz. Açılan listede **Tables** seçeneğine farenin sağ tuşu ile tıklayarak **Add New Table** seçeneğini tıklayarak veritabanımızda tablo ve alanları oluşturacağız.



Tablomuzdaki alanları aşağıdaki gibi oluşturuyoruz.

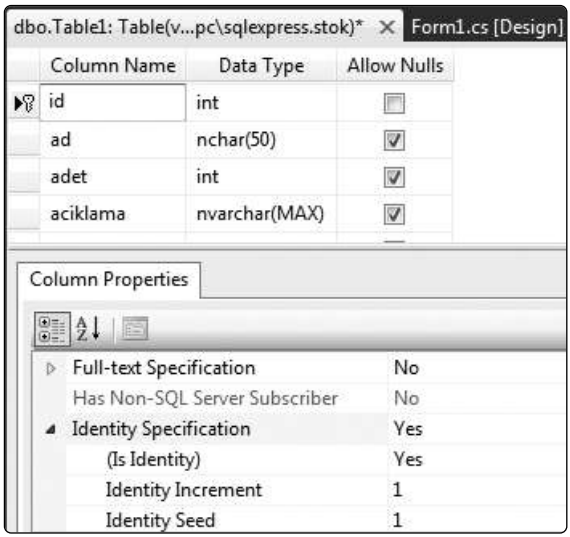
`id` alanında **Allow Nulls** seçeneğinin işaretini kaldırarak boş değer girilmesini engelliyoruz. Diğer alanları da işaretleyerek boş değer giriş izni veriyoruz.

`id` alanındaki değerlerin otomatik olarak arttırılması için **Identity Specification** seçeneğini **Yes** olarak seçiyoruz.

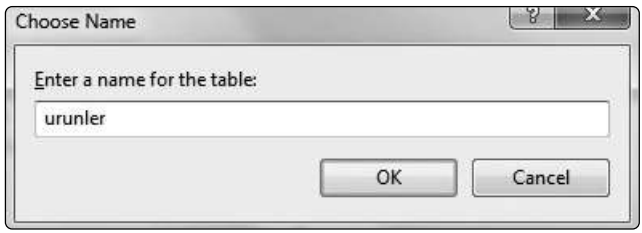
Identity Increment seçeneği artış değerinin kaçar kaçar olacağını belirtir.

Identity Seed seçeneği ise artış işleminin hangi rakamdan itibaren başlatılacağını belirtir.

`id` alanına farenin sağ tuşu ile tıklayarak **Set Primary Key** seçeneğine tıklayıp birincil anahtar olarak belirlenmesini sağlıyoruz.



Tablomuzu kayıt edip urunler ismini veriyoruz.



Alanlar üzerinde daha sonra düzenleme yapmak için tablo adına farenin sağ tuşu ile tıklayarak **Open Table Definition** menüsünü kullanabiliriz. Alanlara kayıt girmek için tablo adına farenin sağ tuşu ile tıklayarak **Show Table Data** seçeneğini kullanabiliriz.

urunler: Query(volk...pc\sqlexpress.stok) | Form1.cs [Design]

id	ad	adet	aciklama	fiyat
1	parsumen ...	100	cizgili	0,25
2	parsumen ...	100	cizgisiz	0,25
3	kursun kalem ...	500	siyah	1
4	kursun kalem ...	500	kırmızı	1
5	boyama kalemi ...	50	12'lik	2

`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 Server Explorer içerisindeki veritabanı ismine tıklayıp **Properties** penceresinde **Connection String** özelliğine bakabiliriz.

Properties

volkan-pc\sqlexpress.stok.dbo Connection

(Name)	stok
Case Sensitive	False
Connection String	Data Source=VOLKAN-PC\SQLEXPRESS;Initial Catalog
Owner	volkan-pc\volkan
Provider	.NET Framework Data Provider for SQL Server
State	Open
Type	Microsoft SQL Server
Version	10.00.2531

Stok örneği için kullanacağımız bağlantı cümlesi şu şekildedir:

```
SqlConnection baglantı = new SqlConnection("Data Source=
VOLKAN-PC\\SQLEXPRESS; Initial Catalog=stok;
Integrated Security=True; Pooling=False");
```

SQLCOMMAND

SQL veritabanı işlemleri ile ilgili kullanılan sorguları çalıştırmak için kullanılır.

`OleDbCommand` nesnesinde olduğu gibi komutları yürütmek için dört 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.

KAYIT LİSTELEME

Veritabanındaki 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=VOLKAN-PC\\SQLEXPRESS;Initial Catalog=stok;
    Integrated Security=True;Pooling=False");
  SqlCommand komut = new SqlCommand();
  SqlDataAdapter adaptor = new SqlDataAdapter();

  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.GetString(1).ToString();
    }
```

```
    reader.Close();
    adaptor.SelectCommand = new SqlCommand
      ("select ad,fiyat from urunler ", baglantı);
    adaptor.Fill(ds);
    dataGridView1.DataSource = ds.Tables[0];
    baglantı.Close();
  }
}
```

Texbox 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 `reader.GetString(1)` metodlarının her ikisi ile birden okunan verilerin textbox içerisine aktarımı yapılır.

KAYIT 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=VOLKAN-PC\\SQLEXPRESS;Initial Catalog=stok; ↓
    Integrated Security=True;Pooling=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();
      }
    }
}
```

KAYIT 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=VOLKAN-PC\\SQLEXPRESS;Initial Catalog=stok; ↓
    Integrated Security=True;Pooling=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();
      }
    }
}
```

KAYIT SİLME

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=VOLKAN-PC\\SQLEXPRESS;Initial Catalog=stok; ↓
    Integrated Security=True;Pooling=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 birleştirerek bir uygulama yapalım.

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=VOLKAN-PC\\SQLEXPRESS;Initial Catalog=stok; ↓
        Integrated Security=True;Pooling=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.GetString(1).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();
  }
}

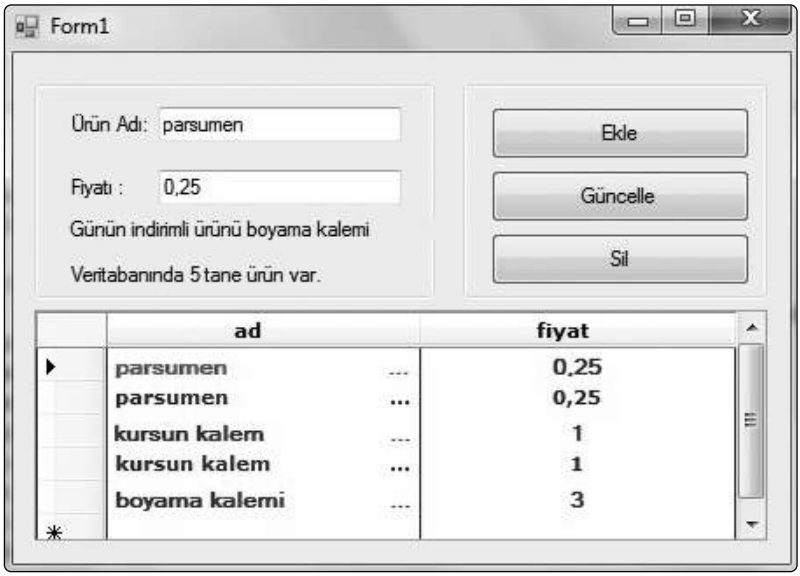
private void btnGuncelle_Click(object sender, EventArgs e)
{
  try
  {
    DataSet ds = new DataSet();
    if (baglantı.State == ConnectionState.Closed)
     baglantı.Open();
    ds.Clear();

    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 veritabanındaki 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 dört 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.

KAYIT LİSTELEME

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=VOLKAN-PC\\SQLEXPRESS;Initial Catalog=stok; ↓
    Integrated Security=True;Pooling=False");
  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 datasete 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["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.

KAYIT 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=VOLKAN-PC\\SQLEXPRESS;Initial Catalog=stok; ↓
    Integrated Security=True;Pooling=False");
  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=VOLKAN-PC\\SQLEXPRESS;Initial Catalog=stok; ↓
    Integrated Security=True;Pooling=False");
  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=VOLKAN-PC\\SQLEXPRESS;Initial Catalog=stok; ↓
    Integrated Security=True;Pooling=False");
  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 birleştirerek bir uygulama yapalım.

`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=VOLKAN-PC\\SQLEXPRESS;Initial Catalog=stok;
      Integrated Security=True;Pooling=False");
    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

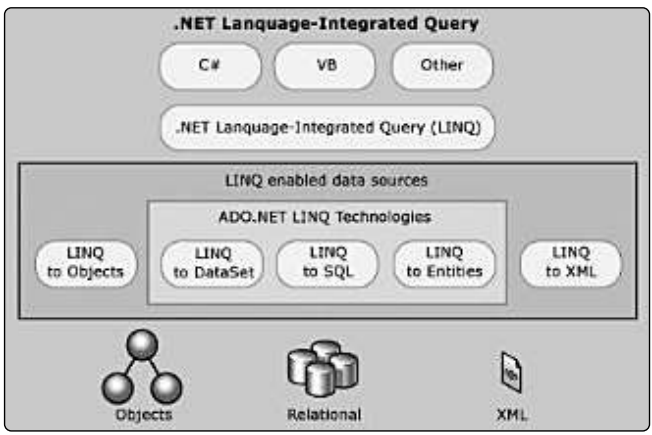
```
"DELETE FROM urunler WHERE ad='" + txt_ad.Text + "'"
```

sorgusu ile `urunler` tablosundaki `ad` alanındaki veri `txt_ad` adlı text kutusundaki değere eşit ise bu satırdaki veri silinir.

26

LINQ (LANGUAGE INTEGRATED QUERY)

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` adlı 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 alıp 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 `structure` oluşturup bu yığına değer atayalım. Bu başlık altındaki örneklerimizi bu yığın sınıfı üzerinden yapacağız.

```
using System;
using System.Linq;
```

```
using System.Text;
namespace linq2
{
  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.

Öğrencilerin hem ad hem de soyadlarını görüntülemek için sorgumuzu aşağıdaki gibi değiştiriyoruz.

```
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());
}
```

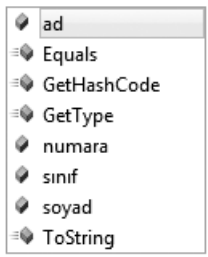


Yukarıdaki örneğimizde öğrencilerin yalnızca ad bilgisine erişmiştik. Ad, soyad, sınıf, numara bilgilerinin hepsine birden erişebilmek için select kısmını biraz değiştiriyoruz.

```
var gelenogr = from ogr in ogrenciler
                           where
                                ogr.sınıf == "3-C"
                           select ogr;
```

```
foreach(var isim in gelenogr)
{

    Console.WriteLine(isim.
```



Gördüğünüz gibi ad, soyad, numara ve sınıf bilgilerinin hepsine birden eriştik.

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 yığınımızdaki 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` metodunu 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 OPERATÖRÜ

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.

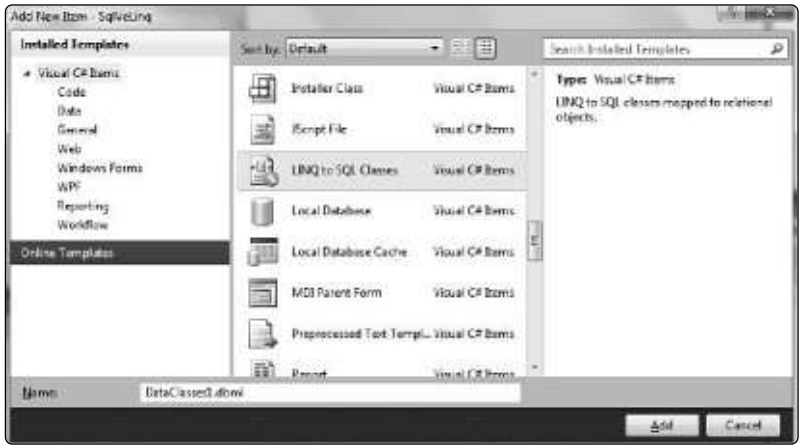
```
var sayı = (from ogr in ogrenciler where
                    ogr.numara>140 select ogr.numara).Count();
Console.WriteLine
   ("Numarası 140'dan büyük olanların sayısı= " +sayı);
   Console.ReadKey();
```

Örneğimiz `ogrenci` adlı yapı içerisindeki numarası `140`'dan büyük olanların sayısını bularak ekrana yazdırıyoruz.

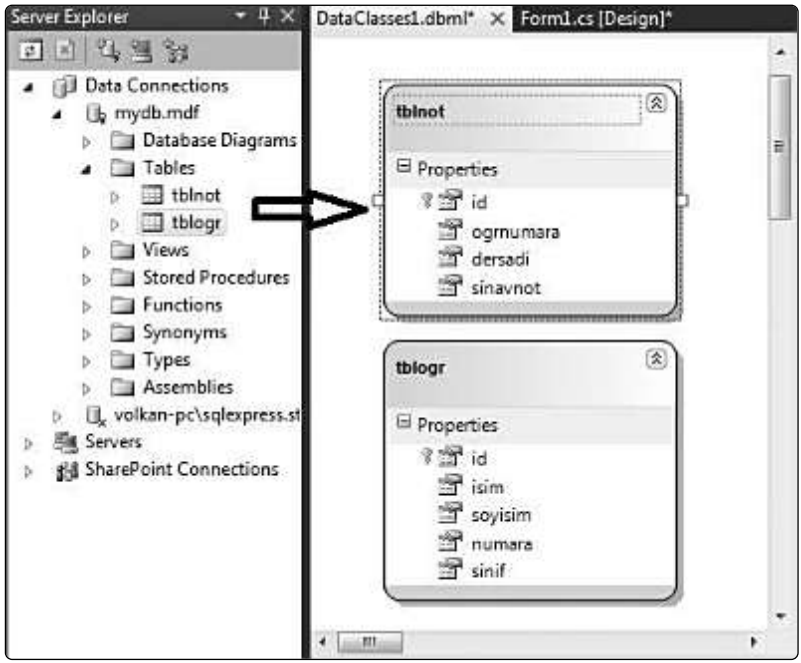


SQL ve LINQ

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 projemize sağ tıklayıp **Add >> New Item** dedikten sonra bir adet **LINQ to SQL classes** ekliyoruz.



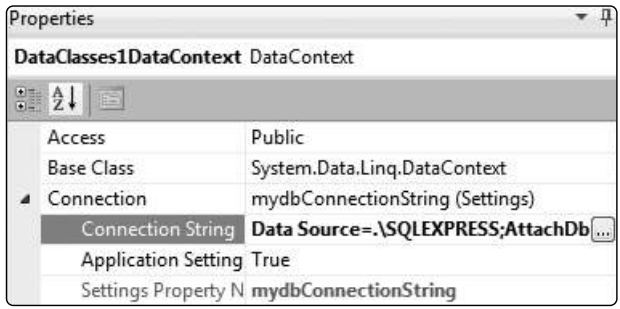
Tablolarımızı oluşan dosya üzerine sürükleyerek atıyoruz.



Artık kod kısmına geçebiliriz.

Kayıt ekleme kodumuz aşağıdaki gibi olacak:

```
string cs = "Data Source=.\\SQLEXPRESS;
  AttachDbFilename=D:\\kitaplar\\c#\\ornek\\linq\\ ↓
  SqlVeLinq\\mydb.mdf; ↓
  Integrated Security=True;User Instance=True";
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 seçili iken 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.

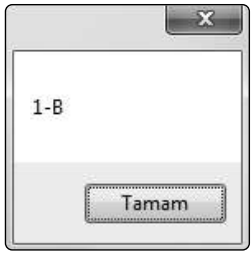
166 numaralı öğrencinin sınıfını bulalım:

```
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());
```



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();
```



Update olayında dikkat edilirse mevcut kaydı alıp üzerinde değişiklik yapmak sureti ile update olayını gerçekleştirdik.

166 numaralı öğrenciyi silelim:

```
string cs = "Data Source=.\\SQLEXPRESS; ↓
  AttachDbFilename=D:\\kitaplar\\c#\\ornek\\linq\\SqlVeLinq\\mydb.mdf; ↓
  Integrated Security=True;User Instance=True";
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 belirtmiştik. 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. Birinci olarak 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 `Kisi` 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 sonuç 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>
```

27

CRYSTAL REPORTS

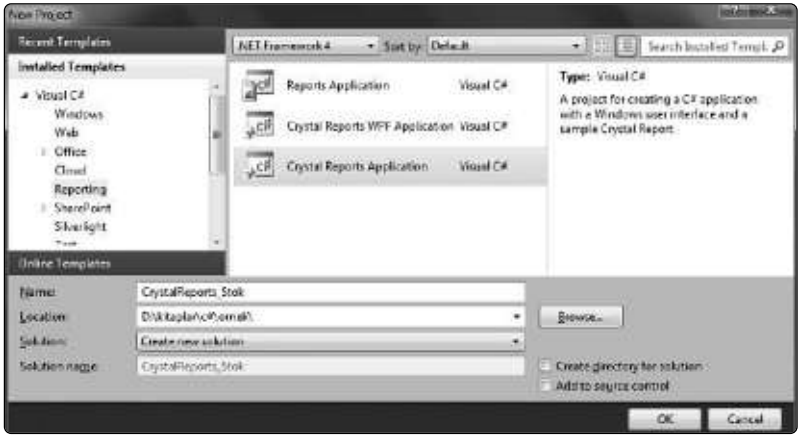
Crystal Reports, verilerinizi raporlamak, grafiklerle ifade etmek için pek çok kullanışlı araca sahip olan bir raporlama aracıdır. .NET altında bulunan crystal report şablonunu kullanarak kolayca raporlama işlemelerinizi yapabilirsiniz.

Öncelikle *http://www.businessobjects.com/jump/xi/crvs2010/default.asp* adresinden **Crystal Reports for Visual Studio 2010** adlı dosyayı indirip kurmamız gerekmektedir. Örneğin `stok` isimli bir Access veritabanı oluşturup, bu veritabanını kullanarak bir rapor oluşturalım.

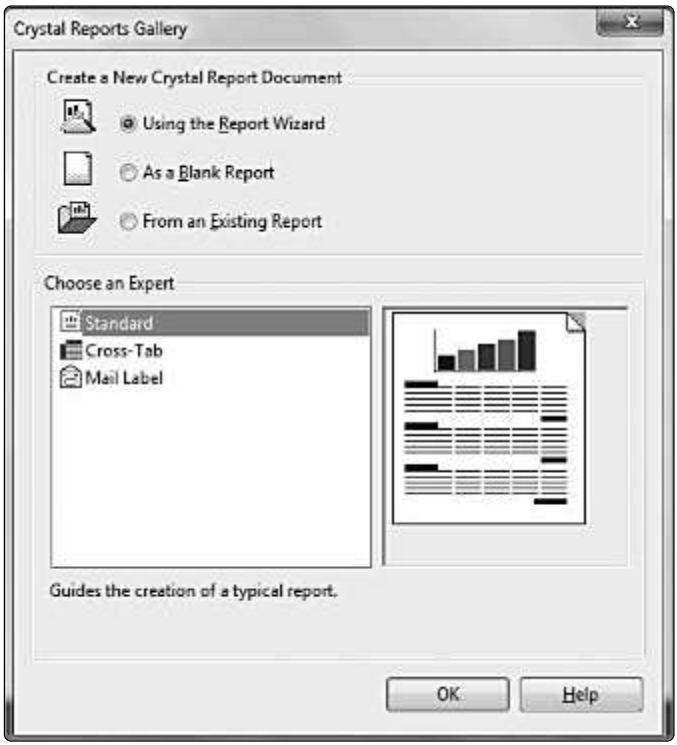
Veritabanının tablo yapısı yandaki gibidir:



Yeni bir C# projesi açarak **Reporting** başlığı altındaki **Crystal Reports Application** isimli proje tipini seçelim.



Karşımıza ilk olarak **Crystal Reports Gallery** başlıklı pencere gelmektedir.



Bu pencerede;

Using the Report Wizard seçeneği ile, rapor sihirbazını kullanarak otomatik olarak bir rapor oluşturulabilir.

As a Blank Report seçeneği ile boş bir rapor oluşturularak daha sonra bu rapor üzerinde değişiklikler ve eklemeler yapılabilir.

From an Existing Report seçeneği ile daha önceden oluşturulmuş olan bir rapor dosyası açılabilir.

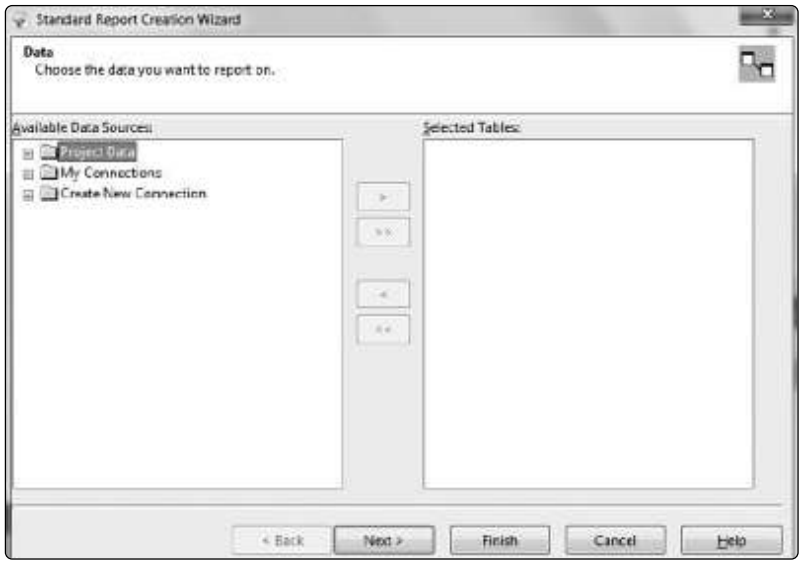
Report Wizard (Rapor Sihirbazı) Kullanarak Rapor Oluşturma

Rapor Sihirbazı altında üç tane seçenek vardır:

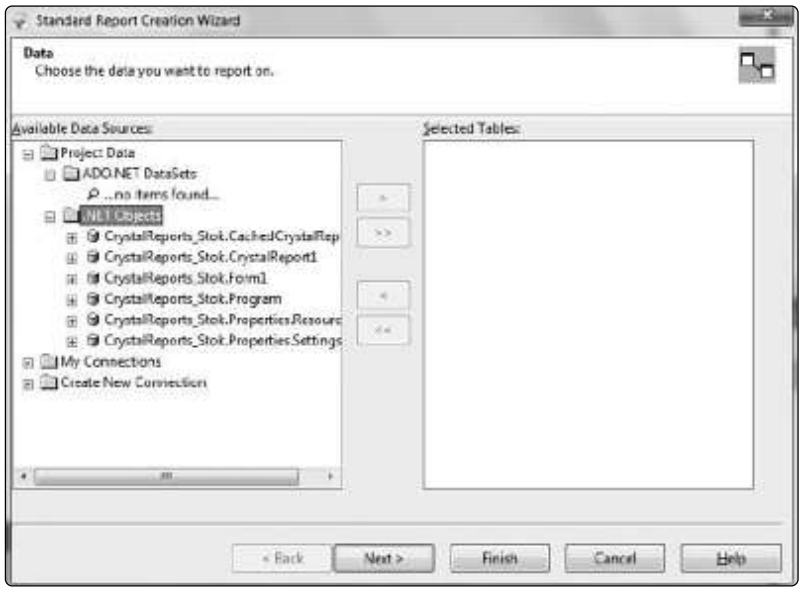
- **Standard:** Sütunlardan oluşan ve gruplandırma, sıralama, ve özetleme gibi işlevlere sahip standart bir rapor hazırlamak amacı ile kullanılır.
- **Cross-Tab:** Çarpraz sorgular içeren rapor hazırlamak amacı ile kullanılır.
- **Mail Label:** Mektup etiketleri hazırlamak amacı ile kullanılır.

Standard

Bu seçeneği seçtiğimizde karşımıza **Standard Report Creation Wizard** başlıklı pencere gelecektir.



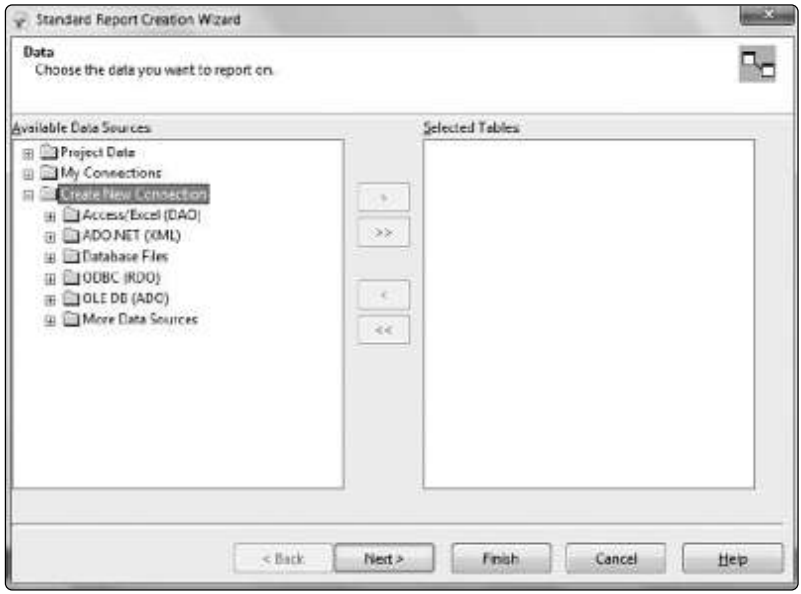
Bu pencere altındaki **Project Data** seçeneği altında iki adet başlık mevcuttur;



- **ADO .NET DataSets** başlığı altında, eğer proje içerisinde veritabanı ile bağlantı kurmak için önceden hazırladığımız bir dataset mevcutsa listelenir.
- **.NET Objects** başlığı altında ise projemizde mevcut olan .NET nesneleri listelenir.

My Connections başlığı altında, eğer veritabanı ile bağlantı kurmak için önceden oluşturduğumuz bir **connection string** (*bağlantı cümlesi*) mevcutsa listelenir.

Create New Connection başlığına tıklayarak ise, yeni bir veritabanı oluşturarak onun ile bağlantı kurabilir ya da daha önceden oluşturmuş olduğumuz bir veritabanı ile bağlantı kurabiliriz.



Buradaki seçeneklerden;

Access / Excel (DAO) ile yeni bir Access ya da Excel dosyası oluşturabiliriz.

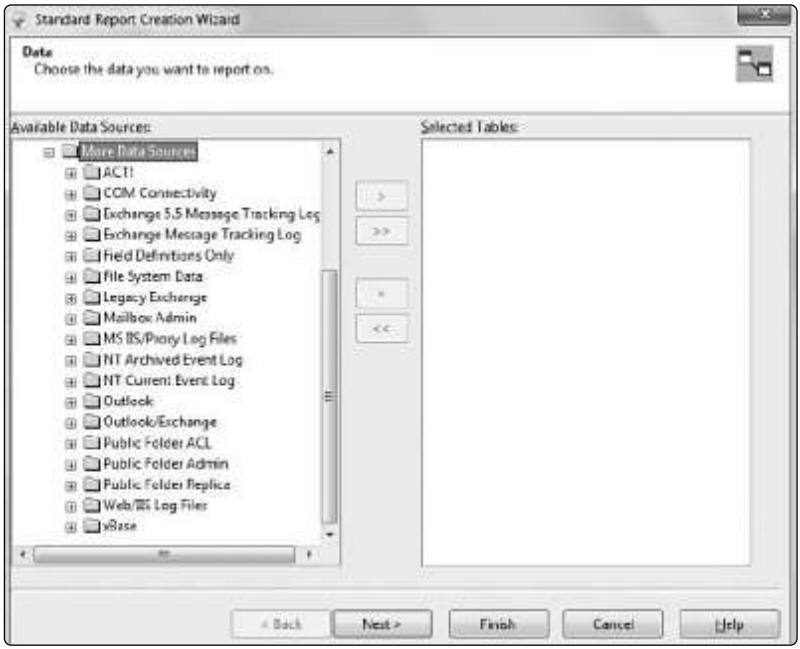
ADO.NET (XML) ile mevcut bir XML dosyasını veritabanı olarak kullanabiliriz.

Database Files seçeneği ile daha önceden oluşturmuş olduğumuz bir veritabanını seçebiliriz.

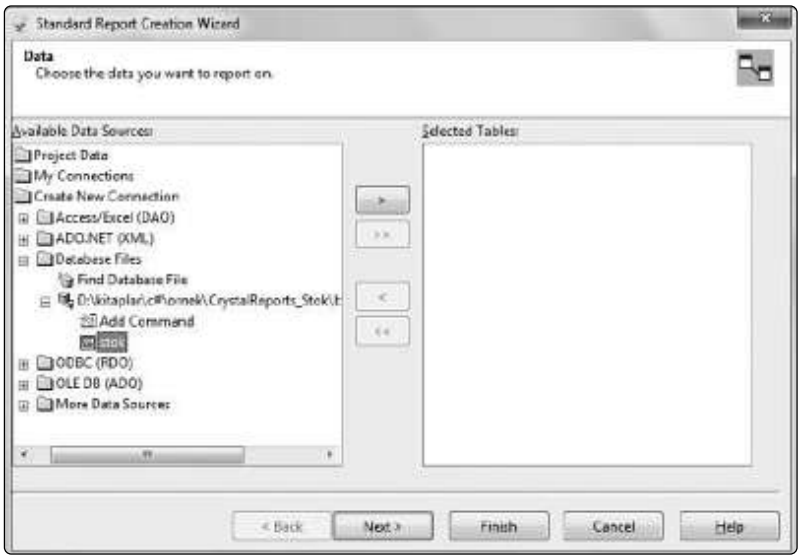
ODBC (RDO) seçeneği ile ODBC bağlantısını kullanarak veritabanları ile bağlantı kurabiliriz.

OLE DB (ADO) seçeneği ile OLE DB bağlantısını kullanarak veritabanları ile bağlantı kurabiliriz.

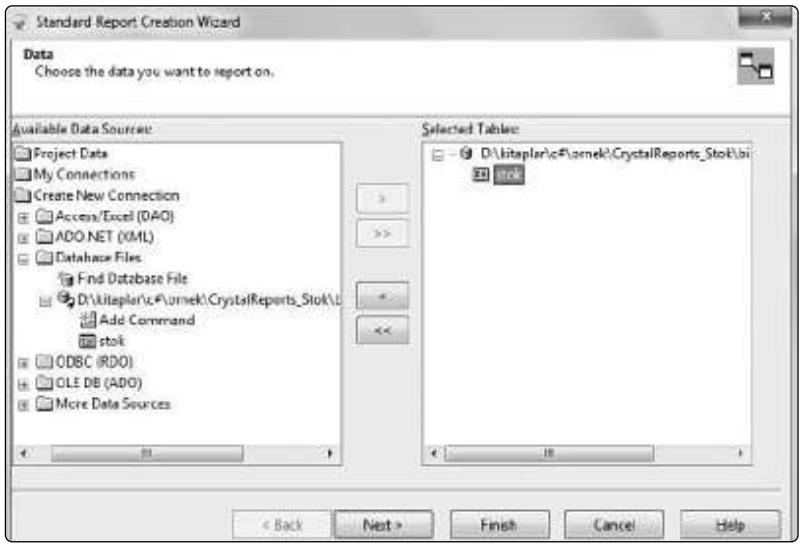
More Data Sources seçeneği ile ise, yukarıdaki seçenekler dışındaki veritabanları listelenir. Pekçok veritabanına destek verilmektedir.



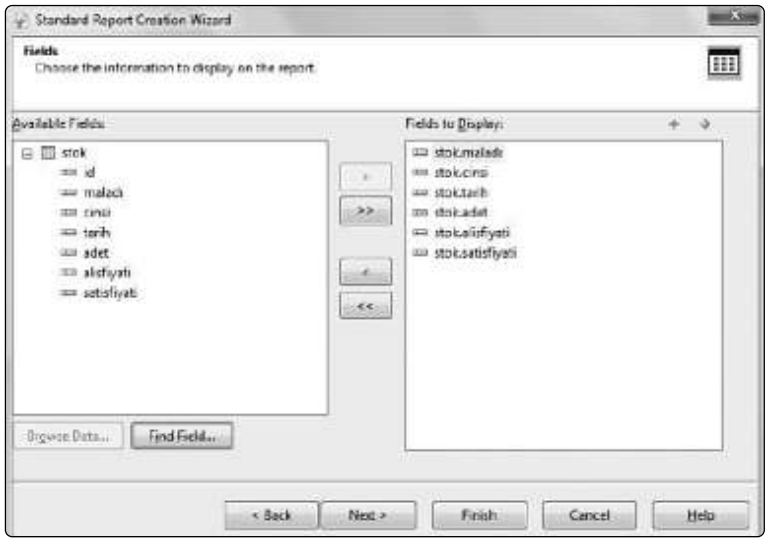
Yapmakta olduğumuz örneğimiz için **Database Files** seçeneğini seçelim ve gelen pencereden stok.mdb isimli Access veritabanımızı seçelim.



Veritabanına ait tablo isimleri pencerenin sol tarafında listelenecektir. İster tablo ismine çift tıklayarak istersek üzerinde sağa doğru ok işareti bulunan butona tıklayarak üzerinde işlem yapacağımız tabloları seçebiliriz.

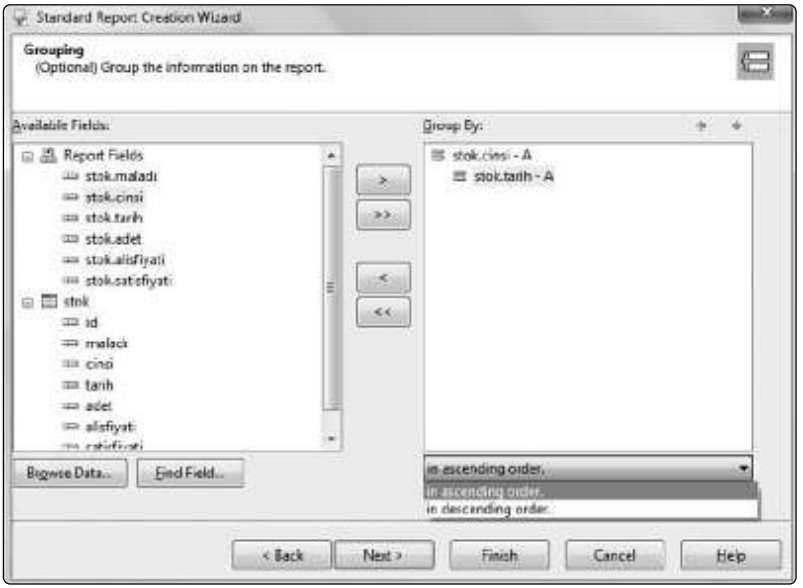


Bu işlemin ardından **Next** butonu ile tabloların içerisindeki alanları seçme aşamasına geçeriz.



Yine aynı şekilde ister alan ismine çift tıklayarak istersek üzerinde sağa doğru ok işareti bulunan butona tıklayarak üzerinde işlem yapacağımız alanları seçebiliriz. Ya da çift ok ile tüm alanları seçebiliriz.

Bir sonraki aşamada ise alanları gruplandırmak için kullanacağımız alanları seçeriz. Örneğin; ben tarih ve malın cinsi alanlarına göre gruplandırma yapmak istediğim için bu alanları seçiyorum.

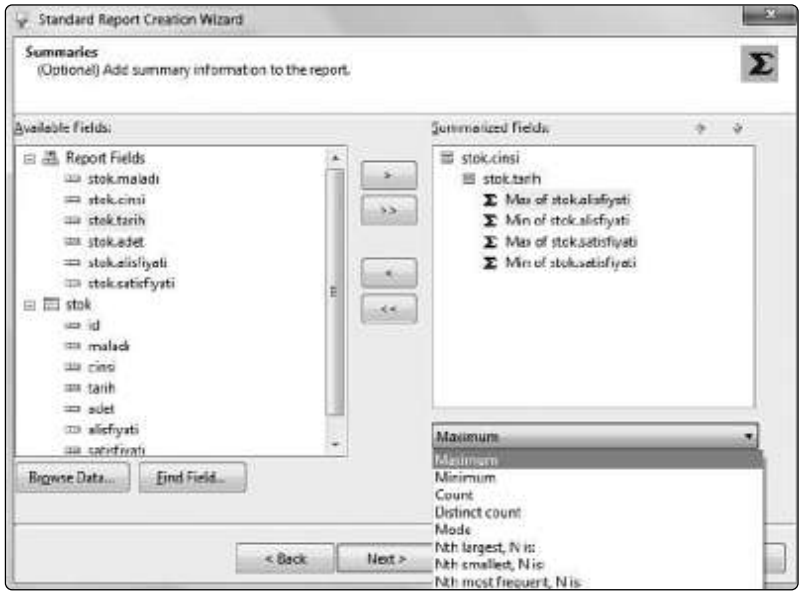


Burada **in ascending order** seçeneği ile artan biçimdeki **in descending order** seçeneği ile de azalan şekilde sıralama yapabiliriz.

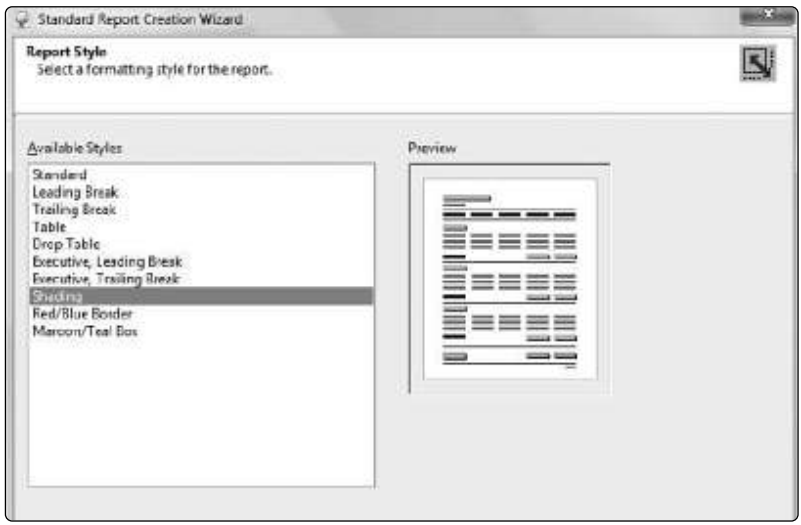
Ayrıca herhangi bir alan seçili iken **Browse Data** butonuna tıklayarak alan içerisinde mevcut olan verileri listeleyebiliriz.

Bir sonraki adımda gösterilecek olan özet işlemleri için alan seçimi ve özet işlemi türünü seçebiliriz.

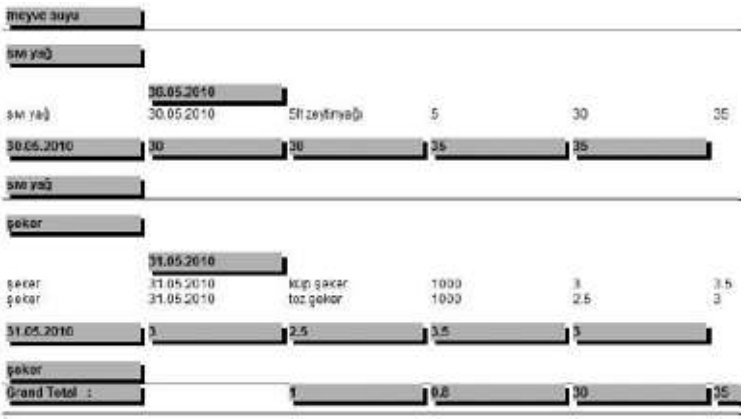




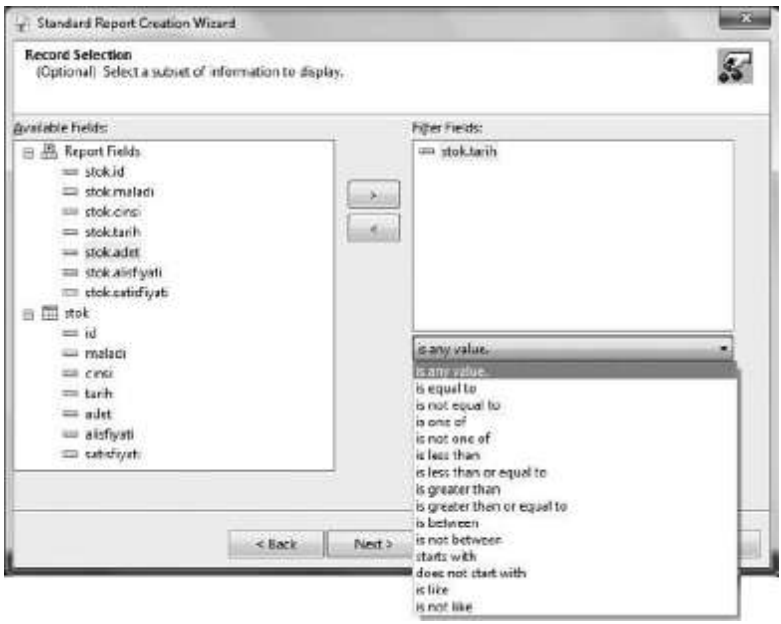
Örneğin ben alış ve satış fiyatının maksimum ve minimum değerlerini görüntületiyorum. Bir sonraki aşamada grup bazlı sıralama için bir biçim seçmek istiyorsak o biçimi seçeriz. Ben herhangi bir biçim seçmek istemediğim için boş bıraktım Sonraki aşamada rapor sitilini seçeriz.



Grouping işlemi sonucunda elde edeceğimiz rapor dosyasının önizlemesi aşağıdaki gibidir:



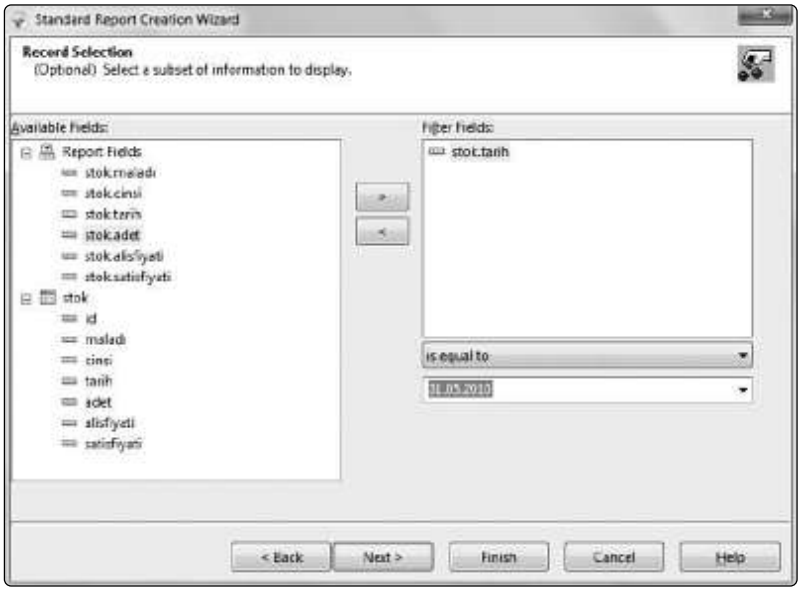
Rapor dosyasını incelediğimizde gruplama işleminin stok cinsi alanına göre yapıldığını ve onun altında stok tarihi alanının eklendiğini görürüz. Raporun sonunda ise genel toplam değerleri listelenmiştir. Grouping işlemini yapmadan diğer adımları yaparak oluşacak olan rapor dosyasına bakalım. Grouping işlemini boş bırakarak geçtiğimizde bir sonraki adımda filtreleme işlemi için alanları seçebiliriz.



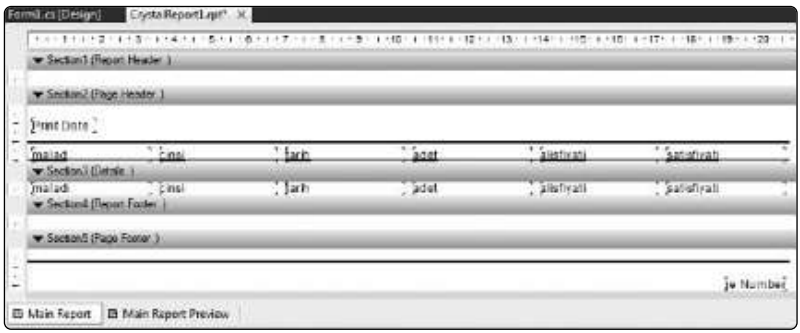
Filtreleme işleminde kullanılacak alanla birlikte bu alan için yapılacak filtreleme işleminin biçimini de seçebiliriz. Burada kullanılabilecek seçenekler ve anlamları aşağıdaki gibidir:

- **is any value:** Herhangi bir değer olabilir.
- **is equal to:** Eşitlik.
- **is not equal to:** Eşit değil, farklı.
- **is one of:** Bir grup değer içerisinden birini seçmek amacıyla kullanılır.
- **is not one of:** Bir grup değer içerisindeki elemanlardan belirtilen eleman dışındakileri seçmek amacıyla kullanılır.
- **is less than:** Küçük.
- **is less than or equal to:** Küçük veya eşit.
- **is greater than:** Büyük.
- **is greater than or equal to:** Büyük veya eşit.
- **is between:** İki değer arasında.
- **is not between:** İki değer arasında değil.
- **starts with:** Belirtilen bir değer ile başlayan değerleri belirtir.
- **does not start with:** Belirtilen bir değer dışındaki karakterler ile başlayan değerleri belirtir.
- **is like:** `*` , `?` gibi özel karakterleri kullanarak filtreleme seçeneklerini genişletir.
- **is not like:** Belirtilen özel karakter dışında kalan değerleri listeler.

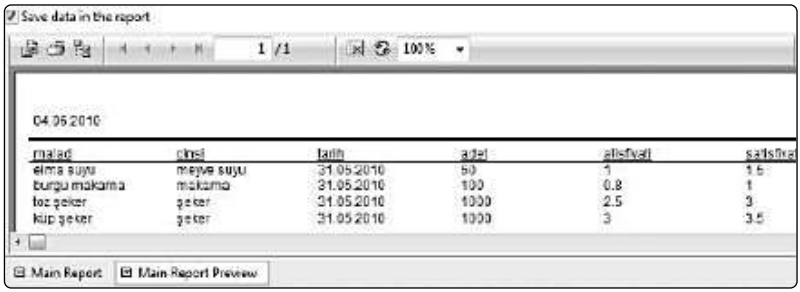
Örneğin tarih alanını seçerek işlem türü olarak da **is equal to** işlem türünü seçtik. Listelenen değerlerden de `31.05.2010` değerini seçerek tarih değeri `31.05.2010` olan verilerin listelenmesini sağlayacağız.



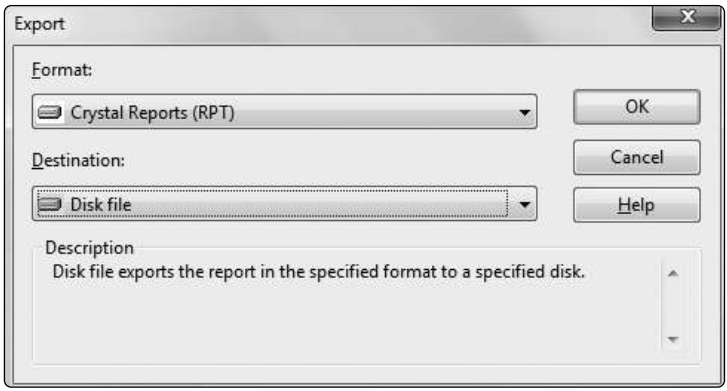
Bir sonraki aşamada rapor görünümünü seçerek işlemimizi bitiriyoruz. Rapor dosyamız **rpt** uzantılı olarak oluşturulur. Oluşturulan proje dosyamızın Form görünümünü incelediğimizde **Main Report** ve **Main Report Preview** adlı iki sekme bulunduğunu görüyoruz. **Main Report** sekmesinde oluşturulan rapor dosyamızın üzerinde değişiklik ve düzenleme yapılabilecek biçimde görüntülendiğini görürüz.



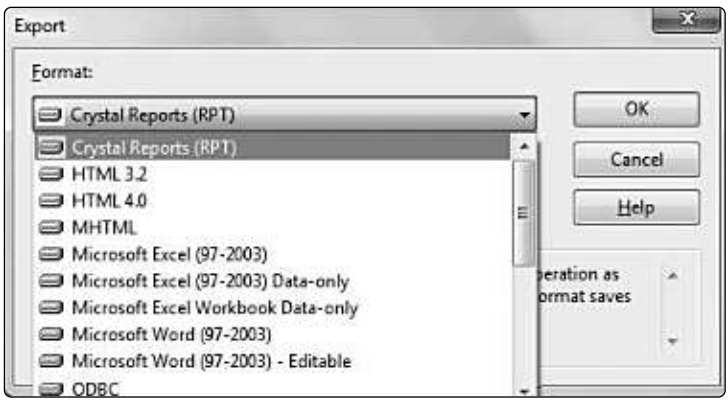
Main Report Preview sekmesinde oluşturulan rapor dosyasının ön izleme biçiminde görüntülendiğini görürüz. Dikkat edilirse yaptığımız filtreleme işlemi sonucunda yalnızca `31.05.2010` tarihli kayıtlar listelendi.



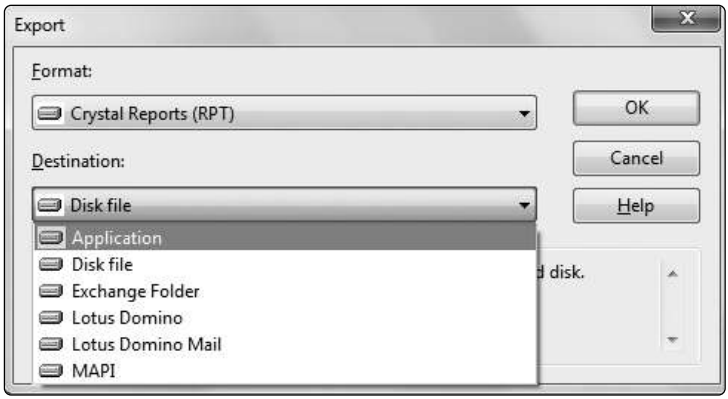
Araç çubuğundaki butonlardan **Export Report** butonu ile raporumuzu ister crystal report dosyası biçiminde istersek farklı biçimlerde kaydedebiliriz.



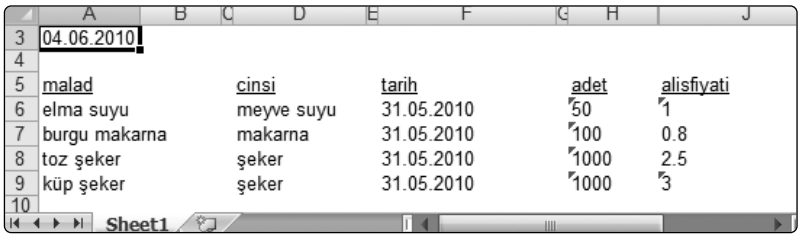
Format başlığı altındaki seçeneklerle rapor dosyamızın kayıt türünü belirleriz.



Destination bölümünde ise raporumuzun kayıt edileceği yeri belirleriz.



Örneğin format türü olarak Microsoft Excel (97-2003), **Destination** seçeneği olarak da **Disk file** seçeneğini seçerek rapor dosyamızın Excel dosyası olarak disk üzerinde belirteceğimiz bir yere kaydedilmesini sağlayalım.

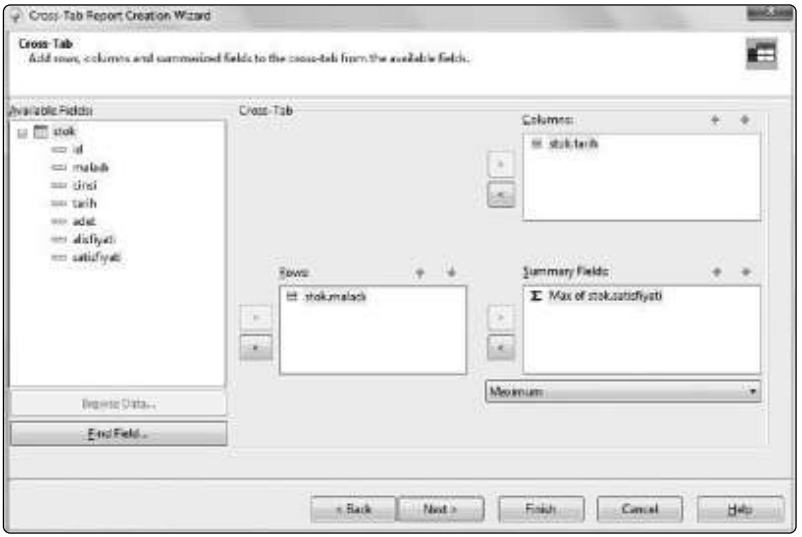


Cross-Tab

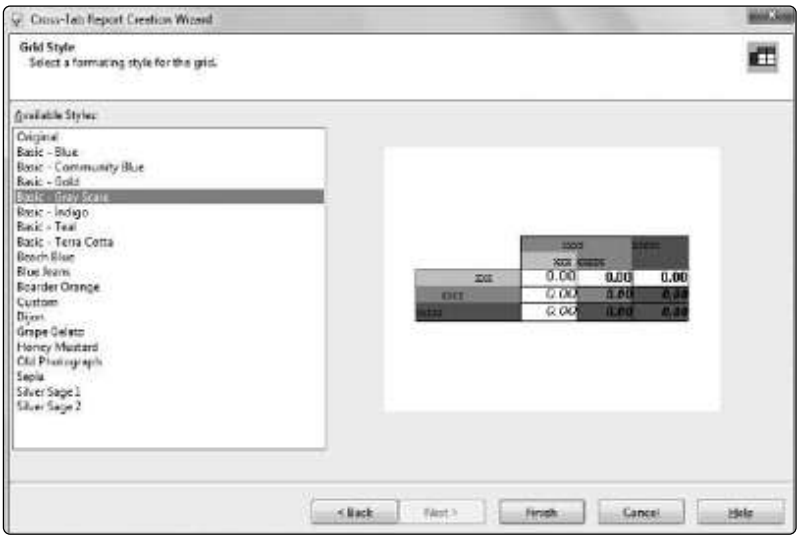
Çapraz sorgular içeren rapor hazırlamak amacı ile kullanılır.



Stok isimli veritabanını kullanarak standart rapor şeklinde hazırladığımız raporu, **cross-tab** biçiminde hazırlayalım. Ortak olan işlemleri atlayarak, yalnızca farklı olan işlemleri görelim.



Örneğimizde belirtilen tarihlerde satış fiyatı en yüksek olan malların listelenmesini sağlayalım. **Rows** bölümüne yazdığımız maladı alanı ile, **Columns** bölümüne yazdığımız tarih alanı çapraz sorgulama yapılarak bu sorgu sonucuna uyan maksimum satış fiyatı değerlerinin listelenir. Son aşamada ise rapor içerisinde listelenecek olan sonuçların görünümü seçilir.



Raporumuzun sonucu aşağıdaki gibi olacaktır:

	30.05.2010	31.05.2010	Total
5lt zeytinya ğı	35		35
burgu makarna		1	1
elma suyu		1.5	1.5
küp şeker		3.5	3.5
margarin	2.5		2.5
toz şeker		3	3
Total	35	3.5	35

04.06.2010

Manuel Olarak Rapor Oluşturma

Sihirbaz kullanmadan **As a Blank Report** seçeneğini kullanarak da boş bir rapor oluşturup, manuel olarak rapor tasarımı yapabiliriz. Bu işlem sonucunda oluşturulan boş rapor içerisinde şu başlıklar mevcut bulunmaktadır:



• **Report Header:** Raporun ilk sayfasının üst kısmında bulunur. Yeni bir raporun başlangıcını gösterir veya rapor kapağı olarak kullanılabilir.

• **Page Header:** Her bir sayfanın üst kısmında bulunur. Sütun başlığı, rapor başlığı, sayfa sayısı gibi bilgileri görüntüler.

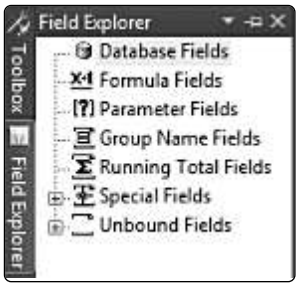
• **Details:** Rapor içerisinde bulunan her bir kayıt için detay bilgileri görüntülenebilir. Görüntü alanına sığmayan verilerin detaylıca gösterilmesi için de kullanılabilir.

• **Report Footer:** Raporun son sayfasının alt kısmında bulunur. Varsayılan olarak görüntülenir. Genellikle rapora dair kayıt sayısı, yazdırma tarihi gibi özet bilgileri görüntüler.

• **Page Footer:** Her bir sayfanın alt kısmında bulunur. Sayfa sayısı, yazdırma tarihi, zamanı gibi bilgileri görüntüler.

Rapor içerisine ekleyebileceğimiz alanlar ise **Toolbox** içerisinde **Field Explorer** başlığı altında bulunmaktadır.

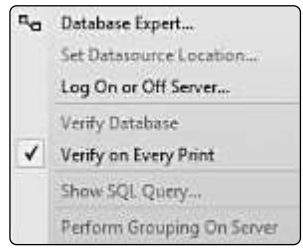
Bu alanlar başlıca şu başlıklar altında toplanmaktadır:

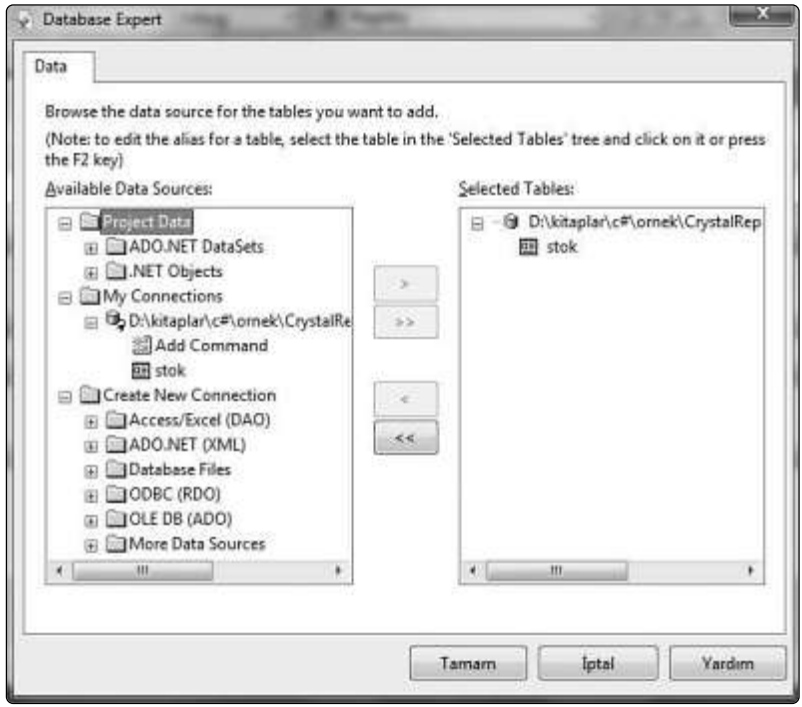


• **Database Fields:** Rapor içerisinde görüntülenecek olan verilerin ait olduğu veritabanı ve tablolar ile bağlantı yapmak amacı ile kullanılır.

• **Formula Fields:** Çeşitli matematiksel işlemler ya da olayları gerçekleştirmek için formüller ve metodları tanımlamak amacı ile kullanılır.

- **Parameter Fields:** Programın çalışması esnasında parametre almak amacı ile kullanılır.
- **Group Name Fields:** Rapor içerisinde gruplandırmaları tanımlama amacı ile kullanılır.
- **Running Total Fields:** Veriler üzerinde matematiksel işlemler yapılarak sonuçlarının görüntülenmesi amacı ile kullanılır.
- **Special Fields:** Rapor üzerinde görüntülenecek tarih, zaman, sayfa sayısı gibi bilgileri görüntülemek amacı ile kullanılır.
- **Unbound Fields:** Çeşitli değişken türlerinden değerlerin görüntülenmesi amacı ile kullanılır.
- **Database Fields: Database Fields** alanına farenin sağ tuşu ile tıklayarak gelen menüden **Database Expert** seçeneğine tıklayarak **Database Expert** penceresini görüntüleriz.





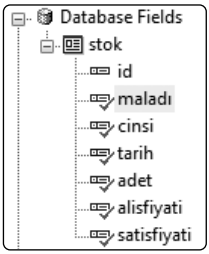
Database Expert penceresindeki...

ADO .NET DataSets başlığı altında, eğer proje içerisinde veritabanı ile bağlantı kurmak için önceden hazırladığımız bir dataset mevcutsa listelenir.

.NET Objects başlığı altında, projemizde mevcut olan .NET nesneleri listelenir.

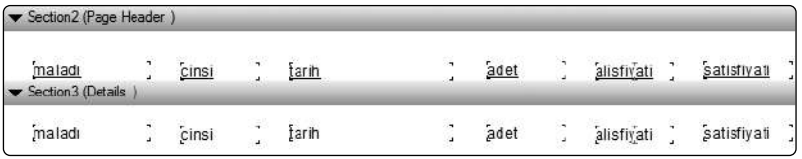
My Connections başlığı altında, eğer veritabanı ile bağlantı kurmak için önceden oluşturduğumuz bir connection string mevcutsa listelenir.

Create New Connection başlığına tıklayarak ise, yeni bir veritabanı oluşturarak onun ile bağlantı kurabilir ya da daha önceden oluşturmuş olduğumuz bir veritabanı ile bağlantı kurabiliriz.

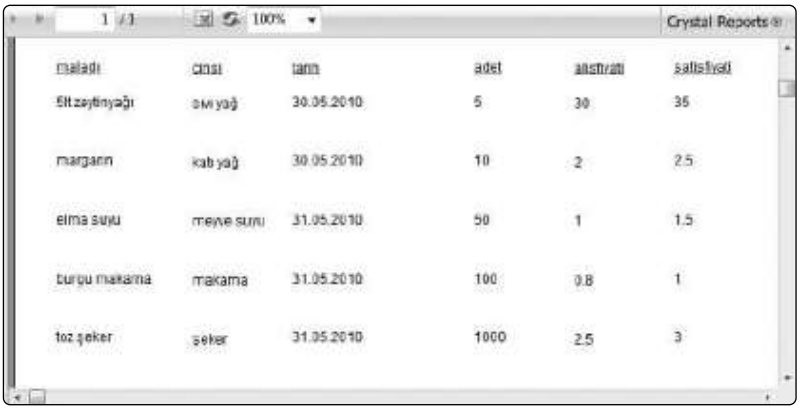


Daha önceden oluşturmuş olduğumuz connection string'i kullanarak stok isimli veritabanı ile bağlantı kuruyoruz. **Toolbox** içerisinde **Database Fields** bölümüne baktığımızda stok isimli veritabanı ile bağlantı kurularak alan isimlerinin listelendiğini görürüz. Burada listelenen alanları fare ile rapor içerisindeki details alanına sürükleyerek bırakıyoruz.

Bu işlemi yaparken **Page Header** kısmında alan başlıklarının otomatik olarak oluşturulduğunu görebiliriz.

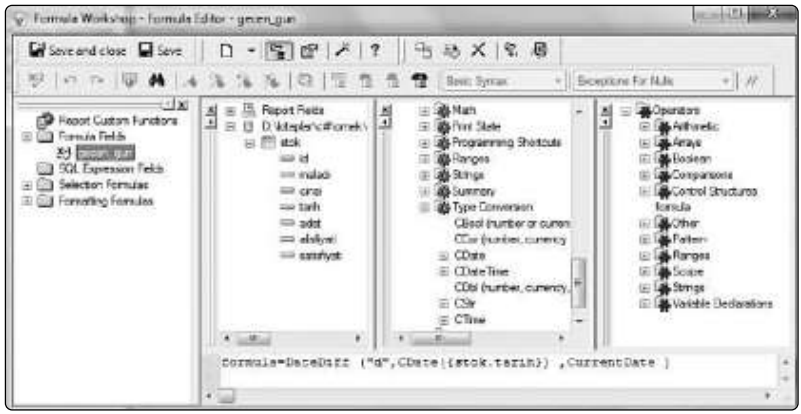


Raporu önizleme yaptığımızda alan başlıklarının ve alanlar içerisindeki değerlerin listelendiğini görebiliriz.



- **Formula Fields: Crystal Reports** içerisindeki formül editörü ile yeni formüller oluşturarak, oluşturduğumuz formül alanlarını rapor dosyamıza ekleyerek sonuçları görüntüleyebiliriz. **Crystal Reports** içerisinde **Crystal Syntax** ve **Basic Syntax** isimli iki dil yardımıyla formüllerimizi yazabiliriz. Basic Syntax'ın kod yapısı Visual Basic'e çok benzemektedir.

Örnek olarak stoktaki ürünlerin tarihi ile o günkü tarih değeri arasındaki farkı hesaplayarak ürünün stoğa girişinden bu yana geçen gün sayısını hesaplayan formülü yazalım.



Tarih değerleri arasındaki farkı hesaplamak için `DateDiff` metodunu kullanacağız. Sonucun görüntülenmesi için `formula` isimli nesneye aktarıyoruz.

`DateDiff` metodunun kullanımı şu şekildedir:

```
DateDiff(interval type value, başlangıç tarihi, bitiş tarihi)
```

Interval type value: tarihler arası fark alınırken gün, ay ya da yıl değerlerinden hangisi üzerinde işlem yapılacağını belirler. Aşağıdaki değerleri alabilir:

- d: (day) Gün
- yyyy: (year) Yıl
- m: (month) Ay
- h: (hour) Saat
- n: (minute) Dakika
- s: (second) Saniye

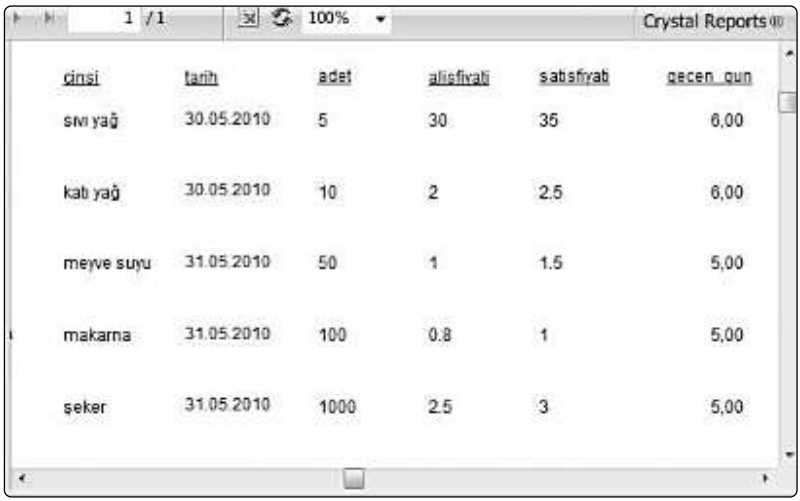
Tarihler arası farkı alırken kullandığımız `CurrentDate` ise o anki zamanı belirtmektedir. `CDate` ile tarih alanının değerini tarih türüne dönüştürüyoruz.

Formülümüzün doğrululuğunu test etmek için araç çubuğundaki **Check** simgesi ya da **Alt+C** kısayol tuşları kullanılabilir.



Eğer formül doğru ise *"No errors found"* biçiminde bir uyarı penceresi görüntülenir. Aksi takdirde oluşan hata ile ilgili bilgilendirme içeren uyarı penceresi görüntülenecektir.

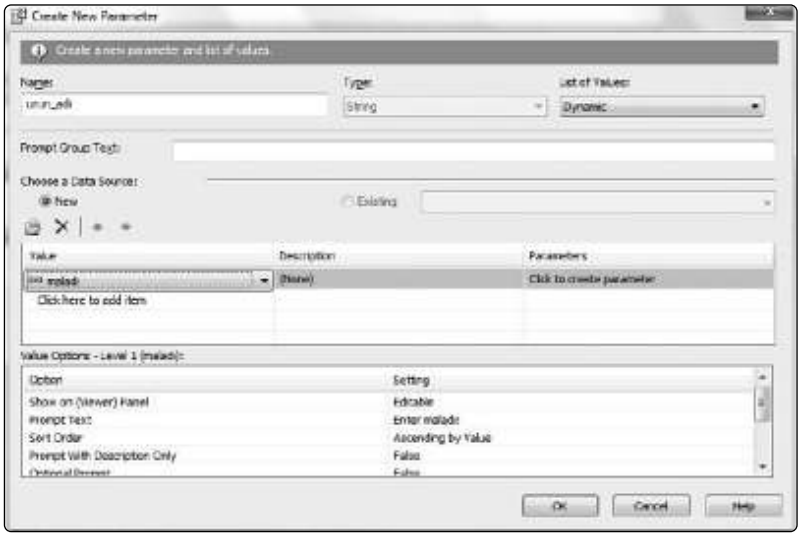
Eklediğimiz formülün sonucu rapor üzerinde aşağıdaki gibi görüntülenecektir:



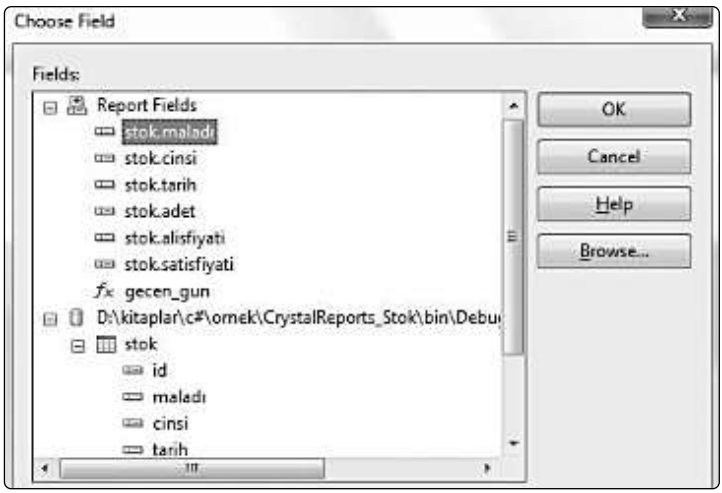
cinsi	tarih	adet	alisfiyati	satisfiyati	gecen_gun
sıvı yağ	30.05.2010	5	30	35	6,00
katı yağ	30.05.2010	10	2	2.5	6,00
meyve suyu	31.05.2010	50	1	1.5	5,00
makarna	31.05.2010	100	0.8	1	5,00
şeker	31.05.2010	1000	2.5	3	5,00

- **Parameter Fields:** Programın çalışması esnasında parametre almak amacı ile kullanılır. Örnek olarak veritabanımızdaki mal adı adlı alanın dinamik olarak dışarıdan parametre almasını sağlayarak yalnızca belirttiğimiz malların listelenmesini sağlayalım.

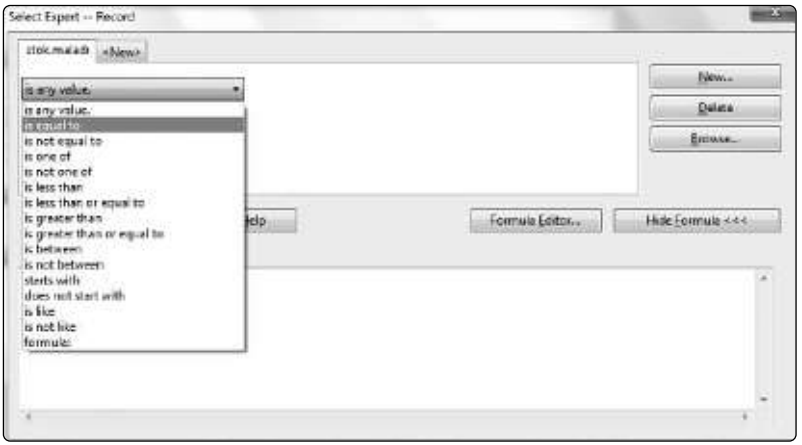
Field Explorer içerisindeki **Parameter Fields** seçeneğine farenin sağ tuşu ile tıklayarak **New** seçeneğine tıklayalım. Karşımıza parametrelerimizi oluşturabileceğimiz bir ekran gelecektir.



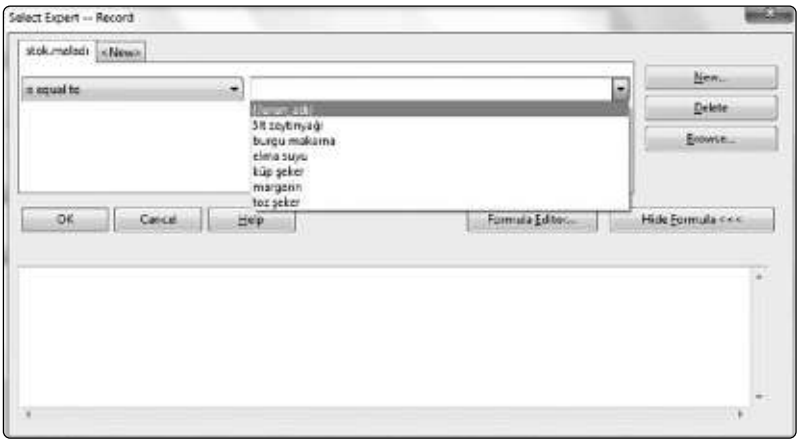
Name bölümünde parametremize bir isim veriyoruz. **List of Values** bölümünden parametremizi **Dynamic** ya da **Static** olarak belirleyebiliriz. Biz **Dynamic** seçeneğini seçiyoruz. **Value** bölümünde üzerinde işlem yapacağımız alanı seçiyoruz. Örneğimiz için maladı alanını seçiyoruz. Araç çubuğundan **Select Expert** butonunu tıklayarak **Select Expert Record** isimli pencereyi görüntülüyoruz. Öncelikle üzerinde işlem yapacağımız alanı seçiyoruz.



Alan seçiminin ardından **OK** butonuna tıklayarak bir önceki penceremize geri dönüyoruz. Bu pencerede parametre üzerinde yapacağımız işlem türünü belirliyoruz. Örneğimiz için **is equal to** seçeneğini seçiyoruz. Bu şekilde belirteceğimiz kayıtların listelenmesini sağlayacağız.



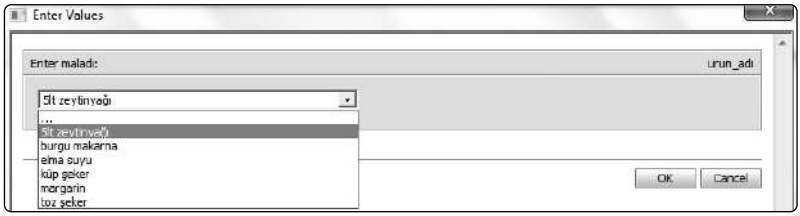
İşlem türünün sağ tarafında parametre değeri ile birlikte alan içerisindeki veriler listelenecektir. Biz ilk sıradaki parametre değerini seçiyoruz.



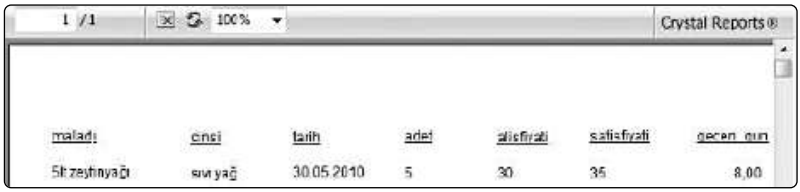
OK butonuna tıklayarak işlemlerimizi bitiriyoruz.

Rapor dosyamıza önizleme yaptığımızda öncelikle **Enter Values** isim-

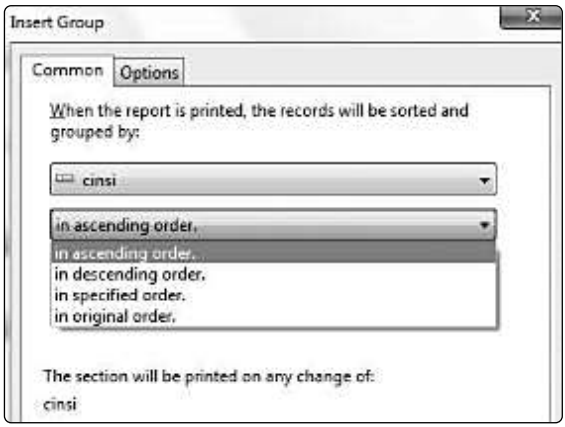
li bir pencere karşımıza gelerek maladı alanı için değer seçmemizi isteyecektir.



Burada seçtiğimiz değer rapor ekranında listelenecektir.

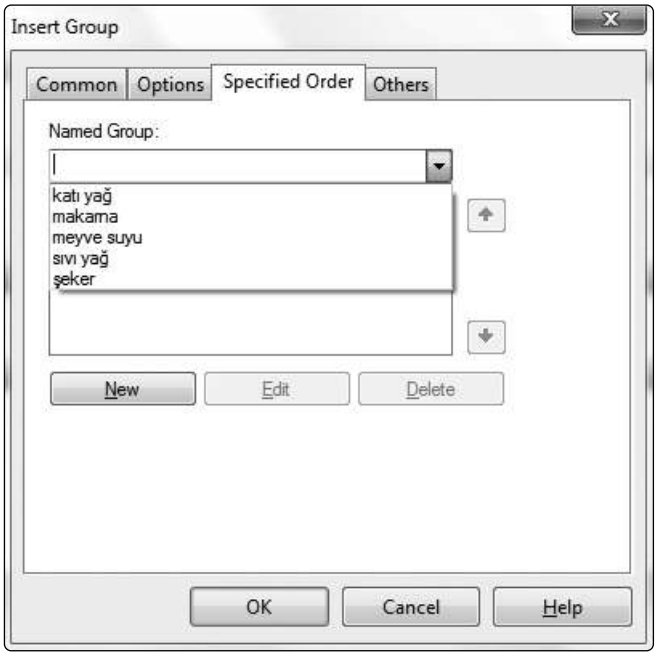


Group Name Fields: Rapor içerisinde belirteceğimiz alanlara göre gruplama yapmak amacı ile kullanılır. Örnek olarak stok cinsine göre gruplandırma yapalım. Öncelikle **details** bölümünde listelenen alanlardan cinsi alanını silelim. Çünkü bu alanı zaten grup ismi olarak ekleyeceğiz. **Field Explorer** içerisinde **Group Name Fields** alanına farenin sağ tuşu ile tıklayarak **Insert Group** seçeneğine tıklayalım.



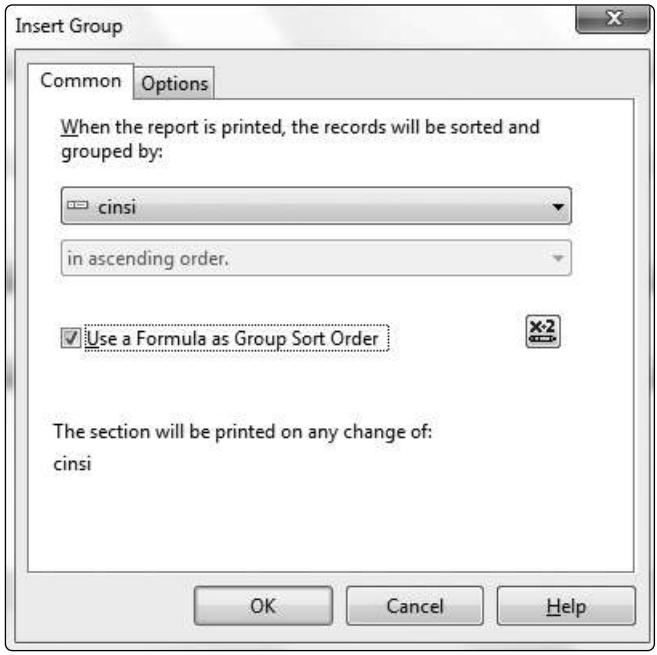
Buradan **cinsi** alanını seçiyoruz. Alan seçim bölümünün alt tarafından sıralama seçeneklerini belirleyebiliriz.

- **in ascending order (artan sıralama)**
- **in descending order (azalan sıralama)**
- **in specified order (özel sıralama biçimi):** Bu seçenek seçildiğinde **Specified Order** sekmesi aktif olur ve buradan stok cinsi alanındaki listelenen değerlerden birini seçerek bu değere göre sıralama yapılmasını sağlayabiliriz.

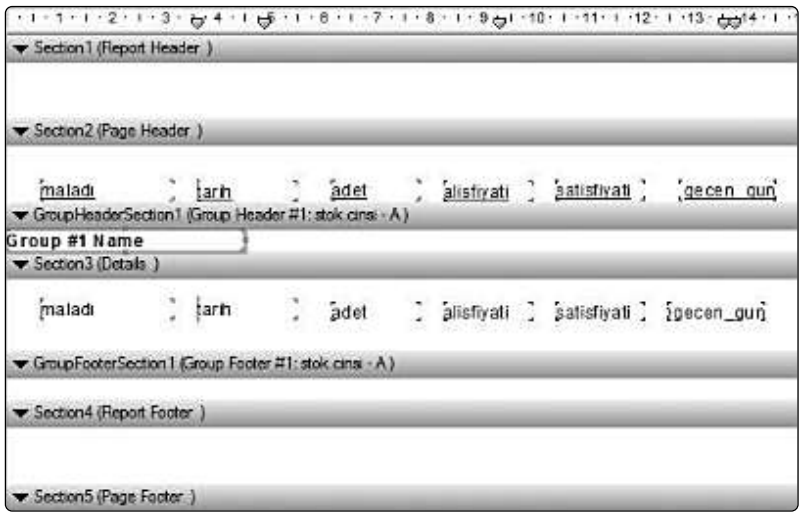


- **in original order (orijinal sıralama biçimi)**

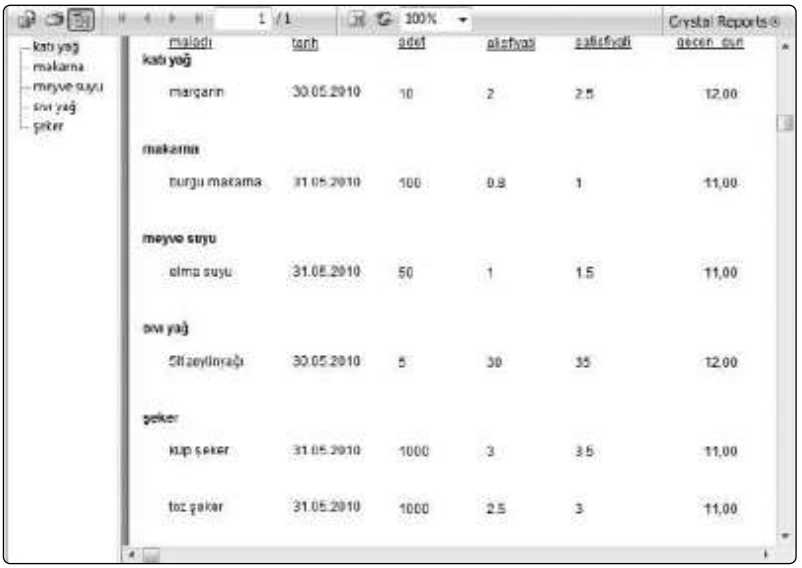
Ayrıca pencerenin alt tarafında bulunan **Use a Formula as Group Sort Order** seçeneğini işaretleyerek sıralama işlemi için formül editörü yardımıyla sıralama işleminde kullanılacak formüller oluşturabiliriz.



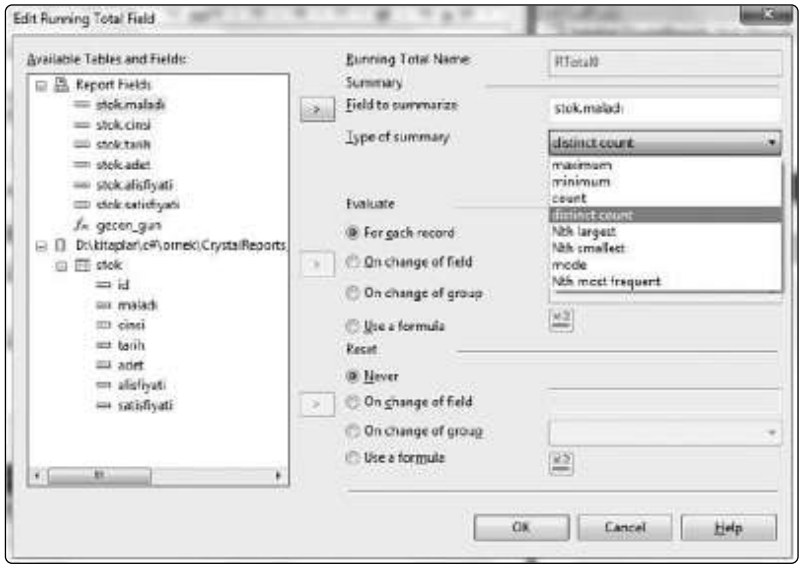
Örneğimiz için **ascending order** seçeneğini seçiyoruz. Raporumuz içerisine **Group Header** ve **Group Footer** alanları da otomatik olarak eklenecektir.



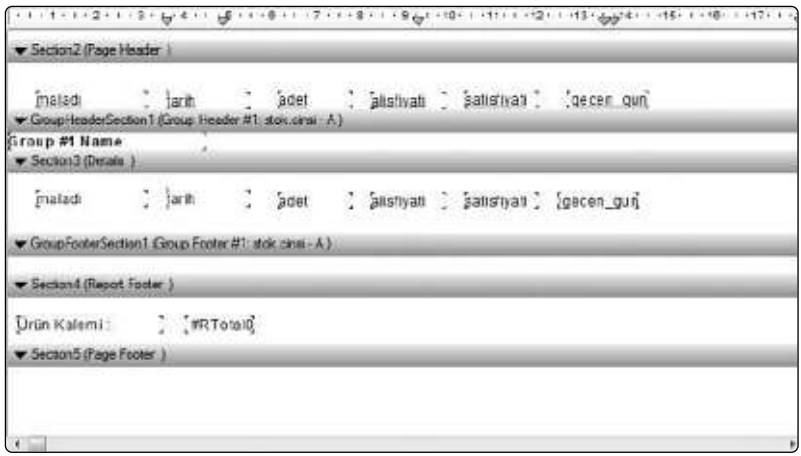
Rapora önizleme yaptığımızda maladı alanındaki ürünlerin stok cinsine göre gruplandırıldığını görebiliriz.



- **Running Total Fields:** Veriler üzerinde matematiksel işlemler yapılarak sonuçlarının rapor dosyası içerisinde görüntülenmesi amacı ile kullanılır. **Field Explorer** içerisinde **Running Total Fields** bölümü içerisinde farenin sağ tuşuna tıklayarak **New** menüsüne tıklıyoruz. Gelen pencerede **Field to summarize** bölümünden üzerinde işlem yapacağımız alanı, **Type of summary** bölümünde ise işlem türünü seçiyoruz. Örnek olarak ben distinct count işlemini seçtim. Bu işlem üzerinde belirtilen alandaki değerlerin adedini buluyoruz. Ancak alan içerisinde aynı değerden birden fazla varsa yalnızca birisi sayılıyor.

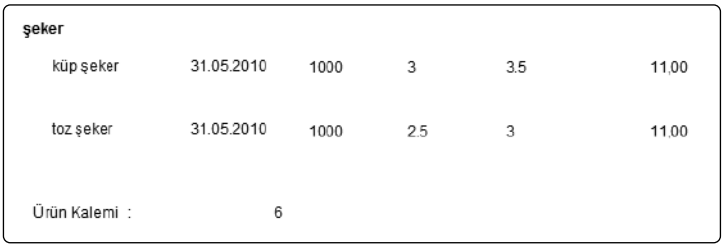


Oluşturulan total field alanını **Field Explorer** içerisinde **Running Total Fields** bölümünden fare ile istediğimiz alana sürükleyip bırakabiliriz. Ben, **Report Footer** alanına ekleyerek oluşturulan rapor dosyasının en alt kısmında görüntülenmesini sağladım.

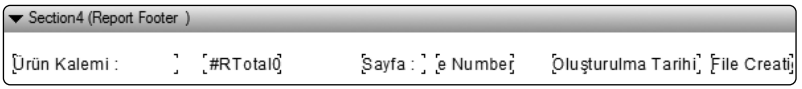


Rapor dosyamızın sonucu aşağıdaki gibi olacaktır. Ben yalnızca `Ürün Kalemi` alanını görmek amacı ile diğer alanları listelemedim.

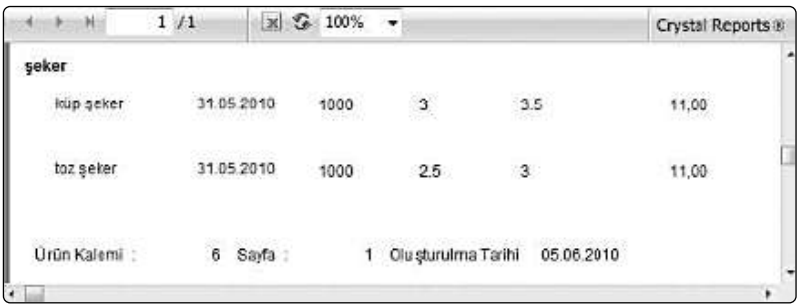
"Ürün Kalemi" yazısı için toolbox içerisinden **Text Object** ekliyoruz.



- **Special Fields:** Rapor üzerinde görüntülenecek tarih, zaman, sayfa sayısı gibi bilgileri görüntülemek amacı ile kullanılır. Örnek olarak **Report Footer** bölümüne **Page Number** (*Sayfa Sayısı*) ve **File Creation Date** (*Dosya Oluşturulma Tarihi*) alanlarını ekleyelim.



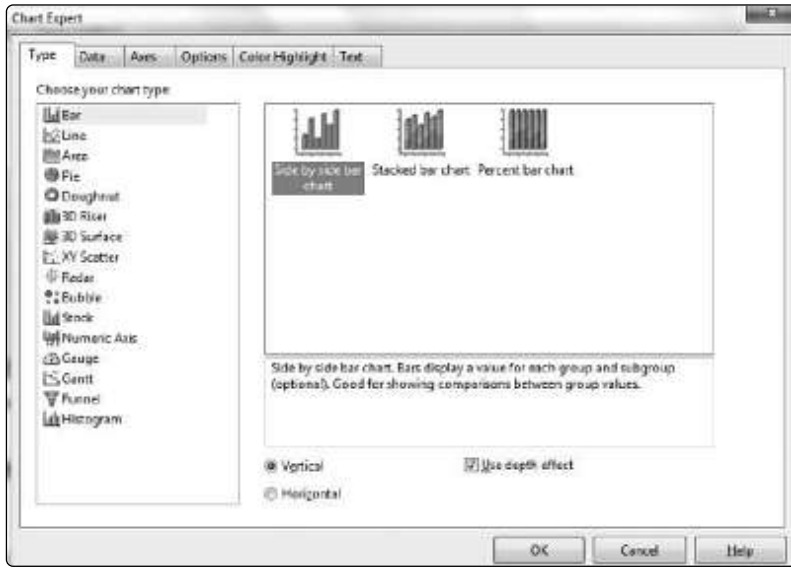
Görüldüğü gibi bu alanlar içerisinde değerler otomatik olarak gösterilmektedir.



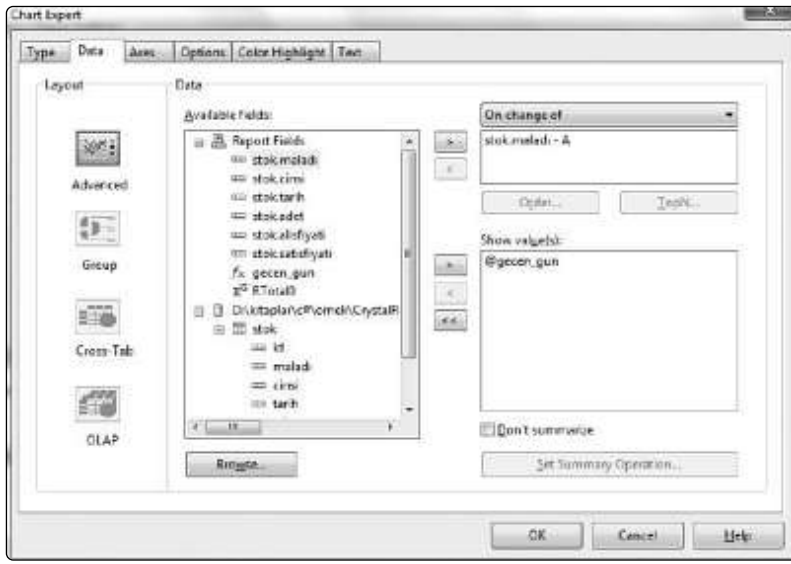
GRAFİK EKLEMEK

Rapor dosyasına grafik eklemek için **Crystal Reports** menüsünden Insert menüsü altındaki **Chart** seçeneği ya da araç çubuğundaki simgesi kullanılabilir. Karşımıza **Chart Expert** isimli grafik işlemlerini yapacağımız pencere gelecektir.

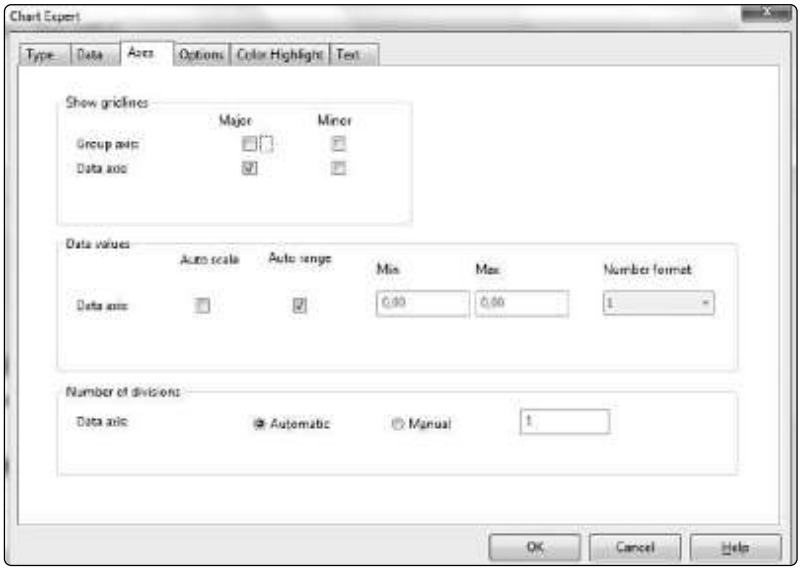
Type sekmesi altında grafiğimizin tipini seçiyoruz. Örneğin ben **Bar** başlığı altındaki Side by side bar chart seçeneği ile sütun grafiğini seçtim. Derinlik efekti vermek için de **Use depth effect** seçeneğini işaretledim.



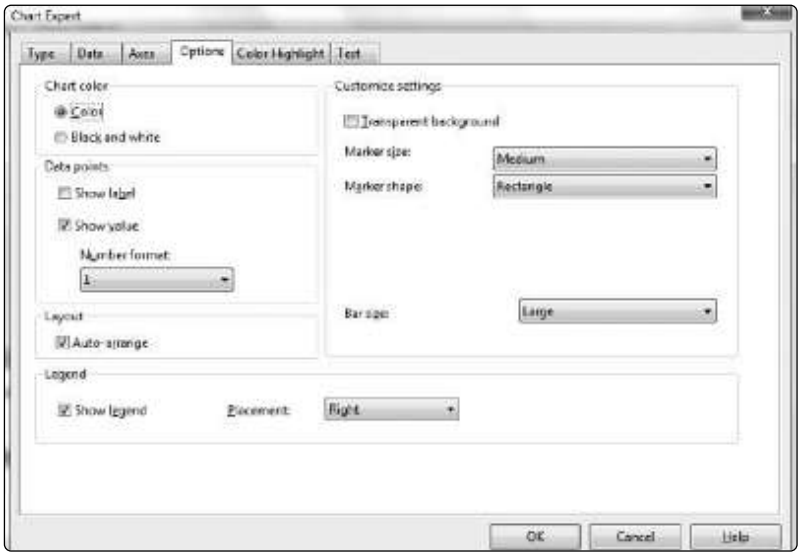
Data sekmesinde ise grafik üzerinde görüntülenecek olan alanları belirliyoruz. On change of bölümünde grafik üzerinde görüntülenecek alanları seçerken, **Show value(s)** bölümünden görüntülenecek olan değerleri seçebiliriz. Burada ister bir alan, istersek bir formül ya da parametre seçebiliriz.



Axes sekmesinde grafiğimizin eksenlerinde görüntülenecek olan alanlarla ilgili ayarlamaları yaparız.



Options ekranında grafiğimizin görünümü ile ilgili çeşitli ayarlamaları yapabiliriz.



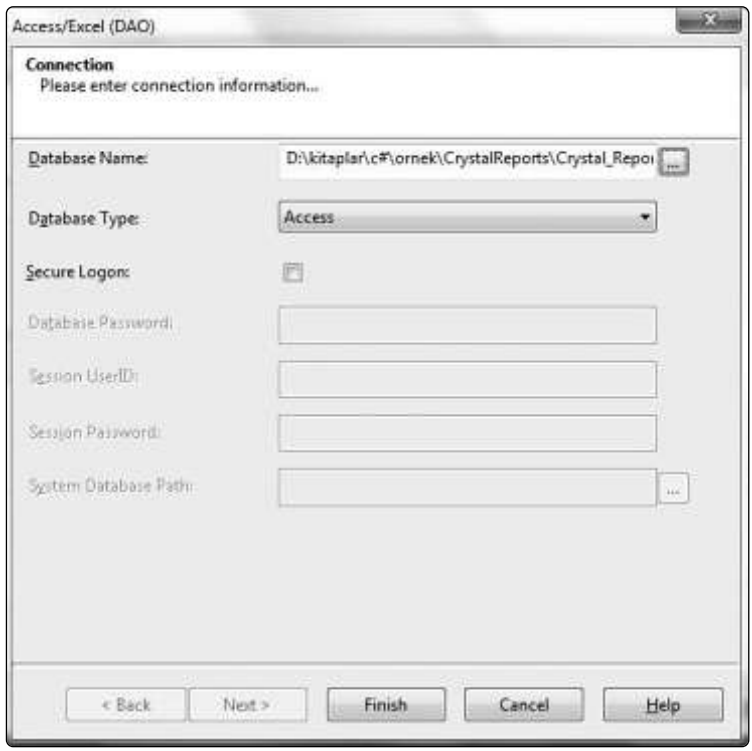
Bu ekrandaki;

Chart color bölümünde, grafiğimizin **renkli** (*color*) ya da **siyah-beyaz** (*black and white*) olmasını ayarlayabiliriz.

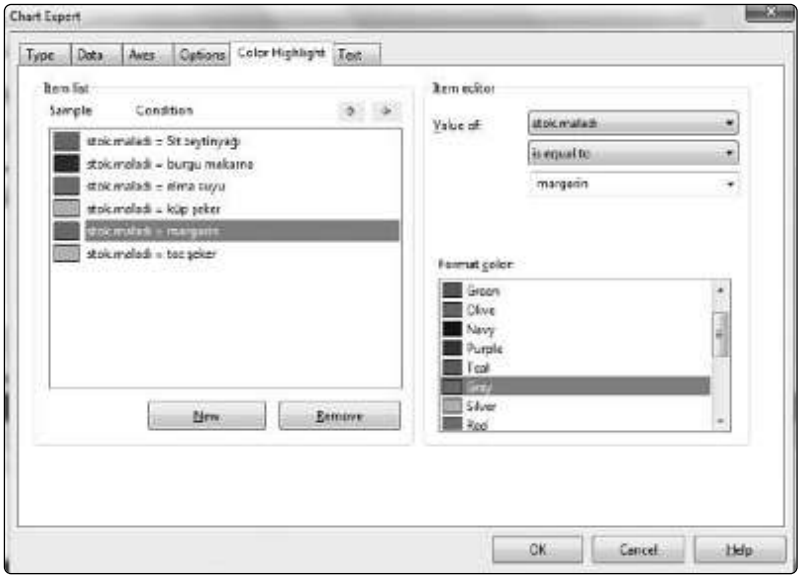
Data points bölümünde: **Show label** seçeneği ile alan etiketlerinin gösterilmesini; **Show value** seçeneği ile alan değerlerinin gösterilmesini sağlayabiliriz.

Layout bölümünde grafiğin yerleşim biçimini belirleriz.

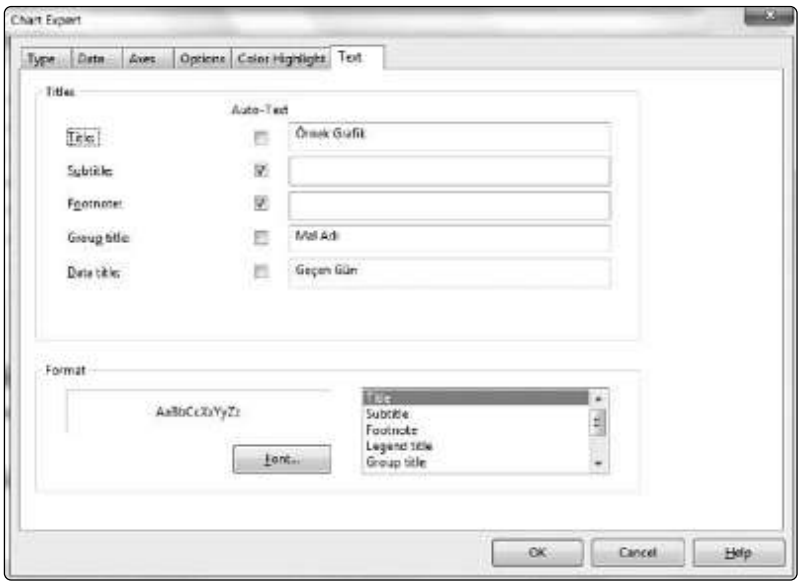
Color Highlight sekmesinde grafiğimiz içerisinde görüntülenecek olan değerlerin hepsi ya da ayrı ayrı her biri için renk ayarı yapabiliriz. Ancak bunun için öncelikle, gelen ekranda kullanılacak veritabanı seçimini yapmalıyız.



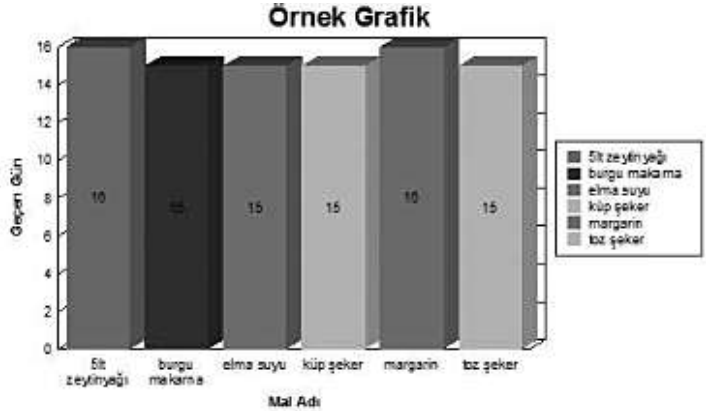
Bu işlemin ardından renk seçimini yapabiliriz.



Text sekmesinde grafik üzerinde görüntülenecek olan yazıları belirleyebiliriz.



Grafiğimizi rapor üzerinde uygun bir alana ekleyebiliriz. Örneğin, ben **Report Footer** bölümüne ekledim.



OLUŞTURULMUŞ OLAN RAPOR DOSYALARINI CRYSTALREPORTVIEWER İLE GÖRÜNTÜLEME

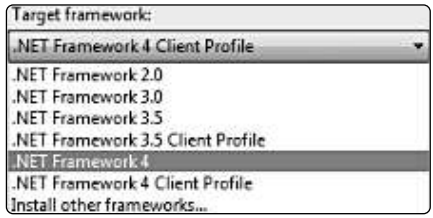
Daha önceden oluşturduğumuz rapor dosyalarını Windows formları içerisinde `CrystalReportViewer` nesnesi yardımı ile görüntüleyebiliriz. Bu işlemi de rapor dosyasının ismini belirterek ya da `ReportDocument` nesnesini kullanarak gerçekleştirebiliriz.

RAPOR DOSYASININ İSMİNİ BELİRTEREK RAPOR GÖRÜNTÜLEME

`CrystalReportViewer` nesnesi içerisinde rapor dosyasının ismini belirterek rapor dosyasını görüntüleyebilmek için `ReportSource` özelliğini kullanıyoruz.

Örnek olarak yeni bir Windows form projesi açalım. İçerisinde iki adet form olsun. İkinci formu rapor dosyamızı görüntülemek amacı ile kullanacağız.

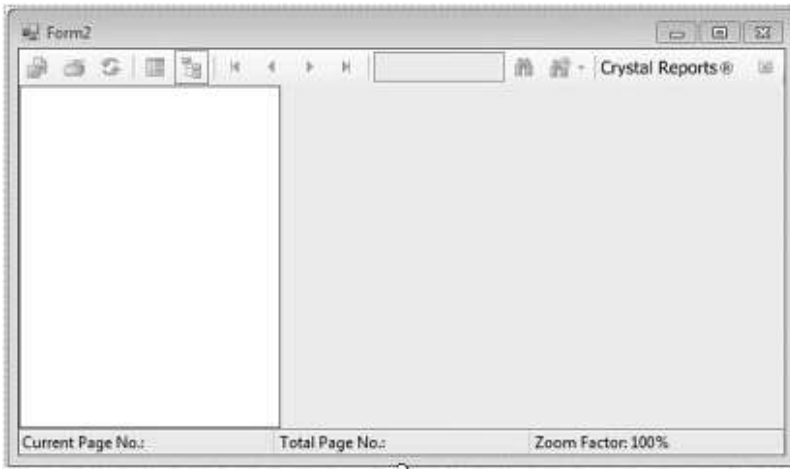
Proje içerisine daha önceden oluşturmuş olduğumuz örnek bir crystal report dosyası ekleyip, bu raporu ikinci formun içerisinde `CrystalReportViewer` nesnesi yardımı ile görüntüleyeceğiz. Bunun için ikinci form içerisine Toolbox içerisinden `CrystalReportViewer` nesnesini ekleyeceğiz. Bu nesne toolbox içerisinde **Reporting** başlığı altında yer almaktadır.



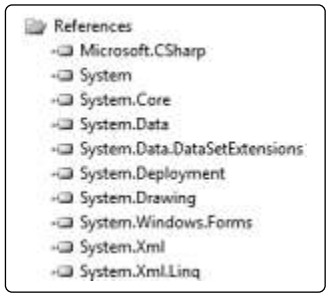
Eğer bu nesne pasif durumda ise **Project / Proje Adı Properties** menüsünden **Application** sekmesinde **Target Framework** bölümünü kontrol etmeliyiz.

Burada **.NET Framework Client Profile** seçeneği seçili ise bunu **.NET Framework 4** olarak değiştirdiğimizde `CrystalReportViewer` nesnesi aktifleşecektir.

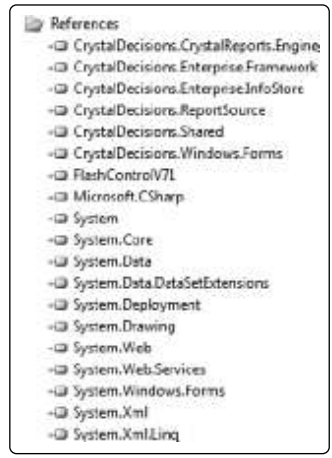
`Form2` üzerine `CrystalReportViewer` nesnesini eklediğimizde formumuzun tasarım görünümü aşağıdaki gibi olacaktır:



`CrystalReportViewer` nesnesini eklediğimiz anda gerekli referans dosyaları da otomatik olarak eklenecektir. Aşağıdaki ekran görüntülerinde `CrystalReportViewer` nesnesi eklenmeden önce ve eklendikten sonraki referans dosyalarının listeleri görülmektedir.



CrystalReportViewer nesnesi eklenmeden önce.



CrystalReportViewer nesnesi eklendikten sonra

`Form1` içerisinden `Form2`'yi çağırmak için

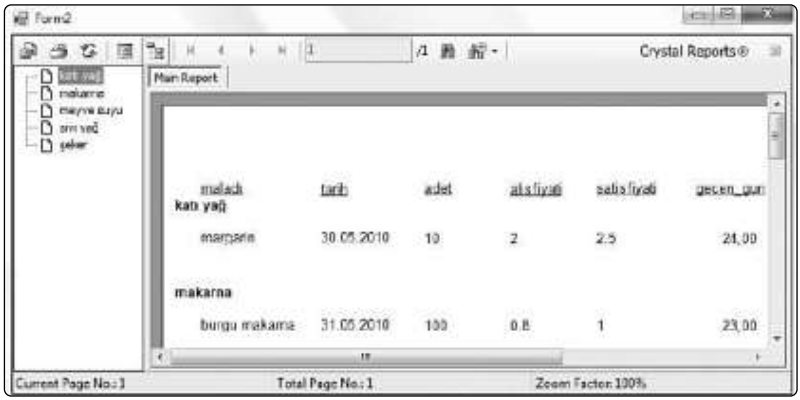
```
private void btnRaporGoruntule_Click(object sender, EventArgs e)
{
     Form2 frmRapor = new Form2();
     frmRapor.Show();
}
```

kodunu yazıyoruz.

Rapor dosyasının `crystalReportViewer` içerisinde görüntülenmesi için `ReportSource` özelliğini kullanıyoruz.

```
private void Form2_Load(object sender, EventArgs e)
{
     crystalReportViewer1.ReportSource=
          Application.StartupPath + "\\CrystalReport1.rpt";
}
```

Form2'yi görüntülediğimizde rapor içeriği görüntülenecektir.



REPORTDOCUMENT NESNESİNİ KULLANARAK RAPOR GÖRÜNTÜLEME

CrystalReportViewer nesnesi içerisinde ReportDocument nesnesini kullanarak rapor dosyalarını görüntüleyebiliriz. Bunun için ReportDocument nesnesinin Load özelliğini kullanıyoruz.

ReportDocument nesnesini kullanabilmek için CrystalDecisions.CrystalReports.Engine referansının projemizde ekli olması gerekmektedir.

Form2 içerisine CrystalReportViewer nesnesini ekleyerek formun Load olayı içerisine aşağıdaki kodları yazalım:

```
private void Form2_Load(object sender, EventArgs e)
{
     CrystalDecisions.CrystalReports.Engine.ReportDocument rapor = new
          CrystalDecisions.CrystalReports.Engine.ReportDocument();
     rapor.Load(Application.StartupPath + "\\CrystalReport1.rpt");
     crystalReportViewer1.ReportSource = rapor;
}
```

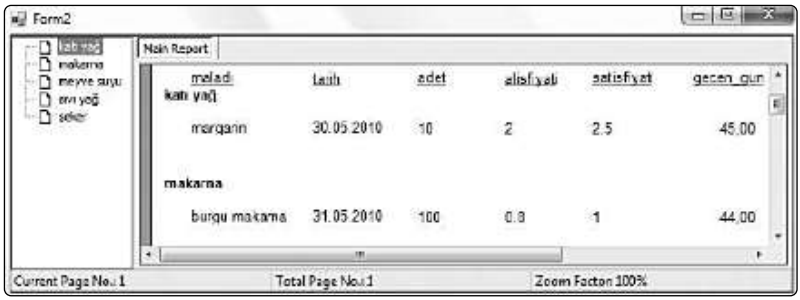
Yazdığımız kod sonucunda Form2 içerisinde CrystalReport1.rpt adlı rapor dosyası görüntülenecektir.

CrystalReportViewer Nesnesinin Özellikleri

DisplayToolbar: Araç çubuğunun görüntülenmesi ya da gizlenmesi için kullanılır.

DisplayStatusBar: Durum çubuğunun görüntülenmesi ya da gizlenmesi için kullanılır. Örneğin rapor dosyasını bu özelliklere false değerini atayarak kullandığımızda;

```
crystalReportViewer1.DisplayToolbar = false;
crystalReportViewer1.DisplayStatusBar = false;
```



biçiminde görüntülenecektir.

Ayrıca `ShowCloseButton`, `ShowExportButton`, `ShowGoToPageButton`, `ShowGroupTreeButton`, `ShowPageNavigateButtons`, `ShowParameterPanelButton`, `ShowPrintButton`, `ShowRefreshButton`, `ShowTextSearchButton`, `ShowZoomButton` özelliklerine `true` ya da `false` değerlerini atayarak araç çubuğunda sadece belirleyeceğimiz butonların görüntülenmesini ya da gizlenmesini sağlayabiliriz.

CrystalReportViewer Nesnesinin Metodları

ExportReport: Rapor dosyasının CrystalReport dosyası formatında ya da farklı formatlarda (*Word*, *Excel*, *Acrobat*, *Xml* gibi) kaydedilmesini sağlar.

GetCurrentPageNumber: O an görüntülenen sayfa numarasını belirtir.

PrintReport: Rapor içeriğini yazdırmak amacıyla kullanılır.

ShowFirstPage: Rapor içeriğindeki ilk sayfayı görüntülemek amacı ile kullanılır.

ShowLastPage: Rapor içeriğindeki son sayfayı görüntülemek amacı ile kullanılır.

ShowNextPage: Görüntülenen rapor sayfasından bir sonraki sayfayı görüntülemek amacı ile kullanılır.

ShowPreviousPage: Görüntülenen rapor sayfasından bir önceki sayfayı görüntülemek amacı ile kullanılır.

ShowNthPage: Rapor içeriğindeki belirtilen sayfayı görüntülemek amacı ile kullanılır. `ShowNthPage(int sayfa numarası)` biçiminde kullanılır.

Zoom: Rapor içeriğini yakınlaştırmak amacı ile kullanılır. `Zoom(int yakınlaştırma yüzdesi)` biçiminde kullanılır. Örneğin `%125` oranında yakınlaştırmak için `Zoom(125)` biçiminde yazılır.

SearchForText: Belirtilen bir string değeri aramak amacı ile kullanılır. `SearchForText(string Aranan Metin)` biçiminde kullanılır.

Geriye boolean değer döndürür. Eğer aranan metin bulunursa `true`, bulunamazsa `false` değerini döndürür.

`CrystalReportViewer` nesnesinin `PrintReport` metodunu kullanarak rapor dosyasını yazıcıdan çıkartabileceğimiz gibi, `ReportDocument` nesnesinin `PrintToPrinter` metodunu kullanarak da rapor dosyasını yazıcıdan çıkartabiliriz.

```
private void Form2_Load(object sender, EventArgs e)
{
    CrystalDecisions.CrystalReports.Engine.ReportDocument rapor =
      new CrystalDecisions.CrystalReports.Engine.ReportDocument();
    rapor.Load(Application.StartupPath + "\\CrystalReport1.rpt");
    crystalReportViewer1.ReportSource = rapor;
    rapor.PrintToPrinter(1, true, 1, 2);
}
```

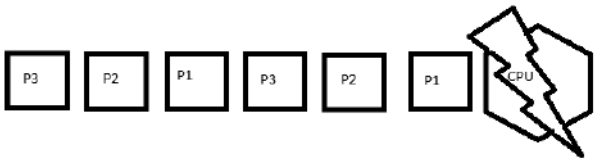
`PrintToPrinter` metodunun kullanımı aşağıdaki gibidir:

```
PrintToPrinter(Kopya Sayısı, Harmanlama, Başlangıç Sayfası,
Bitiş Sayfası)
```

28

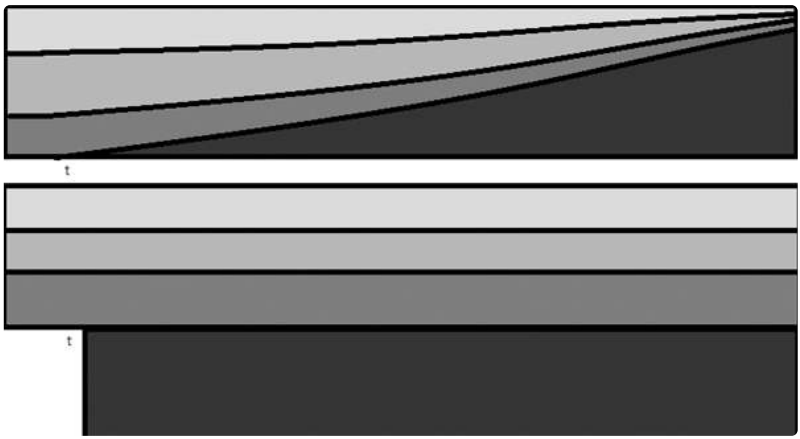
THREADING

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 pek 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 `20 ms` 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 birde 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.



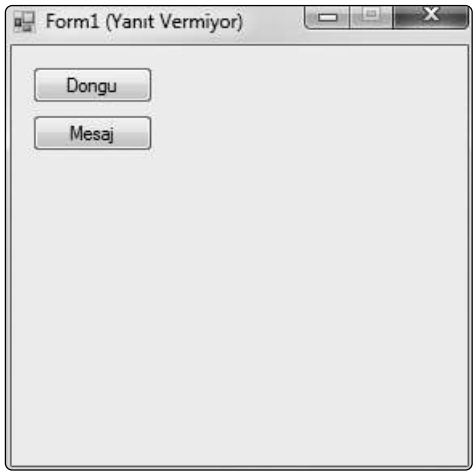
1. Grafikte thread kullanılmaması sonucu `t` anında çalışmaya başlayan koyu gri renkli işlemin sistem kaynaklarını kullanmasının diğer işlemleri kısıtlaması şematize edilmiştir.
2. Grafikte ise `t` anında çalışmaya başlayan koyu gri renkli işlemin ayrı bir thread içerisinde çalışması şematize edilmiştir.

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_1(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 iki adet label nesnesi ekleyelim ve bu labeller 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 yarımşar saniye olarak belirliyoruz. Ayrıca `sayici2` adlı thread içerisinde yazdırılan rakam `15` ise, `3000 milisaniyelik` ek bir bekleme işlemi daha yaptırıyoruz.




```
CheckForIllegalCrossThreadCalls = false;
```

ifadesini şu amaçla yazdık;

Bu ifade 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 başlatma butonuna bastığımızda öncelikle `kanal1` içerisindeki sayma işleminin gerçekleştirildiğini, `kanal2` isimli thread içerisindeki sayma işleminin `kanal1` içerisindeki sayma işleminin bitmesinin ardından başladığını görüyoruz.

29

PARALEL PROGRAMLAMA

Artık standart bir hal 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** (**T***ask* **P***arallel* **L***ibrary*) geliştirilmiştir. Bu kütüphane çok çekirdekli işlemcileri kullanmak ve bu amaçla kolayca kod yazabilmek amacıyla tasarlanmıştır.

.NET içerisinde paralel programlama başlığı altında **Parallel LINQ**, **Parallel sınıfı**, **Task yapıları** bulunmaktadır. Bu yapılar `System.Threading.Tasks` sınıfı altında yer almaktadır.

Parallel Sinifi

Parallel sınıfı altında üç temel metod bulunmaktadır. Bunlar:

- `Parallel.Invoke`
- `Parallel.For`
- `Parallel.ForEach`

Parallel.Invoke Metodu

Bir metod dizisini aynı anda çalıştırmak amacıyla kullanılır. Kullanım biçimi şu şekildedir:

```
Invoke(ParallelOptions, Action())
```

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;
```

ifadesini şu amaçla yazdık;

Bu ifade 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 metod ismini çağırarak da işlem yaptırmamız mümkündür. Aşağıdaki örnekte hem işlem yapan kodu `Parallel.Invoke` içerisinde tanımlayarak çalıştırdık, hem de işlem yapan diğer bir kodu metod 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 Metodları

For ve ForEach komutları yardımıyla paralel işlemler gerçekleştirmek amacıyla kullanılır.

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.

Parallel.For ile

```
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.

İçiçe 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();
}
```

Bir dizi içerisindeki elemanları konsol ekranına yazdıran elemanları `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 Oluşturmak

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();
```

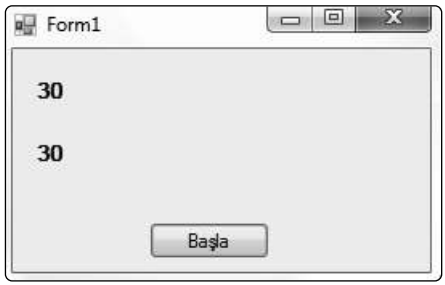
StartNew metodunun kullanım biçimi ise şu şekildedir:

```
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 + " ";
                }
            });
        }
    }
}
```



`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 metodlarını çağırırız.

Task İçerisinde Yürütülen İşlemleri Durdurmak

Task içerisinde yürütülen işlemleri durdurmak amacıyla `Wait`, `WaitAll` ve `WaitAny` metodları kullanılır.

Wait: Belirtilen bir işlemi durdurmak amacıyla kullanılır.

`TaskİsmiWait()` biçiminde kullanılır.

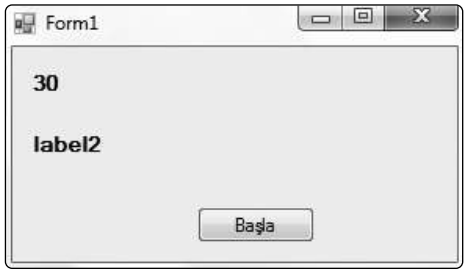
WaitAll: Belirtilen işlemleri durdurmak amacıyla kullanılır.

`Task.WaitAll(TaskİsmiTaskİsmi,...)` biçiminde kullanılır

WaitAny: Herhangi bir işlemi bekletmek amacıyla kullanılır.

`Task.WaitAny(TaskİsmiTaskİsmi,...)` biçiminde kullanılır.

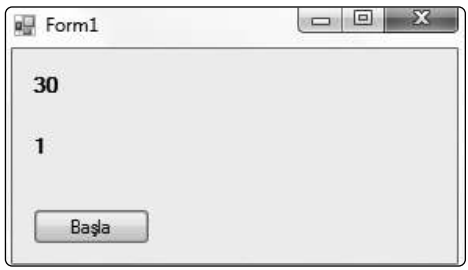
```
private void btnBasla_Click(object sender, EventArgs e)
{
     CheckForIllegalCrossThreadCalls = false;
     Task task1 = new Task(() =>
     {
          for (int i = 1; i <= 30; i++)
          {
               Thread.Sleep(500);
               label1.Text = i + " ";
          }
     });
     Task task2 = new Task(() =>
     {
          for (int j = 1; j <= 30; j++)
          {
               Thread.Sleep(500);
               label2.Text = j + " ";
          }
     });
     task1.Start();
     task2.Wait();
}
```



CONTINUEWITH( ) METODU

Task içerisinde yürütülen bir işlemin ardından başka bir işlemi başlatmak amacıyla kullanılır.

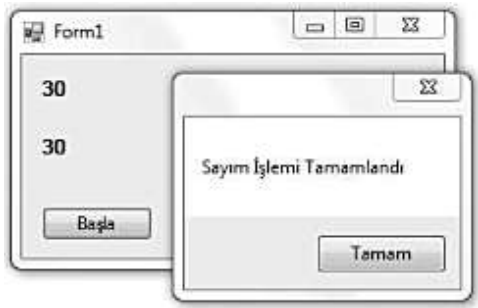
```
private void btnBasla_Click(object sender, EventArgs e)
{
     CheckForIllegalCrossThreadCalls = false;
     Task task1 = Task.Factory.StartNew(() =>
     {
          for (int i = 1; i <= 30; i++)
          {
               Thread.Sleep(500);
               label1.Text = i + " ";
          }
     })
     .ContinueWith((task2) =>
     {
          for (int j = 1; j <= 30; j++)
          {
               Thread.Sleep(500);
               label2.Text = j + " ";
          }
     });
}
```



IsCompleted( ) Metodu

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.

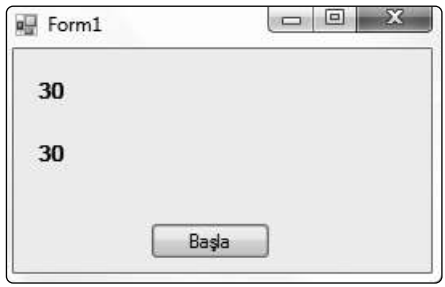
```
private void btnBasla_Click(object sender, EventArgs e)
{
     CheckForIllegalCrossThreadCalls = false;
     Task task1 = Task.Factory.StartNew(() =>
     {
          for (int i = 1; i <= 30; i++)
          {
               Thread.Sleep(500);
               label1.Text = i + " ";
          }
     })
     .ContinueWith((task2) =>
     {
          for (int j = 1; j <= 30; j++)
          {
               Thread.Sleep(500);
               label2.Text = j + " ";
          }
    if (task2.IsCompleted == true)
               MessageBox.Show("Sayım İşlemi Tamamlandı");
     });
}
```



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);
    });
}
```



30

SETUP HAZIRLAMA İŞLEMLERİ

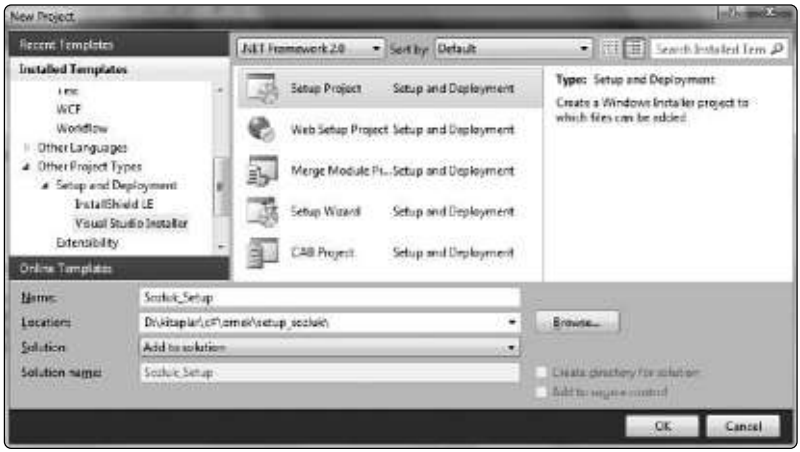
.Net projelerimizin exe dosyalarını WindowsVista'dan önceki işletim sistemlerinin yüklü olduğu bir bilgisayarda çalıştırmak için, XP sistemlerde .Net Framework'un işletim sistemi kurulumu sırasında yüklenmemesi nedeni ile .Net Framework'un yüklenmesi gerekmektedir.

Windows Vista ya da Windows 7 yüklü herhangi bir bilgisayarda ise .Net Framework'un yüklü olması nedeni ile rahatlıkla çalıştırabiliyoruz. Ancak Vista varsayılan olarak .Net Framework 2.0 versiyonu yüklü olarak gelirken, Windows7 ise Framework 3.5 versiyonu yüklü olarak gelmektedir. Daha üst Framework sürümünü kullanarak bir uygulama geliştirdiğimizde uygulamamızın çalışması için o Framework sürümünün yüklü olması gerekmektedir.

Bütün bunların dışında ise projelerimizin içerisinde kullandığımız ek component'ler ya da dosyalar olabilmektedir. Bu gibi durumlarda karşılaşabileceğimiz sorunların hepsini aşarak projemizi çalıştırabilmemiz için setup projesi hazırlamamız gerekmektedir.

Yeni bir setup projesi hazırlamak için setup oluşturacağımız projemiz açık iken **File / New Project / Other Project Types / Setup and Dep-**

loyment / Visual Studio Installer menülerinden **Setup Project** seçeneğini seçiyoruz. **Solution** kısmında **Create new solution** seçeneği yerine **Add to solution** seçeneğini seçmeliyiz.



Setup projemizin adını da verip **OK** düğmesini tıklıyoruz. Karşımıza aşağıda görüldüğü gibi setup ile ilgile gereken ayarlamaları yapabileceğimiz ekran gelecek.



File System on Target Machine başlığı altındaki bulunan seçenekler şunlardır:

Application Folder

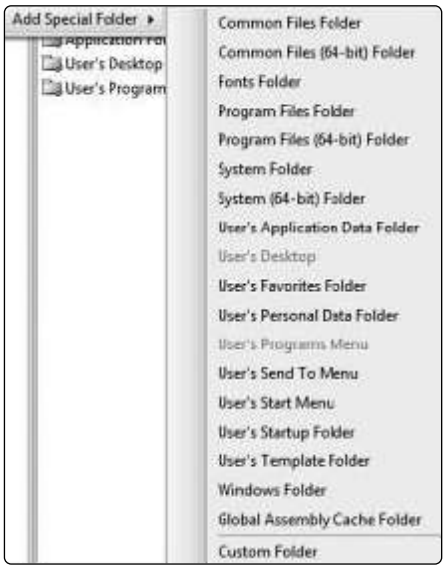
Bu seçenek altında kullanıcının bilgisayarına yükleyeceğimiz exe dosyamız, projemizde kullandığımız harici dosyalar (veritabanı dosyası, ini dosyası, şablon dosyaları, yardım dosyaları gibi), projemiz içerisinde kullandığımız dll dosyaları, kullanacağımız resim, simge, ses ve video dosyaları gibi dosyaların eklenme işlemi ve bu dosyalarla ilgili ayarlamalar yapılır.

User's Desktop

Kullanıcının bilgisayarında masaüstünde programın kısayolunun oluşturulması ve bu kısayolun simgesinin belirlenmesi işlemi bu bölümde ayarlanır.

User's Programs Menu

Kullanıcının bilgisayarında programlar menüsünde programın kısayolunun oluşturulması ve bu kısayolun simgesinin belirlenmesi işlemi bu bölümde ayarlanır.

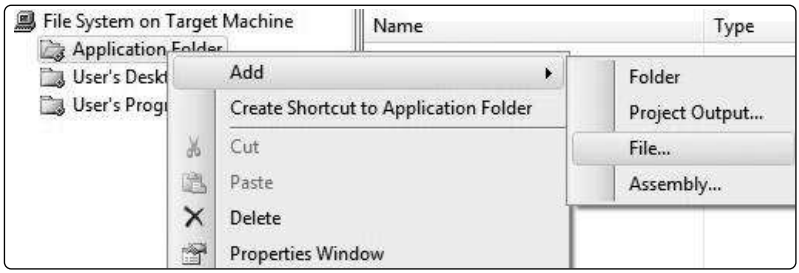


Bunların dışında klasör eklemek için ise panel içerisine farenin sağ tuşu ile tıklayarak **Add Special Folder** seçeneği tıklanır

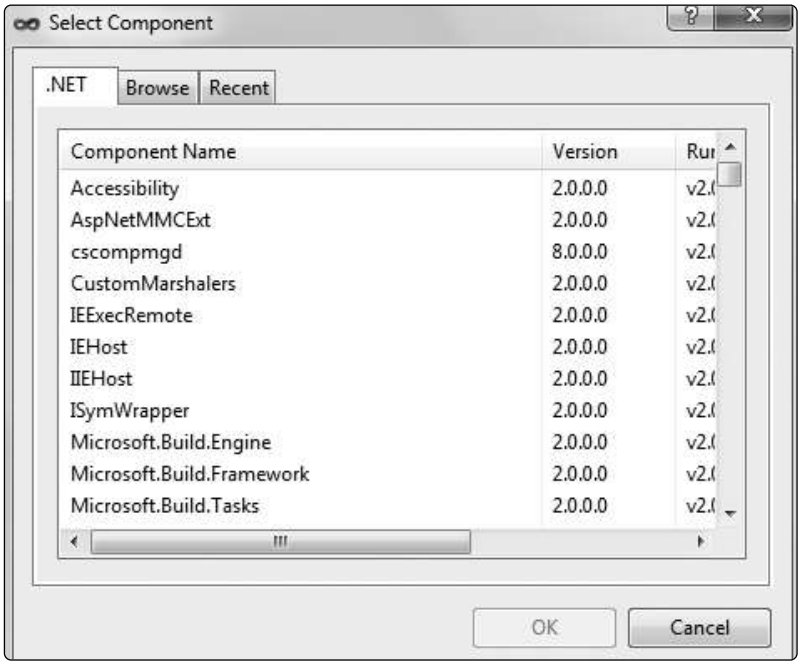
Buradaki örneğin **User's Startup Folder** seçeneği ile programımızın başlangıçta çalıştırılmasını, **User's Send To Menu** seçeneği ile programımızın gönder menüsüne eklenmesini sağlayabiliriz.

İlk olarak exe dosyamızı ekleyelim.

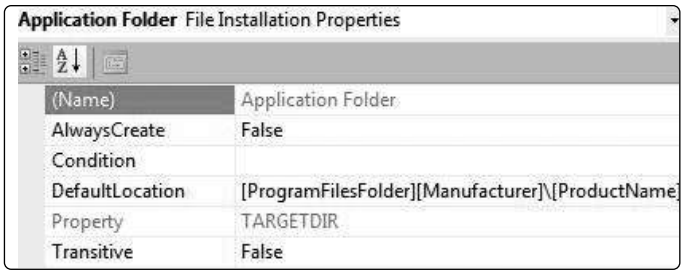
Application Folder seçeneğine farenin sağ tuşu ile tıklayarak **Add / File** seçeneğine tıklıyoruz.



Projemizin **Bin/Debug** klasörü altına gidip exe dosyamızı seçiyoruz. Exe dosyası ile birlikte proje içerisinde kullandığımız dll dosyaları da otomatik olarak eklenecektir. Eğer bunların dışında kullandığımız dll dosyası varsa exe dosyamızı eklediğimiz alanda farenin sağ tuşu ile tıklayarak **Add/Assembly** seçeneğini tıklarız. Karşımıza .Net referans dosyalarını ekleyebileceğimiz pencere gelir. Eklemek istediğimiz dll dosyalarını buradan ekleyebiliriz.



Uygulamamızın simgesi için kullanacağımız simge dosyasını da exe dosyamızı eklediğimiz alanda farenin sağ tuşu ile tıklayarak **Add/File** seçeneği ile ekleyelim.



Application Folder seçeneği seçili iken properties penceresinde **DefaultLocation** seçeneği ile programımızın kurulacağı yeri değiştirmemiz mümkündür.



Manufacturer, **ProductName** gibi ayarları değiştirmek için ise **Solution Explorer**'da **Setup** projemizin ismine fare ile tıklayıp properties penceresinde bu ayarları ve programımızın kurulumu ile ilgili daha bir çok ayarlamaları yapabiliriz.

AddRemoveProgramIcons seçeneği ile programımızın **Denetim Masası/Program Ekle Kaldır** bölümündeki program listesinde hangi simge ile görüntüleneceğini belirleriz. Bunu belirlemek için bu seçeneğe tıklayıp **Browse** seçeneği ile dosya seçme penceresini açıyoruz. Buradan

Browse seçeneği ile gelen pencereden **Application Folder** klasörünü seçip bu klasör altına daha önceden eklediğimiz simge dosyasını seçiyoruz.

Kullanıcının masaüstünde kısayol oluşturmak için **User's Desktop** klasörünün içerisinde farenin sağ tuşuna tıklayarak **Create New Shortcut** seçeneğine tıklayalım. Gelen pencereden **Application Folder** klasörünü seçerek exe dosyamızı seçiyoruz. Kısayolumuzu seçerek farenizin sağ tuşu ile tıklayarak rename seçeneği ile ismini düzenliyoruz. Yine kısayolumuz seçili iken properties penceresinden icon seçeneğine tıklayarak gelen pencereden **Application Folder** klasörü altındaki daha önceden eklediğimiz simge dosyasını seçiyoruz.

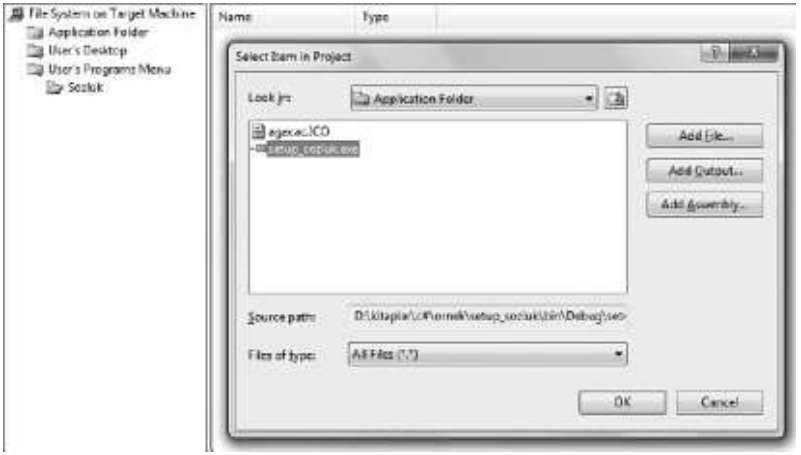
ShowCmd seçeneği altında uygulamamızın hangi ekran modunda çalışacağını belirleyebiliriz. Burada bulunan seçenekler şu şekildedir:

vsdscNormal: Uygulamamızın varsayılan ekran boyutları ile çalışmaya başlamasını sağlar.

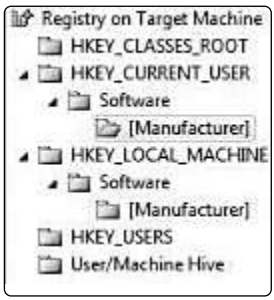
vsdscMaximized: Uygulamamızın tam ekran modunda çalışmaya başlamasını sağlar.

vsdscMinimized: Uygulamamızın simge durumunda çalışmaya başlamasını sağlar.

User's Programs Menu klasörü altındaki seçenekleri kullanarak uygulamamızın **Başlat** menüsüne yüklenmesi ile ilgili ayarları yapalım. Farenin sağ tuşu ile **User's Programs Menu** klasörünün içerisinde iken tıklayarak **Add Folder** seçeneğine tıklayıp **Başlat** menüsünde uygulamamızın klasörünü oluşturalım. Klasörü farenin sağ tuşu ile tıklayarak **rename** seçeneği ile ismini değiştirelim. Daha sonra klasörün içerisine farenin sağ tuşu ile tıklayarak **Create New Shortcut** seçeneğine tıklayalım. Gelen pencereden **Application Folder** klasörünü seçerek exe dosyamızı seçiyoruz. Kısayolumuzu seçerek farenizin sağ tuşu ile tıklayarak rename seçeneği ile ismini düzenliyoruz. Yine kısayolumuz seçili iken properties penceresinden icon seçeneğine tıklayarak gelen pencereden **Application Folder** klasörü altındaki daha önceden eklediğimiz simge dosyasını seçiyoruz.



Solution Explorer'dan **Registry Editor**'u açarak programımızla ilgili yapacağımız registry düzenlemelerini yapabiliriz.

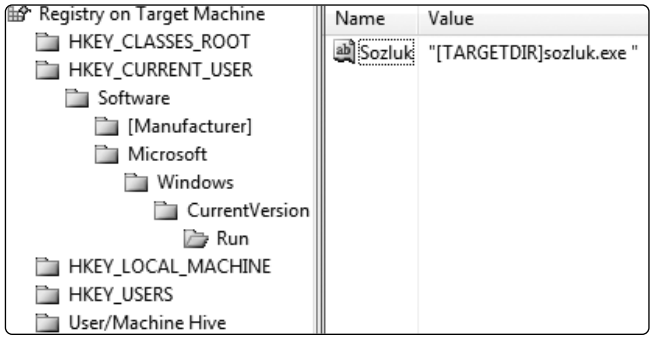


Registry Editor içerisindeki **HKEY_CURRENT_USER / Software / [Manufacturer]** ve **HKEY_LOCAL_MACHINE / Software / [Manufacturer]** klasörleri registry içerisinde uygulamamız için oluşturulan klasörleri temsil etmektedir. Bu klasör altında uygulamamız için anahtar ve değerler oluşturabiliriz. Örnek olarak ben setup oluşturduğumuz proje içerisinde programın ilk defa kurulup kurulmadığını belirleyen kod satırları kullandım. Bunun için gerekli olan anahtar ve değerleri registry içerisinde oluşturalım.

Manufacturer klasörü içerisinde iken sağ tuşla tıklayarak **New / String Value** değerini seçelim ve `CalismaSayisi` adı ile bir değer oluşturalım. Eğer istenilirse değerimiz seçili iken **Properties** penceresinde value özelliğine bir değer girilerek bu özelliğimize varsayılan bir değer verebiliriz.

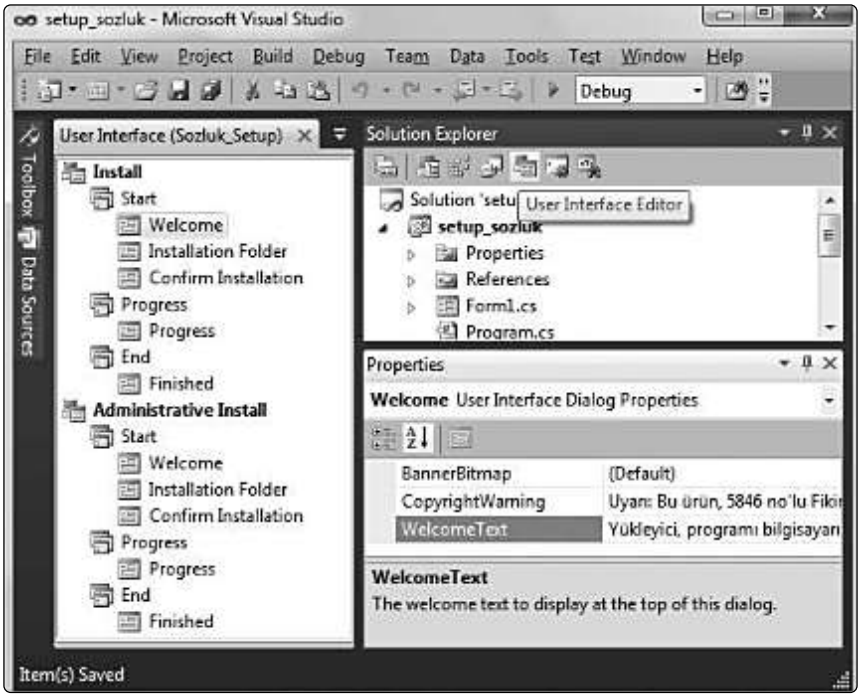
Yine **Registry Editor** vasıtası ile programımızın bilgisayarın açılışında otomatik olarak çalışması için gereken değerleri oluşturalım. **HKEY_CURRENT_USER / Software** klasörlerini seçerek farenin sağ tuşu ile klasör ismine tıklayıp **New / Key** seçeneği ile anahtarları alt alta oluşturmaya başlıyoruz.

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run şeklinde tüm anahtarları alt alta oluşturuyoruz. Son anahtarımız olan Run anahtarının içerisinde **New / String Value** diyerek yeni bir string değer oluşturuyoruz. Değerimizi sağ tıklayıp **Rename** seçeneği ile uygun bir isim veriyoruz. Örneğin ben Sozluk ismini verdim. Properties penceresinden value özelliğine uygulamamızın çalışma klasörünü ve adını `[TARGETDIR]uygulamaadı.exe` şeklinde yazacağız. Ben, `[TARGETDIR]sozluk.exe` şeklinde yazdım.



Solution Explorer'dan **User Interface**'i açarak programımızın kurulum ekranları ile ilgili gerekli düzenlemeleri yapabiliriz.

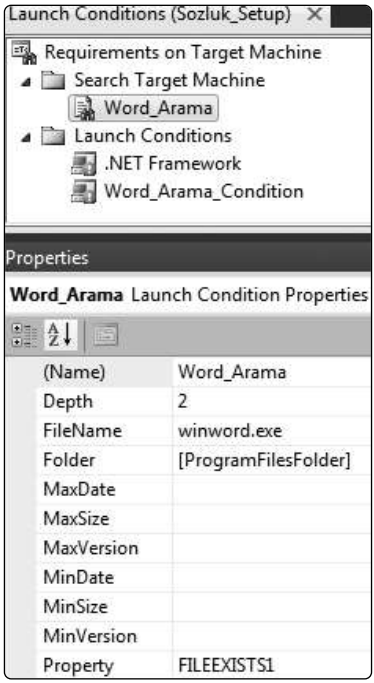
Örneğin programın kuruluşu esnasında ekrana gelecek olan ilk formu değiştirelim. Bu form **Start** başlığı altında **Welcome** adı ile yer almaktadır. Bu formun uyarı metnini properties penceresinden **Copyright-Warning** seçeneği ile değiştirelim. Burada yer alan İngilizce metin yerine *"Uyarı: Bu ürün, 5846 no'lu Fikir ve Sanat Eserleri Kanunu ile korunmaktadır. İzinsiz kopyalanması ve bu programın tamamının veya herhangi bir bölümünün dağıtımı yasaktır."* şeklinde Türkiye Bilişim yasalarına uygun bir metin yazalım.



WelcomeText seçeneğinde yer alan İngilizce metni ise *"Yükleyici, programı bilgisayarınızda [ProductName] yüklemek için size yol gösterecektir."* şeklinde değiştirelim.

BannerBitmap seçeneği ile form üzerinde programımız için görüntülenecek varsayılan resim dosyasını değiştirebiliriz. Örnek olarak hazırladığımız resim dosyasını **File System** bölümünden **Application Folder** içerisine yükleyelim ve **BannerBitmap** seçeneğinden **Browse** seçeneğine tıklayarak resim dosyamızı ayarlayalım.

Solution Explorer'dan **Launch Conditions** simgesini tıklayarak programımızın kurulacağı bilgisayar ile ilgili ayarlamaları yapalım.



Örneğin kurulumunu hazırladığım projede word belgelerine aktarma işlemi yaptığım için bilgisayarda ofis programının yüklü olması gerekmektedir. Ofis programının yüklü olup olmadığını kontrol ederek eğer yüklü değilse uyarı verilmesini sağlayalım.

Launch Conditions içerisinden **Search Target Machine** klasörünü seçip farenin sağ tuşu ile tıklayarak **Add File Search** seçeneğine tıklayalım. Oluşturulan **Search for File1** dosyasını *"Ofis_Arama"* olarak değiştirelim. **Properties** penceresinden **FileName** özelliğine **winword.exe** yazalım. **Folder** özelliğinden **[ProgramFilesFolder]** seçeneğini seçelim. **Depth** seçeneği ile belirttiğimiz klasörün kaç seviye alt klasörüne inileceğini belirliyoruz. Buraya da 2 değerini giriyoruz. **MinVersion**, **MaxVersion** değerlerine word programının minimum ve maksimum versiyonları yazılabilir. Böyle bir sınır koymak istemediğim için boş bıraktım.

Launch Conditions klasörünü seçerek farenin sağ tuşu ile **Add Launch Condition** seçeneğini tıklıyoruz. Oluşturulan *"Condition1"* değerinin ismini *"Ofis_Arama_Condition"* şeklinde değiştirdim.

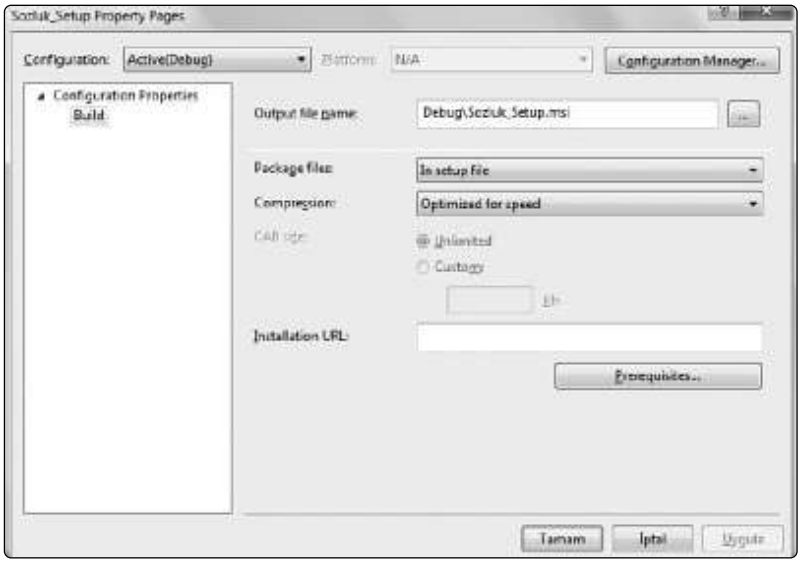
Properties penceresinden condition seçeneğinden "FILEEXISTS1" değerini seçiyoruz. **Message** özelliğine *"Uygulama çalışabilmek için Word 2000 ve üstü versiyonlara ihtiyaç duymaktadır. Deneme sürümünü indirmek ister misiniz?"* mesajını giriyoruz.

InstallUrl seçeneğine Ofis 2007 programının 60 günlük deneme sürümünü indirme adresi olan *http://trial.trymicrosoftoffice.com/wwtrial/product.aspx?re_ms=oo&family=officepro&culture=en-GB* adresini giriyorum.

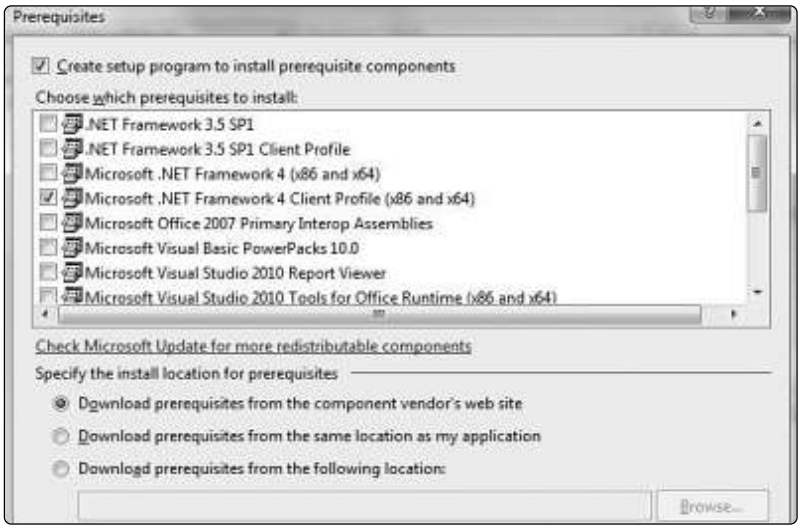


Son aşamamızda framework'un yüklenmesi ile ilgili ayarları yapacağız. Programımızın kurulacağı bilgisayarda .Net Framework programının yüklü olmaması durumu için kurulum projemiz içerisinde .Net Framework'un yüklenmesini sağlayabiliriz. Programımızın yüklenmesi esnasında framework yüklü olup olmadığının kontrolü yapılmaktadır. Eğer yüklü değilse kullanıcının bir link vasıtası ile framework dosyasını indirmesini sağlayabileceğimiz gibi, kurulumun yapıldığı bilgisayarda internet bağlantısının olmaması gibi durumlar için framework dosyasını setup projesi içerisine de eklenebilir.

Solution Explorer'da setup projemizin adını seçip farenin sağ tuşu ile **Properties** seçeneğine tıklıyoruz. Gelen **Property Pages** sayfasından **Prerequisites** butonunu tıklıyoruz. Buradan programımızın çalışması için gerekli olan yeterlilikleri ayarlayacağız.



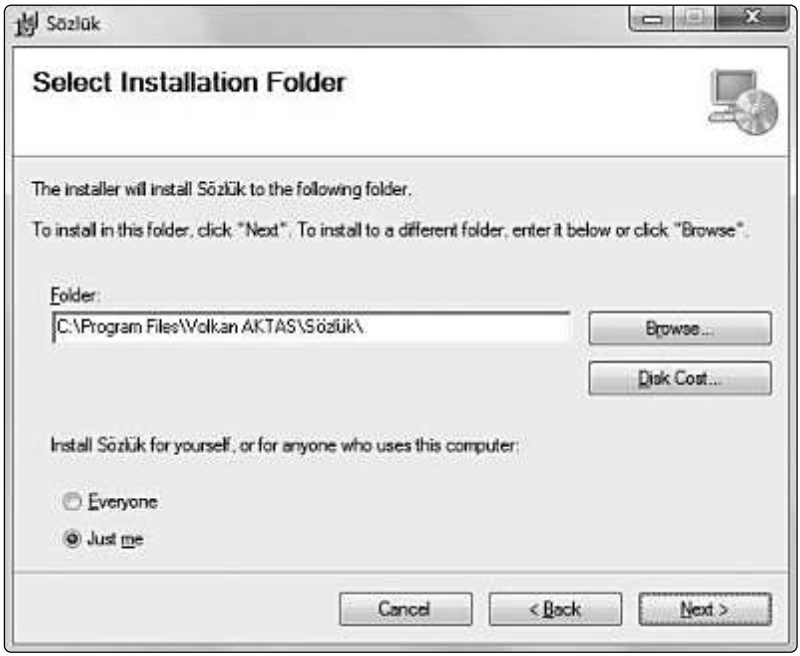
Buradan kullanacağımız Framework versiyonunu seçiyoruz. **Specify the install location for prerequisites** bölümündeki ilk seçenek olan **Download Prerequisites from the component vendor's web site** seçeneği işaretli iken seçili olan dosyalar bilgisayarda mevcut olmadığı zaman internet sitesinden yüklenir. Biz bu seçeneği seçiyoruz.



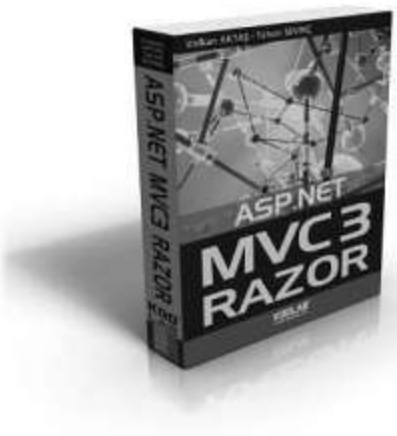
İkinci seçenek olan **Download Prerequisites from the same location as my application** seçeneği ile setup uygulamamız içerisine eklemiş olduğumuz dosyalardan yükleme yapılmasını sağlarız. Diğer seçenek olan **Download Prerequisites from the following location** seçeneği seçilip **Browse** butonu ile belirtilen klasörden yükleme yapılması sağlanır. Ancak bunun için öncelikle framework dosyasının eklenmesi gerekiyor. Projemizi geliştirirken Framework'un hangi sürümünü kullanıyorsak onun eklenmesi gerekiyor. **Solution Explorer**'dan **File System** simgesine tıklayarak **Application Folder** klasörünü seçiyoruz. Farenin sağ tuşu ile tıklayarak **Add File** seçeneği ile framework dosyamızı ekliyoruz.

Son olarak **Solution Explorer**'dan setup projemizi seçerek **Build** seçeneği ile setup projemizi derleyelim. Setup dosyalarımız Setup projemizin **Bin/Debug** klasörü içerisinde oluşturulacaktır.

Setup dosyamızı test ettiğimizde projemizin hatasız olarak kurulduğunu göreceğiz.




SILVERLIGHT 4
Silverlight


ASP.NET MVC3 RAZOR
ASP.NET
MVC3
RAZOR


WPF
WPF


FRAMEWORK 4.0
FRAMEWORK
4.0


4.0
C# ile
ASP.NET


ALGORİTMA
ALGORİTMA


HER YÖNÜYLE
HTML5
HTML5 • JavaScript • CSS3


SQL SERVER
2008 R2


OBJECTIVE-C


C++ EĞİTİM SETİ
C++


10 ADIMDA
YAZILIM GELİŞTİRME


PROGRAMLAMA
&
VERİTABANI

KODLAB®
"Bilişim Yayıncılığının Yeni Yüzü"