Fatih KAPLAN



BİLGİSAYAR MÜHENDİSLİĞİNE GİRİŞ

KODLAB®
www.kodlab.com

BİLGİSAYAR MÜHENDİSLİĞİNE GİRİŞ

Fatih Kaplan

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

BİLGİSAYAR MÜHENDİSLİĞİNE GİRİŞ

FATİH KAPLAN

ISBN 978-605-2118-52-8
Yayıncılık Sertifika No: 13206

1. Baskı: Ekim 2020

Genel Yayın Yönetmeni
Hüseyin Üstünel

Grafik Tasarım
Mustafa Said Şahin

Editör
Gizem Aksan

Satış ve Pazarlama
Can Üstünel
Y. Emre Üstünel

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

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

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

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

Fatih Kaplan

Yazarımızın bilgisayar ve programlama dünyası ile tanışması lise yıllarına dayanıyor. Konya Meram Anadolu Ticaret Meslek Lisesi Bilişim Teknolojileri Alanı ve Veri Tabanı Programcılığı bölümünde lise eğitimimi tamamladıktan sonra Selçuk Üniversitesi Bilgisayar Programcılığı ön lisans programını ve ardından KTO Karatay Üniversitesinde Bilgisayar Mühendisliği bölümünde eğitimini tamamlıyor. Lisansüstü eğitim hayatını ise yine KTO Karatay Üniversitesi Fen Bilimleri Enstitüsünde Elektrik Bilgisayar Mühendisliği Tezli Yüksek Lisans Programında tamamlayan yazarımız, Konya Teknik Üniversitesi Bilgisayar Mühendisliği Doktora programında öğrenciliğine devam etmektedir. İş hayatına ise üniversitede bilgi teknolojileri araştırma ve uygulama direktörlüğünde yönetici olarak devam etmektedir.

Aileme ve tüm sevdiklerime ithafen…

ÖN SÖZ

Kitabımızda bilgisayar kavramı irdelenmiş olup geçmişten günümüze bilgisayar konu başlığı altına giren alt başlıklara yer verilmiştir. Bilgisayar için akılda kalıcı öz bir tanıma yer verilmiş olup sırasıyla Bilişim Teknolojilerinin Temelleri, Yazılım, İşletim Sistemleri Detayları, Sunucu Sistemleri, Bilgisayar Sistemleri ve Sayısal Elektronik, Algoritma ve Programlama Giriş, Nesne Tabanlı Programlama, Veri Tabanı, Veri Yapıları, Bilgisayar Mimarisi, Algoritmaların Analizi, Otomata Teorisi, Bilgisayar Ağları, Yapay Sinir Ağları-Makine Öğrenmesi ve Derin Öğrenme, Pattern-Anti Pattern-Secure Design Pattern Kavramları ve son olarak Web Programcılığı hakkında bilgiler verilmiştir.

Bu kitabın amacı, bu denli geniş başlıkların her birini detaylı anlatmak değildir. Kitabımızın amacı; ilgili bölümlerde okuyan üniversite öğrencilerine eğitim süresince ne tür konularla karşılaşabilecekleri ve kendileri üzerine ne koyabilirler bunun bilincini oluşturmak ve kılavuzluk etmek, sektörde yer alan Junior-Senior bilişime gönül vermiş kimseler için ise ilgili konu başlıkları ile ilgili Türkçe kaynak sağlamaktır.

Kitabımız içerisindeki konular okuyuculara kültür oluşturacak, eğitim sürecinde olan okurlar için kendilerini bilişimin hangi alanında geliştirmek istedikleriyle ilgili yol gösterecek, mezun durumundaki kimseler için ise mülakatlarda gelebilecek sorulara karşı bilgi birikimi oluşmasını sağlayacaktır. Özellikle eğitim müfredat içeriklerine yakın hazırlanmış olması öğrencilere yönelik bir kitap olduğu düşüncesini oluşturacaktır. İlgili konuların her birinin tek bir yerde toplanması sayesinde ilgi duyan herkes kitabımızdan istifade edebilecektir.

Donanım konularının yer aldığı ilk bölümlerde okuyuculara bilgisayarın donanımının ne olduğu, eski teknolojilerden yeni teknolojilere doğru gelişiminin ne şekilde seyrettiği ve bir bilgisayarın bileşenlerinin neler olduğu, nasıl çalıştığı gibi kavramlar anlatılmıştır. Bu bölümde bir bilgisayarın içerisinde bulunan donanım parçaları tek tek anlatılmış okuyucuyu donanım katmanındaki işler bu şekildeyken yazılım katmanında ne şekilde olduğuna dair merak uyandırarak üçüncü bölüm olarak yazılım anlatılmıştır. Genel olarak yazılım tanımı, çeşitleri, günümüzdeki yerinden bahsedilen üçüncü bölümde donanımların çalışması için gerekli olan işletim sistemlerinin de bir yazılım olduğundan bahsedilerek dördüncü bölümde işletim sistemleri anlatılmıştır.

Günümüzde genç okurlarımızın çocukluk yaşlarından beri bir şekilde bilgisayarlar ile tanıştığını düşündüğümüzde işletim sistemi kullanma oranının çok yüksek olduğundan söz edebiliriz. Ancak bir işletim sisteminin aslında ne şekilde çalıştığı, çeşitleri, detaylı özellikleri gibi teknik olarak arkada nelerin döndüğü ayrıntılı bir şekilde anlatılmıştır. İşletim sistemlerinin son kullanıcı, terminal, telefon, tablet gibi çeşitli yerlerde kullanım örneklerinin bulunduğu detayı da verildikten sonra beşinci bölümde sunucu işletim sistemleri hakkında bilgilendirmeler yapılmıştır.

Özellikle öğrenci okuyucularımızın sunucular için farklı, PC'ler için farklı işletim sistemi versiyonunun olduğu ve kurulumdan işletmeye çeşitli farklılıklar olduğunun bilinci bu bölümde oluşturulmuştur.

Altıncı bölümde artık bilgisayarın gözümüzle gördüğümüz iki önemli unsuru olan donanım ve yazılımı iyice pekiştirdiğimizi düşünerek donanımda işlerin nasıl yürüdüğü hakkında fikir sahibi olunması açısından bilgisayar sayı sistemleri bölümü hazırlanmıştır. Bu bölümde elektronik ortamda mantıksal işlerin ne şekilde yapılabildiği hakkında bilgiler verilmiştir.

Altıncı bölümün meslek liselerinde Temel Elektronik Ölçme, Ön Lisans programlarında Sayısal Elektronik ve Lisans programlarında Digital Design olarak karşılıkları hep vardır. Bilgisayar için önemli bir konu başlığıdır. Bizlerin bu bölümü gayet iyi özümlemesi gerekmektedir. Donanımdan bahsederek yazılıma ihtiyaç duyulduğunu hemen arkasından yazılımdan bahsederek işletim sistemlerinin de bir yazılım olduklarını söyledik. Daha sonra donanımlarında elektronik ortamda ne şekilde çalışabildikleri ile ilgili olarak altıncı bölüme yer verdik.

Artık yedinci bölümle birlikte yazılımlarında arka tarafta ne tür süreçlerden geçtiğini anlayabilmenin vakti geldi diye algoritma ve programlama giriş başlığında gerekli konuları işledik. Dikkat edilecek olursa konular bilgisayar biliminin farklı disiplinlerine ait olsalar da sonuç itibariyle hepsi birbiriyle ilişkili ve her bir konunun ayrı ayrı iyi öğrenilmesinin iyi bir bilgisayar bilimcisi olmak için ne kadar önem arz ettiği görülmektedir.

Sekizinci bölümde programlama konseptlerine giriş yaparak günümüz dördüncü seviye programlama dilleri özelliklerinden nesne tabanlı programlama kavramına yer verilmiştir. Programlamadan bahsederken özellikle de yeni nesil programcılıktan bahsederken veri tabanı kavramını hemen arkasından anlatmamak olmazdı. Bu yüzden donanımı, yazılımı, işletim sistemlerini, sayısal elektroniği, programlama dillerini öğrendikten sonra veri tabanı hakkında konuşabilmek için dokuzuncu bölüm hazırlanmıştır.

Programlama dilleri ile yazılan kodlarda prosedürler, metotlar, değişkenler ve bir dize kurallar vardır. Veri tabanı haberleşmede de bu kurallarla birlikte veri tabanı kuralları devreye girer. Veri üzerinde çalıştığımız bu bölümün hemen arkasından onuncu bölümde veri yapılarına ve algoritmalara değinilmiştir. İçerik olarak aslında bir donanım bir yazılım konu başlıklarıyla gidiyor olsak da konuların her biri kendinden önce gelen konu ile bir şekilde ilişkilidir. Veri yapıları soyut bir kavram olduğu için örneğin işletim sistemi algoritmalarından veya sıralama algoritmalarından bahsederken donanım katmanında işlerin nasıl olduğunun detayına inebilmek için on birinci bölümde bilgisayar mimarisine yer verilmiştir. On ikinci bölümde on ve on birinci bölümün cilası niteliğinde algoritmaların analizlerine değinilmiştir. Bu katmanda yer alan konuların biraz daha üst seviyeye çıkarılması ile ilgili olarak on üçüncü bölüm hazırlanmış ve otomata teorisi anlatılmıştır.

Donanım, yazılım ve mimarilere değindikten sonra olmazsa olmaz bilgisayarın iletişimi için ağ kavramına on dördüncü bölümde değinilmiştir. Temel network konuları ve network'ün işleyişte ne tür cihazlar ile ne şekilde hayatımızda olduğuna dair yararlı bilgiler verildikten sonra on beşinci bölümde YSA, Machine Learning ve Deep Learning kavramları anlatılmıştır.

On altıncı bölümde yazılım mühendisliğinin olmazsa olmazı tasarım şablonlarına yer verilmiştir. Son bölümde ise web programcılığına ve web ile yakından ilgili günümüzde hayatımıza bir şekilde girmiş olan konulara yer verilmiştir. Kitabımızın tüm okurlara fayda sağlamasını, okurlarımızın edinecekleri bilgileri çok daha öteye taşıyacak işler yapmalarına zemin hazırlanmasını ve bu bilgi birikimini yayınlarla, kitaplarla, makalelerle başka insanların öğrenmelerine katkı sağlayacakları projelere imza atmalarını temenni ederim.

Fatih KAPLAN

İÇİNDEKİLER

1 GİRİŞ **1**
Temel Kavramlar 2
Bilgisayar Nedir? 2
Bilgisayar Mühendislerinin Çalışma Alanları 2

2 BİLİŞİM TEKNOLOJİLERİNİN TEMELLERİ **5**
Anakart 7
Yonga Seti 7
Veri Yolları 8
İşlemci (CPU) 10
Ön Bellekler 13
İletişim Yolları 13
İşlemci Mimarileri 17
CISC Mimarisi 17
RISC Mimarisi 17
İşlemci Teknolojileri 18
Centrino İşlemciler 18
Çift Çekirdekli İşlemciler (Dual Core) 18
Hyper-Threading 18
Ön Bellek Teknolojisi 18
Turbo-Boost Teknolojisi 19
İşlemci Soğutma Yöntemleri 19
Havayla Soğutma Yöntemi 20
Su ile Soğutma Yöntemi 20
Isı Borulu Soğutma Sistemi 20
Bellekler 21
RAM (Random Access Memory) 21
ROM 24
Kasa ve Güç Kaynağı 25

Kasa Çeşitleri 25
Güç Kaynağı 26
Depolama Birimleri 27
Sabit Diskler 27
Kart Okuyucular ve Hafıza Kartları 30
Optik Sürücüler 31
Ekran Kartı ve Diğer Kartlar 32
Ekran Kartı 32
Ses Kartı 34
TV Kartı 34
Ethernet Kartı 35
Diğer Kartlar 36
İşletim Sistemleri 36
Sistem Geri Yükleme 40
Disk Birleştirme, Temizleme, Denetleme 41
Kayıt Defteri 42
Kullanıcı Hesapları ve Güvenlik 43
Bios 44
Sesli Hata Mesajları 50
Yazılı Hata Mesajları 50

3 YAZILIM 51
Yazılım Nedir? 51
Yazılım Mühendisliği Nedir? 52
İşletim Sistemi Yazılımları 53
Paket Yazılımlar 54
Sektörel ve Ticari Yazılımlar 54
Gömülü Sistem Yazılımları 55
Web Yazılımları 55
Web Servis Yazılımları 56

4 İŞLETİM SİSTEMLERİ DETAYLARI 59
Sistem Özellikleri 59
Bilgisayar Adı Sekmesi 62
Donanım Sekmesi 65
Sistem Geri Yükleme Sekmesi 67
Uzak Masaüstü Sekmesi 68
Regedit 69
Kayıt Defteri Anahtarları 70
Sistem Yapılandırması 72
Genel Sekmesi 72
Hizmetler Sekmesi 74
Başlatma/Başlangıç Sekmesi 74
Güvenli Kip 74
Yönetimsel Araçlar 76
Bileşen Hizmetleri 76
Bilgisayar Yönetimi 77
Hizmetler 77
Olay Görüntüleyicisi 78
Performans 80
Temel İşletim Sistemi Detayları 80
IRQ Kesme İstekleri 80
DMA Doğrudan Bellek Erişimi 82
I/O Giriş Çıkış Adresleri 82
İşletim Sistemi Bileşenleri 82
Çekirdek 82
Dosya Yönetim Sistemi 83
Kullanıcı Arabirimi 84
İşletim Sistemi Türleri 84
Multiuser 84
Multitasking 84

Multiprocessing 85
Multithreading 85

5 SUNUCU SİSTEMLERİ 87
Sunucu İşletim Sistemleri 88

6 BİLGİSAYAR SAYI SİSTEMLERİ VE SAYISAL ELEKTRONİK 91
Sayı Sistemleri ve Dönüşümler 91
ASCII Tablosu 97
Lojik Kapılar 98
VE (AND) Kapısı 99
VEYA (OR) Kapısı 101
DEĞİL (NOT) Kapısı 102
VE DEĞİL (NAND) Kapısı 103
VEYA DEĞİL (NOR) Kapısı 104
ÖZEL VEYA (XOR) Kapısı 105
ÖZEL VEYA DEĞİL (XNOR) Kapısı 106
Boolean Matematiği 107
De Morgan Teoremi 108
Karnaugh Haritaları 109

7 ALGORİTMA VE PROGLAMLAMAYA GİRİŞ 113
Algoritma 114
Akış Diyagramları (Flowchart) 116
Örnekler 118
Programlama Dilleri 123
Programlama Dillerinin Tarihçesi 124
Programlamada İşlemler 126
Matematiksel İşlemler 127
Karşılaştırma İşlemleri 128
Mantıksal İşlemler 131
Programlama Dili Çeşitleri 133

8 NESNE TABANLI PROGRAMLAMA (OBJECT ORIENTED PROGRAMMING) 137

Nesne Nedir? 138
Sınıf Nedir? 139
Nesne ve Sınıf İlişkisi 140
Soyutlama (Abstraction) 140
Kapsülleme (Encapsulation) 141
Kalıtım/Miras Alma (Inheritance) 141
Çok Biçimlilik (Polymorphism) 142

9 VERİ TABANI 147

Veri Tabanı Nedir? 147
Veri Tabanı Yönetim Sistemi 149
Yapısal Sorgulama Dili 149
SQL Server 150
Kısa Kısa SQL Server 151
Management Studio 151
T-SQL 156

10 VERİ YAPILARI (DATA STRUCTERS) 169

Linked List (Bağlı Liste) 170
Stack (Yığın) 172
Queue (Kuyruk) 173
Recursion (Özyineleme) 174
Hashing 174
Tree (Ağaç Veri Modeli) 175
Bubble Sort (Kabarcık Sıralaması) 176
Quick Sort (Hızlı Sıralama) 177
Selection Sort (Seçmeli Sıralama) 178
Insertion Sort (Eklemeli Sıralama) 179
Merge Sort (Birleştirmeli Sıralama) 180

11 BİLGİSAYAR MİMARİSİ **183**
RISC Mimarisi 186
CISC Mimarisi 187
İşlemci Bellek Hiyerarşisi 188

12 ALGORİTMALARIN ANALİZİ (ANALYSIS OF ALGORITHMS) **191**
Karmaşıklık Sınıfları (Complexity Classes) 194
Birleştirme Sıralaması Algoritması (Merge Sort) 195
Taban Sıralaması Algoritması (Radix Sort) 198
Seçerek Sıralama Algoritması (Selection Sort) 199
Hızlı Sıralama Algoritması (Quick Sort) 200
Yığınlama Sıralaması Algoritması (Heap Sort) 202
İkili Arama Algoritması (Binary Search) 204
Doğrusal Arama Algoritması (Linear Search) 205
Açgözlü Yaklaşımı Algoritması (Greedy Algorithm) 205

13 OTOMATA TEORİSİ (AUTOMATA THEORY) **209**
İçerikten Bağımsız (CFG-CFL) 210
Backus Normal Form (BNF) 211
Sonlu Durum (Finite State) 211
Turing 213
Chomsky (CNF) 213
Pompalama (Pumping Lemma) 214
YACC 214
LEX 215
YACC-LEX Uygulaması 215

14 BİLGİSAYAR AĞLARI **223**
Ağ Çeşitleri ve Topoloji 225
Kablo Çeşitleri 227
Kablosuz Bağlantılar 229

OSI Referans Modeli 230
Protokoller (Portlar) 235
VLAN, Statik-Dinamik VLAN 236
IP Adresi Çeşitleri ve Sınıfları 238
Ağ Cihazları 239
Örnek Bir Kurgu 240

15 YAPAY SİNİR AĞLARI-MAKİNE ÖĞRENMESİ VE DERİN ÖĞRENME KAVRAMLARI 243
Yapay Sinir Ağları (YSA) 243
YSA Çalışma Prensibi 247
Makine Öğrenmesi 250
Derin Öğrenme 253

16 DESIGN PATTERNS-ANTI-PATTERNS VE SECURE DESIGN PATTERNS KAVRAMLARI 257
Design Patterns 259
Anti-Patterns 261
Secure Design Patterns 262
Mimari Seviye Güvenli Tasarım Desenleri 264
Tasarım Düzeyi Güvenli Tasarım Desenleri 274
Uygulama Düzeyi Güvenli Tasarım Desenleri 291

17 WEB PROGRAMCILIĞI 311
HTML 314
CSS 315
Javascript 316
JQuery 317
PHP 317
AJAX 317
JSON 318
XML 318

ASP.NET 318
JSP 319
ASP.NET MVC 319
ASP.NET MVC CORE 319
ADO.NET 320
LINQ TO SQL 320
ENTITY FRAMEWORK 321
DAPPER 321
IIS 322
APACHE 322
Tomcat 322
SSL Çeşitleri 324

1

GİRİŞ

Bilgisayar veya yazılım mühendisliği bölümlerinden bir tanesinde öğrenim görüyorsanız veya mezun olduysanız henüz birinci sınıftayken derslerde **bilgisayarın tanımı** ile **karşılaşmayabilirsiniz**. Mühendislik disiplinlerinde bölüme özel derslerin yanı sıra **calculus**, **signal and systems**, **differential equations**, **fluid mechanics**, **numerical analysis** veya **fizik** gibi temel dersler, öğrencilere özellikle birinci sınıfta verilmeye başlandığı için çoğu öğrenci temel derslerin vermiş olduğu kaygı ile bölümüne ait derslerin temel kavramları ile ilgilenmemektedir. Herhangi bir mühendislik disiplini için **temel kavramlara hâkim olmak**, sonraki sınıflarda alacakları derslerde veya karşılaştıkları projelerde **kolaylık sağlayacaktır**. Örneğin; birinci sınıfta fizik 2 dersini alan bir bilgisayar mühendisliği öğrencisi, sonraki dönemlerde alması muhtemel olan elektrik elektronik mühendisliğine giriş dersi için gerekli altyapıyı sağlamış olacaktır.

Mühendislik fakülteleri, yapısı gereği meslek liseleri veya ön lisans programlarında öğretilen temel bilgileri öğrencinin bildiğini kabul ederler veya müfredatlarında olmadığı için birinci sınıf öğrencisinin ilk tanıştığı meslek dersi genelde **Bilgisayar Mühendisliğine Giriş** ve **C programming 101** olmaktadır. Elbette, üniversiteden üniversiteye değişkenlik gösteren müfredatlar, akademisyenden akademisyene farklılık gösteren ders anlatımları ile tüm üniversitelerde temel kavramlar verilmiyor diyemeyiz ama geneli düşünerek ve öğrencinin ihtiyaç duyduğu konuları tek bir yerde toplayarak *henüz aday öğrenciler için de bilgisayar mühendisleri ne tür dersler ile ilgileniyor* sorusunun cevabını verebilmek amacıyla, **temel kavramlardan** başlayıp sırası ile müfredatta yer alan yaygın konular üzerinde durulacaktır.

Temel Kavramlar

Bilgisayar Nedir?

Bir bilgisayar, icat edildiği ilk günden beri kullanıcıdan aldığı veriler ile yine kullanıcının istediği işlemleri mantıksal bir çerçevede yapabilen, verileri fiziksel belleklerde istenildiği kadar **saklayabilen**, saklanılan bilgileri kullanıcının istediği şekilde **tekrar işleyip sunabilen elektro manyetik bir cihazdır**. Manyetik denilmesinin sebebi, hard disk yapısından kaynaklanır ve tanım çoğu yerde bizim tanımımıza benzer şekilde yapılmaktadır. Bilgisayar çeşitli donanım birimlerinden oluşur ve çalışabilmesi için **işletim sistemi** denilen bir yazılıma ihtiyacı vardır.

Bilgisayar Mühendislerinin Çalışma Alanları

Günümüz iş dünyasında, farklı disiplinde eğitim almış bir mühendis de bilgisayar mühendislerinin yaptığı işleri yapabilmektedir. Örneğin programlamaya veya networke ilgi duyan elektrik elektronik mühendisliği mezunu birinin de bilgisayar mühendisi birinin yaptığı işi sektörde yaptığı görülmektedir. Aynı şekilde bilgisayar mühendisleri de ilgi duydukları elektrik elektronik disiplini konusunda çeşitli işler yapabilmektedir. Mühendislik disiplini ortak paydasında buluşan insanlar, aldıkları eğitim ile araştırma yaptıkları **diğer konular üzerinde fikir yürütebildikleri için farklı mühendislik kollarında profesyonelce olmasa da işler yaptıklarını görmekteyiz**.

Matematik bilgisayar bölümü mezunu bir kişinin **çok iyi programcı olduğuna**, elektrik elektronik mühendisliği bölümü mezunu birisinin de **çok iyi network uzmanı hâline geldiğine** dair sizlerin çevresinde de kişiler olduğundan eminim. Farklı bölüm mezunlarının yaptıkları işleri örneklendirirken ve meslek kollarından bahsederken özellikle **programcılık ve ağ** (*network*) **yöneticiliği** kavramlarından gidiyorum. Buradan hareketle, aslında ülkemizde ve dünyada bilgisayar mühendisliği bölümü mezunlarının genel olarak **iki iş kolunda** çalıştıklarını söyleyebiliriz. Bunlardan bir tanesi, **yazılım geliştiriciliği** iken diğeri **network ve sistem** kolunda çalışan mühendislerdir. Bilgisayar mühendisleri öğrenim süreçlerinde bir programlama dilini aslında ileride geliştirici olarak kullanmak için öğrenmezler, öğrenim hayatları ve iş hayatları boyunca karşılaşacakları sorunları giderebilmek için **kullanacakları dili öğrenirler**. **Bilgisayar mühendisinin ilgilendiği asıl alan;** *bilgisayarın mimarisi, dilin mimarisi ve bilgisayar elektroniğinin mimarisidir*. Bu yüzden bilgisayar mühendisliği mezunu bir kişiden mezun olduğu zaman çok iyi bir geliştirici olmasını beklemek yanlış olur. Aynı şekilde çok iyi ağ uzmanı veya sistem yöneticisi olmasını da beklemek yanlış olur. Ülkemizde işlemci, ekran kartı veya programlama dili geliştiren firmalar bulunmadığından mühendislerimizin mimari tarafında aldıkları eğitimi, sektörde çok yerinde kullandıklarını söyleyemeyiz.

Henüz öğrenciyken kendisini yazılım, sistem, ağ ve elektronik haberleşme gibi konularından birine yakın hisseden bir öğrenci, **mezun olduğu zaman hayatına seçtiği alana göre yön verebilmektedir**. Bilgisayar mühendisleri, genellikle **yazılım** ve **network** alanlarına işler yapmaktadırlar. Piyasada bulunan yetkin programcı açığı veya benzer şekilde yetkin network-sistem uzmanı ihtiyacı, devlet dairelerinde alım şartlarının mühendislik mezunu olunması kriteri, meslektaşlarımızı bu alanlara yönlendirmiştir. Kurumsal firmalarda, çeşitli devlet dairelerinde *özellikle belediyelerde*, üniversitelerde ve bankalarda çalışan bilişim personellerinin büyük çoğunluğu **mühendislik mezunudur**. Çok iyi programcı olarak mezun olmayan mühendisler dahi, verilen problemi çözebilme ve kod yazmayı sadece bir **araç** olarak gördükleri için **başarılı işler yapabilmektedirler**. Özetle bir bilgisayar mühendisi, kendisine öğrenci olduğu sıralarda belirlediği alan doğrultusunda ilerlediğini varsaydığımızda; yazılım geliştiriciliği, veri tabanı yöneticiliği, sistem yöneticiliği, network uzmanlığı, ar-ge mühendisliği, robotik programcılığı ve eğitmenlik gibi alanlarda çalışmaktadır.

BİLİŞİM TEKNOLOJİLERİNİN TEMELLERİ

2

Bilgisayarın **donanım** ve **işletim sistemi** ile ilgili temel konularının yer aldığı bu bölümde, donanım birimlerine ve işletim sistemlerine ait birtakım özelliklere değineceğiz. Çoğu öğrencinin biliyor kabul edilerek belki hiç görmeden mezun olduğu bu bölüm, **iyi bir altyapı için oldukça önemlidir**.

Bilgisayar tanımını yaparken **veri** kelimesinden bahsetmiştik. Verinin ne olduğunu hepimiz biliyoruz fakat yine de kısa bir tanım ile açıklayacak olursak veri; **bilgiyi oluşturan parçalara denilmektedir**. Bir bilginin bizim için **veri hükmünde olmasının tek şartı**, *anlamlı sonuçlar döndürüp döndüremediğidir*. Eğer tek başına anlam ifade etmiyor ise başka verilerle bir araya geldiğinde anlam ifade edebilir. Örnek olarak bir bilgisayarın; adı, modeli, fiyatı, üretim yılı birer **veridir** ve hepsinin taşıdığı **değer türü faklıdır**.

Verinin tanımını yaparken **bilgi** kelimesinden bahsettik. *Peki bilgi, bir bilgisayar mühendisine neyi ifade etmelidir?* Bu sorumuzun cevabını ise yine çok yalın bir ifade ile **verilerin işlenerek anlamlı hâle geldiklerinde oluşturdukları sonuçlardır**. Veri ile bilginin farkından bahsedecek olursak da yukarıdaki örnekte olduğu gibi **bilgisayarın üretim yılı bir veridir fakat üretim yılı ile ilgili yapılan bir hesaplamanın sonucu bilgiyi oluşturur**.

Bilginin de tanımını yaptığımıza göre artık **bilgilerin büyüklük birimlerine** değinebiliriz. Gündelik hayatta kullanılan büyüklük birimleri aşağıdaki gibidir.

Bit: Bilgisayarın en küçük veri biriminin adı bittir.

Byte: 8 bitten oluşan veri büyüklüğüne byte denir. 8 bit, 1 byte eder.

Kilobyte (KB): 1024 byte 1 KB oluşturur.

Megabyte (MB): 1024 kilo byte 1 MB oluşturur.

Gigabyte (GB): 1024 mega byte 1 GB oluşturur.

Terabyte (TB): 1024 giga byte 1 TB oluşturur.

Petabyte (PB): 1024 tera byte 1 PB oluşturur.

Exabyte (EB): 1024 peta byte 1 EB oluşturur.

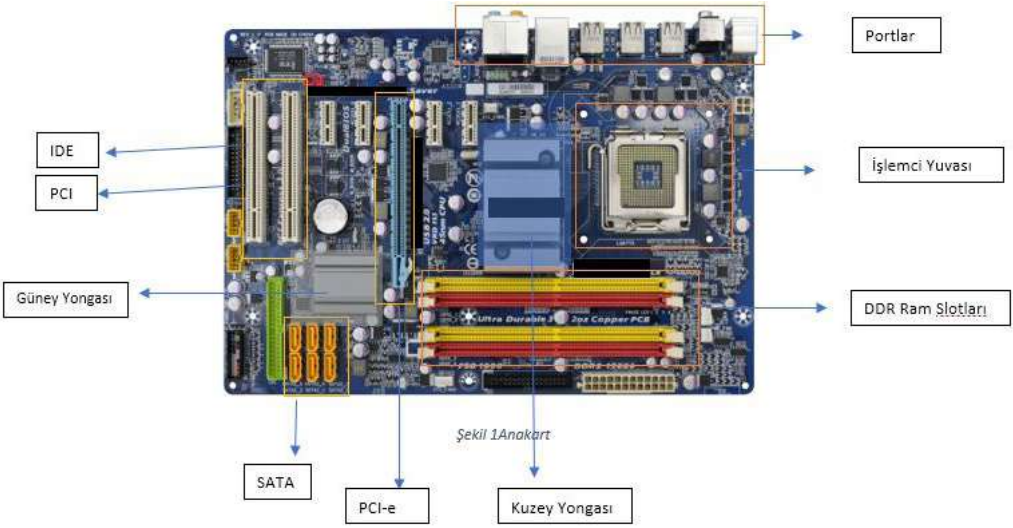
Zettabyte (ZB): 1024 exa byte 1 ZB oluşturur.

Yottabyte (YB): 1024 zetta byte 1 YB oluşturur.

Daha büyük bilgi büyüklük birimleri de vardır. Günümüzde son kullanıcılar **terabyte** büyüklüğünü kullanmaktadırlar. Birimler arası dönüşüm yaparken **küçük olan birim, büyük olan birime dönüştürülmek istenirse** dönüştürülmek istenen değer **1024'e bölünmelidir**. Örneğin 25,384 byte değeri, kendisinden büyük olan **KB değerine dönüştürülmek istenirse 25,384 byte/1024 = 24,789 KB** şeklinde hesaplanır. Eğer büyük olan birim, küçük olan birime dönüştürülecekse de **dönüştürülecek olan değer 1024 ile çarpılmalıdır**. Örneğin 6,84 GB değeri kendisinden küçük olan MB cinsine dönüştürülmek istenirse **6,84 GB * 1024 = 7,004 MB şeklinde hesaplanır**.

Anakart

Bilgisayarın donanım birimlerinden **iç donanım birimlerini bir arada tutan**, birbirleri arasında **iletişimi belirli kriterler ile kontrol ederek sağlayan** elektronik devrelerdir. Anakart, **özel alaşımlı** bir yapıya sahiptir. Devre kartı üzerinde *işlemci soketi, ram yuvaları, çeşitli devreler* ve *iletişimde trafik polisi rolü oynayan yongalar* bulunur. Bir anakart üzerindeki genişleme yuvaları; **ISA**, **PCI**, **AGP**, **PCI** ve **PCI-e** standartları yapısındadır. Ayrıca BIOS, dâhilî donanım devreleri, veri yolları ve bağlantı noktaları da bulunur.



Şekil 2-1: Anakart

Yonga Seti

Şehir trafiğinde araçları ve yayaları kontrollü bir şekilde bir yerden başka bir yere ulaştıran bir dizi trafik kuralları bulunmaktadır. Yaya ve araç trafiğinde uyulması gereken kuralları denetlemek amacıyla da trafik polisleri yer almaktadırlar. Doğanın ve günlük hayatın birebir kopyası şeklinde çalışan teknolojik ürünlerde olduğu gibi anakartlarda da **gerçekleşen iletişimin denetimi sağlayan**, **gerektiği yerde kesmeler yapan** ve **elektronik iletişimin denetimini yapan chipsetlere yonga seti** denilmektedir.

Anakartlarımız üzerinde çok fazla yonga bulunmakta olup **yönetici rolünü üstelenen iki adet yonga seti** bulunmaktadır ve bunların bir tanesinin adı **Kuzey Köprüsü** diğerinin adı ise **Güney Köprüsü**dür. Anakart üzerinde farklı görevleri üstelenen bu iki yonga setinden **North Bridge** yani kuzey köprüsü; işlemci, bellekler, grafik kartı için AGP veya PCI-e veri yollarından sorumludur. Gelişen teknoloji ve optimizasyon algısı sebebiyle kuzey yonga seti çalışma prensibi, **üreticilere göre değişiklik gösterebilir**. Kuzey yongası; işlemci, bellek ve PCI-e gibi slotlardan sorumlu olduğu için **anakart üzerindeki konumu sorumlu olduğu yerlere yakın olmalıdır** bu yüzden de kuzey yongası **anakartın üst kısmında aslında kuzeyinde konumlandırılmaktadır**.

South Bridge, yani **Güney Köprüsü** ise temelde giriş-çıkış birimlerinden, PCI veri yolundan, USB ve diğer anakarta entegre edilmiş olan Ethernet, ses ve güç yönetiminden sorumludur. Tahmin edeceğiniz üzere anakart üzerindeki konumu, **aşağı kısımda olduğundan güney köprüsü** adını almaktadır. Bu iki yonganın tek bir devre üzerinde toplanmayıp ikiye ayrılmasının altında aslında bir **mühendislik çalışması** yatmaktadır. Üreticiler farklı kullanıcılara hitap eden farklı farklı modellerde anakart üretmek yerine üretilmiş mevcut anakart üzerinde **ufak tefek değişikler yaparak** model farklılaştırmaktadırlar. Yeni bir özelliği devrenin tamamına uyarlamak yerine örneğin, sadece güney yongasında çeşitli değişiklikler yaparak ikinci bir model anakart piyasaya sürmüş olurlar. Aynı zamanda arıza durumunda da ilgili yonganın **değişmesi**/**tamir edilmesi** mümkün kılınmaktadır.

Veri Yolları

Yonga seti başlığı altında **kuzey** ve **güney chipsetlerini** incelerken veri iletişimini **kurallı** bir şekilde sağladıklarından bahsetmiştik. *Peki, anakart üzerindeki bu veri iletişimi nereden sağlanmaktadır* diye düşünecek olursak elbette anakart üzerindeki veri yolları aracılığı ile gerçekleşen bir iletişimden, **trafikten bahsediyoruz**. Bir anakart üzerindeki iletişim **veri yolları aracılığı ile** sağlanırken anakarta dışarıdan bağlanan donanımlar da veri yollarının bağlı oldukları **slot** veya **portlar** sayesinde iletişim kurabilmektedirler. Veri yolları da geçmişten bugüne dek üretilen anakartlarda farklı standartlar ile gelmişlerdir. Yonga setlerinin kontrol ettikleri bileşenlerden bahsederken adından söz ettiğimiz **ISA**, **PCI**, **AGP**, **PCI-e** kavramlarının her biri farklı standart ve ihtiyaçlara göre geliştirilmiş olan veri yollarından bazılarıdır.

PCI Veri yolu (Peripheral Component Interconnect)

1993 yılında geliştirilmiş bir veri yoludur ve **64 bit bant genişliğine** sahiptir. Önceleri ayrı ayrı kart şeklinde PCI slotlarına takılabilen TV kartı, Ses kartı, Ekran kartı, Ethernet kartı gibi **tak çalıştır** kartlar PCI slotları üzerinde çalıştığı için PCI slot sayısı anakart üzerinde fazlaca bulunurdu fakat daha sonraları bu kartların çoğunun on board yani ana kart ile bütünleşik olarak gelmesinden dolayı PCI slotlara eskisi gibi ihtiyaç kalmadı. PCI slotları **32 bit** ve **33** veya **66 MHz hız frekansında çalışmaktadırlar**. 32 bit ve 33 MHz hızı ile çalışan PCI veri yollarının kapasitesi ise **130MB/SN** dolaylarındadır.

ISA Veri yolu (Industry Standart Architecture)

Temel kavramların başlangıcında mühendislik disiplinin **mimari** ile ilgilendiğinden bahsetmiştim. ISA veri yolu standardının adından da anlaşılacağı üzere **endüstriyel standart mimarisi** şeklinde bilinmektedir. ISA'lar **çok eski slot türleridir** ve **8-16 bit veri yoluna sahiptirler**. **Tak çalıştır özelliği olmayan** bu veri yoluna ait slotlar genelde **siyah** renktedir ve hızı **8.33 MHz** frekansındadır.

Tak çalıştır özelliği literatürde **Plug and Play** adı ile anılır ve adından da anlaşılacağı üzere tak çalıştır donanımlar bilgisayarın çalışmasını etkilemeksizin ve herhangi bir ayar da yapmaya gerek kalmadan **istenildiği zaman takılıp istenildiği zaman sökülebilen sistemleri ifade eden** bir özelliktir.

AGP Veri yolu (Accelerated Graphics Port)

Yine isminden çıkarım yapabileceğimiz bir veri yolu standardıdır. **AGP açılımına bakıldığında ivmelendirilmiş/hızlandırılmış** grafik portu olduğunu görürüz. AGP'ler **yalnızca ekran kartlarında** kullanılırlar. **533 MHz veri yolu hızına çıkabilir** ve **kanalı 32 bit genişliğindedir**. 2xAGP, 4xAGP ve 8xAGP modları bulunan AGP veri yolu özel bir sinyallaşme yöntemi sayesinde aynı clock hızında **2,4** ve **8 kat** hıza çıkabilir.

PCI-e Veri yolu

PCI Express olarak bilinen bu veri yolu, ekran kartlarında kullanılır AGP veri yolundan üstün olmasının en belirgin sebebi **güç tüketimi ile ilgili yapılan iyileştirmelerdir**. PCI-e veri yolu, belleği daha verimli kullanarak maliyetleri aza indirgemektedir. **PCI-e 1.1** ve **PCI-e 2.0** olmak üzere iki modeli bulunmaktadır.

PCI-X Veri yolu

PCI-express veri yolu ile **karıştırılmaması gereken** bir veri yolu standardıdır. Sonundaki **X** harfi express'i değil **büyük sistemleri ifade etmektedir**. Bu yüzden **sunucu platformlarında** (XEON vb.) **kullanılan bir veri yoludur** diyebiliriz. Sunucu platformlarının ihtiyaç duyduğu yüksek hızları sağlamak için (*gigabit Ethernet gibi*) gereken bant genişliğini sağlamaktadır. Yüksek teknolojili anakartlar üzerinde **yeşil renkte slotlar** ile karşımıza gelmektedirler.

Veri yollarından bahsederken sürekli bant genişliği terimini kullandık. **Bant genişliği**, çoğunuzun bildiği üzere web sitelerinde kullandığımız **bandwidth terimi** ile **aynı karşılığı sağlamaktadır**. Yani birim, zaman içerisinde aktarılabilecek **maksimum veri miktarına** bant genişliği denilmektedir.

İŞLEMCİ (CPU)

İşlemciler, bir dönem tek bir işi gerçekleştirmek için üretilmişlerdi. Daha sonraları daha çoğul işleri gerçekleştirmek için üretilen işlemcilerin maliyeti çok yüksek ve yaptıkları işlemler çok zayıftı. Bugünün işlemci modellerinin temellerinin atıldığı 1970'li yıllarda, **mikroişlemciler** üretilmeye başlandı. 1971 yılında üretilen **ilk intel mikroişlemci modeli intel 4004 olarak bilinir** ve bu işlemciler günümüzdeki gibi soket yuvasına sahip olmayıp ram bellekler gibi slot yuvaya sahip işlemcilerdir. Intelin en geniş çapta kullanılan ve **marka değeri için önem arz eden**, **herkes tarafından bilinen işlemcisi ise 1974 yılında üretilen Intel 8080 versiyonudur**. Intelin bu işlemcisi, günümüzdeki küçük işlemcilerin tasarımı için model olmuştur.

Daha önceki başlıklarda, **bilgisayar sistemlerinin tasarımlarının doğanın birer kopyası mantığı ile işlediğinden** bahsetmiştik. İşlemcileri bu bağlamda insan beyni gibi düşünebiliriz. Bir işlemci içerisinde tıpkı beynimizde olduğu gibi çeşitli işlemleri yapabilmek için çeşitli birimler bulunmaktadır. Bir işlemci, **bilgisayardaki tüm işlemleri gerçekleştiren merkezî bir işlem birimi**dir. Bu yüzden işlemcilere mib, yani **merkezî işlem birimi** de denilmektedir. İngilizce karşılığı **Central Processing Unit** şeklinde yine merkezî işlem birimi anlamına gelmektedir. İşlemcilere **mikroişlemci** de denilmektedir. İşlemci, insan beyninde olduğu gibi farklı birimlerden oluşup her birim kendisine özgü işlemleri gerçekleştirmektedir. Bu bağlamda bir işlemcinin **dört ana birimi** bulunmaktadır. Bu birimler aşağıdaki gibidir:

- Merkezî işlem birimi
- Bellek
- Giriş/Çıkış ünitesi
- Giriş/Çıkış Aygıtları

İşlemciler bilgisayar üzerindeki işlemleri yaparken **makine komutlarını işlerler** ve **sonuçlarını çevre birimlerine ya da belleklere gönderirler**. Gönderilen ve işlenen makine komutları **1** ve **0**'dan oluşan **komut setleri**dir. İşlemci, kullanıcının yaptığı işlemleri komut setine dönüştürerek işler. Gönderilen bu komutlara göre **işlemci şu işlemleri gerçekleştirir**:

- **ALU (Arithmetic Logic Unit)**, Aritmetik mantıksal işlemlerin yapıldığı birimdir. Bir işlemcinin en önemli birimidir. *Toplama, çıkarma, çarpma* ve *bölme* komutlarını gerçekleştirir. Bu işlemleri, işlemci yapısında bulunan milyonlarca elektronik devreden (transistor) geçerek elektrik sinyalleri sayesinde gerçekleştirir.
- **Komut Çözücü (Decoder)**, işlemcinin gerçekleştireceği işlemlerin çalışabilmesi için gereken komut setini çalıştırır.
- **Veri Yolu (Bus)**, işlemcinin işlemleri gerçekleştirme senaryosunda donanım birimleri ile bağlantısını sağlayan elektriksel yollardır.

Komutların çalıştırıldığı bölüme, işlemci içerisinde **Çekirdek (Core)** denilmektedir. İngilizce kaynaklarda **Execution Unit** olarak da bilinir. Çekirdek ünitesi komutları çalıştırır ve boru hattı (pipe line) diye bilinen elektronik yollar ile **okuma**, **çalıştırma** gibi işlemleri yapar. Bir işlemci çekirdeği içerisinde **ALU en başta bulunmakta olup çeşitli sayaçlar** ve **register bulunur**.

İşlemci içerisinde gerçekleşen işlemlerde, bir sonraki aşamada çalıştırılacak olan komutun **hafızadaki adres değeri tutulur** ve o değerler çalıştırılacağında kontrol ünitesi **komutu işleyebilmek için hafızadan alıp komut kaydediciye işlenmesi için aktarır**. Bu işlemi yerine getiren çekirdek birimine **program sayacı** denir. **Program counter** şeklinde bilinir. İşlemcinin dâhilî geçici hafızası olarak bilinen kaydedicilere **register** denir.

İşlemci üzerindeki komutların çalıştırıldıktan sonra **hesaplamaları yeniden yapan** ve **durum kaydedicisinin değerini değiştiren** bir önceki işlemden kalan sonuçları saklayan, gerektiğinde kontrol ünitesine yazmaçtaki değeri kullanması amacıyla gönderen komutların **tamamını çalıştıran yapıya** ise **durum kaydedicisi (Status Register)** denir.



Şekil 2-2: İşlemci Yapısı

İşlemciler çok hızlı çalışan sistemlerdir ve bir bilgisayarda **işlemciye hızı en yakın olan diğer donanım ram bellektir**. İşlemler yapılırken işlemcinin hızına gelen veriler yetişemezler. Bu hız problemini ortadan kaldırmak için işlemciler içerisinde **yüksek hafızalar bulunur** ve **bu hafızalara ön bellek denir**. Ön bellekler çalışan programa ait komutların, geçici olarak saklandığı yüksek seviyeli hafızalardır. Ön bellekler **L1**, **L2** ve **L3** şeklinde üç tanedir.

Ön Bellekler

L1 (**cache**) **ön belleği**, 2 kb ile 256 kb arasında kapasiteye sahiptir. Veriler bellekten L1 ön belleğine **kopyalanır** ve işlemci çok daha hızlı bir şekilde L1 de yer alan verilere ulaşabilir. L1 ön belleği gelen **veriler için Data Cache** ve **kodlar için Code Cache** şeklinde iyi kısımdan oluşmaktadır.

L2 (**cache**) ön belleği, 256 kb ile 2 mb arasında kapasiteye sahiptir. L2 önbellekler önceleri işlemciye yakın yerlerde konumlandırılmış olsa da yeni işlemcilerde **işlemci çekirdeğine entegre edilmiş** bir şekilde konumlanmaktadır.

L3 (**cache**) ön belleği, 2 mb ile 256 mb arasında bir kapasiteye sahiptir. Diğerlerine göre **daha yeni bir teknolojidir**. Çekirdekte konumlandırılan bu önbelleklerde çok çekirdekli işlemcilerin her birine ön bellek koymaktansa **bütün çekirdeklere tek bir L3 belleği ile hizmet verilmesi amacıyla tasarlanmıştır**.

İletişim Yolları

Anakartların anlatıldığı başlık altında, anakart üzerindeki iletişimi sağlayan bir **veri yolundan** bahsetmiştik. İşlemcilerde de hem işlemcinin kendi içerisindeki iletişimi için hem de işlemcinin diğer birimler ile iletişimi için kullanılan **elektriksel iletişim yolları** bulunmaktadır. İşlemci modeline ve üreticiye göre işlemci üzerindeki iletişim hatlarının sayısı değişebilir.

Veri yolu bilgisayar iletişiminde **veri yolu ne kadar büyükse o kadar büyük bir iletişim sağlanmış olur**. İşlemci, çevresel birimlerle birbirine **paralel iletken** hatlar üzerinden iletişim kurmaktadır. Bu iletken **hatların sayısı 32 ise veri yolu 32 bitlik** demektir. Veri yolu üzerindeki iletişim **çift yönlü** bir iletişimdir.

Kontrol yolu okuma yazma sinyalleri, **irq kesme sinyalleri** ve **clock sinyallerinin iletildiği iletişim yoludur**. Bu iletişim yolunda, işlemci diğer birimleri yönetmek için **sinyal göndermektedir**.

Adres yolu, işlemci birimlere ulaşmak amacıyla **adresler** kullanır. Bilgi yazılan ve okunulan her hafıza hücresinin veya çevre birimlerinin **birer adresi vardır**. Bu adresler **ikili sayı sistemi ile oluşturulan grup sayılardır**, işlemcinin ulaşabildiği maksimum adres sayısı ise adres yolunda bulunan hat ile ilişkilidir.

Adres yolunu çoğunlukla kullanan işlemci için **adres yolu trafiğinin tek yönlü bir iletişimde olduğu söylenebilir**.

Yol arayüzü **bus interface** olarak bilinen bu yapıda, işlemciye **veri** ve **kod karışımı bilgiler gelmektedir**. Bus interface bu verileri **ayırır** ve işlemcideki ünitelerin **bu anlamlı verileri kullanabilmesine olanak sağlar**. İşlemcilerin adresleyebileceği hafızayı bu arayüzün genişliği belirlemektedir.



Şekil 2-3: İşlemci İletişim Yolları

İşlemci Hızı

İşlemcilerin gerçekleştirdikleri işlemlerin hızı, belirli bir hız birimi ile ifade edilir ve bu birim **saat hızı** olarak bilinen, **aslında saniyede ne kadar çevrim yapabileceğini gösteren bir birimdir**. İşlemci bir işlemi gerçekleştirirken **saat vuruşu** ile çalışmaktadır. Anakart üzerinde bu saat vuruşlarını üreten bir **yonga** bulunmaktadır. Esasında bu yonganın içerisinde bulunan elemanlar, yüksek hassasiyete sahip **kristal titreşimlerden başka bir şey değildir**. İşte, bu saat kristallerinin oluşturduğu vuruşların hızına ise birim olarak **Hertz** ve bu hızın ölçüldüğü veri yoluna **FSB** denilmektedir. FSB ingilizce açılımı **Front Side Bus** olarak geçmektedir. Hatırlarsanız anakart üzerindeki kuzey ve güney köprülerinin iletişimi sağladığından bahsetmiştik, işte **FSB hızı işlemci ile kuzey köprüsünün arasındaki veri yolu yoludur**. Hertz, saniyedeki tek vuruşun ölçüsüdür.

1 Hertz (Hz) saniyede **1** çevrime eşittir.

1 Megahertz (MHz) saniyede **1.000.000** çevrime eşittir.

1 Gigahertz (GHz) ise saniyede **1.000.000.000** çevrime eşittir.

Peki, işlemcinin hızı fabrika verileri ile belirlenmiş ve sayısal değerler ile belirtilmiş ise bu hızı değiştirmek, arttırmak mümkün olabilir mi? Hemen söyleyelim, elbette çeşitli yöntemler ile işlemci hızını arttırmak yani üreticinin belirlediği hız değerlerinden daha yüksek seviyelere çıkarmak mümkündür. **Anakartlar üzerinde yapılacak değişiklikler ile işlemci hızlarını arttırmak mümkündür**. Sistem hızını, yani **FSB ile çarpan değerlerinde yapılacak değişiklikler ile işlemci hızı arttırılabilir**. Bu işleme **hız aşımı** denilmektedir. Hız aşımı kaynaklarda ve sektörde **Overclock** olarak bilinir. Adından da çıkarım yapabileceğimiz gibi işlemci içerisindeki saat vuruşu yani **çarpan değerlerinin aşılması demektir**. Donanımın ömrünü azalttığı için *çok tavsiye edilen bir işlem değildir*. **Çünkü işlemci hızı yalnızca, işlemcinin kendisinin hızlı olması anlamına gelmemelidi**r. Bilgisayardaki diğer donanım birimlerinin de işlemci ile aynı ritimde çalışması gerekeceğinden **tüm donanımı yoracaktır**. Bunu, otomobillerdeki chip tuning uygulamasının turbonun ve motorun ömrünü erken tükettiğine benzetebiliriz.

İşlemciler bu kadar hızlı çalışıyorken diğer donanım birimlerinin de işlemci kadar hızlı olmadığından bahsetmiştik. Bir işlemciyi, **kontrollü bir şekilde kullanabilmek için ona hükmetmek gerekir**. İşlemci içerisinde yer alan programlar vasıtasıyla neyi nasıl yapması gerektiğini söylemek gerekmektedir. Örneğin sabit diskler işlemcilere göre **aşırı yavaş aygıtlardır.** Bir veri sabit diskten alınıp doğrudan işlemciye getirilemez, aralarında muazzam bir frekans ve zaman damgası farkı vardır. Eski bilgilerinizi biraz canlandırdığınızda sabit diskteki verilerin öncelikle geçici belleğe, yani **RAM üzerine** yüklenerek oradan da **işlemciye ön bellekler vasıtasıyla ulaştırıldığını** hatırlarsınız. Aynı şekilde işlemcinin geri döndürdüğü işlem sonuçları da öncelikle RAM daha sonra da sabit diske yüklenir. Bu yüzden işlemci ile sabit disk arasındaki iletişime **çift yönlü** bir iletişim diyebiliriz.



Şekil 2-4: Soket İşlemci



Şekil 2-5: Slot İşlemci

İşlemci Mimarileri

Cisc Mimarisi

Complex instruction set computer kelimelerinin baş harflerinin bir araya gelmesi ile **CISC** şeklinde ifade edilen mimarinin Türkçe karşılığı: **Karmaşık komut setli bilgisayar** anlamına gelmektedir. CISC, bir bilgisayar işlemcisinin komut yapısını ifade etmektedir. Adından da anlaşılacağı üzere komut yapısı karmaşık olan bir işlemciden bahsedilmektedir. Mühendisliğin temel çalışma alanlarından olan **maliyet** ve **optimizasyon konusunda çok başarılı bir mimari diyemeyiz** çünkü karmaşık yapısı yüzünden her bir komutun **işlemci tarafından çözümlenmesi** (*decode*) **uzun sürmekte** ve bu karmaşıklık sebebiyle **devre kartının şekli de karmaşıktır**. CISC işlemci mimarisinin bu fazla komut ve maliyetli yapısına karşın **RISC** işlemci mimarisi geliştirilmiştir ve RISC mimarisine sahip işlemciler komut sayısını azaltarak daha yalın bir hâle getirilmesini ve performans elde edilmesini sağlamıştır. CISC işlemci mimarisi, bu yönden dezavantajlı gibi görülse de **tek bir komut ile yaptığı işlem**, **RISC işlem mimarisi ile birden fazla komut ile yapılmaktadır**. Mühendislik disiplininin diğer çalışma alanlarından biri olan optimizasyon ile RISC mimarili işlemciler, bir işi daha fazla komutla yerine getiriyor olsalar da daha yüksek clock frekansı ile yaptığı işlemler **daha güçlü** olmaktadır.

Risc Mimarisi

Reduced instruction set computing kelimelerinin baş harflerinin bir araya gelmesi ile **RISC** şeklinde ifade edilen mimarinin Türkçe karşılığı; **İndirgenmiş komut setli bilgisayar** anlamına gelmektedir. Adından da anlaşıldığı gibi CISC mimarisinde olup biten birtakım işlemler indirgenmiş yorumunu yapabiliriz. Bu mimari, aslında bilgisayar işlemci mimarisinde **önemli dönüm noktalarından birisidir**. RISC mimarisinde, **daha az sayıda komut bulunur**. Daha az sayıda adresleme ve sabit uzunluklarda komut yapısına sahiptirler. RISC mimariler, işlemleri gerçekleştirirlerken her seferinde belleğe erişmezler **yalnızca okuma** ve **yazma işlemlerinde erişirler** bu, işlemlerin direkt bellek üzerinde yapılmayıp **saklayıcılar üzerinde yapılması anlamına gelmektedir**.

İşlemci Teknolojileri

Centrino İşlemciler

Centrino'lar, **dizüstü bilgisayar için** üretilmişlerdir. Amacı tahmin edeceğimiz üzere daha **az enerji harcayıp aynı zamanda sistemin daha az ısınmasını sağlamaktır**. Negatif korelasyon ile geliştirilen bu işlemcilerde boyut, enerji ve ısınma gibi nitelikler küçülürken aynı zamanda performans, wifi kullanımı gibi yapılabilecek işlemlerin de **arttırılması** sağlanmıştır.

Çift Çekirdekli İşlemciler (Dual Core)

İki adet çekirdeğin **tek bir işlemcide aynı frekansla çalıştığı** işlemci teknolojisidir. Çekirdeklerin her ikisi de aynı belleği, aynı paketi ve dolayısıyla aynı devre elemanını kullanırlar. Bu yapısı ile kullanıcının **aynı anda farklı uygulamaları çalıştırabilmesine** olanak sağlamaktadır. Üreticisine göre farklılık gösteren çift çekirdekli işlemcilerin çekirdeklerinde **Hyper-Threading** özelliği bulunabilir. Bu özellik (HT), iki çekirdeğin ayrı ayrı iki tane **virtual (sanal)** simetrik işlemci gibi davranan **dört işlemci gibi** çalışmasına olanak verir. Bu teknoloji **sanal çift çekirdek** diye bilinmektedir.

Hyper-Threading

Bu teknoloji çift çekirdek konu başlığında da bahsedildiği üzere, tek bir işlemcinin çok sayıda işi **eş zamanlı** olarak yapabilmesi için sanal işlemci yapısını kullanması şeklinde bilinir. HT teknolojisine sahip bir işlemci, sanal olarak **iki adet işlemci** bulundurur. Her bir işlemci, aslında fiziksel olarak **aynı devre (circuit) üzerinde** konumlandırılmış olmasına rağmen sanallaştırmadan dolayı **farklı komutları işleyebilmektedirler**. Donanım ile gerçekleştirilen fazladan iki işlemci olmayıp sanal olduğu için gerçek işlemcili bir bilgisayar performansı beklenemez fakat **güzel bir optimizasyon örneği olmaktadır**.

Ön Bellek Teknolojisi

İşlemciler konu başlığında ön belleklerden bahsetmiştik. Ön bellek teknolojisi aslında yeni bir teknoloji değildir. Klasik işlemcilerin her bir çekirdeğinin içerisinde yer alan L1 ve L2 önbelleklerinin içerisinde 1 MB kadar ön bellek bulunur bu ön bellekler için yeni teknoloji diyemeyiz fakat ön bellek teknolojisini farklı bir başlık altında işlemci teknolojisi olarak ele almamızın sebebi ek olarak gelen L3 ön belleği ve 8 MB'lık alanıdır. Bu farkı ile L3'ler smart ön bellek olarak da bilinirler.

Turbo-Boost Teknolojisi

Yine otomobil dünyasından bir örnekle dilim döndüğünce tanımlamaya çalışacağım bu işlemci teknolojisini, yeni nesil üretilen **1.0 motorlu turbolu** otomobiller gibi düşünebiliriz. Aslında motor hacmi epeyce küçük olan bu otomobiller, turbo sayesinde daha büyük motor hacmine sahiplermiş gibi etkileyici bir performans gösterebilirler. Turbo-Boost teknolojili işlemciler de kendilerince birtakım kriterleri ele alarak **performans artışı** gösterirler. İşlemciler **over-clock** işlemini **ihtiyaca göre otomatik olarak gerçekleştirip performansı arttırıp azaltıyor** diyebiliriz. Tıpkı küçük motorlu turbolu otomobillerde olduğu gibi işlemci de performansını arttırmak için çeşitli kriterleri ele alacaktır. Bu kriterler; *işlemcinin çekirdek sayısı, anlık güç tüketimi, sağlıklı akım miktarı* ve *ve ısı problemleridir*. Bu kriterlere göre gerektiğinde **işlemci performansını arttırır** ve **turbo teknolojisi sayesinde hissedilir performans artışı sağlanır**.

İşlemci Soğutma Yöntemleri

Bu kadar otomobil örnekleri verdikten sonra işlemciler ile son bölüm olan soğutma bölümünü de otomobiller ile ilgili bir örnekle başlatmak yerinde olacaktır. Isınan otomobil motorunun ve diğer aksamlarının **otomatik olarak soğutulması ihtiyacını işlemcilerimiz** ve **diğer donanımlar da duymaktadırlar**. İşlemcilerin soğutulması ile ilgili olarak çeşitli yöntemler bulunur. Bunlardan en yaygın olanı çoğumuzun ev bilgisayarlarında bulunan hava yani **fanlı soğutma** yöntemidir. Elektronik aygıtların ısınma problemi, içerisinde gerçekleşen **saniyelik işlemlerden** kaynaklanır ve çalışan bir elektronik aygıtın ısınması kadar normal bir olay yoktur. Doğru soğutma sistemi ile çalışan bir bilgisayar için ısınmadan endişe duyulmasına gerek yoktur. Bilgisayar soğutma işlemi denilince herkesin aklına *fanlar, soğutucular, termal macunlar* ve *sıvı soğutmalı sistemler* gelir. Şimdi, soğutma ile ilgili bilmemiz gereken **temel konuları beraber inceleyelim**.

Havayla Soğutma Yöntemi

Yapılabilecek **en basit** ve **etkin** çözümdür. Normal bir bilgisayar kasasını açtığınız zaman işlemci üzerinde sürekli **dönen bir fan** görürüz. Bu fanın altında ise bir soğutucu metal, metalin altında termal macun ve onun altında ise işlemci bulunur. **Düzenli aralıklar ile termal macunu yenilemek faydalı fakat dikkatli yapılması gereken bir işlemdir**. Bu işlemi de araçların filtre bakımı gibi düşünebiliriz.

Havayla soğutma işlemi basittir, işlemci ısınır termal macun ısıyı soğutucuya iletir, soğutucu bu ısıyı emer ve fan vasıtasıyla hava sirkülasyonu sağlanmış olur. **Ters çalışan bir fan**, ısınan havayı soğutucu üzerinden alırken aynı zamanda da **soğutucuya soğuk hava üflemiş olur**. Günümüz bilgisayar sistemlerinde bilgisayarların eskisinden daha önemli ve çok çalışması gerektiği **render alan** veya **deep learning** algoritmalarının çalıştırıldığı gibi uygulamalarda fan ile soğutmadan **daha efektif soğutucu görevi gören yöntemler geliştirilmiştir**.

Su İle Soğutma Yöntemi

İşlemcileri anlatırken otomobillerden fazla örnek verdiğimin farkındayım fakat bu konular çok temel konular olmasına rağmen *unutulmaya çok müsait konulardır*. Bu yüzden akılda kalıcı olması açısından otomobil **radyatörlerinin içerisinde dolaşan suyun motoru soğutması yöntemi ile birebir aynı olan** bir yöntemden bahsetmek istiyorum. Genellikle bilgisayar kasasının dışında kasa boyutlarından daha küçük bir radyatör ile kasanın içerisindeki işlemci, ekran kartı gibi ısınan donanımların üzerinden **borular ile soğuk su dolaştırılmasını sağlayan sistemler**dir. Kulağa hoş gelen bu sistem fan ve fan motoru olmadığından **daha az gürültü** yapacaktır. Tabii ki her güzel şey gibi fanlı sistemlerden **daha pahalı bir kasa soğutma** sistemleridir.

Isı Borulu Soğutma Sistemi

Suyla soğutma sistemine çok benzer. İşlemcilerin üzerinde **ısı boruları bulunur** ve bu boruların içi **ısı ile yoğunlaşabilen özel bir sıvı** ile doludur. İşlemci ısı oluşturdukça boruların içerisindeki sıvı buharlaşır, yoğunlaşır ve yukarı doğru hareket gerçekleştiren **sıvı soğuyarak yaptığı soğuma hareketi sayesinde işlemcilerden ısıyı uzaklaştırmış olur**.

Bellekler

Bilgi kelimesinin geçtiği hemen her yerde, bu bilginin saklanması gerektiğini biliriz. Bilgisayarlar üzerinde **bir bilginin saklanıldığı donanım birimlerine bellek** denilmektedir. Bellek teriminin yanı sıra *hafıza, disk, veri kütüğü, flash* gibi terimler de kullanılmaktadır. Günlük hayatta hangi terim ile hangi bellek türünü kastettiğimizi ayırt edebiliyorsak hangisini kullandığımız çok önemli değildir. Bilgisayarlarımız bellekler üzerinde tuttukları her türlü video, text, image gibi bilgiyi kendi anlayacağı dilden yani **makine dili türünden tutarlar**. 1 ve 0 şeklinde tutulan bu bilgiler belleklerde **elektriksel** ve **manyetik** olarak saklanabilmektedir.

RAM (Random Access Memory)

Rastgele erişimli bellek anlamına gelen RAM bellekler, bilgisayarlarımızın **ana bellekleri**dir. Ana bellek denilince akla hafıza büyüklüğünün fazla olması sebebiyle sabit diskler gelebilir fakat bir bilgisayarın ana belleği **RAM bellekler**dir. RAM bellek, **geçici hafıza depolama birimi**dir. Bilgisayar çalıştığı sürece bilgiler RAM üzerindeki transistorlerde **doldur boşalt** şeklinde sürekli bir enerji trafiğine maruzdur ve bilgisayar kapandığı zaman yani elektrik kesildiği zaman güç olmayacağından RAM bellekte üzerindeki güç trafiğini boşaltır. Bu yüzden RAM belleğe **geçici bellek** denilmektedir ve bilgisayar üzerinde önemli işler yapılırken anlık elektrik kesintisi gibi kötü senaryolar için sürekli **yapılan işin sabit diske kaydedilmesi önerilir**. Günümüz yazılımlarında bu iş için otomatik olarak belirli periyodlarda kullanıcıdan **bağımsız olarak kaydet** işlemi gerçekleştirilmektedir. RAM, bilgileri geçici olarak tuttuğu için üzerinde ne kadar çok bilgi var ise o kadar fazla miktarda bilgiyi işlemciye yetiştirmesi gerekecektir. Bu durum, işlemcinin hızlı olması gerektiği kadar bir bilgisayarda **RAM kapasitesinin de yüksek olması gerektiği** sonucunu doğurmaktadır. Eğer RAM performansı düşük olursa bu durum bilgisayarın da performansının düşmesine yol açar ve yapılan işlem türüne göre program çalıştıramama veya yavaş çalıştırma gibi sonuçlar doğurur.

RAM'lerin elektronik yapısı **matris** gibi dizayn edilmiştir. Her adresin bir referans numarası bulunur ve sabit diskten okunan veriler **bu referans değerlerine yerleştirilir**. İşlemci ise bu değerleri ram üzerinde okuyarak işlemleri gerçekleştirir. İşlemci konusunda işlemcilerin çok hızlı donanımlar olduğundan ve kendisine en yakın hızda çalışan **ikinci donanım biriminin ram** olduğundan bahsetmiştik. Bu yüzden kullanılmadığı esnada sabit disk üzerinde veya hâricî disk, DVD gibi ortamlarda bekleyen veriler **yalnızca kullanılacakları zaman ram üzerine yüklenirler**.

Çağrılan program veya veri ram kapasitesinden büyük olduğu durumlarda ise işlemi gerçekleştirebilmek için RAM bellek, **sabit diskin önceden tanımlanmış bir bölümünü takas eder** ve **sabit diskin bu alanı ram gibi davranması amacıyla rezerve edilir**. Bu işleme **swap** (*takas*) bellek yöntemi denir ve işletim sistemi üzerinde bilgisayarım kısmından sabit disk özelliklerinden swap alanının büyüklüğünü ayarlayabilirsiniz. Yapılan bu işleme yani sabit diskin bir bölümünü RAM gibi davranma eğilimine **çarşaflama** veya **PageFile** da denilmektedir.

NOT

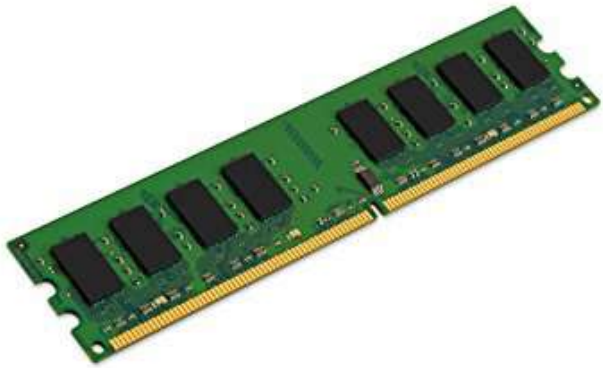
Sanal makine yazılımları ile uğraştıysanız sanal makine üzerine bir işletim sistemi kurarken sabit diske ne kadar ayırayım, RAM olarak kaç gb ayırayım ve sanal bellek için yani swap alanı için ne kadar yer istiyorsunuz şeklinde ayarlar yapılan **diyalog kutularını** hatırlayacaksınızdır.

Tek Sıralı Hafıza Modülü olarak bilinen kısaca **SIMM** şeklinde ifade edilen ve İngilizce karşılığı **Single inline memory modüle** olan bu çözüm, sistemlerdeki RAM artış ihtiyacına karşın geliştirilen **RAM yongaları çözümü**dür. Bilmemiz gereken bir diğer RAM türü ise **Statik RAM**'lerdir. Statik RAM'ler büyük bir mimari kullanmaktadırlar bu sebeple **maliyetlidirler** fakat bunun yanı sıra **yüksek hızlara destek verirler**. Statik RAM'lerin yüksek hızlı oluşu, maliyetli oluşu ve mimari yapısının gelişmiş oluşu, bize işlemcileri anlatırken bahsettiğiniz **önbellekleri** (**cache**) hatırlatabilir. Statik RAM'ler genellikle **küçük boyutlu olarak önbelleklemede** kullanılırlar. İşlemci devresinde **entegreli** oldukları için bir **yenisiyle değiştirilmesi gibi bir durum söz konusu olmayacaktır.**

Önbellek dışında son kullanıcı olarak rahatlıkla takıp çıkartabildiğimiz ve **sistem belleği** olarak adlandırdığımız bellek türüne ise **DRAM** denilmektedir. DRAM adından da çıkarım yapılabileceği üzere **Dinamik RAM** kelimelerinden oluşmaktadır. RAM geçici depolama birimi yani **Random Access Memory** olarak ifade edildiğinden DRAM ise **Dynamic Random Access Memory** olarak ifade edilir. DRAM'ler SRAM'lere göre **daha düşük maliyetlidirler**. Hız olarak sistem belleği görevini üstlenebilecek kadar hızlı fakat **önbelleklere göre daha yavaş ve makul bir bellek türüdür**.

DIMM RAM olarak bilinen başka bir Ram bileşeni daha vardır. Bu paket, **Çift Yönlü Hafıza Modülü** olarak bilinir ve İngilizce karşılığı **Dual inline memory modüle** olarak bilinir. Buffering ve ECC fonksiyonları için **ek pinleri** bulunur. Hepimizin bildiği ve bilgisayarlarında kullandığı **double data rate** teknolojisi ise iki kat veri transferi sağlar. **DDR** teknolojisi **iki kat veri transferi sağladığından oldukça popülerdi**r. *DDR2, DDR3 gibi çeşitleri bulunur*.

RAM'ler ile ilgili anlatılacak çok fazla husus bulunmaktadır fakat kitabımızı yazma amacım zaten internetten araştırılınca bulunabilecek veya hocaların ders notlarında bulunan hatta ve hatta diğer bilgisayar kitaplarından da temin edilecek detaylı bilgileri tekrar edip hem **okuyucuya temiz bilgi sunmak** hem de bazı konularda detay verip bazı konularda temel bilgileri aktararak **başucu niteliğinde bir eser hâline getirme** düşüncesini benimsemiş olmamdır.



Şekil 2-6: Random Access Memory

ROM

ROM belleğin RAM bellekten farkı, çoğu kaynakta tanım olarak **genelde sadece okunabilir belleklere ROM denildiği** şeklindedir. Bu doğru tanımı biraz açmak ve örneklendirmek gerekirse **Read Only Memory** yani sadece okunabilir bellekler otomobillerdeki Ecu yani *aracın beyni* olarak düşünülebilir. Fabrika tarafından bir kez kodlanan ve daha sonra **sadece araç tarafından okunan belleklerdir**. Bilgisayar sistemlerinde de anakart üzerinde üretici fabrika tarafından bir kez kodlanan ve sistem tarafından her post işleminde veya bios ile ilgili işlemlerde okunan bellek türüdür. Sadece okunabilir diye geçmesine rağmen bilgisayar sistemlerinde de otomobil sistemlerinde de **ROM bellekleri çeşitli yöntemlerle programlamak mümkündür**.

Bu yöntemler elektriksel yöntemlerdir. Rom bellekler **dört grupta** incelenirler.

PROM: **Programlanabilir ROM**: programlayıcı devresi ile yalnızca bir defa programlanabilen ROM türüdür.

EPROM: **Silinebilir ve Programlanabilir ROM**: kızılötesi ışık ile içerisindeki verilerin silinebildiği ve sonra yeniden programlanabilen bir ROM türüdür. EPROM yongası üzerindeki quartz pencereden gönderilen kızılötesi ışıklar ile işlem gerçekleştirilir.

EEPROM: **Elektrikle Silinebilir ve Programlanabilir ROM**: EPROM gibidir fakat silinme işlemi kızılötesi ışıklarıyla değil elektriksel olarak yapılmaktadır.

MPROM: **Maske Programlı ROM**: veriyi maskelemek amacıyla üretici tarafından programlanan ROM çeşididir.



Şekil 2-7: Rom

Kasa ve Güç Kaynağı

Kasa Çeşitleri

Bilgisayar kasaları başlığını ele almamdaki en önemli sebep, güncel bir eser yazmaya çalışırken geçmişte daha popüler olan konu başlıklarından çok da uzaklaşmak istememdir. Kitabın okuyucu hedef kitlesi mühendislik öğrencisi olsun olmasın, mühendislik disiplini içerisinde alacağı dersler kapsamında hemen hemen **hiçbir ders içeriğinde olmadığını tespit ettiğim temel konulara ilk bölümde yer vermeyi hedeflediğim için** donanım ile ilgili olan bu son bölümde, **kasalara** da yer vermek istedim. Bu bölümden sonra işletim sistemlerine değinecek olup daha sonrasında ise mühendislik derslerinde yer verilen konulardan işinize yarayacak kısımları derlemeye çalışacağım.

Bir bilgisayar kasası, bu bölüme kadar anlatmaya çalıştığım **iç donanım birimlerinin bir arada bulunduğu fiziksel bir koruyucu parçadır**. Organizer gibi de düşünebileceğimiz kasalar, çeşitli malzemelerden üretiliyor olsa da **son kullanıcı seviyesindeki kasalar alüminyum malzemeden yalıtılarak üretilmiş kasalardır**. Kasalarında multimedya gibi özellikleri veya çeşitli fonksiyonları bulunabilir. Bir işlemcinin, soğutucunun, belleğin ve ekran kartının bağlı olduğu anakart, **doğrudan kendisine ait vida yerlerine vidalanarak kasa üzerinde yerini alır**. Bir masaüstü bilgisayar toplarken *işlemci, soğutucu, ekran kartı ve RAM* **anakart üzerine montajlanır**. Sabit disk ise anakart üzerinde kendisine ait olan alana bağlanır. Birbirleri ile uyumlu olarak satın alındığını varsaydığımız *iç donanım bileşenleri* **anakart ile birlikte kasaya montajlanır**. Kasanın güç butonları ön panel bağlantıları ile ilgili bağlantılar da yapıldıktan sonra kasa kullanıma hazırdır diyebiliriz. Bir kasanın içerisinde DVD, CD-ROM sürücülerinin bağlanacağı panel yuvaları, güç kaynağının konumlandırılacağı güç kaynağı paneli, anakartı sabit konumlandırmak için özelleştirilmiş deliklere sahip anakart montaj havuzu, ön tarafta usb, kulaklık veya mikrofon gibi giriş portları takabilmek için kasanın dizaynına göre şekillendirilmiş ön panel aralıkları ile arka tarafında soket boşlukları yer alır.

Bilgisayar kasaları boyutlarına ve anakart form boyutlarına göre **çeşitlere** sahiptir.

Desktop Kasalar: Tüm son kullanıcıların bildiği masaüstü kasalardır. Bu kasalar eskiden devlet dairelerinde **çok sık gördüğümüz** üzerlerinde monitör taşıyan **yatay kasalardır**.

Tower Kasalar: Kule kasalar büyük daha büyük kasa çeşitleridir ve farklı büyüklük formlarında üretilmişlerdir. En büyük kule kasa **Full Tower** olarak adlandırılır ve büyüklüğü 60-90 cm yüksekliğindedir. Büyük oluşundan ötürü çok fazla sürücü veya aygıt takılabilir. Büyük bir kasayı ve fazla donanım barındırabilen bir kasayı tahmin edeceğimiz üzere **büyük bir güç kaynağı** beslemektedir. Güç kaynağının büyük olması çok fazla soğutma elemanına da destek vermektedir.

Full Tower kadar büyük olmasa da orta boyutlara sahip diğer kule kasa çeşidi **Mid-tower** kasalardır. Çoğu yönden full tower kasalara benzeyen mid-tower kasalar, boyut olarak biraz daha kısa kasalardır.

Mini Tower kasalar ise kule kasalar türündeki en küçük kasa türüdür. **50-70 cm yüksekliğine sahip olabilir**. Çoğu son kullanıcı ve ev kullanıcısı bilgisayar kasaları mini tower olabilmektedir. Masaüstünde kullanımda dahi çok fazla yer kaplamamaktadır.

Slim Kasalar: İnce kasalardır. Boyutları mini tower büyüklüğünde olup hem **yatay** hem de **dikey** olarak kullanılabilir.

Güç Kaynağı

Güç kaynağı kasanın içerisinde yer alan, **power supply** olarak da bilinen bilgisayarın **ihtiyaç duyduğu tüm enerjiyi sağlayan cihazdır**. Güç kaynağı üzerinde anakart için ayrı fanlar için ve sürücüler için ayrı kablolar ve konnektörler bulunur. Güç kaynağı kasanın da büyüklüğüne göre faklı watt gücünde olmaktadır. Son kullanıcı bilgisayarlarının enerjisini karşılayacak olan bir güç kaynağı **400-600 watt** güç üretmektedir. Bilgisayar içerisinde yer alan birimler doğrudan şehir şebekesinden gelen elektrik ile çalışamazlar. Küçük voltaj değerlerinde çalışan donanım birimlerinin ihtiyaç duyduğu 5-12 Volt gibi değerleri sağlayan güç kaynağı, şehir şebekesinden aldığı **220 Volt alternatif akımını doğrusal akıma çevirir**. Şehir elektriğinden gelen akımda dalgalanma olmaması açısından alternatif akımı (**AC**), doğrusal akıma (**DC**) çevirir.

Sağlıklı bir bilgisayar sisteminin yapı taşlarından bir tanesi de **güç kaynağı**dır. Güç kaynağı ne kadar sağlamsa yani malzeme kalitesi ne kadar iyiyse ve gücü ne kadar fazlaysa sistemi o kadar iyi besleyecek demektir. Bu yüzden **güç kaynağı seçimi, bilgisayar toplama sürecinde oldukça önemlidir**.

Depolama Birimleri

Sabit Diskler

Bilgisayar donanım parçalarından bir diğeri ise **sabit diskler**dir. Sabit diskler; cep telefonlarımızdan, tablet bilgisayarlarımıza veya bir otopark bariyer sisteminden güvenlik kamerası kayıt cihazlarına kadar **verinin tutulduğu her yerde** bulunur. Hard disk olarak bilinen bu disklerin çeşitleri bulunur ve teknolojisine göre farklı çalışma prensiplerine sahiptirler.

Bir hard disk, bilgisayarlarımızdaki kalıcı verilerin tutulduğu **saklama depolama** alanıdır. Kullanıcı, yedekleme senaryolarının bulunmadığı sistemlerde bilgisayarından ziyade hard diske gözü gibi bakmalıdır. Hard diskler **fiziksel aygıtlar** oldukları için ve teknolojisine göre farklılık göstermesine karşın içerisinde plakalar ve okuyucu kafası gibi fiziksel elektronik parçalar barındırdığından boyutları **diğer iç donanım birimlerine oranla büyük** ve **ağır** olan cihazlardır. Bilgisayar çalıştığı esnada RAM içerisindeki transistörlerin, yani elektronik yongalarının içerisi tamamen **elektrik** ile doldurulur ve boşaltılır. Bu şekilde bilgisayardaki işlemler yapılmış olur fakat elektrik enerjisi kesildiğinde RAM üzerinde yapılan bir işlem varsa da **elektrik boşalacağı için işlem kaybolacaktır**. Bu yüzden işlemler gerçekleştirilirken **sürekli kaydedilmesi** önerilir. Oyunlarda veya paket ofis programlarında belirli aralıklarla otomatik kayıt işlemi de bu yüzden gerçekleştirilmektedir. Buradaki kayıt işlemi sabit diske, yani **hard diske** yapılır ve bu bilgiler kullanıcı silene kadar kalıcı bilgilerdir.



Şekil 2-8: Sabit Disk

Şekil 2-8'de bir hard diskin **kesit görüntüsü** yer almaktadır. Manyetik olan diske ait bu görüntüdeki parlak levhalar **manyetik plakalar**dır. Hemen üzerindeki kafa ise **okuma ve yazma kafası**dır. Ön tarafında ise **bağlantı girişleri** bulunmaktadır. Bir hard diskin dış kapağını açtığınızda şekildeki gibi bir görüntüsü olacaktır. Bu görsel, **manyetik SATA türündeki bir sabit diske aittir**. Günümüzde artık son kullanıcı seviyelerinde dahi **SSD (Solid State Drive)'**ler kullanılmaya başlanmıştır.

Manyetik sabit disklerin içerisindeki plakaların, **dakikadaki dönüş hızı Rpm ile ifade edilir** ve **dakikada 5000-7000-15000** gibi devirlere sahip diskler vardır. Manyetik disklerde okuma yazma kafa plakaları manyetik ile işlerler. Okuma ve yazma işlemi **manyetizma** ile gerçekleştirilmiş olur. Plaka üzerindeki iz ve sektörler okuma kafasının çalışma prensibi ile **aynı anda aynı konumda hareket edecek şekilde** okuma yazma işlemi gerçekleşir.

Sabit disk üzerindeki veriyi depolama senaryosunda **kümeler** kullanılır. **Kümeleme** (*cluster*), disk yönetimine bağlı olarak değişen bir yer alanıdır. İşletim sistemi tarafından kullanılan **dosya sistemi ne ise küme alanı da ona göre bir boyuttadır**.

HDD, yani hard diskler manyetik plakalardan dolayı **hassas cihazlar**dır. Taşıması önemlidir. Manyetik plakalar taşınma veya yer değiştirme esnasında **sert darbelere maruz kalmamalı**dır. HDD içerisinde barındırdığı okuma kafası, manyetik plakalar ve motor gibi parçalardan dolayı hassastır fakat **maliyeti açısından oldukça ucuz diskler**dir. 1 TB HDD 120 GB SSD ile aynı

fiyatlarda olabilmektedir. *Peki, SSD tam olarak nedir* ve *neden pahalıdır* sorusunu inceleyecek olursak; **Solid State Drive** yani kısaca **SSD**'ler verileri manyetik plakalar yerine bellek yongasına benzeyen flash yongalarda tutmaktadır. Hard disklerde olduğu gibi veriyi plakaların üzerinde bulunan sektörlerde tutmak yerine **NAND Flash yongalarında saklanırlar**. SSD diskleri, USB flash belleklere benzetebiliriz.

USB flash belleklere çok benzeyen **SSD** disklerin içerisinde manyetik plakalar, okuma kafası, motor gibi parçalar bulunmadığından **tak çıkar işlemine daha duyarlıdır**. Veriler elektriksel olarak okunup yazıldığı için **okuma yazma hızı oldukça yüksektir**. Veriyi okumak veya yazmak için sektöre, izlere veya manyetik diğer prosedürlere ihtiyaç duymadığından elektriksel olarak **doğrudan verinin saklandığı adrese erişebilmektedir**.

Yine mekanik parçalar olmadığı için çalışma esnasında bir **ses söz konusu değildir**. Aynı zamanda mekanik parçanın olmayışı çok daha az elektrik harcamasına da neden olur. Kitabımızın başında, mühendisliğin amaçlarından bir tanesinin **hız** ve **optimizasyon** olduğundan bahsetmiştik. SSD disklerde iyi bir optimizasyon yapılmıştır. Hem **enerji tasarrufu** hem **ısı probleminin olmayışı** hem de **mekanik parça olmadığından** kolaylıkla taşınabilmesi buna güzel bir örnektir.

Bu kadar olumlu bir teknoloji ile ilgili tam olarak olumsuzluk olmasa da satın alırken bilmemiz gereken ve insanları bu konuda bilgilendirmemiz gereken mühendis olarak sorumluluğumuz da olan **iki küçük ayrıntıyı** paylaşarak SSD konusunu tamamlayalım. **Flash yongalar**, üretim şekillerine göre **iki şekilde** üretilirler. Bunlardan bir tanesi sektörde **SLC** adı ile bilinirken diğeri **MLC** olarak tanınır.

SLC–**Single Level Cell** flash yongaları, her bir hücrede **1 bit** veri tutma kapasitesine sahiptirler. Bu durum daha az kapasite sunarken bu devre tipleri *çok daha fazla performans* sağlarlar.

MLC–Multi Level Cell flash yongalar ise her bir hücrede **2 bit** ve daha fazla bit veri tutma kapasitesine sahiptirler. Kapasiteden dolayı tercih sebebi olan MLC yongaları *performans olarak SLC kadar iddialı değildir*.

Sabit diskler anakart ile iletişimlerini **veri kabloları** ile gerçekleştirirler. HDD veya SSD, SCSI hangi tür olursa olsun **IDE, SATA, PATA** gibi veri kablolarından

kendisine uygun olan ile bağlanırlar. Sata veri kabloları **Serial Ata** olarak bilinirler (*Serial Advanced Technology Attachment*). Sata veri kabloları, en fazla **90-100 cm** olarak kabul edilebilir. Veri transferi yenilenen sata kablolarında **590-600MB/s** hıza cevap verebilmektedir. 80 Pin IDE kablolarından 7 pine sahip olması sebebiyle boyut olarak çok daha **küçüktür**. Pinler sırasıyla **GND, TXP, TXN, GND, RXN, RXP** ve **GND** şeklindedir.

Pata veri kabloları (*Parallel Advanced Technology Attachment*), **paralel satalardır**. Kablo üzerine iki tane sürücü tanımlanması mümkündür. İki ayrı sürücünün çalışma esnasında **öncelikli olanının** belirlenmesi **master/slave** ayarları ile mümkündür. Sabit disklerin üzerinde bulunan master/slave pinleri, bir tanesi **master** diğer **slave** (*efendi/köle*) olacak şekilde ayarlanarak kabloya bağlanırlar. Paralel bir veri iletişiminin söz konusu olduğu bu veri kablosuna, sadece **iki adet disk bağlamak yerine** master pin **disk için ayarlanıp**, slave pinini de **CD-ROM** olarak ayarlayıp, hard diskteki pini master, CD-ROM'daki pini slave olarak ayarlayıp kabloya her ikisini de bağlamak mümkündür.

NOT

Bir hard disk alırken dikkat edilmesi gereken husus, öncelikle diskin **dönüş hızı** olmalıdır. HDD seçiminde işletim sisteminin desteklediği dosya boyutu ve kapasitesine de dikkat edilmelidir.

Kart Okuyucular ve Hafıza Kartları

Günümüzde, cep telefonlarından araç içi kameralara kadar **hafıza kart**larının girmediği yer yok diyebiliriz. Öyle ki otomobillerde bulunan navigasyon sistemlerindeki haritalar bile **hafıza kartı modülleri** ile desteklenmektedir. Veri transferinin **hızlı** gerçekleştirilmesi, **maliyetin ucuz olması** ve **taşınabilir** olması sayesinde kolayca herkesin sahip olduğu bu diskler, flash hafızalar olarak tanınırlar. Bu aygıtlar enerji yönünden **pasif** oldukları için yalnızca bilgisayara takıldıklarında bilgisayarın USB portu veya okuyucunun göndermiş olduğu enerji ile işlemlerini gerçekleştirebilirler. Bu sebeple **donanımı güvenle kaldır** demeden flash bellekleri veya hafıza kartlarını enerji transferi henüz devam ediyorken çıkartmak doğru bir davranış değildir. Windows işletim sistemine sahip değilseniz bile "**Donanımı Güvenle Kaldır**" benzeri bir güvenli kaldırma aracı, kullandığınız işletim sistemi tarafından sunulur. Kart okuyucular çeşitli kartları okumaya yarayan **okuyucu donanım birimleri**dir. Bir kart okuyucu, yalnızca masaüstü bilgisayarların kasalarında eski disk okuyucu boyutlarında gelebileceği gibi dizüstü bilgisayarlarda da bütünleşik olarak

gelmektedir. Ayrıca bir kart okuyucusu ile *kameralarda, araçlarda, fotoğraf makinelerinde, cep telefonlarında ve yazıcılarda* da karşılaşmamız mümkündür. **SD**, **MiniSD**, **MicroSD** gibi boyutlarda hafıza kartları ve bu kartları okuyacak yine o boyutlarda okuyucu haznesine sahip kart okuyucuları piyasada **çeşitli markalar** ile bulunmaktadır.

Optik Sürücüler

CD, DVD ve Blueray disklerin ne olduklarını biliyoruz. **Compact Diskler** (CD), **Digital Versatile Disk DVD**, **Mavi Lazer teknolojili** (BlueRay) **diskler** optik disk sürücülerinin en temelleridir. Üzerlerindeki **çukur** ve **tümseklere** göre (1-0) veri okuma yazma işlemi gerçekleştirirler ve kapasite olarak farklılık gösterirler. CD'ler **CD-R** ve **CD-RW** olarak ikiye ayrılırlar. CD-R'ler sadece **bir kere yazılabilen** ve **daha sonra değiştirilemeyen CD'lerdir**. CD-RW ise **ReadWrite edilebilir**. Yani tekrar üzerine veri yazılması mümkündür. 700 MB kapasiteye sahip günümüz CD'leri 12 mm boyutunda ve 1.2 mm kalınlığındadır. CD RW sürücülerinin üzerinde yer alan sayısal değerler X birimi gösterilir ve yazma, okuma hızlarını belirtir. Her bir CD için **üç farklı hız birimi** vardır. Bu hızlar; CD Okuma hızı, CD Yazma hızı ve CD tekrar yazma hızını belirtmektedir. Örneğin; **52 x 32 x 52** gibi.

Günümüzde hâricî bellekler cepte taşınabilecek kadar küçük ve ucuz oldukları için **CD**, DVD gibi ortamlar veri taşımak için pek **kullanışlı çözümler sayılmazlar**. İlk çıktığı yıllarda (1983) eski tür disklere göre büyük bir yenilik ve büyük kapasitesi ile uzun yıllar çok kullanılsa da artık yerini **USB Flash bellekler** almıştır. CD'ler **polikarbonat plastikten** üretilmiştir. Bu plastik alüminyum tabaka ile kaplanır. Yazma işlemi yakma diye bilinir ve CD üzerinde bulunan tabakadaki çukur ve tümseklerine **yazma işlemi** (*burning*) gerçekleştiği zaman CD içerisine veriler aktarılmış olur. CD yazma yazılımlarında genelde CD yaz anlamında **Burn** terimi kullanılır.

CD'den en büyük farkı **kapasitesi** olan DVD'ler ise **4.7 GB** kapasiteye sahiptirler. Yaklaşık olarak **6 CD boyutuna eşit** olan DVD ortamında veri kümeleri birbirlerine **daha yakın** konumlandırılarak daha çok veri sığdırılması ilkesi ile büyük bir boyuta kavuşmuştur. İki katmanına da veri yazılabilen türü de bulunan **çift katmanlı çift yüzlü DVD ortamları 17 GB kadar veri kapasitesine sahiptirler**. DVD ortamları üzerinde de yazılı olan 12x, 16x ve 24x değerleri yazma hızını belirtmekte olup **x birimi yaklaşık 1320 KB** değerindedir. (*Farklı kaynaklarda 1318 şeklinde de ifade edilir.*)

Blue-Ray sürücüler, **mavi lazer teknolojisi**ni kullanırlar. Bu teknoloji, **daha yenidir** ve kapasitesi tabaka üzerinde **daha hassas odaklama yapabilmesinden ötürü daha yüksektir**. Fiziksel kapasite olarak CD ve DVD ile boyuta sahip Blue-ray diskler **25 gb kapasiteye sahiptirler**. DVD diskler **720*480p çözünürlük** desteklerken blue-ray diskler, **Full HD 1080p** destek vermektedir. Daha büyük veri kapasitesi ve video dosyalarındaki çözünürlüğü ile günümüz optik disklerinin en güncelidir diyebiliriz. Sadece okunabilen blue-ray ortamlarına **BD-ROM**, yazılabilir blue-ray ortamına **BD-R,** yeniden yazılabilir olan blue-ray ortamlarına ise **BD-RE** denilmektedir. Tek katmanında **25-27 gb** veri depolayabilirken çift katmanlılarında **54 gb** kapasite bulunur.

Ekran Kartı ve Diğer Kartlar

Belki 15 yıl öncesine kadar geliştirilen anakartlarda çeşitli donanım kartlarının takılabilmesi için çeşitli **slotlar** fazlaca üretilirdi. Anakart başlığında işlediğimiz *PCI, AGP, ISA gibi yuvalar ekran kartı, ses kartı, TV kartı* gibi donanım kartlarının **sonradan takılabilmesine olanak sağlayan yuvalardır**. Daha yeni anakart devre tasarımlarında, çoğu kartın **tümleşik** olarak üretilmesi sebebiyle özellikle PCI slotuna da ihtiyaç eskiye oranla azalmıştır.

Ekran Kartı

Bilgisayarımızı topladık, diskleri işlemciyi her şeyi montajladık ve güç butonu ile açtık. İşte, bu aşamadan sonra **ekrana görüntünün gelmesini sağlayan donanım ekran kartıdır**. Ekran kartları işlemcide gerçekleşen işlemlerin sonucunu kullanıcıya monitör vasıtasıyla aktaran **grafik işlemci**leridir. Bir bilgisayardaki grafiğin kalitesi işlemciden ve monitörden etkilendiği gibi ekran kartı ile de doğrudan ilgilidir. **İyi bir ekran kartı iyi bir grafik sunmaktadır**. Ekran kartları ise iyi bir grafik sunabilmek için iyi bir fiziksel yapıya ve veri yoluna sahip olmalıdırlar. Ekran kartları veri yolu arayüzü olarak **CGA**, **VGA**, **SVGA** gibi arayüzler kullanırlar.

Ekran kartları, bilgisayar sistemimizdeki anakart ile **enerji**, işlemci ile **grafik oluşumu**, RAM ile işlemci ekran kartı arasındaki geçici depolama ve monitör ile de nihai sonucun kullanıcıya gösterilmesini sağlarlar. Görüntünün oluşturulması ve kullanıcıya ulaşmasındaki en önemli bileşenleri ise şunlardır:

VGA BIOS: Tıpkı anakart üzerindeki bios'un yaptığı gibi ekran kartlarının çalışması için gerekli olan komutları içerir.

GPU: Ekran kartı işlemcisidir. (*Grafik Processor Unit*) Ekran kartının görüntü hesaplama işlemlerini, matrislerde gerçekleşen işlemlerin yapılabilmesini sağlar ve bu işlemleri ana işlemciye yani **mikroişlemciye yansıtmadan** direkt kendi üzerinde yapar.

Video RAM: Ekran kartının kendi biosu, kendi işlemcisi olduğu gibi kendisine ait bir geçici belleği de vardır. Bu bellek işlemleri yaparken geçici depolamayı ana bellek üzerinde yapmayarak bilgisayar performansına **olumlu** katkıda bulunarak kendi belleği yani **video RAM** üzerinde gerçekleştirir.

Ramdac: (*Dijital-Analog Çevirici*) Ekran kartlarının geçici belleğinde bulunan, belirli bir dizi matris değerleri içeren sayısal (*dijital*) verilerin, kullanıcı tarafından monitörde görünebilecek analog sinyallere dönüştürülmesini sağlayarak bu sinyalleri ekran kartının monitör bağlantı port çıkışına gönderir.

Ramdac dönüştürücüsünün **verileri aktarma hızı oldukça önemli bir değerdir**. Öyle ki bu değer bize ekranın **refresh** (*yenileme*) hızını vermektedir. Yenileme hızını Hz birimi ile ölçeriz ve genelde standart bir monitörde yenileme hızı işletim sistemi üzerinden **70 Hz** olarak ayarlanır. 70 Hz yenilenme hızına sahip bir ekran kartı saniyede sahip olunan görüntüyü **70 defa yenilemektedir**. Burada verilmesi gereken bir diğer bilgi ise **LCD ekranların zaten dijital sinyalleri görüntüleyebildikleridir**. Ekran kartı belleğindeki veriler, RAMDAC dönüştürücüsüne gitmeden direkt olarak ekran kartının sahip olduğu DVI portu vasıtasıyla LCD ekran monitöre aktarılmaktadır.

Veri yolu standartlarına göre **ISA**, **PCI**, **AGP**, **PCI-e** ve **PCI-X** şeklinde gruplandırılan ekran kartları üreticiler tarafından **farklı kullanımlara göre çeşit çeşit üretilmektedirler**. Bant genişliğinin yüksek olması sebebiyle günümüzde **en çok kullanılan ekran kartı modeli PCI-e** ekran kartlarıdır.

ARAŞTIRMA: Ekran kartı montajı ve kullanım ihtiyaçlarına göre modellenmiş olan SLI ve Crossfire teknolojisini kendiniz araştırınız.

Ses Kartı

Günümüzdeki anakartlarda uzun yıllardır ses kartları **tümleşik** (*onboard*) olarak gelmektedir. Hâricî bir ses kartı takılması gerektiği durumlarda **PCI**, **ISA** veya **PCI-e** genişleme yuvalarından bir tanesine takılabilir. **Hâricî ses kartı takılacağı durumlarda bios üzerinden bütünleşik olan ses kartı devre dışı bırakılmalıdır**. Ses kartlarının da ekran kartlarında olduğu gibi temel bileşenleri bulunur.

DSP (*Dijital sinyal işlemcisi*): Ses kartının sesi üreten işlemcisidir.

FM-Wavetable: Ses kartından gelen dijital sinyali midi sesine dönüştürür.

DAC/ADC: Analog sinyalleri dijital sinyallere çevirir. Dijital ses verilerini çıkış olarak analog sinyallere çevirir.

Ses kartları mikrofon girişi, diğer player cihazlarındaki sesin aktarılabilmesi için **Line in**, hoparlörlerden ses çıkışı verebilmek için **Line out**, çeşitli kartlar arasında bağlantı sağlamak için **AUX-in**, 3D sesler için **Rear out**, MIDI ve Joystick aygıtları bağlanabilsin diye **Joystick-MIDI portları**, modem ile bağlantı kurularak telefondan gelen sesin aktarılması için **TAD**, CD-ROM üzerindeki audio in girişi için **CD Audio** bağlantısı ve DAT türdeki kaynaklardan gelen verilerin sorunsuz aktarılması için **S/PDIF bileşenlerine** ve **portlarına** sahiptir.

TV Kartı

Televizyon yayınlarının bilgisayarlarımızdan izlenebilmesini sağlayan kartlardır. Standart bir anakart üzerinde **tümleşik olarak gelmez**. Hâricî olarak **PCI yuvasına takılırlar**. Analog yayınların izlenebilmesini sağlayan **Analog TV kartı** ve dijital yayınların izlenebilmesini sağlayan **Dijital TV kartı** şeklinde iki türü vardır. Analog için **normal bir anten**, dijital için ise televizyonlarda olduğu gibi **uydu anteni bağlantısı** gerekmektedir.

TV kartlarında **Tuner bileşeni** bulunur. Tuner bileşenin görevi anten üzerinden gelen farklı frekanslara ait yayınlardan istenilen yayını almak ve diğerlerini filtrelemektir.

Ethernet Kartı

Bir bilgisayarın, internete bağlanmasını sağlayan donanım birimi **Ethernet kartıdır**. Günümüz anakartlarında uzun yıllardır Ethernet yongası **tümleşik olarak** (*onboard*) gelmektedir. Ethernet kartları da **PCI yuvalarında çalışırlar**. Üzerlerinde **RJ-45 konnektörü** ve kablolu internet için **BNC konnektörü bulunur**. Ethernet kartlarına sektörde **Ağ Kartı** da denilmektedir. Elektronik mağazalarında veya ürün data sheetlerinde NIC yani **Network Interface Card** ve **LAN kartı** olarak da geçmektedir.

Koaksiyel kablolar ile iletişimin sağlandığı internet ortamlarında koaksiyel kablo ile bağlanan BNC konnektörlü Ethernet kartları kullanılır ve en fazla **10 Mbps veri aktarımına** olanak sağlar.

Twisted-pair yani **eş bükümlü bakır kablolar** ile internet erişiminin sağlandığı ortamlarda ise **RJ-45 konnektörü** kullanılır. **10,100,1000 Mbps** hızlarda veri iletimine olanak sağlar. Kablo çeşidi olarak en yaygın **cat5**, **cat6 kablolar** kullanılır.

Ağ ile ilgili konuları anlatacağımız bölümde tüm detaylarına değineceğimiz için bu kısımda çok detay vermeden son olarak **iletişimin şekline** ve **paketlere** değinmek istiyorum. İnternet erişiminde bilgi, **paketler hâlinde transfer edilmektedir**. Paketler hâlinde gönderilmesi hem trafiği yormaz hem de olası bozulmalarda bilginin tamamının bozulması önlenmiş olur. Bir veri paketi **4 bileşenden oluşur**. Bunlar; **Hedef MAC adresi**, **Kaynak MAC adresi**, **Veri** ve **Checksum CRC kodu**dur. MAC numarası 48 bit genişliğe sahip bir adrestir. İlk 24 biti üreticiyi tanımlayan son 24 biti ise yine üretici tarafından üretilen unique yani benzersiz bir ID numarasıdır. **Her mac adresi dünyada yalnızca bir kez kullanılmaktadır**. Paket içerisindeki hedef mac karşı tarafın, yani paketin teslim edileceği tarafın mac adresini tanımlar. Kaynak mac ise göndericinin mac adresidir. Data, **gönderilecek veriyi ifade eder** ve CRC ise **veri ile birlikte paketlenen bir kontrol kodudur**. CRC kodu başlangıçta ve teslim aşamasında check edilir ve fark yok ise veri bozulmadan transfer edilmiştir anlamına gelir.

Diğer Kartlar

Güvenlik Kartı: Bilgisayarın **CMOS** ve **BIOS** bilgilerinde kullanıcı tarafından yapılacak olası değişiklikler ve *bilgisayarı zararlı dosyalara karşı korumak, dosyaların elektrik kesintisi* olmak üzere çeşitli sebeplerle bozulması gibi sorunlara karşı sistemi koruyan, bilgisayarı yeniden başlatınca **eski hâline dönmesini sağlayan** kartlardır.

SCSI Kartı: IDE ile bağlanabilen birçok donanım örneğin optik sürücüler gibi SCSI ile bağlanabilir. Bir SCSI kartına 10-15 adet cihaz bağlanabilir ve IDE'ye göre çok **hızlıdır**. PCI yuvasından çalışan SCSI kartlarının kendilerine ait işlemcileri bulunur bu yüzden **hızlı** ve **güçlü** üretilmiş cihazlardır.

İşletim Sistemleri

Bu başlıktan önceki konularda **bilgisayarın donanım birimlerinden** ve **hiç bilmeyenler için** bir bilgisayar kasası içerisinde yer alan donanımlardan bahsedildi. Kasalar anlatılırken kasanın güç butonuna basıldığında bilgisayarın açıldığını biliyoruz. *Peki, bilgisayar tüm bu donanımları çalıştırdıktan sonra bizimle nasıl iletişim kuruyor* biraz da bu konuyu inceleyelim. **Kullanıcı ile iletişimi olan her türlü bilgisayarın işletim sistemi vardır**. İşletim sistemleri; bilgisayarın donanım birimlerinin kullanıcı tarafından yönetilebilmesini, kullanılmasını ve yapacağı tüm işlemleri yapabilmesini sağlayan bir arayüzdür. Bilgisayarın tanımını yaparken donanım ve yazılım bileşenlerinden oluşan bir aygıt olduğunu söylemiştik. İşte burada, **yazılım** kısmına değineceğiz. İşletim sistemi adı üzerinden çıkarım yapabileceğimiz gibi işlemlerin çalıştırıldığı yani **işletimin söz konusu olduğu** bir programdır.

Cep telefonları, bilgisayarlar ve ATM cihazlarını kullanmamızı sağlayan donanım ile uyumlu çalışan **sistem yazılımları işletim sistemleri**dir. Bu bölümde, son kullanıcı seviyesindeki işletim sistemlerin ait konularını işleyecek olup *sunucu işletim sistemlerini ve gelişmiş özellikleri kitabımızın ilerleyen bölümlerinde başka bir bölüm başlığı altında inceleyeceğiz*.

İşletim sistemleri çok geniş bir konudur ve bu başlık altında yazılacak çok şey vardır. Üniversitelerde veya meslek liselerinde bu konu güzel bir şekilde işlenmiş olup **kurulumu** ve **gelişmiş** özellikleri ile ilgili bilgiler **Windows işletim sistemleri için müfredat hâline** getirilmiştir.

Fakat **işletim sisteminin internet araştırmalarında kolaylıkla karşımıza çıkmayan** çalışma mekanizması ile ilgili algoritmaları olan, yani **mühendisliğin ilgilendiği konular** bulunmaktadır.

Örneğin, **Multi Level Feedback Queue** (*MLFQ*) algoritması gibi işletim sistemi algoritmaları çalışma mantıkları örnek kodlar ile simülasyon uygulamaları, daha ileri başlıkların altında **işletim sistemi mühendislik konuları** başlığı ile anlatılacaktır. Birinci bölümün içerisinde olduğumuz Bilişim Teknolojilerinin Temel ilkeleri veya Bilgisayar Mühendisliğine Giriş veya Fundamentals Of Computer Science çerçevesinde olduğumuz bu bölümde, *biraz daha temel bilgiler, işletim sisteminin ne olduğu, kaç çeşit işletim sisteminin piyasada varlığını sürdürdüğü, yeni bir sistemin nasıl kurulduğu, Windows işletim sistemlerinin çekirdek olarak kodlanan versiyonlarını ve yama olarak geliştirilen versiyonlarını, profesyonel bir kullanıcının işletim sisteminin ne kadarına müdahale edebileceği ile ilgili bilgilerin verilmesi ve sunucu işletim sistemleri ile ilgili bilgilerin* **bu bölümde verilmesi amaçlanmıştır**.

İşletim sistemleri günümüzde son derece **kullanıcı dostu** olarak tasarlanan ve **geliştirilen** yazılımlardır. Klavye ve Mouse ile oldukça kolay kullanılabilen bu yazılımlar, geçmişten günümüze üreticilerin edinmiş oldukları **user experiements**, yani **kullanıcı deneyimi** sayesinde oldukça iyi seviyelere getirilmiştir. Bir işletim sistemi kullanıcı ile bilgisayar arasındaki iletişimi sağladığından **kullanımı kolay olmalıdır**. Aynı zamanda arka tarafta da **donanım birimleri ile uyumlu bir şekilde çalışmalıdır**. Kullanıcı dostu ama donanım ile uyumsuz çalışan bir işletim sistemi bu uyumsuzluk sebebiyle zaten kullanıcıya stabil olmayan tepkiler verecektir. Günümüz son kullanıcıları özellikler öğrenciler, küçük yaşlardan itibaren akıllı cep telefonları ile fazla içli dışlı oldukları için işletim sistemi kullanma kültürüne sahiptirler. Bu kullanım kültürü sayesinde farklı işletim sistemleri ile karşılaştıklarında bile, kullanıcılar zorluk çekmeden yeni bir telefonu veya bilgisayarı biraz kurcalama ile kolaylıkla kullanabiliyorlar.

Mühendisliğin gelişimi, programlama ortamlarının gelişimi kadar tasarımlar ve kullanıcıyı doğru adıma götürme adına diyalog kutularının yerleşimi, işletim sistemi kullanımı esnasındaki **iş akışı** da oldukça önemlidir. Bunu en güzel gerçekleştiren yazılım, **Microsoft işletim** sistemleridir. Özellikle **XP** versiyonu ile birlikte günümüz kullanıcılarının çoğunda oturmuş bir kullanım kültürü oluşturulmuştur. Amerika yapımı filmlerde çok görülen siyah ekran karşısında işlem yapan bilgisayarlar ve işletim sistemlerini muhtemelen bugünün üniversite öğrencileri hiç görmemiş olabilirler. **1980**'lerde Microsoft'un **MSDOS** işletim sistemi vardı ve işlemler siyah bir konsol ekranı üzerinden gerçekleştiriliyordu. IBM, Sun ve MS işletim sistemlerinde o yıllarda grafik desteği yoktu, öyle ki kodlama ortamları bile bugünün nesne tabanlı veya görsel programlaması kadar ileride olmadığı için **kod geliştirme ortamları bile daha ilkeldi**.

Geliştirilen işletim sistemleri açık kaynak kodlu yani herkes tarafından geliştirilmeye açık **ücretsiz** ve **lisanslı** olup **ücretli olan şekilde iki ayrı grupta piyasada bulunurlar**. Ticari kaygı gütmeyen ve herkesin geliştirmesine izin veren yazılımlara **GPL** (**General Public Lisans**) **lisanslı** denir. **Ücretsiz** olarak sunulan bu sistemlerden en yaygın olanı **Linux** dağıtımlarıdır. Açık kaynak kodlu olan bu işletim sistemlerinden bilinen diğerleri ise aşağıdaki gibidir.

UNIX, FREE-DOS, SUN SOLARIS, SUSE, UBUNTU, RED HAT, PARDUS, DEBIAN

Ticari olarak satışı mümkün olan ve lisanssız kullanımda ilgili kuruluş tarafından cezai işlem uygulanabilen işletim sistemlerinde ise hiç şüphesiz herkesin aklına **Microsoft çözümleri** gelmektedir. Geçmişten günümüze geliştirilen Microsoft işletim sistemleri aşağıdaki gibidir.

MS-DOS, WINDOWS 3.1, WINDOWS 95, WINDOWS 98, WINDOWS ME, WINDOWS MILENIUM, WINDOWS XP, WINDOWS VISTA, WINDOWS 7, WINDOWS 8, WINDOWS 10

Mobile işletim sistemi grubunda; **Windows Phone, Apple IOS, Android, Blackberry, Symbian, Tizen, Windows Mobile, Ubuntu, WebOs, Bada** işletim sistemleri bilinenler arasında yer almaktadır.

Bir işletim sistemi kurulmadan önce sahip olduğumuz bilgisayarı iyi tanıyor olmalıyız. Donanımın desteklediği bir işletim sistemi temin ederek kurulum yapılmalıdır. **İşlemcimiz 32 bit ise kurulumunu yapacağımız işletim sistemi de 32 bit olmalıdır**. Aynı şekilde işlemcimiz 64 bit ise 32 bit işletim sistemi de kurulabilir fakat kesinlikle 64 bit işletim sistemi kurulumu yapılmalıdır. İşletim sistemi kurulumu sonrasında yüklenecek olan diğer programlarda da aynı şekilde 32 veya 64 bit ayrımı söz konusu olduğu için 64 bit bir bilgisayara 32 bit işletim sistemi kurulması durumunda kurulacak diğer programlarda da **karışıklıklar, donanım uyumsuzlukları stabil çalışmayan aygıtlar** gibi sorunlar teşkil edecektir.

İşletim sistemleri kurulumu ile ilgili olarak üç çeşit kurulumdan söz edebiliriz. Bunlardan ilki son kullanıcı tarafından format atmak olarak bilinen sistemin **eski sistem silinerek** (*biçimlendirilerek*) **yeni sistemin kurulması** şeklinde gerçekleşmektedir. İkinci kurulum şekli aslında tam bir kurulum olmayıp mevcut işletim sisteminin **uygun güncellemesi ile bir üst versiyonuna yükseltmektir**. Bu işlem yükseltme olarak bilinir ve lisanslı bir Microsoft işletim sisteminde **lisans olduğu sürece üst versiyona yükseltme işlemi yapılabilir**. Diğer kurulum yöntemi ise yine **sıfırdan işletim sistemi kurmaktır** ama bir diske **iki ayrı işletim sistemi** kurulmasıdır.

Kitabımızda bir işletim sisteminin nasıl kurulduğuna değinmeyeceğim. Okuyucularımızın çoğunluğunun daha önce *sistem kurduğunu/format attığını* tahmin ediyor, daha önce kurulum gerçekleştirmeyen okurlarımız için ise kısa bir **Google araması** yaparak işletim sisteminin sıfırdan bir bilgisayara nasıl kurulduğunu adım adım görebileceklerini ve bu sayede kurulum yapabileceklerini düşünüyorum.

İşletim sistemi kurulurken bilmemiz gereken en önemli husus, **donanıma uygun işletim sistemi temin ettiğimizden emin olmamız** ve **kurulum aşamasında da diskin biçimlendirilme** ve **eski bilgilerin silinmesi gerektiği durumunu bilmemiz gerekmektedir**. Yeni bir işletim sistemi kurulurken eski işletim sistemi silinir. Eğer disk boyutunda değişiklik yapılmayacaksa eski sürücünün üzerine biçimlendirme işlemi yaparak yenisi kurulur. Eğer diskteki sürücü boyutunda değişiklik yapacaksak veya sürücü sayısını fazlalaştıracaksak bu durumda sistemin yükleneceği sürücünün **biçimlendirileceği** gibi oluşturulan yeni sürücünün de kurulum yapılmadan format esnasında **bir kez** biçimlendirilmesi gerekmektedir.

Kurulum esnasında biçimlendirilmeyen sürücüler daha sonra kullanılmaz duruma gelmektedir. İşletim sistemi kurulumu yapmadan disk sürücülerinde değişiklik yapılacak ise bunu çeşitli **partition bölümleme** programları ile yapabildiğimiz gibi Windows işletim sisteminin kendisinde bulunan **disk bölümlendirme arayüzü** ile de yapabiliriz. Sistem çalışıyor iken yapılacak tüm genişletme veya daraltma işlemlerinde veri kaybı olacağından **öncelikle verilerin yedeklenmesi** veya **bölümlendirme işleminin çok dikkatli bir şekilde yapılması gerekir**.

Sistem Geri Yükleme

İşletim sistemi yüklemek, *sıfır alınan bir bilgisayara mahsus olup bilgisayar bozulana kadar tekrar sistem kurulumu yapmaya gerek yok mudur?* Elbette bu çoğu kullanıcının istediği bir şeydir ama ne yazık ki **sistemler çeşitli sebeplerden ötürü yavaşlarlar** ve **sistem performansı ilk zamanlarda olan kararlılığını gün geçtikçe yitirmeye başlar**. Bu yavaşlığa neden olan çok fazla etken vardır örneğin internet üzerinden gördüğü her şeyi indirip bilgisayarına kuran ve deneyen bir kullanıcının bilgisayarı, işletim sisteminin bulunduğu sürücü (*genelde C:// sürücüsü*) üzerinde çok fazla veri tutan kullanıcıların bilgisayarı veya anti virüs yazılımı kullanmayan birisinin bilgisayarı zamanla yavaşlayacak ve gerek kullanıcı alışkanlıklarından gerek ise donanımın tembelleşmesinden ötürü ara sıra sistemi yeniden kurmak çoğu kullanıcının yaptığı/yaptırdığı bir işlemdir. **Doğrudan her başımız sıkıştığında format atma eyleminde bulunmayıp sistemi ilk günkü hâline geri yükleme adımı** ile bilgisayarlarımızı **daha stabil** kullanabiliriz. Kitabın başından beri verdiğim otomobil örneklerinden birini de burada vermek gerekirse otomobillerimizin rutin bakımlarında olduğu gibi **bilgisayarlarımıza da rutin bakımlar yapmalıyız**. Bu bakımların işletim sistemi üzerinden yapılabilen hâline **Sistem Geri Yükleme** modülünü örnek gösterebiliriz.

Sistem geri yükleme, olası yaşanacak bir olumsuzluk hâlinde **bilgisayarın en stabil çalıştığı güne geri dönmesini sağlayan bir araçtır**. Sistemin en stabil çalıştığı güne geri dönebilmesi için daha önceden o güne ait bir geri yükleme noktası oluşturulması gerekir. Windows işletim sistemimiz üzerinden **Başlat** menüsü arama alanına **Sistem Geri Yükleme** denildiğinde araç ile ilgili diyalog kutusu çalışacaktır. Sistem geri yükleme aracı üzerinden belirli aralıklarla **otomatik** geri yükleme noktaları oluşturulmasını sağlayabiliriz.

Disk Birleştirme, Temizleme, Denetleme

Disk Birleştirme: Disk içerisinde kullanım geçmişi söz konusu oldukça yani diskte **yazma-silme** işlemi olduğu sürece, veriler disk üzerinde **parçalanmış** şekilde tutulabilir. Disk birleştirme işlemi, **parçalanmış olan bu verilerin birleştirilmesi** ve **optimize edilmiş bir disk elde edilmesi işlemidir**. Bilgisayarlarımız biz bir veriyi silip daha sonra başka bir veriyi eklediğimizde eklenen yeni veriyi disk üzerinde **çok farklı boşluklara** yerleştirebilir. Kullanıcının arayüz üzerinden göremeyeceği bu farklılık, verinin okunması için gerçekleştirilen işlemde **bilgisayarı yorabilir**. Farklı boşluklara yerleştirilen aynı veri grubuna ait bilgiler farklı referanslardan okunup getirildiğinde RAM ve işlemci gereksiz yorulmuş olur. Disk birleştirme aracı ile zaman zaman **bu sorunu ortadan kaldırarak manyetik diskin ömrünü de bu sayede uzatmış oluruz**. Disk birleştirme aracı ile parçalanan veri grupları birleştirilmiş olur ve veriler yeniden düzenlenmiş olur. Bu sayede de bilgisayar ve diskin daha verimli çalışması sağlanmış olur. Bu aracı, sistem geri yükleme aracında olduğu gibi **zaman zaman kendimiz çalıştırabileceğimiz gibi otomatikleştirip kendisinin belirli zaman dilimlerinde çalışmasını sağlayabiliriz**.

Disk birleştirici aracını çalıştırmak için Windows işletim sisteminden **Başlat** menüsü arama alanına **Disk Birleştirici** yazıp sonuç listesinde görüntülenen programı çalıştırmak yeterlidir. Önemli bir adım olduğu için varsa işletim sistemine ait **admin kullanıcısının parolası** sistem tarafından istenebilir. Gelen diyalog penceresinden ise birleştirme ile ilgili ayarlamalar yapılarak araç kullanılır.

Disk Temizleme: bu aracın amacı ise **diskte yer açmaktır**. Gereksiz verilerin silinmesi her zaman kullanıcıya **performans artışı** olarak geri dönmektedir. Bu özelliğe akıllı telefon kullanan herkesin aşina olduğundan eminim. Özellikle android işletim sistemine sahip cep telefonlarında bellek sorunu var ise geçici verilerin silinmesi ile ilgili sihirbaz penceresi **otomatik** olarak gelmektedir. Akıllı telefon burada geçici verileri silerek hem **yer kazanımı** hem de **performans optimizasyonunu** sağlamış olur. Windows işletim sisteminde **Başlat** menüsü arama alanından **Disk Temizleme** yazarak çalıştırabileceğimiz bu araç ile geçici dosyaları ve geri dönüşüm kutusu içindeki dosyaları temizleyerek **gereksiz ögelerin kaldırmasını sağlayabiliriz**.

Disk Denetleme: Hataları giderme ve belirli oranda onarım işlemi yapılması da bilgisayarımızı stabil hâlde kullanmak için yapmamız gereken ince ayarlardan

bir tanesidir. Bilgisayarımızdaki **sürücülerde** veya **hâricî belleklerdeki hataları denetleyebiliriz**. Denetleme ve kısmen onarım işlemi ile yine performansta iyileştirme sağlanmış olur. Aynı zamanda denetleme aracı bize diskin stabil çalışıp çalışmadığını da gösterecektir.

Denetleme aracına erişmek için **Bilgisayarım** diyalog penceresi üzerinde görüntülenen sistemde **aktif** olarak çalışan sürücülerden bir tanesine **sağ tıklanır** ve en altta yer alan **Özellikler/Araçlar/Hata Denetimi** altından **Şimdi Denetle**ye tıklanır. Denetleme aracındaki tarama düzeltme ayarlarından yapmak istediğimiz ayar seçilerek onarma işlemi gerçekleştirilebilir. Örneğin **Dosya Sistemi Hatalarını Otomatik Olarak Düzelt** seçeneğini seçip tarama işlemini başlattığımızda bulunan sorunlar **otomatik** olarak düzeltilmiş olur.

Kayıt Defteri

İşletim sistemi, kurulum aşamasında **çeşitli ayarları yapılan** (*bölge, dil, klavye, saat-tarih vb.*) ve **bu ayarlara göre kurulumu tamamlanan bir yazılımdır**. Fakat içerisinde gerçekleşen tüm işlemler ile ilgili ayarların, kullanıcı hesaplarının, profillerin, yüklenen programların ve bu programların değiştirdiği konfigürasyonların yer aldığı, olan biten her şeyin kayıt altına alındığı deftere **kayıt defteri** (**REGEDIT**) denilmektedir. **Regedit.exe** bir tür veri tabanı çalıştırır ve işletim sistemi buradaki bilgilere sürekli olarak başvurur.

Eskiden çok korkulan bir tabir olan *bilgisayarın çökmesi*, *çökertilmesi* işlemi **regedit** üzerinden elle değiştirilecek bir konfigürasyon ile çok kolaydır. Yüklenen programlar kendi içlerindeki yapılandırma ayarları ile gerektiğinde regedit üzerinde **değişiklikleri kendileri yaparlar** bu yüzden son kullanıcı seviyesinde **elle müdahale etmek önerilmemektedir**. Kullanılan programlardan bir tanesi kayıt defterinde birtakım değişiklikler yapılması gerektiğine dair **kullanıcıya mesajlar verebilir** eğer böyle bir durum ile karşı karşıya kalırsak öncelikle **kayıt defterinin bir yedeği alınmalıdır**.

Regedit, **Başlat** menüsü açıkken **regedit** yazılarak çalıştırılabilir. Başlat menüsündeki **Çalıştır** uygulamasından da **Regedit** veya **Regedit.exe** yazılarak da çalıştırılabilir. Regedit çalıştırıldığı zaman bilgisayara ait **tüm konfigürasyon işlemlerinin** buradan yapıldığı anlaşılır. Bizler son kullanıcı seviyesinden daha üst bir seviyede bilgisayar kullanıcı olduğumuz için regedit üzerinden yapılabilecek ve kültür oluşturması açısından bilmemizde fayda olan

birtakım işlemleri *İŞLETİM SİSTEMİ GELİŞMİŞ ÖZELLİKLER VE SUNUCU İŞLETİM SİSTEMLERİ* bölümünde göreceğiz. Bu bölümde, örneğin bilgisayar açılırken **Num Lock** tuşunun sürekli açık gelmesini regedit üzerinden nasıl sağladığımızı öğreneceğiz veya bilgisayarımızda bir klasörü kısayol oluşturduğumuzda ve kopyalayıp yapıştırdığımızda yapıştırılan klasörün adının başına **Kısayol** kelimesinin gelmemesi için regedit üzerinden yapılabilecek işlemi göreceğiz.

Kullanıcı Hesapları ve Güvenlik

Windows işletim sistemlerini kurduktan sonra Denetim Masası üzerinden **Kullanıcı Hesapları menüsü** ile birden fazla kullanıcı oluşturabiliriz. Yani kişisel bilgisayarımızı bile birden fazla kullanıcı ile paylaşmış oluruz. Okullarda, bilgisayar laboratuvarlarından hatırlayacağınız üzere bir bilgisayarda **iki ayrı kullanıcı hesabı** olup birisi admin hesabı diğeri ise öğrenci hesabı olarak tanımlanan kullanıcı hesapları bulunur. **Her bir kullanıcının bilgisayardaki ayarları farklı olabilir**. Farklı bir masaüstü, farklı bir tema ve farklı dosyalar gibi. Standart hesap kullanıcıları birbirlerinin dosyalarına erişemezken yönetici hesabına sahip kişi tüm kullanıcıların ayarlarına ve dosyalarına erişebilir. **Windows 7** itibari ile işletim sistemlerinde **zorunlu bir adet yönetici hesabı** tanımlıdır. Standart kullanıcı hesabı yok ise yönetici hesabı bilgisayarı kullanan kişinin hesabıdır. Eğer standart kullanıcılar var ise yönetici hesabı ile giriş yapan kişi diğer oturumlara ait *yazılım yükleme, silme ve dosyaları görüntüleme* gibi işlemleri gerçekleştirebilir. Yönetici hesapları bilgisayarın bir **etki alanına** bağlı olarak ya da **çalışma grubuna** bağlı olarak çalışmasına göre farklılık gösterebilir. Bireysel kullanımda olan fakat yönetimsel işlerde bir kurum kültürüne bağlı olmak zorunda olan kullanıcılar için farklı yönetim senaryoları bulunmaktadır. Örneğin bir kurumdaki tüm bilgisayarları tek bir yerden yönetmek, aynı anda yazılım yüklemek ve yetkileri merkezî bir yerden vermek için bilgisayarları **bir etki alanına alarak** yapılacak her bir işlem için bir poliçe yazılır.

Windows bu işi **Active Directory** ile gerçekleştirmektedir ve bu konuda kitabımızın ilerleyen başlıklarından bir tanesi olan *İŞLETİM SİSTEMİ GELİŞMİŞ ÖZELLİKLER VE SUNUCU İŞLETİM SİSTEMLERİ* başlığında incelenecektir.

Parola Sıfırlama Diski: Windows işletim sistemlerini kurduktan sonra olası parola unutma durumunda bilgisayarı açabilmemizi sağlayacak **parola sıfırlama diski oluşturulabilir**. İsteğe bağlı olan bu işlemi yapabilmek için

oturum açma ekranındaki parola sıfırlama diski oluştur bağlantısından bir ortam seçilerek (**CD, DVD**) **disk sıfırlama dosyası oluşturulur** ve daha sonra parola unutulma durumunda bu disk sayesinde oturum açma işlemi gerçekleştirilmiş olur.

Güvenlik: İşletim sistemi kurulum işlemi tamamlandıktan sonra temel güvenlik önlemlerinin alınması amacıyla öncelikle **Windows güvenlik duvarının aktif değilse aktif edilmesi gerekmektedir**. Windows güvenlik duvarı (**firewall**) **ateş duvarı** manasına da gelmektedir. Firewall işletim sistemimiz ile birlikte gelen bir programdır ve kimlik avı ile diğer tehditlere karşı **kalkan görevi** görmektedir. Ayrıca bilgisayarımıza güvendiğimiz bir **anti virüs yazılımının lisanslı versiyonunun kurulması** da güncel virüslere karşı koruma sağlayacağı için önemlidir.

Bios

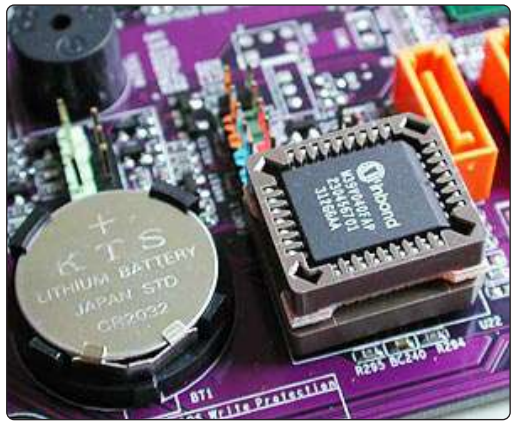
Bios bilgisayarın **en önemli** parçalarından bir tanesidir. **Temel giriş çıkış sistemi** anlamına gelen Bios, anakart üzerinde bulunan bir devredir. Genellikle hemen yanında da **Bios pili** bulunur. Bilgisayarın açılmasından kapanmasına anakart üzerindeki yuvalara yeni bir donanım bağlanması ile ilgili ayarlara, açılışta yüklenecek olan sürücüye kadar çok fazla işlem Bios üzerinden yapılmaktadır.

Basic Input Output System yani **BIOS**, bilgisayarın açılış esnasında **tüm kontrolleri gerçekleştirir ve denetler**. Bu kontroller bir bilgisayarın açılması esnasında hangi birimin önce hangisinin sonra çalışacağı gibi sıra işlemlerini, anakart üzerindeki donanım kartlarından hangilerinin çalıştırılacağının kontrolü ve birimlerden hangisine ne kadar enerji verileceğine kadar tüm kontrolleri kapsar. BIOS anakart üzerindeki **ROM içerisinde çalışır** ve EEPROM oldukları için **kullanıcı tarafından birtakım bilgilerin değiştirilmesine olanak sağlanır**.

Bir bilgisayarın Bios arayüzüne girildiğinde kullanıcıyı karşılayan **sabit yazılar** ve **sabit ayarlar Rom içerisindeki değiştirilemeyen alanda tutulmaktadırlar**. Fakat değişime izin verilen bölümler, Rom içerisinde elektrik ile programlanabilen **Cmos** kısmında tutulur. Cmos'un elektrik gittiğinde verileri hâlâ tutabiliyor olmasının sebebi ise daha doğrusu **flash bellek entegresi** ile aynı olmasına rağmen elektrik olmadığında verilerin

korunmasını Bious'un hemen yanında bulunan **Bios pili** sağlar. Bir bilgisayarın Bios pili çıkartılırsa veya Bios resetleme işlemi yapılırsa Bios ile ilgili **tüm konfigürasyonlar fabrika ayarlarına döndürülmüş olmaktadır**.

Bilgisayarlarımızın saat tarih ayarlarında da bozulmalar meydana geliyor, sürekli farklı bir tarihte bilgisayarı başlatıyorsa **Bios pilinden şüphelenebiliriz**. Bios üzerinde yapılan işlemlerin tekrar eski hâline geldiğini görüyorsanız veya açılış esnasında "**CMOS Checksum Error**" **şeklinde bir hata alıyorsanız da pilin değişme zamanı gelmiş olabilir**.



Şekil 2-9: Bios Entegresi

Bios'a ulaşmak için bilgisayarın başlatıldığı esnada klavyeden **F2** veya **DEL** tuşlarına basılması gerekmektedir. Farklı üreticilerin farklı tuş kombinasyonları ayarladıkları Bios ekranı genellikle **DEL**, **F2** veya **F1** tuşları ile gelmektedir. Bilgisayar açılırken **F8** tuşuna basılması da hangi sürücü ile başlatılması gerektiğini seçebileceğiniz **sürücü listesini** çıkarmaktadır.

Bilgisayarın ilk açılışta **çeşitli kontroller** yaptığını ve bunu **Bios** ile gerçekleştirdiğini söylemiştik bilgisayarın yaptığı bu işleme **POST** denilmektedir. Post **ilk açılış** anlamına gelmektedir. Aslında ilk açılışta yapılan tüm post işlemleridir. *Klavye sağlam mı?* **Sağlam**. *Mouse sağlam mı?* **Sağlam**. Gibi kontroller gerçekleştirilir. Post tam olarak bilgisayara enerji verdiğimizde yani **power tuşuna bastığımızda gerçekleşir**. Kısa bir süre ekranda; işlemci hızı, bellek boyutu, sabit disk kaynağı hakkında bilgiler ekranda belirir. Bios üzerindeki ayarlar ile yine bu ekranın yerine üreticinin logosunun yer aldığı

teknik bilgilerin olmadığı bir ekran da ayarlanabilir. Her iki arayüzde de DEL veya F2 tuşları ile Setup ekranına yani **Bios ekranına** girmek mümkündür. Eğer bu bilgiler yoksa ve daha görsel üreticinin logosunun yer aldığı ekran karşımıza gelmiş ise TAB tuşu ile **Şekil 2-10**'daki gibi geleneksel post ekranına dönülebilir.



Şekil 2-10: Post Ekranı

Bios üzerinde yapılabilecek işlemler aşağıdaki gibidir.

Main: Tarih-saat ayarları, klavye ayarları, sürücülerin önceliği ve BIOS memory bilgilerinin görüntülendiği menüdür. *System Time, System Date, Primary Master, Primary Slave, Keyboard Features, System Memory ve Extended Memory* gibi **alt menüleri** vardır.

Advanced: Daha gelişmiş olan ayarların yapıldığı menüdür. Örneğin açılış sırasındaki ekranın belirlendiği *açılış şekli belirleme ayarları, USB hızı ayarı, USB veri yolu desteği ve dâhilî Touch Pad* gibi ayarlar yapılabilir. **Alt menüleri** *Silent Boot, USB Operation, Remote Power On, Legacy USB Support, Internal Mouse* ve *LCD Panel View Expansion* şeklindedir. Remote Power On Lan üzerinden bilgisayarın açılmasını sağlayan bir özelliktir. **LCD Panel View Expansion** ise LCD monitörler için kullanılan bir görüntü destekleme özelliğidir.

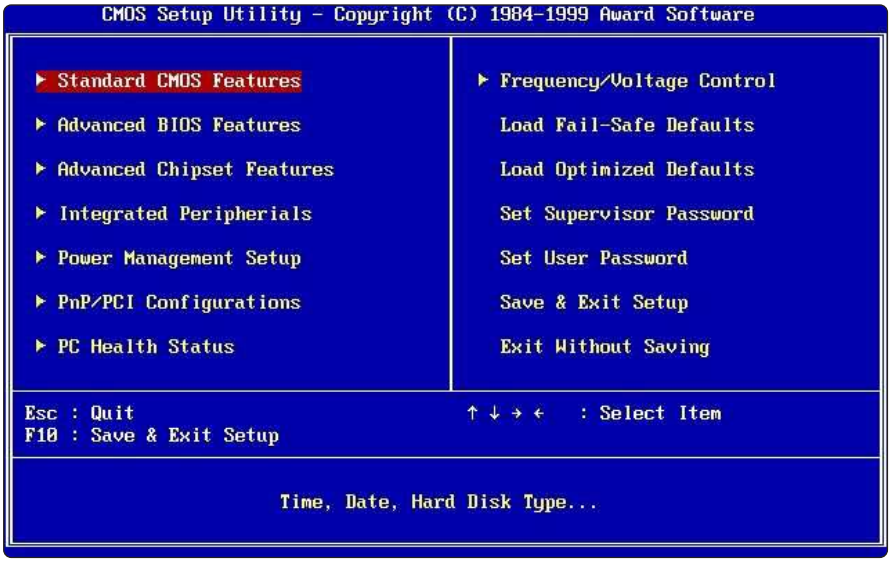
Boot: Sistem kurulumu yapmadan önce bilgisayarın **boot ayarları** yapılır. Yani bilgisayarın hangi ortam ile başlatılacağı belirlenir. Bu ortam **DVD ROM**, **CD ROM** olabileceği gibi Usb bellek veya bilgisayarın kendi **sabit diski** de olabilir. Boot menüsü bilgisayarın açılırken **hangi sürücü ile açılacağını** ve **sürücülerin işletilme sırasının ayarlandığı** menüdür. Birincil olarak sabit disk, ikincil ve üçüncül olarak farklı ortamlar belirlenebilir. Format medyasını (*CD, DVD, USB Bellek*) bilgisayara takıp açılışta bilgisayarın **direkt o ortam ile başlamasını beklemek yanlış olur,** bunun için öncelikle **boot** menüsünden ayar yapılmalıdır. Bu işleme de boot yapmak boot etmek denilmektedir.

Security: Adından da anlaşılacağı üzere **Bios güvenlik ayarları bu menüden gerçekleştirilmektedir**. Güvenlik ayarlarından bir tanesi "**Set Supervisor Password**" menüsüdür ve Bios ayar menüsüne erişim için şifre koymaktır. Diğer bir ayar ise "**Set User Password**" boot menüsündeki menülerin şifrelenmesini sağlar. Bilgisayarın boot menüsü yerine direkt açılışına işletim sisteminden bağımsız olarak şifre konulmak istenirse "**Password On Boot**" menüsü kullanılabilir. Security menüsü altında yapılabilecek son ayar ise "**Fixed Disk Boot Sector**" seçeneğidir. Boot, sektör virüslerine karşı *onarım* ve *koruma* sağlar. Aktifleştirmek için "**Write Protect**" aktif hâle getirilmesi gerekir.

Post işleminin bilgisayarın açılırken gerçekleşen bir dizi donanım kontrol testi olduğundan bahsetmiştik. **Power On Self Test,** yani Post gerçekleşirken bilgisayar klavyeden RAM'e kadar tüm birimlere test sinyali yollar ve **çalışır olup olmadıklarını kontrol ede**r. Eğer birimlerin herhangi birinde hata varsa o hata **özel bip** mesajı verir. Bu sesi hoparlörlerden değil anakart üzerindeki küçük diyafondan verir ve **her bir bip sesisin özel bir anlamı vardır**. Örneğin klavyesi takılı olmayan bir bilgisayarı açtığınızda başlangıçta gelecek olan bip sesinin neye ait olduğunu bilmiyorsanız ana kart kitapçığından faydalanabileceğiniz gibi PCI slotuna takabileceğiniz bir **Post Card** yardımıyla da hem **hatanın kodunu** hem de **ne anlama geldiğini** basitçe öğrenmiş olursunuz. Düzgün çalan bir bilgisayarda **kısa bir bip sesinden sonra** sistem başlatılır. Bilgisayarın güç düğmesine bastığınızda kısa bir süre içerisinde klavyedeki ışıkların üçünün de yanıp söndüğünü görürsünüz. Bu, post işleminin klavyeyi test etmesinden dolayıdır.

CMOS Setup

Bios üzerinde **değişiklik yapabildiğimiz** alandır. CMOS üzerinde çok fazla işlem yapılmaktadır bu işlemler sırasıyla aşağıda anlatılmıştır.



Şekil 2-11: CMOS Setup Ekranı

Standart CMOS Features: En temel işlemlerin gerçekleştirildiği bölümdür. Sistem ayarları, sistem saati, monitör tipi, sabit disklere ait sürücüler ile ilgili işlemler yapılır. Ayrıca **Halt-On** olarak bilinen seçenek ile bilgisayarın post esnasında karşılaştığı bir hatada ne tepki vereceğini ayarlayabilirsiniz. "**No Errors POST**" post sırasında karşılaşılan hiçbir hatada durmayıp devam etmesini sağlar. "**All Errors BIOS**" hata bulunduğunda durur ve hatanın düzeltilmesi için uyarı verir. "**All, But Keyboard POST**" klavye hataları dışındaki tüm hatalarda durur. "**All, But Diskette POST**" disket sürücüsünde meydana gelecek hatalar dışındaki tüm hatalarda durur. "**All, But Disk/Key POST**" bu ayar ise klavye ve disk sürücülerindeki hatalar dışındaki tüm hatalarda açılış işleminin durdurulmasını sağlar.

Advanced BIOS Features: Sistem performansının artırılması için "**Cpu Internal Cache**", "**CPUR Fast String**", Bios güncelleme modülünün işlemciye gerekli bilgilerin iletilmesi için "**Bios Update**", sistemin hangi sürücüden başlatılacağı ile ilgili "**Boot Sequence**", boot virüslerini önlemek için "**Virus Warning**", post işleminin 3 veya 4 kez yapılması yerine yalnızca 1 kez yapılması için "**Quick Power On Self Test**" gibi ayarlar yapılabilir.

Advanced Chipset Features: Anakart ve chipset üzerindeki birimlere ait özelliklerin değiştirildiği ayar menüsüdür. *İşlemci sıcaklığı, bellek erişim zamanları* gibi ayarlar yapılabilmektedir. Donanımın ne şekilde tepki vereceğini bilmeden yapılacak olan ayarlar sisteme ciddi zararlar verecektir o yüzden **varsayılan ayarlarda** kalması tavsiye edilmektedir.

Integrated Peripherals: Onboard kartların ayarlarının yapıldığı ayar menüsüdür.

Power Management Setup: Tamamen **güç seçenekleri** ile ilgili ayar menüsüdür. Güç tasarrufu aktif edilebilir, devre dışı bırakılabilir veya hemen devreye girmemesi sağlanabilir.

PNP/PCI Configurations: PCI veri yoluna çok çeşitli aygıtlar bağlayabildiğimizden bahsetmiştik. PCI veri yoluna takılan donanımlar **Tak-Çalıştır** desteklerler. Bu ayar menüsü ise Tak-Çalıştır özelliğini destekleyen aygıtlar ile ilgili ayarlar yapılmaktadır.

Load BIOS Defaults: Bios ile ilgili tüm ayarların varsayılan döndürülmesini sağlar. Tüm performans ayarlarını kapatır.

Load Setup Defaults: Bios'un yine varsayılana dönmesini sağlar fakat üretici tarafından optimum seviyeye göre ayarlanmış bir versiyona geri döner.

HDD Low Level Format: Disk üzerindeki bozuk sektörlerin kurtarılması için kullanılan bir nevi **bad sector kurtarma** ve **format aracıdır** diyebiliriz.

Save & Exit Setup: Bios üzerinde yapmış olunan tüm değişikliklerin kaydedilip Bios menüsünden çıkılmasını sağlar. Kısa yolu **F10**'dur.

Exit Without Saving: Bios üzerindeki değişikliklerin uygulanmadan yani **kaydedilmeden çıkış yapılmasını sağlar**.

Sesli Hata Mesajları

HATA ŞEKLİ	AÇIKLAMASI
Sürekli Beep Sesi	Güç Kaynağı Arızası
Birden Fazla Kısa Beep Sesi	Anakart Arızası
1 Uzun Beep Sesi	Bellek Tazeleme Ünitesi Hatası
1 Uzun 1 Kısa Beep Sesi	Anakart ya da Beep Çipi Arızası
1 Uzun 2 Kısa Beep Sesi	Ekran Kartı Arızası
1 Uzun 3 Kısa Beep Sesi	Ekran Kartı Arızası
2 Uzun 1 Kısa Beep Sesi	RAMDAC Kaynaklı Ekran Kartı Arızası
2 Kısa Beep Sesi	Bellek Parity Hatası
3 Kısa Beep Sesi	Bellek ilk 64K Hatası
4 Kısa Beep Sesi	Timer Hatası
5 Kısa Beep Sesi	İşlemci Hatası
6 Kısa Beep Sesi	Klavye Hatası
7 Kısa Beep Sesi	İşlemci Hatası
8 Kısa Beep Sesi	Ekran Kartında Okuma/Yazma Hatası
9 Kısa Beep Sesi	BIOS ROM Hatası
10 Kısa Beep Sesi	CMOS Hatası
11 Kısa Beep Sesi	Tampon Bellek Hatası

Yazılı Hata Mesajları

HATA ŞEKLİ	AÇIKLAMASI
Cmos Checksum Error	Muhtemel Bios Pili veya Bios Yongası arızasını bildirir.
Cmos Battery Failed	Bios pilinin bittiğini bildirir.
Memory Test Fail	Bios'ta yer alan bellek ayarlarında veya belleğin modüllerindeki arızayı bildirir.
Hard Disk Fail	Bilgisayarda bağlı olan Hard Disk ile Bios üzerinden ayarlanan Hard Disk ayarlarının aynı olmadığını bildirir.
Floppy Disk Fail	Bilgisayarda bağlı olan Floppy Disk ile Bios üzerinden ayarlanan Floppy Disk ayarlarının aynı olmadığını bildirir.
Keyboard Error or No Keyboard Present	Klavye algılanmadığı durumlar olabilir. F1 ile yeni bir klavye takılmak üzere devam edilmesi istenir.

YAZILIM

3

Önceki bölümde bilgisayarın donanım kısmı ile tanışıp **donanım birimleri** ve bunların **Bios üzerinden yönetimi** gibi konulara değinmiştik. Belki bildiğiniz konuların üzerinden hatırlatma yapmış belki de bilmediğiniz konuları beraber incelemiştik. Bilgisayar tanımında **yazılım** ve **donanım** diye iki ayrı bölümde incelendiğinden bahsetmiştik. Bu bölümde, yazılım kelimesinin bize ne çağrışımda bulunması gerektiğine ve yazılım çeşitlerine değineceğiz. Bu bölümde **programlamaya giriş yapmayacağımız için** *kategorileri* ve bir *mühendisin kültür olarak bilmesi gereken temel bilgileri* vermeye çalışacağım.

Yazılım Nedir?

Kitaplarda, internet üzerinde pek çok tanım bulunmakta ve güvenilir yerlerdeki tanımların hepsi de geçerli ve doğrudur fakat biz akılda kalıcı ve bize ifade etmesi gereken şeyleri bir tanımda birleştirecek olursak bana göre yazılım; *bilgisayarın donanım birimlerinde gerçekleştirilen işleri kullanıcıya çeşitli arayüz ortamlarında sunan ve kullanıcıdan gerektiğinde veri girişi alıp bu verileri de tekrar çeşitli kurallara ve kriterlere göre işleyip bir sonuca varan komutlar bütününe yazılım* denir. **Programlama ise** *bir problemin çözümü için kullanılan matematiksel ifadelerdir*. Programlama ile çözülen problemler, yazılım çözümlerinde kullanıcıya sunulurlar bu yüzden **farklı platformlardaki işleri yapan farklı kategorilerde yazılımlar** bulunmaktadır.

Yazılım; kısacası bir dizi kodlanmış komuttan oluşturmaktadır. Yazılım geliştirme çok yeni bir süreç olmayıp 1960'lı yıllardan beri hızla gelişmiştir.

Bilgisayarın anladığı dil, yani makine dili ile geliştirilen yazılımlar günümüzde görsel programlamaya ve **hatta nesne tabanlı programlama** seviyesine kadar gelişmiştir.

Farklı platformlardaki çeşitli işlere göre geliştirilen yazılımlar *masaüstü, web, cep telefonları ve cep bilgisayarları, ATM, bulaşık makineleri ve otomobillerde* yerini almıştır. Masaüstünde çalışan yazılımlar, herkesin sıklıkla kullandığı *Media Player, Office Word, Muhasebe Programı* gibi yazılımlardır. Web üzerinde çalışan yazılımlar ise *web siteleri, web servisler ve web tabanlı* yazılımlardır. Akıllı telefonlarımızda çalışan yazılımlar başta **işletim sistemleri** ve **sosyal medya uygulamaları gibi uygulama marketlerden temin edilen çeşitli yazılımlardır**. ATM cihazlarında, aslında bir tür web/masaüstü karışımı **hibrit bir sistem** bulunur ve genellikle **distributed** (*dağıtık*) çalışabilen yazılımlardır. Evlerimizdeki tek programlı veya üç programlı diye ifade ettiğimiz bulaşık makinesi veya diğer elektronik eşyalarda da çalışabilmesi için yazılım vardır, sanayilerde kullanılan otomasyon ile çalışan makinelerde de otomobillerin beyinlerinde (*ecu*) yazılım vardır. **Kısacası yazılım çevremizdeki hemen her şeyde bulunmaktadır**. Kolumuzdaki saate kadar her şeyin içerisinde micro işlemcilerin yer edinmesi bir taraftan da yazılımların yer edinmesi anlamında gelmiştir. Giyilebilir teknoloji ürünlerde de **gömülü sistem (embeded)** yazılımlara rastlayabiliriz.

Yazılım Mühendisliği Nedir?

Bilgisayar programcılığı gibi anlaşılan **yazılım mühendisliğinin** ne olduğunu hakkını vererek anlatalım. Yazılım mühendisliği, sektörde eğitimli bilgisayar programcılığı gibi algılanmakta olup **tamamen yanlış bir düşüncedir**. Çünkü eğitimli bilgisayar programcılarını Yazılım Mühendisliği bölümü değil **Bilgisayar Programcılığı ön lisans bölümleri** yetiştirmektedir. Her iki bölümün de vermiş olduğu eğitim şüphesiz ki kendi çerçevelerinde gayet yeterlidir. Fakat yazılım mühendisliği bir **mühendis disiplinidir** ve **çalışmaları baştan aşağı bilim** ve **matematiğe** dayanmaktadır.

Yazılım Mühendisliği; yazılım geliştirme döngüsünde diğer mühendislik disiplinlerinin *sistem, düzen, ölçülebilir, gerçekleştirilebilir ve optimize edilebilir* yaklaşımlarını miras alarak yazılımın üretiminde ve bakımında uygulanan bir disiplindir. En nihayetinde yazılım geliştirme işinin de mühendisliğe uyarlanmasıdır.

Çekirdekten yetişen, herhangi bir eğitim almadan klavye başına geçince harikalar yaratan programlar olduğunu biliyor ve onlarla da ülkemize beşeri katkıda bulundukları için gurur duyuyoruz. Mühendislik eğitimimden önce ön lisans da okuyan birisi olarak, yazılım geliştirme işi kurallarını bilen bir geliştirici için **normal bir iş süreci gibi** geldiğini söyleyebilirim fakat büyük yazılımların geliştirilmesi ve savunma sanayi gibi sistemlere entegre edilmesi **kesinlikle mühendislik eğitimi** gerektirmektedir.

Bilgisayar mühendisliği ders müfredatlarında da yazılım mühendisliğine çok yakın konular verilmektedir. Programlama *dili yapısı, veri yapıları, otomatlar, object oriented yaklaşımı* çeşitli programlama dilleri ile ilgili dersler her üniversitede okutulmaktadır. Fakat iş doğrudan yazılım mühendisliğine geldiğinde iş sistemi tanımaktan ve analiz etmekten başlar ve sistemin kullanım sürecindeki yaşam döngüsüne kadar uzanan süreçleri **sistematik olarak pratiğe geçirmeye** hedefler. Bu sistematik çalışmada yine matematik ve bilimi kullanır.

İşletim Sistemi Yazılımları

İşletim sistemleri başlığında aslında işletim sistemlerini **lisanslı** ve **GPL lisanslı** olarak örneklendirmiştik. İşletim sistemleri de çalışan diğer yazılımlar gibi bir yazılımdır. Farkı **sistem yazılımı olması** ve **doğrudan donanım ile haberleşiyor olmasıdır**. Microsoft Windows, Apple IOS, LINUX sürümleri masaüstü işletim sistemi yazılımlarıdır. Mobile işletim sistemlerinde ise yine Microsoft Windows Phone, Apple IOS, Açık kaynak Android Sürümleri işletim sistemi yazılımlarına örnektir.

İşletim sistemi yazılımı denilince sadece **güçlü bir arayüzden oluşan kullanıcı dostu işletim sistemleri akla gelmemelidir**. Otomasyonlarda kullanılan veya panolarda kullanılan işletim sistemleri de **işletim sistemi yazılımlarına örnek** olarak gösterilebilir.

Paket Yazılımlar

Paket yazılımların, sektörel veya masaüstü yazılımlarından ayrı tutulmasının nedeni bu yazılımların **bir bütünlük içermesi** ve **genellikle kurulumlarında birden fazla paketin art arda kurulmasıdır**. En güzel paket program örneği, **Microsoft Office** yazılımıdır. İçerisinde *Word, Excel, Access, PowerPoint, One Note, Planner, Delve, Todo* gibi programları bulundurur. Microsoft Office dağıtımına, açık kaynak işletim sistemlerinde **Open Office** paket yazılımı karşılık gelmektedir.

Sektörel ve Ticari Yazılımlar

Programlamanın ülkemizde popüler olduğu yıllarda öne çıkmış yazılım firmalarını ve programlarının isimlerini şu an anımsıyoruzdur. Yerli yazılım firmalarımız *muhasebe, kasa, stok, e-ticaret, dijital arşiv, elektronik belge yönetim sistemi, öğrenci bilgi sistemi, şehir içi toplu ulaşım e-bilet yazılımları, turnike yazılımları, banka yazılımları* gibi pek çok konuda sektörde **ihtiyaç duyulan yazılımları geliştirmişler** ve bugün **birçoğu hatırı sayılır firmalar hâline gelmişlerdir**.

Sektörel yazılımlar, **bir sektöre hizmet eden yazılımlardır** ve bir yazılım **birden fazla firma tarafından farklı farklı geliştirilmiş olabilir**. Özellikle muhasebe yazılımları çok güçlü birden fazla firma tarafından geliştirilmiş ve hemen hepsi de pazarda müşteri bulmuşlardır. Örneğin otellerin kullandıkları yazılımlar **sektörel bir yazılımdır** ve otelin müşteri kayıtlarından depo ihtiyaçlarına kadar pek çok başlığı kapsamaktadır. Bu tür programlar ücretli ve lisanslı olarak satıldığı için **tamamen ticari yazılımlardır**. Ancak eğitim amacıyla kullanılabilecek yerlerde ücretsiz sürümleri varsa temin edilebilir.

Ne yazık ki ülkemizde geliştirilmeyen ve tüm dünyanın kullandığı başka ticari yazılımlar da vardır. Bunlara en iyi örnek **Adobe** firmasının **Master Collection** veya **CC** paketindeki **Photoshop, Dreamweaver, Fireworks, Indesing** gibi yazılımlardır. Yine benzer şekilde çizim programlarından **Autocad, Solidworks** yazılımları da sektörel ve ticari yazılımlardır.

Gömülü Sistem Yazılımları

Üretim sektöründe robotların ve otomasyon sistemlerinin önemli yer aldığı günümüzde; robot kollar, PLC sistemler, kuyu otomasyonları, sera otomasyonları, enjeksiyon makineleri, otomobiller, LPG beyinleri, evimizdeki buzdolabı veya televizyon, petrol pompa otomasyon sistemlerinde donanım içerisinde yer alan yazılımlardır. Mühendislik eğitiminde micro işlemciler dersinde **Pic16F84 devresi** içerisinde yazmış olduğunuz LED yakma otomasyonu da bir **gömülü sistem** yazılımıdır çünkü bir ROM içerisine veya farklı türdeki devreler içerisine **kodlanmış** (*gömülmüş*) **sistemlerdir**.

Gömülü sistem yazılımlar, normal yazılımlardan farklı olarak **bir işlemi yerine getirmek için** kullanılırlar. Evimizdeki elektronik eşyalarda da bir işlemi yerine getirmek söz konusudur. Örneğin akıllı bir fırında seçilen yemek çeşidine göre kaç dakika, kaç derece de pişirme işleminin gerçekleştirileceği komutlar sayesinde gerçekleştirilir. Bir işin yapılabilmesi için **donanım**, **elektronik** ve **yazılımın birleştiği sistemlere kısaca gömülü** (*embedded*) **sistemler** denilmektedir. İnternetin de yaygınlaşması ile birlikte gömülü sistemlere yeni bir boyut kazandırılarak gömülü sistem olan bir aygıta **uzaktan erişilmesi cep telefonu uygulaması ile kontrol edilebilmesi sağlanmıştır**. Bir kombiyi kendi üzerindeki butonlar ile çalıştırmak belirli dereceye geldiğinde kapanmasını sağlamak **gömülü sistem** ile mümkündür **fakat kombiyi cep telefonu ile kontrol edebilmek IOT yaklaşımını getirmiştir**. **Endüstri 4.0**'da da çokça kendinden bahsettiren IOT **Internet of Things** yani **nesnelerin interneti** anlamında gelmektedir. IOT yaklaşımında içerisinde işlemci ve işletim sistemi bulunan her nesneyi internete kavuşturup çeşitli işlemler yapılabilmektedir.

Web Yazılımları

Masaüstü yazılımlarını web yazılımlarından ayıran unsur, **geliştirme şekli** ve **erişildiği konum**dur. Masaüstü yazılımları bilgisayarlara yüklenmek zorundayken web yazılımları internet sunucularında çalışmaktadır.

Web siteleri veya web yazılımları web sunucularının üzerinde çalıştığı için kullanıcılar bu web sitelerine veya web yazılımlarına **alan adları** vasıtasıyla erişmektedir. İntranet gibi iç ağda çalışan web yazılımlarına **IP adresi** ile giriş yapılabilir. Zaten alan adı da IP adresine karşılık gelerek **Domain Name System** (**DNS**) tarafından çözümlenmektedir.

Bir web yazılımı geliştirme ortamı ve programlama dili olarak **PHP, ASP, ASP.NET, JSP, ASP.NET MVC** veya **ASP.NET CORE** ile geliştirebilir. Url Routing yapılmamış ise php ile geliştirilen web yazılımlarını url sonundaki **.php** uzantısından, asp ile geliştirilen web yazılımlarını **.asp** ve asp.net ile geliştirilen web yazılımlarını url sonundaki **.aspx** uzantısından anlayabiliriz.

Web yazılımlarının hızlı ilerlemesindeki en büyük etken çalışacağı bilgisayara **kurulum yapılmasına gerek olmaması** ve **internet olan her yerden erişimi mümkün olmasıdır**. Klasik masaüstü programları ile yapılabilen işlemler web ortamında da geliştirilebilir, bu yüzden web yazılımları kullanıcının *bilgisayarının çökmesi, bozulması, çalınması* gibi durumlara karşı **daha kullanışlı olmakta** ve çoğu yazılım artık web ortamında geliştirilmektedir.

Web Servis Yazılımları

Farklı programlama dilleri ile geliştirilen yazılımlar, kendilerine ait olan teknolojideki bir sunucuda barınırlar. Örneğin **asp.net** ile kodlanan bir web sitesi **.net framework** kütüphanesine ihtiyaç duyar ve Microsoft sunucusunda **IIS** ile çalışır. Benzer şekilde php ile kodlanan bir yazılım Linux sunucuda **Apachi** üzerinde çalışır. Bu farklılık kendi içlerinde **avantaj** ve **dezavantajlara** sahiptir fakat iki teknoloji arasında iletişim söz konusu olduğunda imdada **ortak bir veri iletişim dili kullanan (XML) web servisler yetişmektedir**.

Bir web servis ile A bankasının Microsoft sunucularındaki veri devletin Linux sunucuna web servisler ile gönderilebilir ve üzerinde işlemler yapılabilir. E-devlet uygulamasındaki yüzlerce servis, **farklı kurumlardan veri çekmektedir**. Örnek olsun diye bir senaryo çizecek olursak örneğin tapu kadastro veri tabanı **MySql** olsun ve kullandıkları geliştirme dili ile platform **php Linux** olsun. E-devlet ise **Java** ile geliştirilmiş olsun ve veri tabanı olarak **Oracle** kullanıyor olsun. Vatandaş olarak e-devlete girip üzerimizdeki tapu kayıtlarını sorgulamak istediğimizde biz farkında olmadan web servis gidip farklı bir teknolojideki veriyi sorgulayıp bize sunacaktır. Web servis mantığının ortak olması **açık standartlar** ile geliştirilmesinden dolayıdır.

Web servisler temel olarak **XML** ve **http** kullanırlar ve aşağıdaki iletişim protokollerini standart olarak kabul ederek çalışırlar.

- **SOAP (Simple Object Access Protocol)**: Basit nesne erişim protokolüdür. Küçük miktarda bilgileri aktaran protokoldür. SOAP http üzerinden **XML formatında** aktarım yapar. Google, çoğu servisinde **SOAP** kullanır.
- **REST (Representational State Transfer)**: Temsili durum transferidir. İstemci sunucu arasındaki **iletişimin en basit hâli**dir. REST ile veri alışverişi yaparken SOAP gibi XML kullanmayı zorunlu kılmaz standart bir tanımlamaya ihtiyacı olmadığı için JSON, XML veya Text formatında bile iletişim kurulabilir. REST mimariside **http** protokolünü kullanmaktadır. Amazon'un **REST** kullandığını söyleyebiliriz.
- **WSDL (Web Services Description Language)**: XML tabanlı web servislerini ve yerini açıklamak, belirlemek için tanımlanmış bir iletişim kanalıdır. **W3C** standardındadır.

Web servis yazılması veya kullanılması ile ilgili ayrı bir bölümümüz olduğu için bu kadar teknik detayı yeterli buluyorum ve fikir edinmesi için bu bölümde geçen standartları ve protokolleri ilgili bölüme geçene dek *incelemenizi tavsiye ediyorum*.

İŞLETİM SİSTEMLERİ DETAYLARI

4

İşletim sistemleri bölümünde **temel bilgileri** edinmiştik. Bu bölümde hem Windows işletim sistemine ait detaylı bilgilere hem de Windows sunucu işletim sistemlerine değineceğiz.

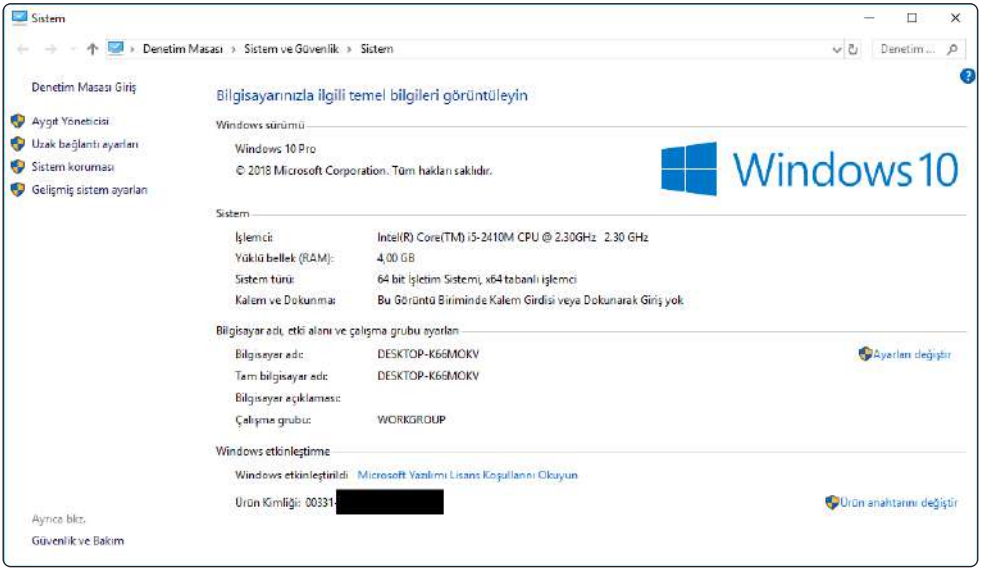
Sistem Özellikleri

Windows işletim sistemlerinde eski adıyla "**bilgisayarım**" yeni adıyla ise "**bu bilgisayar**" masaüstü simgesine **sağ tıkladıktan** sonra **özellikler** menüsü ile eriştiğimiz diyalog kutularından birtakım sistem özelliklerine erişmekteyiz. **Windows XP** işletim sistemlerinde özellikler menüsüne tıklanınca doğrudan **Şekil 4-1**'de olduğu gibi **sistem özellikleri penceresi** gelmekteydi. Fakat **Windows 7** ve sonrasındaki işletim sistemlerinde bu pencere biraz daha **farklı** gelmektedir.



Şekil 4-1: Windows XP Sistem Özellikleri

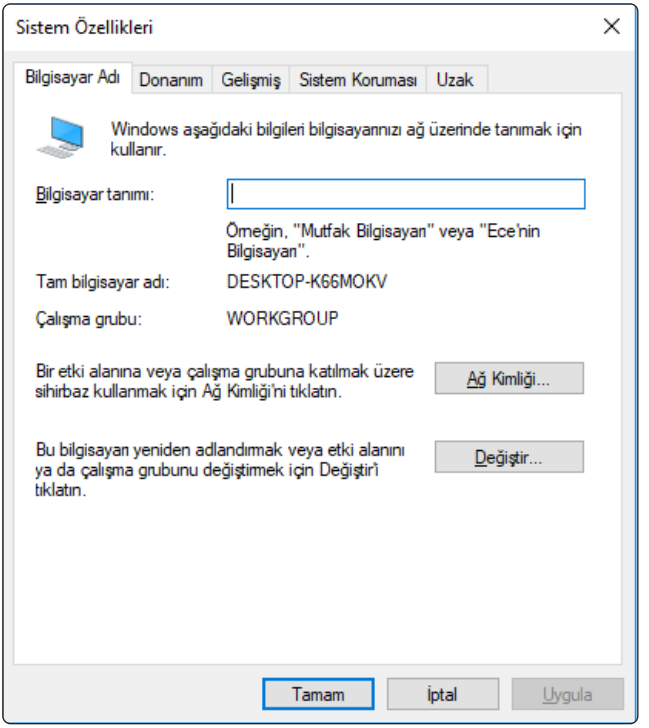
Sistem özellikleri (*system properties*) penceresinden çeşitli ayarlar yapılabilmektedir. Ayrıca bilgisayar hakkında özet bilgi edinebildiğimiz bu pencere bilgisayarın işlemci özelliklerini ve **RAM ile ilgili bilgiler de vermektedir**. Windows 10 işletim sistemi üzerinden bu pencereye erişmek istediğimizde yine "**Bu Bilgisayar**" masaüstü simgesine **sağ tıkladıktan sonra** özellikler menüsüne tıklayıp **Şekil 4-2**'deki gibi bir diyalog penceresi bizi karşılamaktadır.



Şekil 4-2: Windows 10 Sistem Özellikleri Genel Penceresi

Bu görselde yer alan diyalog penceresinden **bilgisayar adı**, **etki alanı** ve **çalışma grubu** ayarları yapılabilir. Sistem ile ilgili bilgiler görüntülenebilir eğer **bilgisayar** adında veya **etki grubunda** değişiklik yapılmak istenirse "**Ayarları Değiştir**" link butonuna tıklanabilir. Bu butona tıklayınca **XP** versiyonundaki gibi daha küçük bir pencere bizi karşılar. **Şekil 4-3**'te Windows 10 işletim sisteminde yer alan sistem özellikleri penceresindeki **Bilgisayar** adı kısmındaki "**Ayarları Değiştir**" butonuna tıklayınca açılan diyalog penceresi yer almaktadır. Bu pencere "**Bilgisayar Adı**" sekmesinde açılır. Genel menü ise *işletim sisteminin versiyonu, kullanıcı ile ilgili bilgiler, işlemcinin özellikleri ve geçici bellek (RAM)* ile ilgili bilgileri vermektedir. Windows 10 işletim sistemindeki pencere (**Şekil 4-3**) biraz daha özet bir görünümde "**Genel**" sekmesi sunmaktadır.

Bilgisayar Adı Sekmesi

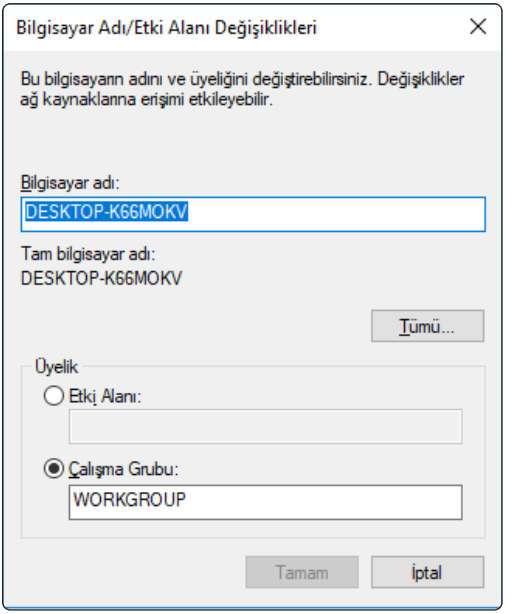


Şekil 4-3: Windows 10 Sistem Özellikleri Diyalog Kutusu

"**Bilgisayar Adı**" sekmesi, yukarıdaki şekilde de görüleceği gibi bilgisayarın adının görüntülendiği ve aynı zamanda bilgisayara yeni bir ad verilmesi durumunda kullanılacak arayüzdür. **Bilgisayar adı oldukça önemli bir ayardır**. Bilgisayarlarımız ağda iletişim kurarken ve oturum yönetiminde **bir isme zorunlu olarak ihtiyaç duymaktadırlar**. "**Bilgisayar Tanımı**" ise bilgisayarlarımıza ait **açıklama alanıdır**. Bu açıklamanın **kısa olması** önemlidir.

İPUCU: Bilgisayar üzerinde çalıştığınız bir uygulamanın veya anlık bir çalışmanın ekran görüntüsünü almak için klavyede bulunan "**Print Screen**" tuşuna basmak gerekir. Print Screen tuşu **ekran alıntısı**, yani **Screenshot** yapacaktır ve herhangi bir metin, resim düzenleme editörüne yapıştır (**CTRL+V**) ile kullanılabilir. **En kolay düzenleme** Windows işletim sistemi ile beraber gelen "**Paint**" uygulaması üzerinden gerçekleştirilebilir. Peki, **Şekil 4-3**'te olduğu gibi alt tarafta "**Sistem Özellikleri**" penceresi hemen üstünde ise "**Bilgisayar Adı**" diyalog kutusu açıkken sadece en üstte çalışan uygulamanın *ekran görüntüsünü nasıl alacağız?* **Print Screen** tuşuna bastığımızda, ekranda görüntülenen ne kadar obje varsa alacaktır. Sadece en üstteki çalışan objeyi almak istediğimizde "**ALT+Print Screen**" tuş kombinasyonunu kullanmalıyız.

Bilgisayarı bir çalışma grubuna üye yapacağımız zaman "**Bilgisayarım veya Bu Bilgisayar**" masaüstü simgesine sağ tıklayıp özellikler dedikten sonra **bilgisayar adı ayarlarını değiştir** butonu ile **Şekil 4-3**'teki pencereye ulaşabileceğimiz gibi "**Denetim Masası**" üzerinde bulunan "**Sistem**" menüsünde de aynı pencereye ulaşmak mümkündür. Çalışma grubuna eklemek için sistem penceresi yukarıda olduğu gibi "**Bilgisayar Adı**" sekmesindeyken "**Değiştir**" alanına tıklamak gerekir.



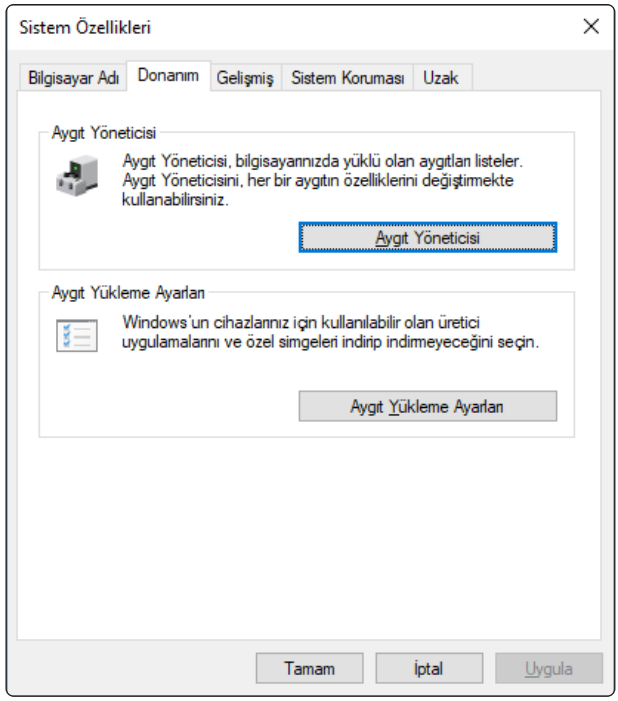
Şekil 4-4: Çalışma Grubuna Dâhil Olma

Şekil 4-4'deki penceredeyken "**Üyelik**" paneli içerisinde bulunan "**Çalışma Grubu**" alanı seçilir ve katılmak istenilen çalışma grubunun adı yazılarak aktifleşen "**Tamam**" butonu ile işlem tamamlanır. Bilgisayarımız yeni girmek istediğimiz çalışma alanına üye olmadan önce farklı bir etki alanına üye olmuş ise o etki alanından **otomatik** olarak çıkartılarak **bilgisayar hesabı devre dışı kalır**. Çalışma grubu etki alanı değil de **Active Directory** gibi ağda bulunan **bir etki alanına üye olunması gerektiğinde** denetim masasından yine sistem menüsüne gelerek yani **Şekil 4-3**'te yer alan penceredeki "**Ağ Kimliği**" alanına tıklayarak ağ kimliği sihirbazı ile ilgili etki alanına giriş yapılabilir. Bu şekilde bir etki alanına dâhil olma işleminde **bilgisayar adının 15 karakteri aşmaması hatta tam 15 karakter olması önerilir**. Çoğu tanımlamada olduğu gibi yine Türkçe karakter (*ü, ğ, ş, ç, ö, İ gibi*) ve ":;*/-+=>|<? gibi karakterleri içeremez.

ARAŞTIRMA: Netbios ve Fqdn terimlerini araştırınız.

Donanım Sekmesi

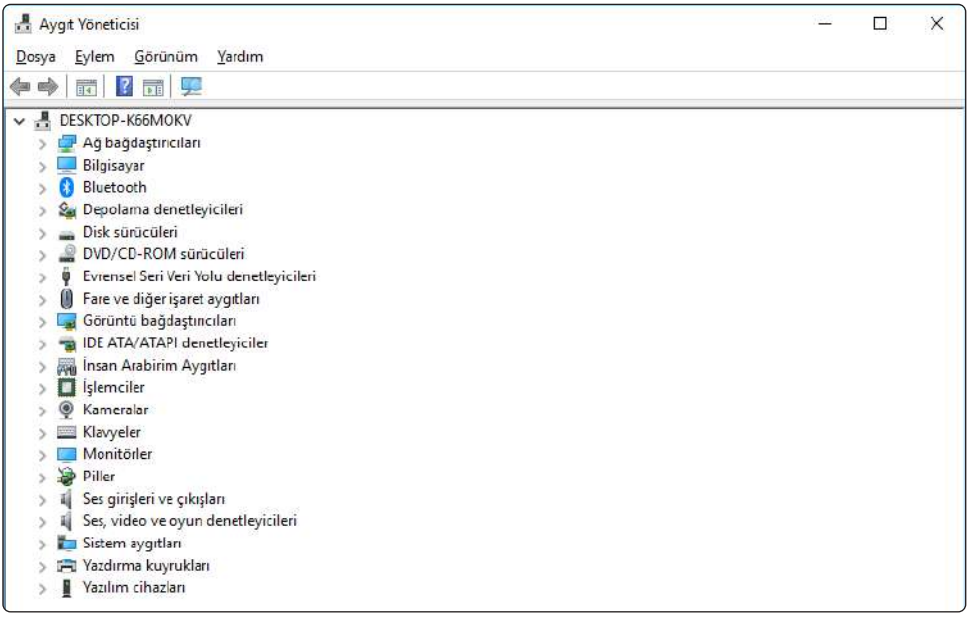
Bilgisayara **bir donanım eklendiğinde** bilgisayarı bu **durumdan haberdar etmek** ve **işletim sisteminin yapması gereken işlemlerin ayarlandığı sekmedir**. Günümüz sistemleri **tak-çalıştır** uyumluluğu yüksek sistemlerdir.



Şekil 4-5: Donanım Sekmesi Diyalog Penceresi

Şimdi, biraz geriye gittiğimizde özellikle **Windows XP** işletim sistemini kurduktan hemen sonra sistemin ihtiyacı olan **temel sürücüleri** yüklemek için **bu menüyü ziyaret ettiğimizi hatırlayabiliriz**. Bu menüyü daha önce kullanmamış olan okurlarımız için **daha önce çok kullanıldığını** özellikle belirtmek istedim. Donanımların işletim sistemi tarafından tanınmadığı durumlarda **Şekil 4-5**'teki "**Aygıt Yönetici**" menüsü oldukça önemli bir işleve sahiptir. **Bir donanımı kaldırmak** veya **ayarlarını değiştirmek** için bu menü kullanılmaktadır. Aygıt yönetici menüsünden "**donanım değişikliklerini tara**" ile tüm değişiklikler de **refresh edilmiş** olur. Eğer tarama sonrası bulunan eksik donanım sürücülerini tamamlama işlemini başaramaz ise **manuel olarak**

sürücüyü şu konumdan yükle gibi seçenekler ile kendimiz de tanıtabiliriz. Bu işlem, bir sürücü yazılımının arayüzünden yapılacak **install işlemi ile aynıdır** diyebiliriz. Aygıt yöneticisine tıkladığımızda açılan donanımlar listesinde seçeneklerin herhangi birinin önünde "**?**" varsa **bu o donanım ile ilgili sorun olduğu anlamında gelmektedir**. Genellikle bu sürücü, **bilgisayara hiç yüklenmediği durumlarda soru işareti çıkmaktadır**. Ama bazı durumlarda gerekli ayarların doğru konfigüre edilmemesi, farklı versiyon sürücü kurulması veya başka bir sürücü ile çakışması gibi durumlardan ötürü de soru işareti çıkmaktadır. Soru işareti olan seçeneğe **çift tıklayıp** sürücünün kaldırılması veya farklı bir kaynaktan yeniden yüklenmesi sağlanabilir.



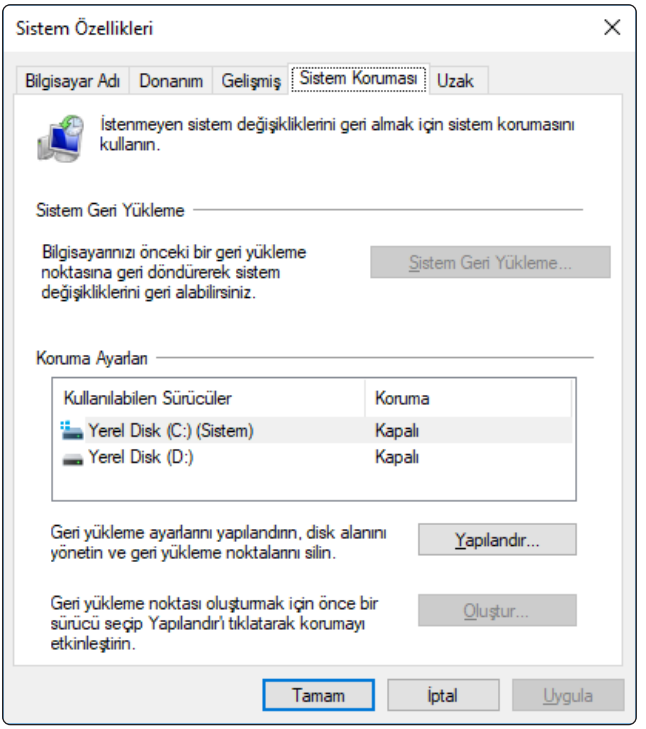
Şekil 4-6: Aygıt Yöneticisi

Şekil 4-6'da yer alan aygıt yöneticisi penceresindeki seçeneklerden herhangi birine **çift tıklanıldığı zaman** seçilen donanım ile ilgili ayarların yapılmasına izin verilir. "**Genel**" sekmesinde donanımla ilgili bilgiler yer almaktadır ve donanım **çalışma durumu ile ilgili** (*status*) **bilgi vermektedir**. Aynı penceredeki "**Sürücü**" sekmesinde programın kullandığı **sürücü yazılımının** ayarları yer alır. Bilgisayar üzerindeki tüm donanımlar (*tak-çalıştır olsa bile*) kendilerini tanıtan birer yazılıma ihtiyaç duyarlar.

Sistem kurulumu yaptıktan sonra (*formattan sonra*), gerekli sürücülerin yüklenmesi (*driver*) derken **bu sürücü yazılımlarından** bahsedilmektedir. Özellikle **tak-çalıştır** özelliği olan **donanımlar** ve **standart donanımlar** işletim sistemi ile birlikte gelmektedir. Bu yüzden Windows çoğu donanım için sürücüleri direkt tanımaktadır. Windows'un bir sürücüyü tanıyamadığı durumlarda ise çeşitli kaynaklardan sürücü **manuel** olarak yüklenebilir.

Sistem Geri Yükleme Sekmesi

İşletim sistemleri bölümünde sistem geri yüklemeden bahsetmiştik. **Sistem geri yükleme adımı** bilgisayarımızda bir sorun çıktığında **verilerimizi kaybetmeden** bilgisayarın önceki bir durumunun geri yüklenmesi işlemidir. Bilgisayarlarımız çalışma zamanında değişiklikleri algıladıkça **geri yükleme noktaları oluşturur** ve bu geri yükleme noktaları sayesinde bilgisayarlarımızı **daha önceki bir zamana geri alabiliriz**.

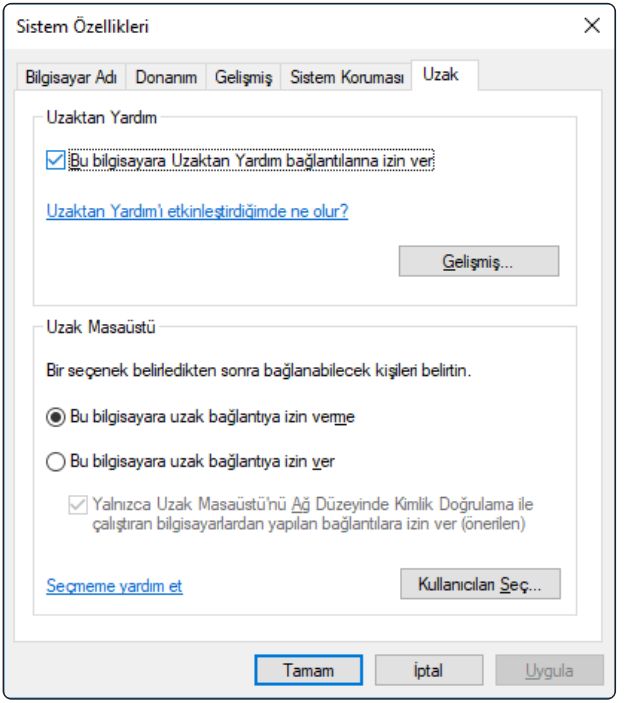


Şekil 4-7: Sistem Geri Yükleme

Uzak Masaüstü Sekmesi

Yardımlaşmanın da **dijital bir boyut** kazandığı günümüzde, bir arkadaşımızın bilgisayarına uzaktan bağlanıp onun masaüstüne olduğu gibi erişip kendi bilgisayarımızdan arkadaşımınız bilgisayarını kullanabiliriz. İş dünyasında uzaktan bağlantı için geliştirilmiş **profesyonel yazılımlar** bulunmaktadır ve bu yazılımlar aynı zamanda birden fazla katılımcı sağlayarak **konferans özelliği** de sunmaktadır. **TeamViewer**, **AnyDesk**, **Alpemix** gibi uzaktan bağlantı programları bunlara örnektir.

Windows işletim sistemine sahip makine başka bir Windows işletim sistemine sahip bilgisayara başka bir yazılıma ihtiyaç duymadan bağlanabilir. **TCP/IP** iletişim protokolü kullanan iki bilgisayar **Uzak Masaüstü** ile birbirine bağlanabilir. **XP** işletim sisteminde bulunan uzak masaüstü bağlantı bileşeninde birbirleri ile uzaktan bağlanacak iki bilgisayarın da XP olma zorunluluğu vardı eğer işletim sistemlerinden biri farklı ise **NetMeeting** ile ilgili **port açık olduğu sürece** bağlantı kurulabiliyordu ve uzak masaüstü bileşeni tüm XP versiyonlarında bulunmuyordu.

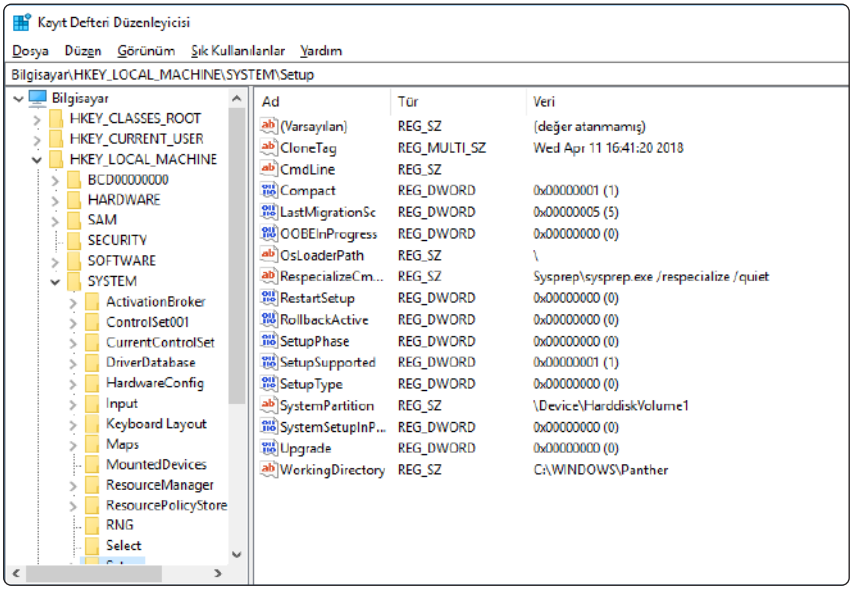


Şekil 4-8: Uzak Masaüstü

Uzak masaüstü, varsayılan ayarlarda **pasif** olarak gelmektedir. **Kullanılması gerektiğinde aktifleştirilmelidir**. Aktif edebilmek için oturum açan kullanıcının **yönetici olması** gerekmektedir. **Şekil 4-8'**de bulunan pencereden, ayarları yapılan uzak masaüstü işleminde bilgisayara uzaktan bağlanacak kullanıcılar "**Kullanıcıları Seç**" butonu ile seçilir. Biz bir bilgisayara uzak masaüstü ile bağlanmak istediğimizde karşıdaki bilgisayarın **IP adresi**, **kullanıcı adı** ve **şifresi** istenir. Bağlantıda internetten verimli faydalanabilmek için görüntü ayarları da kısıtlanabilir. Windows uzak masaüstü ile bağlantı kurulduğunda uzak bağlantı için kullanılan ticari yazılımlardan farklı olarak karşıdaki bilgisayarın kullanımı durdurulur.

Regedit

Windows işletim sisteminin tüm yapılandırma bilgilerinin tutulduğu veri tabanına kayıt defteri (regedit) denildiğini önceki bölümlerde öğrenmiştik. Bu bölümde regedit ile ilgili detaylı bilgilere değineceğiz. Windows işletim sistemi içerisinde varsayılan olan gelen bileşenlerden bir tanesi olan kayıt defterine Windows/Başlat menüsü/Çalıştır aracından "regedit.exe" veya "regedit" ile ulaşabiliriz. Kayıt defteri üzerinde yapılacak işlemlerin dikkatli bir şekilde yapılması gerektiğinden daha önce bahsetmiştik. Bu yüzden ne gibi sonuçlar doğuracağını bilmediğimiz özellikleri değiştirmek yanlış bir eylem olacaktır.



Şekil 4-9: Regedit

Kayıt defteri üzerindeki tüm işlemler **Şekil 4-9**'da yer alan pencereden yapılmaktadır. Pencerenin sol tarafında bulunan klasörlerin her biri **kendi içerisinde kategorilenmiş** alt klasörlerden oluşmaktadır ve regedit üzerindeki tüm değişiklikler bu klasörlerin içerisindeki girdiler üzerinden yapılmaktadır. Bir girdinin üzerinde **çift tıklanıldığı** zaman pencerenin orta kısmında o girdi ile ilgili ayarların yapılacağı **parametrik bir menü** açılmaktadır. Kayıt defteri düzenleyici penceresindeki sol taraftaki üst klasörlere, **kayıt defteri** anahtarları denilmektedir.

Kayıt Defteri Anahtarları

Kayıt defteri anahtarları, varsayılan olarak yerel bilgisayarlarda **beş klasör** şeklinde gösterilir. Bu anahtarların **ne tür bilgileri sakladıkları** ile ilgili aşağıdaki tablo incelenmelidir.

Klasör/Anahtar	Açıklaması
HKEY_CURRENT_USER	Bilgisayarda oturum açmış olan kullanıcının profil bilgileri ile ilgili ayarların tutulduğu yerdir.
HKEY_USERS	Bilgisayardaki tüm kullanıcıların profil bilgileri ile ilgili ayarların tutulduğu yerdir. Current yani aktif profil ayarlarının tutulduğu HKEY_CURRENT_USER, HKEY_USERS klasörünün alt klasörüdür.
HKEY_LOCAL_MACHINE	Herhangi bir kullanıcının bilgisayardaki özel yapılandırma bilgilerini barındıran yerdir.
HKEY_CLASSES_ROOT	HKEY_LOCAL_MACHINE klasörünün alt klasörüdür. Windows üzerinden gezgin ile bir program çağırdığımızda doğru olanın açılmasını sağlayan ayarları barındırır.
HKEY_CURRENT_CONFIG	İşletim sisteminin başlatıldığı esnada sahip olunan donanım birimleri profili hakkında bilgileri barındıran yerdir.

Regedit üzerinden gerçekleştirilecek olan **temel işlemlerden** birkaç tanesini beraber inceleyelim.

1. **Bilgisayar açılırken NumLock tuşunun açık olarak başlatılması**:

   Bu işlem oldukça **temel** fakat sonucu ile de bir o kadar **önemlidir**. Çünkü bilgisayar açılışında NumLock kapalı gelmesi durumu çoğu kullanıcısının özellikle **şifre giriş ekranında rahatsız olduğu** bir durumdur. Bu değişikliği **regedit üzerinden** basit birkaç adım ile düzeltebiliriz. Öncelikle **Başlat/Çalıştır/Regedit** veya direkt başlat çubuğuna regedit yazarak **kayıt defteri düzenleyicisini açılmasını** sağlayalım. Daha sonra sol tarafta yer alan klasörlerden **HKEY_USERS/.DEFAULT/Control Panel/Keyboard** anahtarına gelip pencerenin sağ tarafındaki düzenlenebilir ayarlardan **InitialKeyboardIndicators** alanına **çift** tıklayalım. Gelen küçük pencereden ise **Değer Verisini 2** olarak güncelleyelim. Bilgisayarımız yeniden başlattığımızda **NumLock** sürekli açık olarak gelecektir.

2. **Kısayol olarak yapıştırılan klasörlerin isimlerindeki "kısayol" ekininin kaldırılmasını sağlamak**.

   Regedit düzenleyicisi açıkken sol taraftaki klasörlerden **HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer** yolunu izleyerek gerekli anahtar alanına ulaşalım. Explorer alanına tıklayalım ve pencerenin sağ tarafındaki kayıtlardan adı **link** olan ayara **çift tıklayıp** değerini **00 00 00 00** olarak ayarlayalım. Bu ayarlamadan sonra bilgisayarı bir kez yeniden başlatmakta fayda olacaktır. Herhangi bir klasörü kopyalayıp kısayol olarak yapıştır dediğimizde veya masaüstüne **kısayol gönder** dediğimizde artık dosya adının başına **kısayol ibaresi gelmeyecektir**.

3. **Kilitlenen programların otomatik olarak kapatılmasını regedit ile sağlayalım**.

   Regedit düzenleyicisi açıkken sol tarafta bulunan klasörlerden **HKEY_LOCAL_MACHINE\System\ControlSet001\Control** yolu izlenir ve sağ taraftaki pencerede beliren özelliklerden en altta yer alan **WaitToKillServiceTimeOut** alanına **çift** tıklanarak **değeri 100** yapılır.

4. **Başlat menülerinin daha hızlı açılmasını sağlayalım**.

   Regedit düzenleyicimiz açıkken yine sol tarafta yer alan klasörlerden **HKEY_CURRENT_USER\ControlPanel\Desktop** yoluna girilir ve sağ taraftaki ayarlardan **MenuShowDelay** ayarına **çift** tıklanarak **değeri 1** yapılır.

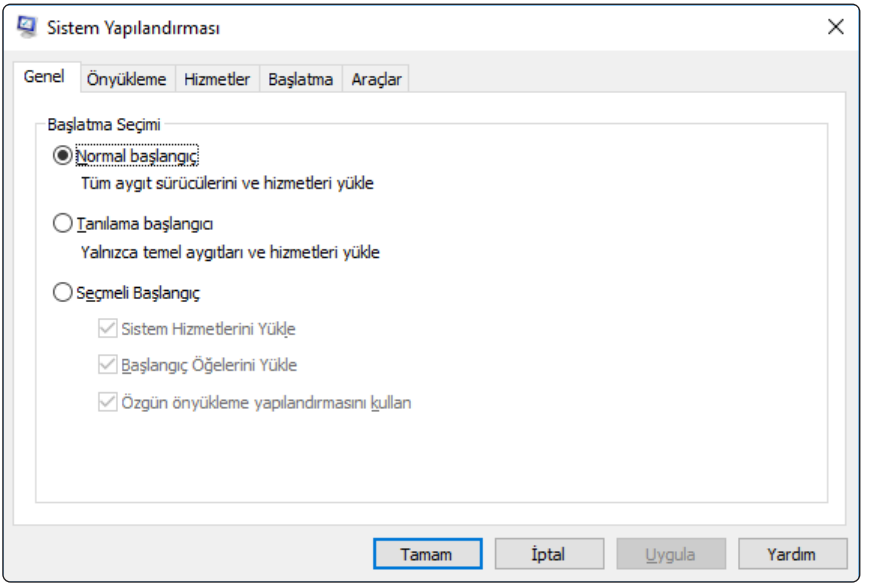
5. **Windows açılışına kendimize göre özelleştirdiğimiz bir açılış uyarı yazısı getirilmesini sağlayalım**.

   Regedit düzenleyicisi açıkken sol taraftaki klasörlerden **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\WinLogon** yolunda WinLogon seçili iken sağ tarafta gelen ayarlardan **LegalNoticeCaptin** alanına **çift** tıklanır ve göstermek istediğimiz mesajın başlığı bu değer alanına yazılır. Hemen altındaki **LegalNoticeText** alanına **çift** tıklayınca da değer alanına göstermek istediğimiz mesajı yazabiliriz.

Sistem Yapılandırması

Genel Sekmesi

Sistem yapılandırması bileşenine Başlat/msconfig yolu ile ulaşabiliriz. Sistem yapılandırma bileşeni Windows işletim sistemlerinde açılışta yapılan işlemlerin efektif hale getirilmesi için kullanılır. Msconfig olarak bilinen bu bileşen İngilizce Microsoft Configuration Utility kelimelerinin kısaltılmış halidir ve Türkçe de Microsoft Konfigursayon Aracı olarak geçmektedir. Başlat/Çalıştır menüsüne "msconfig" yazıp çalıştırdığımızda şekil 4.10'daki gibi bir pencere gelmektedir.



Şekil 4-10: Msconfig

Genel sekmesi ile bizi karşılayan bu pencereden **normal** başlangıç, **tanımlama** başlangıcı ve **seçmeli** başlangıç seçenekleri bulunmaktadır.

Normal başlangıç; işletim sisteminin kullanıcı müdahalesi olmadan yapılan sistem ayarları ile başlatıldığı başlangıç türüdür. İşletim sistemi bu seçenekte yüklenen programların yaptığı değişiklikler ve diğer hizmetlerin yaptığı ayarlar doğrultusunda başlatılır.

Tanımlama başlangıcı; sistemde bir problem olduğu zaman hatanın nereden olduğunu görebilmek için bu seçenek seçilirse bilgisayar açıldığında hatanın kaynağının bulunması kolaylaşır.

Seçmeli başlangıç; bilgisayara yüklediğimiz bir programın, bilgisayarın her açılışında yeniden yüklenmesini isteyebiliriz, bu gibi durumlarda kullanılacak olan başlatma türüdür. Altında çalışan **system.ini**, **win.ini** gibi seçenekler barındır.

Windows XP işletim sisteminde doğrudan **mcsonfig** penceresinde görüntülenen **system.ini** ve **win.ini** sekmeleri **XP** sonrasındaki işletim sistemlerinde doğrudan görüntülenmemektedir. Fakat karşılaşma ihtimaline karşın **system.ini** sekmesinin **çok eski bir sistem dosyası** olduğunun bilinmesi gerekir. İşletim sisteminin başlatılma esnasında yüklenecek olan **DLL** dosyalarını, **sürücüleri** ve **MSDOS** programlarında kullanılacak olan ayarların kaydedildiği dosyadır. **Win.ini** sekmesi ise system.ini kadar **eski bir dosya ayar** dosyasıdır fakat win.ini dosyasının farkı, **işletim sistemi açıldıktan sonra kullanılıyor olmasıdır**. İşletim sisteminin hangi ayarlar ile açılacağı ile ilgili bilgileri tutan dosyadır. Diğer bir dosya boot.ini dosyasıdır ve **NTFS** tabanlı işletim sistemlerinde kullanılır. İşletim sisteminin yüklenme yani boot edilme esnasındaki ayarları barındırır.

ARAŞTIRMA: /Safeboot, /Noguiboot, Bootlog, Basevideo ve /Sos seçeneklerinin ne işe yaradıklarını araştırınız.

Hizmetler Sekmesi

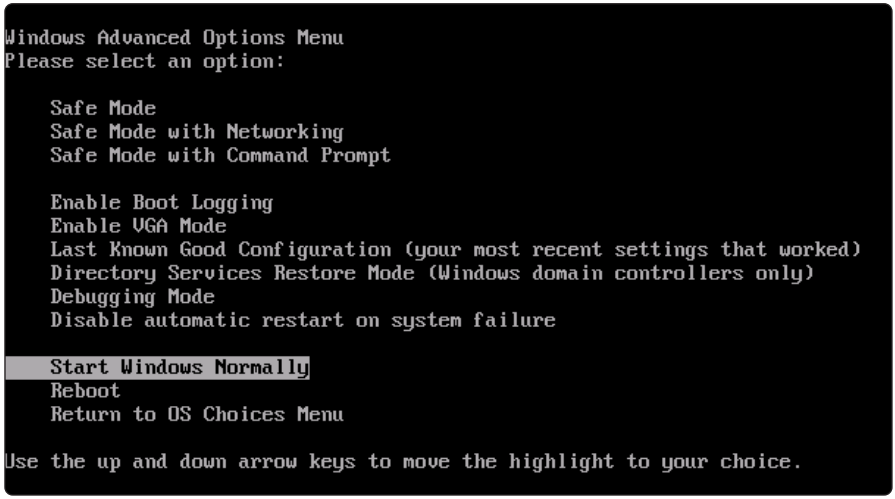
İşletim sistemi çalıştıktan sonra aynı zamanda arka tarafta **bizim işlemlerimiz için** çalışmakta olan **birtakım hizmetler** bulunmaktadır. Bu hizmetler işletim sisteminin özelliklerinin desteklenebilmesi için zaruri olarak çalıştırılan hizmetlerdir. Örneğin bir program takıldığında nasıl görev yöneticisi penceresinden programı sonlandırıyorsak **hizmetler sekmesinden de hizmeti sonlandırabiliriz**.

Başlatma/Başlangıç Sekmesi

Yüklediğimiz yazılımların bilgisayarın her açılışında **otomatik** olarak başlatılması **performans sorunu oluşturacağından istenmeyen bir durumdur**. Seçmeli başlangıç seçeneği ile açılış esnasında hangi programların çalışıp hangilerinin çalışmayacağını belirleyebiliriz. Genellikle **Skype kullanıcılarının başlangıçta yaşadıkları yavaş açılma probleminde en sık başvurdukları özelliktir**.

Güvenli Kip

Bilgisayarımızı bir problemden dolayı düzgün başlatamıyor olabiliriz. Sistem başlatılamadığı için geri yükleme noktasına döndürmek de mümkün olmayacağından sistemi **başlamadan farklı bir yol ile çalıştırmak gerekmektedir**. Güvenli mod olarak bilinen komut istemcisi bilgisayarın açılma esnasında **F8**'e basılması ile devreye girer. Komut istemcisi olarak kullanıcıyı karşılayan güvenli mod ekranı grafik arayüzlerden uzak daha çok **MSDOS** benzeri bir panel olarak gelir.



Şekil 4-11: Güvenli Mod

HATIRLATMA: Güvenli Mod ile açılan bilgisayar grafik arayüzünden **uzak** olduğu için biraz daha **son kullanıcı kullanımına uzak bir görünüme** sahiptir. Bilgisayarın normal hâliyle başlatılamadığı durumlarda güvenli mod yerine "**Boot Hiran**" gibi internette kolayca bulacağınız boot yazılımları ile açılmasını veya disk üzerindeki verilere ulaşılması sağlanabilir.

Şekil 4-11'de yer alan ekranda olduğu gibi güvenli mod ekranı açıldığında **aşağı yukarı yön tuşları ile güvenli mod türü seçilmelidir**.

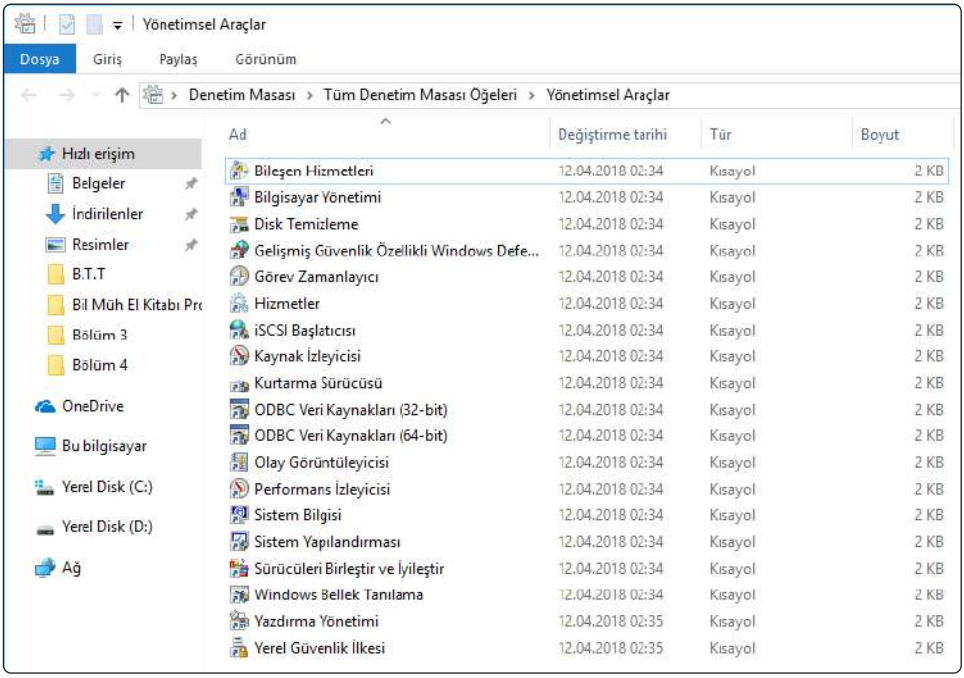
Safe Mod: Bu mod ile başlatılan sistemde **sadece temel işlevler** çalıştırılmaktadır. Ağ bağlantısı bulunmayıp grafik kartı en düşük seviyedeki VGA işleviyle çalıştırılır.

Safe Mode With Networkin: Safe mode'a ek olarak ağ bağlantısı özelliği de eklenerek başlatılır.

Safe Mode With Command Prompt: Bilinen **DOS komut** istemcisi ile başlatılır diyebiliriz. Diğer başlatma modlarından farklı olarak bu mod ile başlat menüsü gibi Windows grafikleri olmadan başlatılır.

YÖNETİMSEL ARAÇLAR

Bu bölümde "**Denetim Masası/Yönetimsel Araçlar**" sekmesi ile ulaşabileceğimiz pencerelerdeki özelliklere değineceğiz.



Şekil 4-12: Yönetimsel Araçlar

BİLEŞEN HİZMETLERİ

Bileşen hizmetlerine **Denetim Masası/Yönetimsel Araçlar** yoluyla ulaşabiliriz. **Şekil 4-12**'de yönetimsel araçlar penceresi ve içerik alanında pek çok aracın olduğunu görmekteyiz. Bileşen hizmetleri **sistem olaylarını görüntülemek** ve **yönetimsel araçların bulunduğu bilgisayarlarda sistem hizmetlerini yönetmek için** kullanılmaktadır. Bileşen hizmetleri **COM** bileşenleri ile **COM+** uygulamalarının **yönetimi** ve **yapılandırılmasında** görevlidir. Bileşen hizmetlerinin görevleri arasında ayrıca sistemi yapılandırmak, başlangıçtaki hizmetleri ayarlamak, bileşen hizmetlerini izlemek ve COM+ uygulamalarını yüklemek yer alır.

COM+ ile bileşen hizmetlerinin yönetilmesi için birtakım ayarlar yapılmalıdır. Bu ayarlar;

COM bileşenlerinin ve uygulamalarının güvenliği için gerekli COM güvenlik ayarları,

Dağıtılmış işlem yöneticisinin başlangıç şeklinin belirlenmesi ve ağ yapılanmasına uygun Windows kümeleme ayarlarının doğru yapılması şeklindedir.

Bilgisayar Yönetimi

Denetim masası üzerinden yine **Yönetimsel Araçlar/Bilgisayar Yönetimi** yolu ile ulaşabileceğimiz bu pencere bilgisayarı yönetmemiz için yardımcı bir program şeklindedir. *Sistem araçlarını, depolamayı, hizmetler ve uygulamalar ile ilgili ayarları* buradan yapabiliriz.

Hizmetler

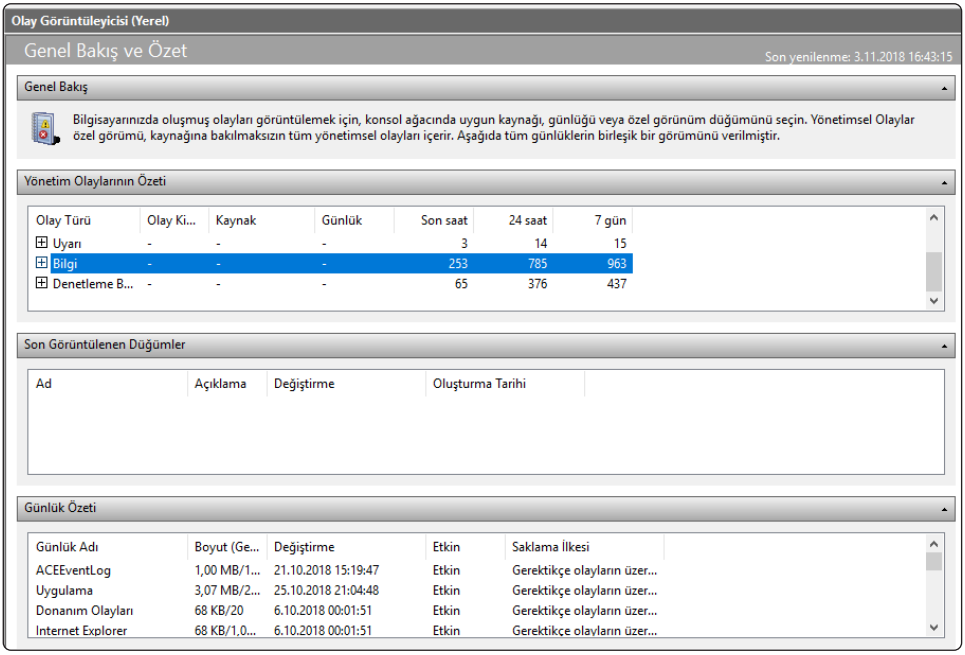
İşletim sistemleri kullanıcının gördüğü programlar dışında arka tarafta **kullanıcı tarafından görülmeyen** fakat o**ldukça önemli işlemleri gerçekleştiren hizmet bileşenlerini** çalıştırırlar. Bu bileşenlere **hizmetler** denilmektedir. Örneğin SQL Management Studio çalışırken *bir hata alınması durumunda SQL Server hizmeti durmuş olabilir mi acaba* diye ilk bakacağımız yerlerden bir tanesi **hizmetler penceresi** olmalıdır veya *ağdaki yazıcımızdan sorunsuz bir şekilde çıktı alabiliyorken sebepsiz ve hata mesajsız bir şekilde çıktı alamadığımız durumlarda da yazıcı servisi çalışıyor mu* diye **hizmetler penceresi ziyaret edilebilir**. Hizmetler penceresini yine **denetim masasından yönetimsel araçlar altındaki Hizmetler yolu** ile açabiliriz. Hizmetler penceresi önemli işlemler içerdiğinden işletim sistemi sizi **admin olmadığınız zaman hizmetler penceresine almayacaktır** bu yüzden admin hesabı ile giriş yapılması gerekir. Açılan hizmetler penceresinden bir hizmet durdurulabilir veya yeniden başlatılabilir. **Hizmet ayarlarına kritik durumlarda kayıt defteri editöründen de erişip değişiklik yapmak mümkündür**.

Olay Görüntüleyicisi

Olay görüntüleyici **oldukça önemli bir bileşendir**. Bilgisayar hakkındaki hemen her işleme ait bir **rapor** elde edilebilir. **Event Listener** olarak da işletim sistemlerinde erişimi mümkün olan bu bileşen, aslında İngilizce adı ile bize ne işe yaradığını doğrudan söylemektedir. Listener yani dinleyici bilgisayardaki hemen **her aktiviteyi dinler ve kayıt altında tutar**. Bu aktiviteler bilgisayarın *ne zaman açıldığı, kaç saat açık kaldığı, kullanıcısının parolasını kaç kez yanlış girdiği gibi bilgileri* tutmaktadır. Windows 10 işletim sisteminde **başlat çubuğuna** direkt "**event listener**" veya Türkçesini "**olay görüntüleyici**" yazdığımız zaman erişebileceğimiz gibi denetim masası üzerinden Y**önetimsel Araçlar/Olay Görüntüleyicisi** yolu ile de ulaşabiliriz.

Olay görüntüleyicisinde Sistem, güvenlik ve uygulama başlıklarında **üç farklı kayıt alanı** bulunur. Sistem kayıt alnında işletim sistemi aygıtlarla ilgili **her türlü olayı kaydetmektedir**. Güvenlik kayıt alanında yalnızca **güvenlik hususları ile ilgili olaylar tutulur**. Uygulama kayıt alanında ise işletim sisteminin dışında bulunan diğer yazılımların kendi çalışmaları ile ilgili **kayıtlar** tutulmaktadır. Kayıtlar yalnızca **LOG** şekline **bilgi amaçlı değil aynı zamanda uyarı ve hata şeklinde de tutulmaktadır**.

Olay görüntüleyici açıldığında listedeki olayların kaynak alanında "**eventlog**" yazdığı olaylar servisin kendi olaylarıdır. Diğer olayların kaydedilebilmesi için öncelikle bu servisin çalışması gerekmektedir. Bilgisayarımızda kapanma işleminde de **en son bu servis kapanacaktır**. Eğer bu servisin kapanmadığı durumlar olmuş ise bilgisayar olması gerektiği biçimde kapanmamıştır anlamına gelmektedir. Veri tabanı başlığında göreceğimiz **SQL Server** Monitor'de **sql** ile ilgili olayların tutulduğu sevistir.



Şekil 4-13: Olay Görüntüleyici

Olay görüntüleyici kurumlarda **çok kullanılan bir hizmettir**. Son kullanıcılar olarak bizler kişisel bilgisayarlarımızda pek ziyaret etmesek de kurumlarda problemin kaynağının tespit edilmesi yönünden ve kullanıcının bilgisayar üzerindeki eylemlerinin takip edilebilmek açısından **oldukça kullanışlıdır**. Öyle ki **AdManager** gibi yazılımlar ile **Active Directory** gibi domain yapısında çalışan bilgisayarların her birinin **olayları**, **tek bir yerden görüntülenebilmektedir**. Çeşitli rakipleri de olan AdManager **ücretli bir yazılımdır** ve domain yapısındaki bilgisayarların yönetildiği **sistem yöneticisinin bilgisayarına kurularak tüm kullanıcıların bilgisayarları ile olayların tek bir yerden görüntülenmesini sağlar**. LDAP sunucuda çalışan bir domain yapısı var ise **LDAP** ile bağlanarak kendiniz de **basit bir event listener** takip yazılımı geliştirebilirsiniz. Gerçek bir kurum için yapacağınızda bağlantıyı **VPN** ile yapmanız gerekecektir.

Performans

Performans bileşeni adından da anlaşılacağı üzere **işletim sisteminin beraber çalıştığı tüm donanımlar** ile arasındaki ilişkinin farklı yönlerden **incelenmesini** ve performansı optimize etmek için **analizler yapılmasını sağlayan bir bileşendir**. Bu bileşene erişebilmek için denetim masası altından **Yönetimsel Araçlar / Performans** yolunu takip edebilirsiniz. İlk açıldığında gelen pencere bize sistemimizin durumunu **grafiksel** olarak gösterecektir. Buradaki kayıtları daha sonra incelemek istersek ayarlayacağımız bir süre aralığında **kaydın tutulmasını da sağlayabiliriz**. Donanımların farklı yönden incelenmesini sağlar demiştim. Bu şu anlama gelmektedir örneğin ekran kartının veya Ethernet kartının gönderdiği-aldığı veri miktarı gibi bilgileri **sayaçlar sayesinde görebiliriz**.

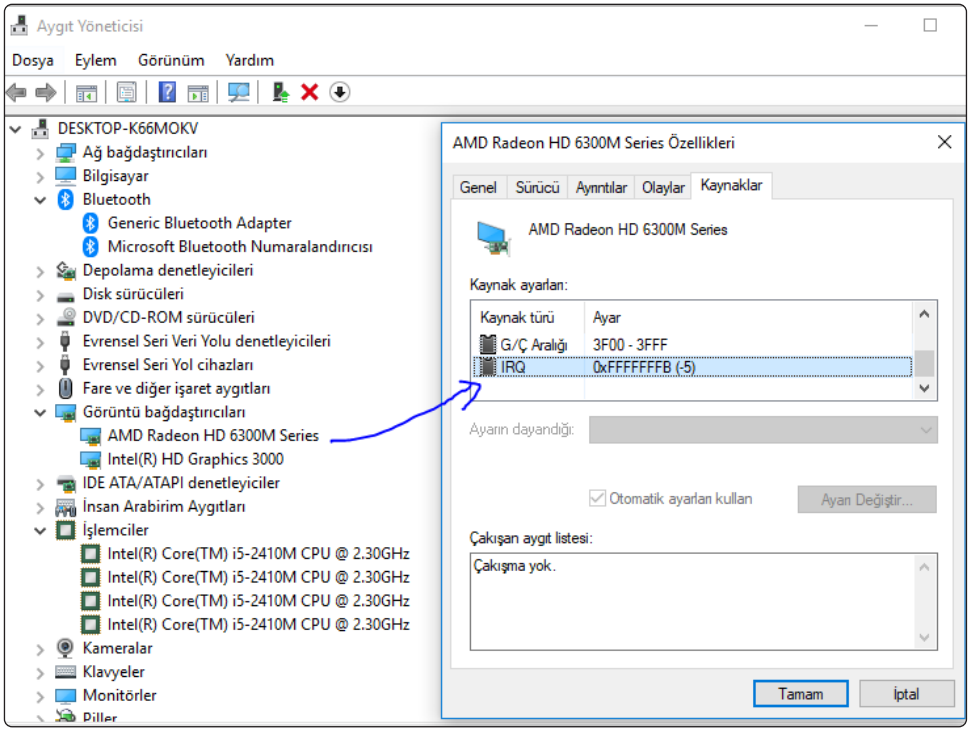
Yönetimsel araçlar penceresinden incelenebilecek daha fazla bileşen bulunmaktadır. *Veri kaynakları, yerel güvenlik ilkesi, kurtarma sürücüsü, kaynak izleyicisi, disk temizleme, sürücüleri birleştir ve iyileştir, yazdırma yönetimi* bunlardan birkaçıdır fakat kitabımızın konu işleyiş çerçevesinde her birine yer vermek yerine **bilinmesi gereken bileşenlere yer verilerek** diğerlerini de açıldığı zaman ne iş yaptıkları, nasıl kullanıldıkları ile ilgili fikir sahibi olunduğunuzu düşünüyorum.

Temel İşletim Sistemi Detayları

IRQ Kesme İstekleri

Interrupt Request olarak geçen ve Türkçe karşılığı **kesme isteği** olan **IRQ**, **donanımların işlemci ile iletişim kurmalarındaki rolü üstlenir**. Donanımların işlemci ile iletişim kurması gerektiği durumlarda IRQ devreye girer ve çevre birimlerin işlemci ile iletişim kurabilmesi için kesme istekleri sağlayan bir programın çalıştırılmasını sağlar. Bilgisayar rutin olarak çalışıyorken işlemciye gelen "**ben daha acilim**" sinyali olması durumunda **bu sinyali gönderen donanım biriminin talebini IRQ alır** ve **işlemcinin diğer işlemlerini keserek bu işi ele almasını sağlar**. Bu işlem elbette sistem tarafından belirlenen **öncelik sıralarına** göre yapılmaktadır. İşlemci bu talep ile ilgilenirken önceki işe ait dönüş adreslerini **yığında saklar** ve **işi bitince kaldığı yerden devam edecek şekilde adresleri işleme alır**.

IRQ, önceleri anakartlar üzerinde **ayrı bir çip** olarak geliyorken şimdiki anakartlarda **bütünleşik bir yonga olarak yerini almaktadır**. Bilgisayarımızda ekran kartının çalışması ile ilgili şüphelerimiz varsa **Şekil 4-14**'de olduğu gibi aygıt yöneticisi üzerinden ekran kartına ait donanım birimine çift tıklayıp açılan diyalog kutusundan "Kaynaklar" sekmesinin altında kaynak ayarları panelinde IRQ ile ilgili mesajları kontrol edebiliriz.



Şekil 4-14: IRQ

DMA Doğrudan Bellek Erişimi

Direct Memory Access anlamına gelen **DMA** doğrudan **bellek erişimi kelimelerinin kısaltmasıdır**. DMA özelliğinin **aktif olması**, **disk sürücüleri için performans artışı sağlayacaktır**. Doğrudan bellek erişimi modülü aktif olduğunda adından da anlayabileceğimiz gibi ihtiyaç hâlinde işlemciye hiç gitmeden bilgileri **direkt bellekten sorgulanacak** ve **performans yönünde artış sağlanmış olacaktır**.

I/O Giriş Çıkış Adresleri

İşlemciler iletişim kurarken iki yol kullanırlar; bu yollardan bir tanesi **bellek adresleri** diğeri ise **I/O adresleridir**. İşlemci diğer çevre birimleri ile iletişim kurarken bu birimleri tanıyabilmek ve iletişimini ayarlayabilmek için birimlerin **I/O** yani **giriş/çıkış** adreslerini kullanır. Genelde bu adresleri sektörde **port adresi** olarak bilmekteyiz. Kaynaklar sekmesinden görüntülenebilecek olan bu giriş/çıkış birimlerinin adres bilgileri son kullanıcılar için önemli olmasa da geliştiriciler için **donanımı tanımak adına kullanılacak tek bilgilerdir** diyebiliriz.

İşletim Sistemi Bileşenleri

İşletim sistemleri de **bilgisayar kasasının içerisindeki bileşenler gibi** bileşenlere sahiptir. Nasıl ki işlemci çalışmasını sağlayabilmek için diğer bileşenlere ihtiyaç duyuyor ise işletim sistemleri de **bileşenlerden** oluşmaktadır. Bu bileşenler *çekirdek, dosya yönetim sistemi ve kullanıcı arayüzünden* oluşur.

Çekirdek

İşletim sisteminde çekirdek **diğer servislerin çalışabilmesi adına işlemler gerçekleştirir**. *Programların başlatılması, çeşitli servislerin ihtiyacı olan paketlerin gönderilmesi, yapılan işlemlerdeki dosyaların izlerinin tutulması ve ağdaki paketlerin postalanması, alınması veya publish edilmesi* gibi işleri yapar. İşlemciler bir bilgisayarın donanımsal bir çekirdeği (core) olarak **kalbi** oldukları gibi işletim sistemlerinin de **yazılımsal kalbi çekirdektir**.

Dosya Yönetim Sistemi

İşletim sistemi bize ihtiyacımız olan sonuçları döndürürken kendi içerisinde **çeşitli kurallar** ve **prosedürlere** göre işlem gerçekleştirir. Dosyalar ile ilgili organizasyon işleminde **dosya yönetim sistemi** kullanılır. Sistem içerisindeki dosya yönetim sistemi **hiyerarşik olarak kurgulanmıştır** ve **bir başlangıç noktasına** sahiptir. Başlangıç noktası **kök dizin** olarak bilinir ve **C:** olarak bizi program kurarken veya farklı işlemlerde sıklıkla karşılar. Eğer işletim sistemi içerisinde bir dosya yönetim sistemi olmasaydı hâricî bellekleri kullandığımızda veya bilgisayarımıza bir oyun yüklediğimizde kaos yaşamamız muhtemel olacaktı. İşletim sistemi dosya yönetimi sorumluluğu sayesinde mantıksal dosyaların **fiziksel belleklere aktarılmasındaki geçişin sağlanmasını sorunsuz yönetir**. Hâricî belleklerin kullanımında bir farklılık, yabancılık olmaması bu belleklerin kullanımının **sorunsuz olması da yine dosya yönetim sisteminden dolayıdır**.

FAT dosya sistemi, **dosya yerleşim tablosu** olarak bilinir. FAT, disklerin **haritasıdır** çünkü diskte bulunan verilerin **kayıtlı oldukları alanları harita üzerinde belirtir**. **File Allocation Table** kelimelerinin kısaltmasından oluşmaktadır. FAT ile Windows dosya paylaşımı kullanılabilir. **255 karakteri geçmeyecek şekilde dosya isimleri verilebilir**. FAT sistemlerde **Unicode karakterler** de kullanılabilmektedir. FAT sistemini sabit diskin yanı sıra taşınabilir belleklerde de kullanabiliriz. **FAT16** ve **FAT32** gibi türleri vardır. Ayrıca sanal dosya sisteminde kullanılan FAT sistemine de Virtual FAT denilmektedir.

NTFS ise **New Technology File System** kelimelerinin kısaltmasıdır ve Türkçe karşılığı **yeni teknoloji dosya sistemidir**. İçinde bulunduğumuz yıl için yeni olduğunu söyleyemeyiz ama FAT ile beraber incelendiğinde **oldukça yeni bir teknoloji olarak algılandığı için** böyle geçmektedir. Bu sistem ise verilerin konumlarını ve diğer bilgilerini her bir **cluster** içerisinde ayrıca tutarak **güvenilir bir yapı sunar ancak disk haritası FAT sistemine göre oldukça geniştir**. 16 gb'a kadar olan dosyaları **tek parça hâlinde saklayabilir**. FAT sisteminden farklı olarak kullanıcı dosya erişimi konusunda kısıt konulmasını desteklemektedir.

Kullanıcı Arabirimi

Kullanıcı arabirimi denilince aklımızda **UI**, **GUI**, **User Experiement gibi kavramların çağrışım yapması** gerekmektedir. **Donanımın kullanıcı** ile **iletişimi arabirimler sağlamaktadır** ve genellikle kullanıcı dostu bir kullanıma sahip olmaları tercih edilir. Bir kullanıcı arabirimi **illa grafiksel olmak zorunda değildir**. DOS, MSDOS veya günümüzdeki Windows 10 gibi işletim sistemlerindeki kullanımı sağlayan kullanıcı ile **doğrudan iletişim hâlinde olan ortamlara kullanıcı arabirimi diyebiliriz**.

İşletim Sistemi Türleri

Gelişen donanıma göre işletim sistemleri de farklı türlerde geliştirilmektedir. Kullanıcıların ihtiyaçları ve kullanım yerlerine göre şekillenen bilgisayar sistemleri donanım tarafından gelişmeye ve şekillenmeye devam ettiği gibi **yazılım ve işletim sistemi yazılımları tarafından da şekillenmeye ve gelişmeye devam etmektedir**.

Multiuser

Birden fazla kullanıcının aynı programlar ve çevre birimleri üzerinde çalışabilmesine olanak veren işletim sistemleridir. *Ağ üzerinden oyun oynama, ağ üzerinden çıktı alma* gibi işlemleri örnek gösterebiliriz.

Multitasking

Kullanıcıların aynı anda birden fazla işlem gerçekleştirebildiği işletim sistemleridir. Çoklu görev işletim sistemlerinde aynı anda birden fazla görev başlatılır ve bu görevlerin işletilmesi işlemci ve diğer birimlerinde çoklu görev mantığında yapılmasına olanak sağlar. Burada bilinmesi gereken farklı işlemler arasındaki zaman diliminin **oldukça küçük** olmasıdır. Bu sayede eş zamanlı iş yapılıyor gibi bir yansıması olmaktadır. **İşletim sisteminin işlemciye aynı anda iki işi yaptırıyor olması düşünülmemelidir**.

Multiprocessing

Günümüz bilgisayarlarının çoklu çekirdeğe veya çoklu işlemciye sahip olduğunu biliyoruz. *Peki, donanımın bu şekilde çoklu işlemciden oluşması işletim sistemi için standart bir durum mudur yoksa işletim sisteminin de çoklu işlemci desteği olması gerekir mi?* **Elbette yeni işletim sistemlerinin hepsi çoklu işlemci desteğini vermektedir**. Çoklu işlemcili işletim sisteminin gelen işleri farklı işlemcilere dağıtabiliyor olması ve bunları kontrol edebiliyor olması gerekmektedir.

Multithreading

Çoklu görevleri destekleyen işletim sistemi türüdür. Gelen bir ham program işletilirken parçalar hâlinde işletilmesi gerekebilir. Gelen işin ayrılmış parçaları farklı farklı kullanıcılar tarafından da işletiliyor olabilir. Bu gibi durumlarda işletim sisteminin **çoklu görev desteği olması gerekmektedir**.

ARAŞTIRMA: STACK ve HEAP kavramlarını araştırınız.

SUNUCU SİSTEMLERİ

5

Bilgisayarlar kullanım alanına göre son kullanıcıların kullandığı **istemci bilgisayarlar** ve yönetim görevini üstlenen **sunucu bilgisayarlar** olarak günümüz bilgisayar piyasasında **iki ayrı kategoride incelenmektedir**. Son kullanıcı bilgisayarları istemci bilgisayarlar olarak bilinmekte olup sadece son kullanıcıların kullanımı için değil aynı zamanda daha büyük bir sistemden aldığı verileri kullanıcıya sunan birçok sistemde kullanılmaktadırlar. **Client** yani istemci bilgisayarların yanı sıra **sunucu (server) bilgisayarlar** ağ üzerindeki yönetimi sağlayan bir dize özellikle donatılmış bilgisayarlardır. Ağda bulunan sunucu bilgisayarlar **tüm ağ üzerinde trafiği denetleyen programlar çalıştırmaktadırlar**. Sunucu bilgisayarlar üzerinde çalışan sunucu işletim sistemleri sayesinde yüksek dereceli işlemleri ve görevleri geliştirilmiş bir kullanılabilirlik ile gerçekleştirirler.

Bir sunucuda **sanallaştırma sayesinde** birden fazla sanal bilgisayar oluşturabilir, her bir bilgisayarda ise ihtiyacımıza göre programlar çalıştırabiliriz. Örneğin bir kurumun personel giriş çıkış turnike sisteminin ve **pdks** yazılımının tutulduğu bir sanal bilgisayar ile kurumun muhasebe yazılımı olan **netsis**'in barındırıldığı bilgisayar **aynı sunucu içerisindeki farklı sanal bilgisayarlar olabilir**. Veya yedek ve kurtarma sistemlerinin sanal sunucularda barındırılabileceğini düşünebiliriz. Bu örnekler bilgi işlem yöneticisinin yeteneklerine göre değişecektir. Elbette kuralına uygun olan esasta yapılması veya varsa bilgi güvenliği yönetim sistemi kapsamındaki talimatlara göre yapılması gerekecektir.

Sunucu İşletim Sistemleri

İşletim sisteminin **donanım** ve **yazılımın** beraber çalışmasını sağlayan kullanıcılar tarafından müdahale edilebilirlik seviyesinin yüksek olduğu kullanıcı ile etkileşimi bulunan yazılımlar olduğunu ve çalışma prensiplerine göre **çok görevli**, **çok çekirdekli** gibi çeşitlerinin olduğunu öğrendik. Bu bölüm bize, sunucu işletim sistemleri hakkında bilgi sahibi olunması konusunda ışık tutacaktır.

Sunucu işletim sistemlerinin en temel işlevi **yüksek kapasite**, **yüksek hız** ve **güvenlik** unsurudur. Verilerin tek yerden dağıtılması ve işlenmesiyle ilgili bir çözüm olan sunucu bilgisayarlar üzerinde çalışan sunucu işletim sistemlerinden en yaygın olanı Microsoft firmasına ait **Windows Server** ailesidir. Hastanelerde ve çeşitli kurumların iş yükünü omuzlayan **Windows Server 2003** ve **Windows Server 2008** işletim sistemini çoğu okurumuz görmüştür.

Bilinen yaygın sunucu işletim sistemi tür ve versiyonları şu şekildedir:

- **Windows Server Ailesi**
  - Windows NT
  - Windows 2000
  - Windows Server 2003
  - Windows Server 2008 (*Standart-Enterprise-DataCenter Edition ve Web Server*)
  - Windows Server 2008 R2
  - Windows Server 2012
- **Linux Ailesi**
  - Debian
  - Fedora
  - Suse
  - RedHat
- **MAC OS X**

Sunucu işletim sisteminin kurulum aşaması normal masaüstü işletim sistemi kurulumu ile oldukça benzerlik göstermektedir. Sunucu işletim sistemi (*örneğin Windows Server 2008*) adım adım nasıl kurulduğu ile ilgili olarak internet araştırması yapıldığı takdirde **çok değerli kaynaklara** ulaşacağınızdan eminim, bu yüzden kurulum adımlarını tek tek göstermek yerine önemli olan noktalara değineceğim.

Sunucu işletim sistemleri üzerinde işlemler genellikle **poliçeler oluşturmak** ve **roller** ile yapılır. Sunucu rolleri ve diğer yapılandırma özelliklerinin **kolaylıkla yapılabildiği sunucu yönetimi** konsolu (*server manager console*) s**ayesinde özellikler kolaylıklar eklenebilir**, **kaldırılabilir** veya **değiştirilebilir**. Windows Server 2008 işletim sistemi ile önceki versiyonlarda meydana gelen **arabirim kaynaklı performans düşüklüğü problemi giderilmiştir**. Bu sorunu gidermek için arabirim yerine **komut satırı** ile de işlemlerin gerçekleşebilmesine olanak sağlayan bir yöntem izlenmiştir. Komut satırından da yönetime izin veren Windows server 2008 işletim sistemi sayesinde **DNS**, **DHCP**, **File Server** gibi servislere hizmet edilmektedir.

Sunucu işletim sistemlerinin masaüstü işletim sistemlerine göre en büyük farkının **ağ üzerinde yönetim görevini üstleniyor olması** ve **yüksek güvenlik ile çalıştığından bahsetmiştik**. Sunucu işletim sistemleri üzerinde yapılacak işlemlerin ise roller ve poliçeler ile yapılabildiğine değindik. Peki, güvenlik ile ilgili ağdaki diğer bilgisayarlara **kısıtlama poliçesi** getirmek istiyoruz ve bu duruma göre **ağdaki diğer bilgisayarların bir davranış benimsemesini istiyoruz**, böyle bir durumda Windows server 2008 işletim sisteminde de barınan **NAP** devreye giriyor. NAP, **Network Access Protection** yani **ağdaki erişimin güvenliği** ile ilgili bir özelliktir. Sunucuya bağlı olarak faaliyet gösteren diğer bilgisayarlara ağ üzerinde **bir güvenlik katmanı oluşturulmasını sağlayan bu özellik** ile ağdaki bilgisayarın **firewall özelliğinin açık kalması gerektiği gibi** poliçeler tanımlanabilir. Network ve donanım bazlı sağlanan güvenlik prosedürlerinin yanı sıra sunucu işletim sistemi üzerinde de bu şekilde poliçeler ile güvenlik önlemleri alınabilmesi mümkündür.

ARAŞTIRMA: Sunucu işletim sistemi kurulum aşaması ile ilgili olarak Unattended ve Normal kurulum yöntemlerini araştırınız. Sanal bilgisayarınıza iki şekilde de kurulum yapınız.

Sunucu işletim sistemlerinin yüksek kapasiteli disk çalışma metodu **raid** denilen **disk yapılandırması** ile mümkün olmaktadır. Sunucu bilgisayarların yüksek hızlarda çalışan bilgisayarlar olduğunu düşündüğümüzde işlemcinin **çok hızlı olacağı** ve **sabit diskin yavaş kalacağını düşünmek** artık bizim için ihmal edilecek bir durum değil çünkü temel konularda işlemcinin çok hızlı fakat sabit diskin çok yavaş çalıştığını öğrenmiştik. *Peki, sunucu işletim sistemlerinde aşırı hızlı çalışan işlemciye karşın disk yapılandırılması nasıl oluyor?* Az önce de adı geçtiği gibi bu yapının ismi **RAID** olarak bilinir ve birden fazla diski eş zamanlı olarak performans kaybı olmaksızın hatta o**kuma-yazma hızlarını güvenli bir şekilde arttırılması görevini** üstlenir. **RAID bir teknoloji kavramıdır** ve **donanımsal Raid** ve **yazılımsal Raid** olarak iki kategoride incelenir. İşletim sistemi içerisindeki araçlar ile kullanılan **Yazılımsal Raid**, devre kartları ile yapılana ise **Donanımsal Raid** denir. **Windows Server 2008 işletim sistemi birçok konuda olduğu gibi yazılımsal RAID konusunda da yeteneklidir**.

BİLGİSAYAR SAYI SİSTEMLERİ VE SAYISAL ELEKTRONİK

6

Sayı Sistemleri ve Dönüşümler

Bu bölümde, bilgisayar biliminin kullandığı **sayı sistemlerini** ve **lojik devreleri** inceleyeceğiz. Bu konu aslında *temel elektronik veya elektrik elektronik mühendisliğine giriş* gibi derslerin ilk konularına oldukça yakındır. Bu bölümde de amacımız kitabımızın genel işleyişinde olduğu gibi konular üzerindeki **dikkat çekici noktaları vurgulamak** ve **karşılaşılması muhtemel konuların tek bir yerde toplanmasıdır**.

Bilgisayarların işlemleri gerçekleştirdiği mantıksal proseslerde **dört çeşit sayı sistemi** bulunmaktadır. Başka bir deyişle lojik sayı sistemlerinde **iki tabanlı** (*binary*), **on tabanlı** (*decimal*), **onaltı tabanlı** (*hexadecimal*) ve **sekiz tabanlı** (*octal*) sayılar olmak üzere dört çeşit sayı sistemi bulunur.

Binary sayı sisteminde bilgisayar sistemini zaten en iyi ifade eden "**1**" ve "**0**" kavramı yer almaktadır. **Binary**, yani ikilik sayı sisteminde değerler "1" ve "0"'dan oluşur. **Voltaj değeri var ise** "1" yok ise **"0" değeri ölçülür**. Voltaj değerinin olması **5V'a** veya **3.3V'a** karşılık gelmektedir. Bilgisayarlar "1" ve "0" değerlerini kullanarak işlem yaparlar tanımı, aslında elektronik olarak buradan gelmektedir. Lojik olarak *toplama*, *çıkarma*, *çarpma* veya *bölme* işlemlerini yapan devre tasarımlarında sayı sistemlerinden oldukça yararlanılmaktadır. Binary sayı sistemi ile hazırlanmış bir sayı ifade edileceği zaman, onluk sayı sistemindeki gibi **basamak değeri varmışçasına okunmaz** bunun yerine her bir değer, **rakam karşılığı olarak** söylenir.

Örneğin ondalık, yani **decimal** olarak **65** sayısının binary olarak hazırlandığı **01000001** ifadesi "**sıfır-bir-sıfır-sıfır-sıfır-sıfır-sıfır-bir**" şeklinde okunur. 65 değeri aynı zamanda klavyedeki **A** harfine karşılık gelmektedir. ASCII tablosundan hangi harf veya klavye değerine hangi binary sayının karşılık geldiğini inceleyebilirsiniz.

Binary sayı sistemindeki bu değerlerin her biri, **bir bit değer** ifade etmektedir. Sinyal sistemlerindeki veya sinüsoidal eğrilerde "0" bit değeri **voltajın olmadığını**, "1" bit değeri ise **voltajın olduğunu** ifade eder. Veya bazı kavramlarda "1" var ise "1" **pulse clock** şeklinde değerin hareketi ifade edilir. Dijital tasarım ile uğraştığınızda binary sayıları decimal sayılara çevirme ihtiyacı doğacağından, binary sayıyı **decimal sayıya** ve tam tersi decimal sayıları da **binary sayılara çevirme** işleminin nasıl olduğunu bilmemiz gerekmektedir. Temel matematik konularındaki ikilik tabandaki sayıları **onluk tabana çevirme** ve onluk tabandaki sayıları **ikilik tabana çevirme** işlemi ile aynı şekilde yapılabilir. Örneğin: 11010101 sayısını onluk tabana çevirmek için aşağıda gösterildiği gibi her bir bit için basamak değerleri **2'nin üssü olarak yazılır** ve "1" biti içeren değerler toplanarak **onluk tabandaki karşılığı bulunmuş olur**. "0" bitlik değerler kendisini götüreceği için toplamaya dâhil etmek bir şey ifade etmeyecektir.

1 1 0 1 0 1 0 1

$2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0=213$

Matematik dersinden de bildiğimiz bu çevrim aslında aşağıdaki gibi bir çözüme sahiptir.

$11010101 = (1 \times 2^7) + (1 \times 2^6) + (0 \times 2^5) + (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 213$

Decimal bir sayıyı, binary bir sayıya çevirmek istediğimizde ise yine matematik dersinden aklımıza gelen **bölme işlemi yöntemini** temel olarak uygulamak yeterli olacaktır. Örneğin decimal **15** değerini binary sayı sistemine çevirmek istediğimizde sayıyı **sürekli olarak ikiye bölmek gerekmektedir**. Çıkan her bir kalan değeri ile en sonda yer alan bölüm değerini **yan yana getirip dizdiğimizde** (*bölümden başlayarak*) **ikilik sayı sistemine çevirmiş oluruz**.

Bu sözel tanımı gerçek bir örnekle ifade etmemiz gerekirse yukarıdaki örnekte bulunan **binary** sayıyı, bu sefer de **213** sayısını **binary sayı sistemine çevirmeyi** deneyerek bulmaya çalışalım.

213/2= 106 Kalan 1 106/2= 53 Kalan 0 53/2= 26 Kalan 1 26/2= 13 Kalan 0
13/2= 6 Kalan 1 6/2= 3 Kalan 0

3/2= 1 Kalan 1

Bölme işlemini bu şekilde ayrıntılı gösterilmesi, işlemin nasıl olduğunu iyice **pekiştirilmesi** açısından önemlidir. Şimdi, yukarıdaki işlemde sondan başlayarak en sonda yer alan bölüm değeri yani "**1**" ve sırasıyla tersten kalan değerleri yan yana yazdığımızda **11010101** şeklinde aynı değeri elde ederiz.

Aşağıdaki tabloda sayı sistemleri ile ilgili taban dönüşüm değerleri yer almaktadır.

Decimal	Binary	Hexadecimal
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10
17	10001	11
18	10010	12
19	10011	13

20	10100	14
21	10101	15
22	10110	16
23	10111	17
24	11000	18
25	11001	19
26	11010	1A
27	11011	1B
28	11100	1C
29	11101	1D
30	11110	1E
31	11111	1F
32	100000	20
64	1000000	40
128	10000000	80
256	100000000	100

Tablo 6-1: Sayı Sistemleri Dönüşüm Tablosu

Programlama konularında, özellikle temel programlama derslerinde öğretim üyelerinin genellikle ilk derslerde konuyu kavratmak için verdikleri program örneği, genelde **sayı tabanları** ile ilgilidir. **MOD alma** yöntemi ile bir sayının **ikiye bölümünden kalan değerleri bularak pek çok probleme cevap bulabiliriz**. Bu, bize verilen çift sayıları bulma probleminde de işe yarayacaktır. Tek sayıları bulma problemine de veya asal sayı olup olmadığını bulma problemlerine de cevap olacak bir işlemdir. Bu yüzden eğitim aldığımız üniversitelerde ya da bilgisayar bilimleri ile ilgili ders verilen yerlerde programlama dersi olsun elektronik dersi olsun veya algoritma dersi olsun **sayı sistemleri** muhakkak verilir.

Decimal sayı sistemi, **günlük hayatta** kullandığımız sayı sistemi olduğu için **onaltılık** ve **sekizlik** tabanlarla konuya devam edelim. Mühendislik disiplininin, kitabımızın ilk konularından itibaren sürekli olarak *performans* ve *optimizasyon* ile ilgilendiğinden bahsediyorum. Sayı sistemlerinden binary sayı sistemi, **bilgisayarın anladığı tek sistemdir** ve **on yıllardır bilgisayarlar "bir" ve "sıfır"dan oluşur** denilmektedir. Çok basamaklı sayıların kullanıldığı işlemlerde bilgisayar hem hızlı işlem yapabilmesi hem de ifadelerin daha yalın olabilmesi için geliştirmeler devam etmiş ve hexadecimal sayı sistemi kullanılmıştır.

Bu sayı sistemi onluk ve ikilik sayı sistemleri arasında **taban çevrimini** mümkün kılmakla birlikte büyük sayılar için **daha az maliyetli** sayılar sunmaktadır. Bu sayı sistemine, **onaltılık sayı sistemi** (*hexadecima*) denilmesinin sebebi sayı sistemi içerisinde **16 adet ifadeye yer verilmesidir**. Bunlar 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F'dir. **A** harfi **10**, **B** harfi **11**, **C** harfi **12**, **D** harfi **13**, **E** harfi **14** ve **F** harfi **15** değerini ifade eder. Dikkat edileceği üzere sıfırdan başladığı için toplamda 16 sayıya denk gelmektedir. Hexadecimal sayılar ifade edilirken **taban yazılmak zorundadır**. Örneğin rastgele bir hexadecimal sayı **(4D)16** şeklinde gösterilebilir.

Hexadecimal bir sayının, binary sayıya çevrilmesi işlemi oldukça kolaydır. Örneğin (4D)16 sayısı binary sayı sistemine çevrilirken; öncelikle 4 rakamının binary karşılığı **100**'dür. D'nin karşılık geldiği 13 rakamının binary karşılığı **1101** olarak hesaplanır. İki değer aynı sırada yan yana yazıldığında **1001101** olarak bulunur. Aynı değerin decimal karşılığı ise 16 tabanda işlem gerçekleştirildiğinde **77** olarak bulunur.

Aşağıdaki tabloda, hexadecimal değerlerin **binary karşılıkları ile ilgili dönüşümler** yer almaktadır.

Hexadecimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111
10	10000
20	100000

Tablo 6-2: Hexadecimal ve Binary Dönüşüm Tablosu

Onaltılık tabandaki sayıların binary sayı sistemine ve decimal sayı sistemine aynı şekilde diğer tabanların da onaltılık sayı sistemine nasıl dönüştüğü ile ilgili tablolardan da yararlanarak *daha fazla pratik* yapabilirsiniz. Diğer bir sayı sistemi ise eski sistemlerde çok kullanılmasına rağmen artık güncelliğini korumadığını söyleyebileceğimiz **sekizlik sayı sistemi**dir. Octal, yani sekizlik sayı sistemleri de adından anlaşılacağı üzere 0, 1, 2, 3, 4, 5, 6 ve 7 değerini alarak **8 bit değer** ile işlem gerçekleştiren bir sayı sistemidir. Octal sayıları da binary sayılara çevirirken diğerlerinde olduğu gibi bir işlem gerçekleştiririz. Örneğin **(57)8** octal sayısını decimal sayıya çevirmek için **5** ve **7 rakamlarının binary karşılıklarını bulur** ve **yan yana yazdığımızda sonuca ulaşmış oluruz**. Bu örneği yaptığımızda 5 rakamının binary karşılığını hesapladığımızda veya yukarıdaki tablolardan baktığımızda 101 ve 7 rakamının ise 111 olduğunu görmekteyiz.

Yan yana yazdığımızda **(57)8 = (101111)2** olacaktır.

Sayı sistemlerindeki dönüşümlerin aslında hep aynı mantıkla işlediğini ve *temel matematik bilgimizle bu dönüşümleri kolayca yapabildiğimizi farz ediyorum*. Bu konular meslek liselerinde, ön lisans programlarında ve mühendislik birinci sınıf derslerinde anlatılmaktadır. Bu konuyu zaten bilen okurlarımız için **tekrar**, yeni öğrenen okurlarımız için ise **giriş niteliğinde** olan bu bölümü sonlandırmadan önce **decimal sayıların binary sayılara dönüşümü** ile ilgili geliştirilmiş bir tür dönüşüm sistemi olan **BCD** dönüşümünden bahsetmek istiyorum. Yine amaçları, **işlemlerimizi kolaylaştırmak** olan bilgisayar bilimcilerinin ortaya koyduğu bu yöntemde, decimal bir sayının binary bir sayıya çevrilmesinde her bir basamak değerinin binary karşılığının **ayrı ayrı hesaplanarak sonuca ulaşılması esasına dayanmaktadır**. Örneğin (2650)10 onluk tabanındaki sayıyı binary, yani ikilik tabana çevirmek için sayıyı tek tek ikiye bölmek yerine rakamların **her birinin binary karşılıklarını hesaplayabiliriz**. Hesaplama yapılırken her bir basamak değeri için **4 bitlik değer** yazılır. Örneğin 1 rakamı binary sisteme dönüşürken sonuç "**1**" çıkacaktır fakat dört bit olarak ifade edilmesi gerektiğinden decimal 1 rakamı binary dönüşümü "**0001**" olarak ifade edilmelidir. Bu şekilde yapılacak hesaplamada birim olarak **BCD** kullanılmalıdır.

2 = 0010 6 = 0110 5 = 0101 0 = 0000 sonuç olarak **(2650)10 = (0010 0110 0101 0000)bcd** olarak bulunur.

Ascii Tablosu

Bilgisayar ve elektronik ile ilgili pek çoğu gibi bu standart da karşımıza **Amerikan standardı** olarak çıkmaktadır. ASCII kod sistemi, **Amerikan standart kod** olarak bilinir. Bu sistem bir tablo ile karşımıza çıkar ve klavye üzerinden girişi yapılan her karakterin, bilgisayarın anlayacağı dil olan binary ifadelere dönüşümü ile ilgili bir dize **standart tablosunu** içerir.

Ascii kod sistemi **dört grupta** toplanmıştır. Bunlardan ilk grup **32 karakterden** oluşur ve **fonksiyon tuşlarını** tutar. İkinci grup yine **32 karakterden** oluşur ve **noktalama işaretleri** ile **sayı** ve **space** gibi karakterleri tutar. Üçüncü grup **büyük karakterleri** tutar. Dördüncü grup ise **sembolleri** ve **küçük karakterleri** tutar. Ascii kod sistemi bu şekilde dört gruptan oluşur ve her biri 32 karakterden oluştuğu için **128 karakterlik ile standartlık sunar**.

Kısacası aşağıdaki ASCII tablosundan da inceleneceği üzere bir bilgisayar çevre birimlerinden klavye ile girişi sağlanan her bir karakter için makine dilinden bir karşılığı dikkate alır.

Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char
0	0x00	000	0000000	NUL	32	0x20	040	0100000	space	64	0x40	100	1000000	@	96	0x60	140	1100000	`
1	0x01	001	0000001	SOH	33	0x21	041	0100001	!	65	0x41	101	1000001	A	97	0x61	141	1100001	a
2	0x02	002	0000010	STX	34	0x22	042	0100010	"	66	0x42	102	1000010	B	98	0x62	142	1100010	b
3	0x03	003	0000011	ETX	35	0x23	043	0100011	#	67	0x43	103	1000011	C	99	0x63	143	1100011	c
4	0x04	004	0000100	EOT	36	0x24	044	0100100	$	68	0x44	104	1000100	D	100	0x64	144	1100100	d
5	0x05	005	0000101	ENQ	37	0x25	045	0100101	%	69	0x45	105	1000101	E	101	0x65	145	1100101	e
6	0x06	006	0000110	ACK	38	0x26	046	0100110	&	70	0x46	106	1000110	F	102	0x66	146	1100110	f
7	0x07	007	0000111	BEL	39	0x27	047	0100111	'	71	0x47	107	1000111	G	103	0x67	147	1100111	g
8	0x08	010	0001000	BS	40	0x28	050	0101000	(	72	0x48	110	1001000	H	104	0x68	150	1101000	h
9	0x09	011	0001001	TAB	41	0x29	051	0101001	)	73	0x49	111	1001001	I	105	0x69	151	1101001	i
10	0x0A	012	0001010	LF	42	0x2A	052	0101010	*	74	0x4A	112	1001010	J	106	0x6A	152	1101010	j
11	0x0B	013	0001011	VT	43	0x2B	053	0101011	+	75	0x4B	113	1001011	K	107	0x6B	153	1101011	k
12	0x0C	014	0001100	FF	44	0x2C	054	0101100	,	76	0x4C	114	1001100	L	108	0x6C	154	1101100	l
13	0x0D	015	0001101	CR	45	0x2D	055	0101101	-	77	0x4D	115	1001101	M	109	0x6D	155	1101101	m
14	0x0E	016	0001110	SO	46	0x2E	056	0101110	.	78	0x4E	116	1001110	N	110	0x6E	156	1101110	n
15	0x0F	017	0001111	SI	47	0x2F	057	0101111	/	79	0x4F	117	1001111	O	111	0x6F	157	1101111	o
16	0x10	020	0010000	DLE	48	0x30	060	0110000	0	80	0x50	120	1010000	P	112	0x70	160	1110000	p
17	0x11	021	0010001	DC1	49	0x31	061	0110001	1	81	0x51	121	1010001	Q	113	0x71	161	1110001	q
18	0x12	022	0010010	DC2	50	0x32	062	0110010	2	82	0x52	122	1010010	R	114	0x72	162	1110010	r
19	0x13	023	0010011	DC3	51	0x33	063	0110011	3	83	0x53	123	1010011	S	115	0x73	163	1110011	s
20	0x14	024	0010100	DC4	52	0x34	064	0110100	4	84	0x54	124	1010100	T	116	0x74	164	1110100	t
21	0x15	025	0010101	NAK	53	0x35	065	0110101	5	85	0x55	125	1010101	U	117	0x75	165	1110101	u
22	0x16	026	0010110	SYN	54	0x36	066	0110110	6	86	0x56	126	1010110	V	118	0x76	166	1110110	v
23	0x17	027	0010111	ETB	55	0x37	067	0110111	7	87	0x57	127	1010111	W	119	0x77	167	1110111	w
24	0x18	030	0011000	CAN	56	0x38	070	0111000	8	88	0x58	130	1011000	X	120	0x78	170	1111000	x
25	0x19	031	0011001	EM	57	0x39	071	0111001	9	89	0x59	131	1011001	Y	121	0x79	171	1111001	y
26	0x1A	032	0011010	SUB	58	0x3A	072	0111010	:	90	0x5A	132	1011010	Z	122	0x7A	172	1111010	z
27	0x1B	033	0011011	ESC	59	0x3B	073	0111011	;	91	0x5B	133	1011011	[	123	0x7B	173	1111011	{
28	0x1C	034	0011100	FS	60	0x3C	074	0111100	<	92	0x5C	134	1011100	\	124	0x7C	174	1111100	|
29	0x1D	035	0011101	GS	61	0x3D	075	0111101	=	93	0x5D	135	1011101	]	125	0x7D	175	1111101	}
30	0x1E	036	0011110	RS	62	0x3E	076	0111110	>	94	0x5E	136	1011110	^	126	0x7E	176	1111110	~
31	0x1F	037	0011111	US	63	0x3F	077	0111111	?	95	0x5F	137	1011111	_	127	0x7F	177	1111111	DEL

Şekil 6-1: ASCII Kod Sistemi Tablosu

Lojik Kapılar

Meslek lisesi bilgisayar veya elektronik bölümlerinde okuyan/mezun okurlarımızın konuya hâkim olacağını söyleyebiliriz. Meslek liselerinde *Temel Elektronik Ölçme ve Teknik Resim* derslerinde sıklıkla kullanılan lojik kapıların işlendiği dersler, iki yıllık eğitim veren önlisans programlarında yani meslek yüksek okullarında genellikle **Sayısal Elektronik** isimli ders ile öğrencilerle buluşmaktadır. Mühendislik seviyesinde bu dersi incelediğimizde ise **Digital Design**, **dijital dizayn** dersi olarak veya **lojik devreler** dersi olarak karşımıza çıkmaktadır. Ders isimleri ve içerikleri elbette üniversite ders içeriklerine göre değişkenlik gösterecektir ama eminim ki tüm üniversitelerimizde bu dersler derinlemesine gösterilmektedir.

Mühendislik ile yeni tanışan ve geçmişinde meslek lisesi veya meslek yüksekokulu da olmayan okurlarımız için **temel düzeyde** değineceğimiz bu konu başlığında, fizik dersinde karşınıza gelme ihtimali olan **mantıksal kapılardan** bahsedeceğim. Lojik/mantıksal kapılar, değişken lojik girişleri vasıtasıyla yapılan işlemlerin sonuçlarını, **devrenin çıkışına aktaran** elektronik devrelerdir. Lojik kapıların, lojik devrelerin temel elektronikte gösterilmesinin sebebi **elektronik** ve **bilgisayarın temelini oluşturmasından ileri gelmektedir**. Lojik devrelerin girişindeki bilgi **lojik 1** veya **lojik 0** olarak adlandırılır. Önceki başlıklardan öğrendiğimiz üzere lojik 1 **var** lojik 0 **yok** demektir ve 1 değeri 5V, 0 değeri 0V olarak anlaşılmalıdır. Dijital sistemlerde lojik 1'in **doğruyu**, lojik 0'ın ise **yanlışı** ifade etmesi **pozitif lojik olarak adlandırılan bir kabulden ileri gelmektedir**. Eğer **kabul negatif lojik olursa** bu sefer lojik 0 **doğruyu**, lojik 1 ise **yanlışı** ifade edecektir. Lojik kapılar bir araya gelerek lojik devreleri oluşturmaktadırlar. Bir lojik kapı; *transistör*, *mosfet* ve *diyot* gibi elemanlar ile gerçekleştirilmektedir. Fakat günümüzde lojik kapılar entegre hâline getirilmiştir ve yaygın olarak kullanılmaktadır.

Elektronikte işlemlerin temelde **mantıksal kapılarla** gerçekleştirildiğini ve bu yüzden elektronik dolayısıyla da bilgisayar sistemlerindeki **en küçük program yapı taşının mantıksal kapılar** olduğunu bilmeliyiz. Üzerine sürekli işlem gerçekleştiren ve sonuç döndüren bu kapılar; evimizdeki beyaz eşyalarda, otomobillerde en tabii ki bilgisayar sistemlerinin olduğu her yerde kullanılmaktadır. Bir hesap makinesinin çalışması için kullanılan entegrede bile mantıksal kapılar bulunmaktadır. Şimdi, mantıksal kapıların çeşitlerini ve ne ifade ettiğini inceleyelim.

VE (AND) Kapısı

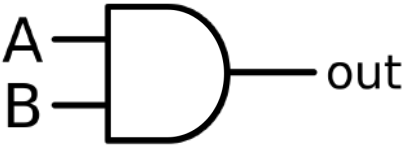
VE kapısını, **cebirdeki çarpma işlemi** olarak ifade ederiz. VE işlemine tabi tutulan bir sistemde yapılan işlem, bilginin **VELENMESİ** yani VE kapısı işlemine tabi tutulması başka bir deyişle **çarpılmasını** ifade eder. VE kapısı, çarpma işlemini gerçekleştiren bir kapıdır ve cebirdeki çarpma işleminin kuralları aynen geçerli olduğundan girişteki bilgilerden bir tanesinin SIFIR (0) olması durumunda çıkış **sıfır** olmaktadır. VE kapısına İngilizce karşılığı olarak **AND** denilmektedir. Programlama konularında göreceğimiz üzere AND yani VE kapısı işlem yaptığı devreye şartların hepsinin **aynı anda** sağlanması gerektiğini söyler. Buna bağlı olarak da VE kapısının aşağıdaki gibi bir **doğruluk tablosu** bulunur.

Doğruluk tabloları tüm kapılarda vardır ve Truth Table olarak bilinirler.

VE (AND) KAPISI		
Giriş A	Giriş B	Çıkış
0	0	0
1	0	0
0	1	0
1	1	1

Tablo 6-3: VE (AND) Kapısı Doğruluk Tablosu

Tablo 6-3, iki girişli tek çıkışlı bir VE doğruluk tablosunu göstermektedir. Cebirsel olarak incelediğimizde girişlerden herhangi birinin **0** olması durumunda sonuç, yani **çıkış 0** olmaktadır. Yalnızca her iki şartında sağlandığı durumda, yani giriş A ve giriş B'nin **1** olduğu durumda sonuç yani **çıkış 1** olmaktadır. Aşağıdaki şekilde ise **iki giriş tek çıkışlı bir VE kapısı diyagramı** gösterilmiştir. VE kapısına ait tüm devre diyagramlarında aşağıdaki gösterim karşınıza çıkacaktır. Mantıksal kapılara ait devre diyagramı yani entegre görünümleri birbirlerine oldukça yakın çizimler olduğu için kapıların çizimlerindeki **küçük farklılıkların birbirinden iyi ayırt edilmesi** gerekmektedir.



Şekil 6-2: VE Kapısı

Bir VE kapısına ait gerçekleştirilen işlem ise **Sonuç = Giriş A x Giriş B** şeklinde ifade edilir. Yani **Q=A.B** şeklinde matematiksel olarak ifade edilmektedir.

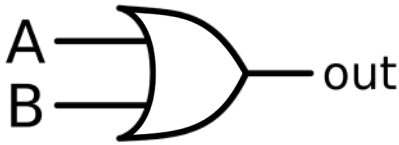
VEYA (OR) Kapısı

VEYA kapısı, cebirdeki **toplama işlemi** olarak ifade edilebilir. VEYA işlemine tabi tutulan bir sistemde yapılan işlem, bilginin **VEYALANMASI** yani VEYA kapısı işlemine tabi tutulması başka bir değişle **toplanmasını** ifade eder. VEYA kapısı toplama işlemini gerçekleştiren bir kapıdır ve cebirdeki toplama işleminin kuralları veya değişme özelliği gibi özellikleri aynen geçerlidir. **VEYA (OR)** kapısında işlem gerçekleştirilirken girişteki bilgilerin bir tanesi **1 olduğu müddetçe sonuç her zaman 1 çıkacaktır**. VEYA kapısının İngilizce karşılığı **OR** olarak bilinir. Ve yine programlama konularında özellikle karar yapılarında çokça kullanacağımız gibi bir sistemde VEYA işlemi yapıldığında şartlardan bir tanesinin sağlanıyor olması yeterlidir. VEYA kapısının **doğruluk tablosu aşağıdaki gibidir**.

VEYA (OR) KAPISI		
Giriş A	Giriş B	Çıkış
0	0	0
1	0	1
0	1	1
1	1	1

Tablo 6-4: VEYA (OR) Kapısı Doğruluk Tablosu

Tablo 6-4, **iki girişli ve tek çıkışlı bir VEYA doğruluk tablosunu** göstermektedir. Cebirsel olarak incelediğimizde girişlerden herhangi birinin 1 olması durumunda sonuç, yani **çıkış 1 olmakta**dır. Şartlardan bir tanesi sağlanması durumunda yani giriş A veya giriş B'nin 1 olması durumunda sonuç yani çıkış 1 olmaktadır. Aşağıdaki şekilde **iki girişli tek çıkışlı bir VEYA kapısı diyagramı** gösterilmiştir. VEYA kapısına ait tüm devre diyagramlarında aşağıdaki gösterim karşınıza çıkacaktır.



Şekil 6-3: VEYA Kapısı

Bir VEYA kapısına ait gerçekleştirilen işlem ise **Sonuç = Giriş A + Giriş B** şeklinde ifade edilir. Yani **Q=A+B** şeklinde matematiksel olarak ifade edilmektedir.

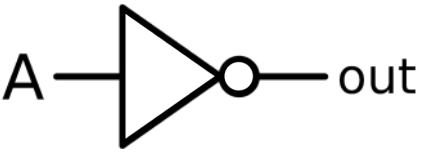
DEĞİL (NOT) Kapısı

DEĞİL kapısı **tek girişe** sahiptir ve girişindeki değeri **mantıksal tersine çevirerek çıkışa** verir. Yani girişteki değer lojik **1 ise çıkış lojik 0 olması anlamına gelmektedir**. **Tümleme devresi** olarak da bilinir ve ifade edilirken **üst çizgi** ile gösterilir. **Tablo 6-5**, **tek girişli bir DEĞİL kapısının doğruluk tablosunu** göstermektedir.

DEĞİL (NOT) KAPISI	
Giriş A	Çıkış
0	0
1	1

Tablo 6-5: DEĞİL (NOT) Doğruluk Tablosu

Q=üst çizgi A olarak gösterilir. Aşağıdaki şekilde, **DEĞİL kapısının devre diyagramlarındaki gösterimi** yer almaktadır.



Şekil 6-4: DEĞİL Kapısı

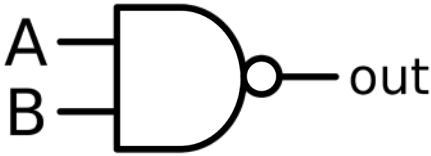
VE DEĞİL (NAND) Kapısı

VEYA DEĞİL kapısı, **VE kapısının arkasına Değil kapısı eklenmiş hâli**dir. Bu yüzden **Q=üst çizgi A.B** olarak gösterilir. VE DEĞİL, yani NAND **kapısına ait doğruluk tablosu Tablo 6-6 da verilmiştir**.

VE DEĞİL (NAND) KAPISI		
Giriş A	Giriş B	Çıkış
0	0	1
1	0	1
0	1	1
1	1	0

Tablo 6-6: VE DEĞİL (NAND) Doğruluk Tablosu

Doğruluk tablosundan da görüleceği üzere aslında yapılan işlem normal bir VE kapısı işlemi iken çıkan sonuç **DEĞİL işlemine tabi tutularak tersi alınıyor**. **VE DEĞİL kapısının devre diyagramındaki gösterimi** ise aşağıdaki şekilde gösterilmiştir.



Şekil 6-5: VE DEĞİL Kapısı

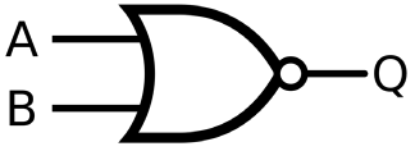
VEYA DEĞİL (NOR) Kapısı

VEYA DEĞİL kapısı, **normal VEYA kapısının arkasına Değil kapısı eklenmiş hâlidir**. Bu yüzden **Q=üst çizgi A+B** olarak gösterilir. VEYA DEĞİL, yani **NOR kapısına ait doğruluk tablosu Tablo 6-7 de verilmiştir**.

VEYA DEĞİL (NOR) KAPISI		
Giriş A	Giriş B	Çıkış
0	0	1
1	0	0
0	1	0
1	1	0

Tablo 6-7: VEYA DEĞİL (NOR) Doğruluk Tablosu

Doğruluk tablosundan da görüleceği üzere aslında yapılan işlem normal bir VEYA kapısı işlemi iken **çıkan sonuç DEĞİL işlemine tabi tutulmakta ve tersi alınarak çıkışa verilmektedir**. VEYA DEĞİL kapısının **devre diyagramındaki gösterimi ise aşağıdaki şekilde gösterilmiştir**.



Şekil 6-6: VEYA DEĞİL Kapısı

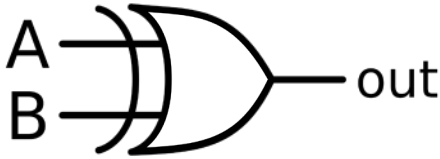
ÖZEL VEYA (XOR) Kapısı

ÖZEL VEYA kapısı, adından da anlaşılacağı gibi **özel yapıda kurulmuş bir lojik sistemdir**. Buraya kadar görmüş olduğumuz kapılardan farklı olarak bu kapıyı özel kılan eylem, **girişlerin birbiri ile karşılaştırılması ile sonucun bulunmasıdır**. İki giriş birbirinden farklı ise çıkışa lojik 1 değeri gönderilir. Q=A B ile gösterilir. Aradaki **daire içerisindeki artı sembolü özel VEYA kapısını simgeler**. ÖZEL VEYA (XOR) kapısına ait doğruluk tablosu **Tablo 6-8**'de verilmiştir. XOR kapısını kaynaklarda **EXOR** olarak da görebilirsiniz.

ÖZEL VEYA (XOR) KAPISI		
Giriş A	Giriş B	Çıkış
0	0	0
1	0	1
0	1	1
1	1	0

Tablo 6-8: ÖZEL VEYA (XOR) Doğruluk Tablosu

Doğruluk tablosundan da görüleceği üzere Giriş A ve Giriş B'deki değerler **farklı ise çıkış lojik 1 değerini almaktadır**. Aynı olduğu durumlarda sonuç daima **lojik 0** dönmektedir. ÖZEL VEYA kapısının **devre diyagramındaki gösterimi ise aşağıdaki şekilde gösterilmiştir**.



Şekil 6-7: ÖZEL VEYA Kapısı

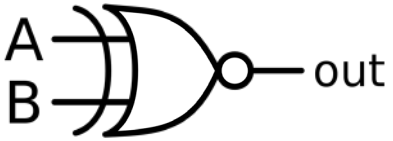
ÖZEL VEYA DEĞİL (XNOR) Kapısı

ÖZEL VEYA DEĞİL Kapısı, artık sizin de tahmin ettiğiniz gibi **ÖZEL VEYA kapısının çıkışına uygulanan bir DEĞİL kapısını ifade ediyor**. Bu yüzden girişler farklılık göstermesi durumunda çıkış artık **lojik 1 değil lojik 0** olmaktadır. Veya başka bir ifade ile girişlerin **birbirleri ile aynı olması durumunda çıkışın lojik 1 olduğu mantıksal kapıdır**. Q=üst çizgi A ⊕ B şeklinde gösterilir. XNOR kapısını kaynaklarda **EXNOR** olarak da görebilirsiniz. **Tablo 6-9**'da XNOR kapısına ait **doğruluk tablosu** verilmiştir.

ÖZEL VEYA DEĞİL (XNOR) KAPISI		
Giriş A	Giriş B	Çıkış
0	0	1
1	0	0
0	1	0
1	1	1

Tablo 6-9: ÖZEL VEYA DEĞİL (XNOR) Doğruluk Tablosu

Doğruluk tablosundan da görüleceği üzere Giriş A ve Giriş B'deki değerler **farklı olduğu zaman sonuç lojik 0, aynı olduğu zaman ise sonuç lojik 1 olarak çıkmaktadır**. ÖZEL VEYA DEĞİL kapısının **devre diyagramındaki gösterimi ise aşağıdaki şekilde gösterilmiştir**.



Şekil 6-8: ÖZEL VEYA DEĞİL Kapısı

Boolean Matematiği

Boole Cebiri olarak da bilinen bu sistem, İngiliz matematikçi **George Boole** tarafından 1850'li yıllarda geliştirilmiştir. Geliştirilmiş olan bu cebir sayesinde sayısal devrelerin tasarımı oldukça kolay bir hâl almıştır. Elektroniğin kullandığı dilden farklı olmayarak **boolean matematiği de yalnızca 1 ve 0 değerlerini kullanmaktadır**. Mantıksal kapılar ve devreler 1850'li yıllarda George Boole tarafından ele alındığında, lojik devreleri **yalınlaştırmak** ve **bir sisteme oturtmak istenmiş** ve bugünkü **Boolean Matematiği** oluşturulmuştur.

Boolean matematiği, ikili sayı sistemi üzerinde yapılan işlemlere **birtakım yeni kurallar** tanımlamıştır. Oluşturulan onca karışık lojik devrenin ifadelerinin **bozulmaya uğramadan** ve **herkes tarafından okunabilir olması adına belirli kurallar bütünü** ile **genişletilebilen** ve **sadeleştirilebilen kullanımına olanak** sağlamıştır. Bu işlemi gerçekleştirirken dikkate alınan **Boolean Matematiği kuralları aşağıdaki tablodaki gibidir**.

Boolean Matematiği Kuralları		
Kurallar	Değerler	Sonuç
Kural 1	0+A	A
Kural 2	1+A	1
Kural 3	A+A	A
Kural 4	A+A üst	1
Kural 5	0.A	0
Kural 6	1.A	A
Kural 7	A.A	A
Kural 8	A.A	0
Kural 9	A	A

Tablo 6-10: Boolean Matematiği Kuralları

Yukarıdaki tabloda **iki değişkenli boolean matematiğinin kurallarını** inceledik. **İki veya daha fazla değişkenli boolean matematiği kuralları ise aşağıdaki gibidir**.

Boolean Matematiği Kuralları	
Kurallar	**Değerler**
A + B = B + A	Toplama işlemi yer değiştirme kuralı
A . B = B . A	Çarpma işlemi yer değiştirme kuralı
A + (B + C) = (A + B) + C	Toplama işlemi birleşme kuralı
A . (B.C) = (A.B).C	Çarpma işlemi birleşme kuralı
A . (B+C) = A.B + A.C	Dağılma Kuralı
A + B.C = (A + B).(A+C)	Dağılma Kuralı
A + A.B = A	Fazlalık Kuralı
A . (A+B) = A	Fazlalık Kuralı

Tablo 6-11: Boolean Matematiği Kuralları

Boolean matematiği ile ilgili olarak bu kuralların bilinmesi yeterli olacaktır. Devre tasarımlarında ya da lojik sistemler ile uğraşırken yaşanacak karışıklıkları **boolean matematiği ile indirgemek** oldukça kullanışlı bir yöntemdir.

De Morgan Teoremi

İngiliz matematikçi ve mantıkçı olan **Augustus De Morgan** tarafından geliştirilmiştir. De Morgan teoreminin amacı görmüş olduğumuz mantık kapılarından **VE**, **VEYA** ile bağlanmış olan iki sistemin **tersinin** (*değilinin*) **indirgenmesidir**. Buna göre **De morgan** teoreminde kural **p** veya **q** önermelerinin değili; p'nin değili ile Q'nun değilinin VE işleminden geçmesine p ve q'nun değili p'nin değili ile q'nun değilinin VEYA işleminden geçmiş sonucuna **eşittir**. Bu karışık ifadeyi aşağıdaki tablodaki gibi **daha okunur** bir şekilde ifade edebiliriz.

De Morgan Kuralı
(p VE q)' DENK p' VEYA q'
(p VEYA q)' DENK p' VE q'

Tablo 6-12: De Morgan Kuralı

De Morgan kuralının **doğruluk tablosunda ispatı ise aşağıdaki gibidir**.

Önermeler	**(p V q)' ≡ p' ^ q'**		**(p ^ q)' ≡ p' V q'**		
p	q	(p V q)'	P'^ q'	(p ^ q)'	P' V q'
1	1	0	0	0	0
1	0	0	0	1	1
0	1	0	0	1	1
0	0	1	1	1	1

Tablo 6-13: De Morgan Teoremi Doğruluk Tablosu

De Morgan teoremini kaynaklarda ve internette sıklıkla karşılaşabileceğiniz bir örnek ile pekiştirelim.

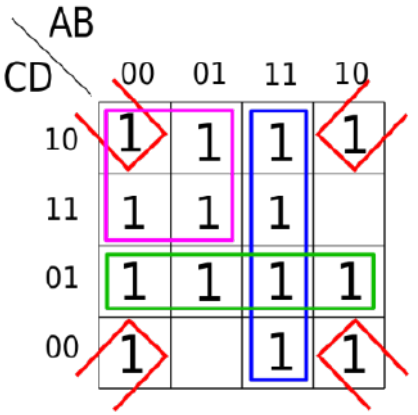
(p ^ q')' ^ (p'V q') önermelerini De Morgan teoremi kuralına göre sadeleştirelim.

- (p ^ q')' ^ (p'V q')
- (p ^ q')' ≡ (p'V q) **De Morgan Kuralı**
- (p'V q) ≡ (p'V q') ≡ p'V (q ^ q') **Dağılma Özelliği**
- p'V (q ^ q') ≡ p'V 0 ≡ p'

Karnaugh Haritaları

Kitabımızın çeşitli bölümlerinde konu **sadeleştirmeye** ve **optimizasyona** gelince sürekli mühendislik disiplininin günümüz dünyasındaki en önemli çalışma konularından birisinin **maliyeti düşük sistemler** geliştirmek olduğundan bahsediyorum. Buradaki maliyet, tahmin edeceğiniz üzere parasal maliyetin yanı sıra **düşük boyut yüksek hız düşük işlem** adımı ile fazla işlem sayısıdır. Dijital elektronikte de en önemli konulardan bir tanesi **düşük maliyetli devreler** oluşturulmasıdır. Boolean matemetiği ve De morgan teoreminde de gördüğümüz üzere **sadeleştirme oldukça önemli kazanç sağlayan bir işlemdir**. Devrelerin en sade hâliyle oluşturulması sonrasında, devre elemanları okunurluğunu da artırmaktadır. Sadeleştirme işlemlerinin bir tanesi de **Karnaugh Maps** diye bilinen **karno haritası yöntemi** ile gerçekleştirilmektedir. Karno yöntemi, temelde Boolean fonksiyonlarının indirgenmesinde yaşanan tekrarlı adımları ve hataların olmasına izin vermemek için basit bir yöntem sunar. **Karno haritası** (*K-Maps*) olarak da bilinir. Aşağıdaki görselde **örnek bir karno haritası** yer almaktadır.

Karnaugh diyagramı 1950'li yıllarda **Edward Veitch** tarafından oluşturulmuş fakat 1953 yılında Maurice Karnaugh tarafından geliştirilerek bugünkü **sadeleştirmede kullanılan yöntemsel hâlini almıştır**.



Şekil 6-9: Karnaugh Haritası

Karno haritaları yukarıdaki şekilde de görüldüğü gibi **karelerden oluşmaktadır**. Her kare **standart çarpım** veya **standart toplam** terimini ifade eder. Karno haritasındaki tabloyu oluşturmak için gerekli olan kare sayısını belirlemek için **giriş değişkenlerinin sayısına ihtiyaç duyulur**. **N giriş** *değişken sayısını*, **2N** ise *kare sayısını* verir. Her bir kareye sıfırdan başlamak suretiyle **soldan sağa doğru numara** verilir ve satır bittiğinde bu sefer alttaki satısın **solundan sağ tarafa doğru** numaralar verilir.

Mantık kapılarında bahsetmiş olduğum **lojik 1 aktif kabul edilmesi durumunda karno haritasındaki hedef en çok 1 karelerini gruplamaktır**. Lojik 1 aktif kabul edildiği bir sistemde kutuların içerisindeki **0 değerleri dikkate alınmaz**. Tam tersi olur ve **lojik 0 aktif olarak kabul edilirse de bu sefer hedefimiz en çok 0 kutularını gruplamaktır**. Genellikle derslerdeki örneklerde lojik 1 aktif olarak kabul edilir ve yukarıdaki örnekte de olduğu gibi hedef **en çok 1 değeri gruplamaktır**. Gruplama yapılırken seçilen kutu sayısı 1, 2, 4, 8, 16 ve katları şeklinde olmalıdır.

Karno haritaları üzerinde gruplama yapılırken **çapraz gruplama yapılamaz**. Grupların **yan yana** veya **alt alta** olması gerekmektedir.

Aşağıdaki şekillerde iki, üç ve dört değişkenli karno haritaları gösterilmiştir.

- İki değişkenli karnaugh haritası **(A, B) 2^2 = 4 kutu** olmalıdır.

B \ A	A'	A
B'	A'.B'	A.B'
B	A'.B	A.B

Şekil 6-10: 2 Değişkenli Karnaugh Haritası

- Üç değişkenli karnaugh haritası **(A, B, C) 2^3 = 8 kutu** olmalıdır.

C \ AB	A'B'	A'B	AB	AB'
C'	A'.B'.C'	A'.B.C'	A.B.C'	A.B'.C'
C	A'.B'.C	A'.B.C	A.B.C	A.B'.C

Şekil 6-11: 3 Değişkenli Karnaugh Haritası

- Dört değişkenli karnaugh haritası **(A, B, C, D) 2^4 = 16 kutu** olmalıdır.

CD \ AB	A'B'	A'B	AB	AB'
C'D'	A'.B'.C'.D'	A'.B.C'.D'	A.B.C'.D'	A.B'.C'.D'
C'D	A'.B'.C'.D	A'.B.C'.D	A.B.C'.D	A.B'.C'.D
CD	A'.B'.C.D	A'.B.C.D	A.B.C.D	A.B'.C.D
CD'	A'.B'.C.D'	A'.B.C.D'	A.B.C.D'	A.B'.C.D'

Şekil 6-12: 4 Değişkenli Karnaugh Haritası

ALGORİTMA VE PROGLAMLAMAYA GİRİŞ

7

Ülkemizde, programlanın oldukça benimsenmiş bir şey olduğunu söyleyebiliriz. Üniversitelerde okutulan dersler kapsamında, internet kaynakları ve çeşitli basılı kaynaklarda programlama hakkında epeyce bilgi edinilmesi mümkündür. Günümüzde programlama öğrenmek imkân olarak bu kadar kolay hâle gelmişken ne yazık ki öğrencilerde veya klavye kahramanlarında oturmayan birçok **programlama prensibi** vardır. Bunun en büyük sebebi insanların programlamaya algoritma ile başlamak yerine proje üzerinde kendilerini yetiştirmeye başlamaları ve benzer sebeplerdir. **Algoritma** ve **akış diyagramları kavramı programlamaya başlamadan önce sindirilmesi gereken önemli kavramlardır**. Algoritmaların anlatıldığı çoğu kaynak insanlara sıkıcı gelmekte ve genelde geçiştirilmektedir. Tavsiyem, **hangi kaynak olursa olsun karşılaştığınız algoritma kaynaklarının incelenmesidir**. Kitabımızın bu bölümünde, öncelikle **algoritmanın ne demek olduğuna değineceğiz**. Daha sonra akış diyagramlarına, şekillerin anlamlarına ve örnek problemlere değineceğiz.

ALGORİTMA

Bilgisayar dünyasında genelde **bir problem ve buna bağlı olarak da geliştirilen birden fazla çözüm bulunur**. Algoritmalar, problemlerin çözülebilmesi için geliştirilen **çözüm yolları**dır. Yukarıda da ifade ettiğim gibi öğreneceğimiz programlama dillerinin hepsinin temelinde **algoritma** yatmaktadır. Problem çözümünde öncelikle matematikten de hatırlayacağımız üzere çözüm yolları belirlenir ve sıraya konur. Bu sıraya göre işlemler takip edilerek **en optimum çözüme ulaşılır**. Algoritma mantığı, bir kez kavrandıktan sonra üzerine hangi programlama dilini katacağınız size kalmıştır. Programlama dilleri gibi **algoritmalar da gelişmektedir**. Algoritmaların gelişmesi elbette programlama konseptindeki gibi versiyon güncellemesi tarzında bir gelişim değildir. Örneğin programlamadaki soyut tasarım desenleri yani örüntüler birer algoritmadır ve programlamanın dünyada yaygınlaştığı yıllarda nasıl tasarım deseni veya güvenli tasarım deseni kavramları çok az veya hiç duymuyorken şimdilerde sıklıkla duyuyorsak algoritmalardaki gelişimin ne şekilde olduğunu da anlayabiliriz.

Mühendislik fakültelerinde *programlamaya giriş veya programlama temelleri* gibi isimlerle okutulan derslerde **algoritmadan** ve **akış diyagramlarından** bahsedilir. Algoritmalar **sözel** bir şekilde işlenir ve problem çözme alıştırmalarına da bu şekilde başlanır. Kod yazma aşamasına geçmeden önce yapılmak istenen işi sözel olarak ifade eden tekniğe **pseudo code** denilmektedir.

Örnek Algoritma Sorusu: Vize ve final notunu kullanıcıdan aldıktan sonra geçme notunu hesaplayan programın algoritmasını oluşturunuz.

Cevap: Bu sorunun cevabı için kod yazmayacağız çünkü henüz bir programlama dili bilmiyoruz. Algoritma konusuna uygun bir şekilde cevabı tamamen sözel olarak ifade edecek adımları yazacağız.

Başla
Kullanıcının adını gir
Kullanıcının vize notunu gir
Kullanıcının final notunu gir
Vize notunu 1.40 ile çarp
Final notunu 1.60 ile çarp
Çarpımlardan elde edilen değerleri topla
Eğer elde edilen toplam 60'dan büyükse ekrana Geçti yaz
Eğer elde edilen toplam 60'dan küçükse ekrana Kaldı yaz

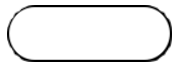
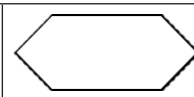
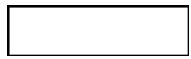
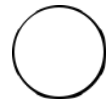
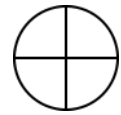
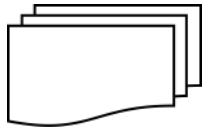
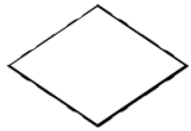
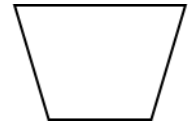
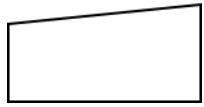
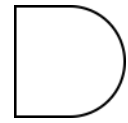
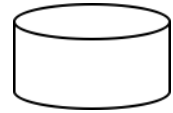
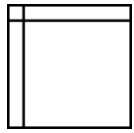
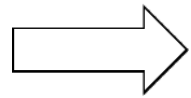
Evet, örneklendirdiğimiz gibi algoritma bir problemi çözmek için tasarlanan yoldur ve pek çok farklı yöntem olabilir. Algoritmaların pseudo code şeklinde ifade edilmeleri oldukça anlaşılırdır fakat **görseller ile diyagram şeklinde de ifade edilebilmektedirler**. Bu ifadelere ise **akış diyagramı** (*flow chart*) denilmektedir.

Akış Diyagramları (Flowchart)

Algoritmanın tanımını yaparken sözel ifadelerden oluştuğunu ve basit kurgulandığını söylemiştik. Akış diyagramları ise **algoritmaların önceden belirlenmiş şekiller ile ifade edilme şeklidir**. Akış diyagramının bir diğer adı ise **akış şeması** olarak bilinir. Akış diyagramları bir algoritmanın sunulması aşamasında **daha efektif** olacağı için kullanılır. **Önceden belirlenmiş akış diyagramı şekilleri programcılar veya yazılım mimarları tarafından bilinirler**. Akış şemalarının öğrenilmesi en az algoritma kavramı kadar önemlidir çünkü bir yazılım geliştirilmeye başlanmadan önce algoritması çıkartılması sonrasında ise akış diyagramı oluşturulmalıdır. Akış diyagramı hem yazılım projesindeki tüm süreçleri daha net görülebilir kılmakta hem de projenin farklı bölümlerinde görev yapan geliştiriciler için ortak dili konuşabilmeye olanak sağlamaktadır. Akış diyagramı üzerinden yapılacak yorumları **geliştirme aşamasına geçmeden önce pek çok sorunun ortadan kaldırılmasını sağlamaktadır**.

Akış diyagramındaki şekiller ve şekillerin ifade ettiği durumlar (*statement*) bilgisayarın programı çalıştırdığı sıra ile aynı şekilde ilerler. Her şeklin bu yüzden **önceden hazırlanmış bir tasarımı** bulunur ve akış diyagramları bu şekilde standartlaşan hâliyle oluşturulur. Aşağıdaki tabloda **akış diyagramlarına ait şekiller ve açıklamaları** yer almaktadır. Office paket programının **şekiller bölümünde tüm akış şemalarını bulabilirsiniz**. Ayrıca pozitif isimli akış diyagramı yazılımı ile de bilgisayar ortamında akış diyagramları oluşturabilir bu sayede işleyişi daha kolay kavrayabilirsiniz.Örnek problemlere geçmeden önce aşağıda yer alan bir problemin çözüm aşamalarının bilinmesi ilerleyen bölümler için yararlı olacaktır. Bir problemin çözüm aşamaları;

1. Problemin tanımı
2. Çözüm yöntemlerinin araştırılması
3. Uygun çözümün belirlenmesi
4. Algoritmanın hazırlanması
5. Akış şemasının hazırlanması
6. Kodlama
7. Deneme / test
8. Kullanıma sunumu

Başlangıç/Bitiş Sürecin nerede başladığı ve nerede son bulduğunu göstermek için kullanılır.		**Hazırlık / Döngü**	
Eylem, Süreç, İşlem İşlemlerin gerçekleştirildiği sembodür.		**Bağlama** Akışın kaldığı yerden devam etmesi durumunda kullanılır.	
Belge, Ekran Mesajı Basılı bir belgeyi, raporu veya ekran mesajını ifade eder.		**Veya**	
Birden Fazla Belge		**Alt Yordam/Önceden Tanımlı İşlem** Daha büyük bir süreç içerisinde gömülü belirli bir görevi gerçekleştiren eylemler dizisi.	
Karar Bloğu Bir kararın alınması gerektiği durumlarda kullanılır.		**Manuel Döngü** El ile durdurulana kadar tekrar edecek döngü.	
Giriş/Çıkış Kullanıcının el ile girdi yapabileceği durumlarda kullanılır.		**Gecikme** Akıştaki işlemde meydana gelecek gecikmeyi ifade eder.	
Veri Depolama Verilerin kaydedildiği adımı ifade eder.		**Veri Tabanı** Gerektiğinde bilgileri veren kayıt kütüğü.	
Dahilî Depolama Akış sırasında bilgilerin bellekte saklanması durumunda kullanılır.		**Bağlantı** Akış yönünü ve diğer semboller arasındaki bağlantıyı gösterir.	

Tablo 7-1: Akış Diyagramları

ÖRNEKLER

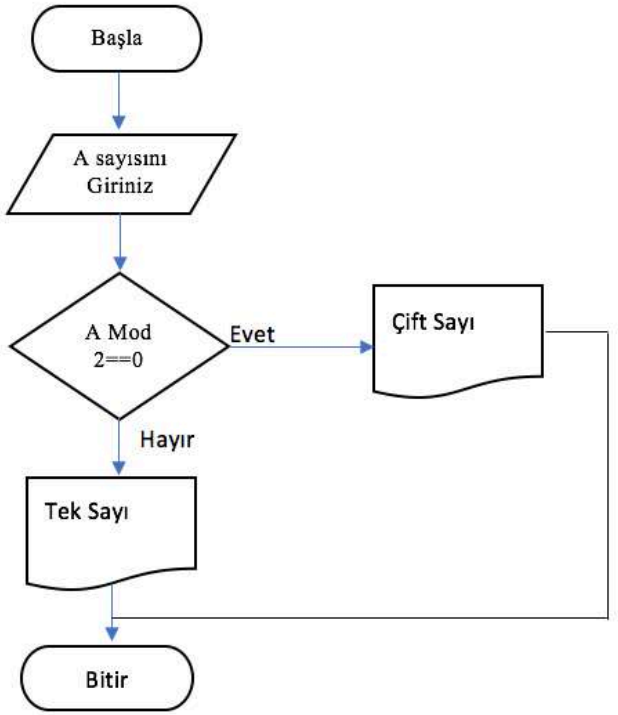
Bu bölümde **akış diyagramı** ve **algoritmalar** ile ilgili yaygın örnekleri inceleyeceğiz.

Örnek 1: Kullanıcı tarafından girilen bir sayının tek mi çift mi olduğunu tespit ederek akış diyagramı ve algoritmasını oluşturunuz.

Çözüm 1:

Algoritma:
Adım 1: Başla
Adım 2: Bir Sayı Giriniz (A)
Adım 3: Eğer A sayısının 2'ye bölümünden kalan 0 ise ekrana çift yaz değilse tek yaz (A mod 2 = 0)
Adım 4: Bitir

Akış Diyagramı:



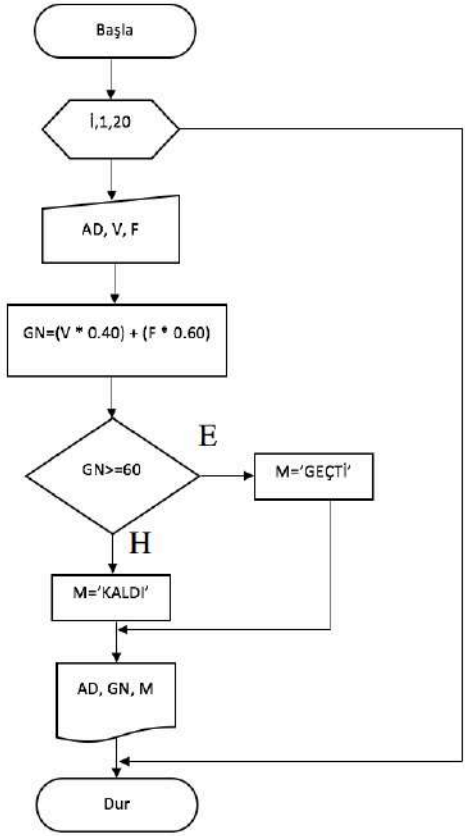
Örnek 2: Dışarıdan girilen 20 öğrencinin adı soyadı, vize, final notu kullanıcılar tarafından girilecek ve geçme notu ile birlikte geçip kalma durumu mesaj olarak yazılacak şekilde algoritma ve akış diyagramını hazırlayınız.

Çözüm 2:

Algoritma:

1. Başla
2. Ad Soyad (AD), Vize (V), Final (F) notlarını giriniz.
3. Geçme notu hesaplanır. GN= (V * 0.40 + F * 0.60)
4. Geçme notu >=60 ise M='GEÇTİ' değilse M='KALDI'
5. Dur

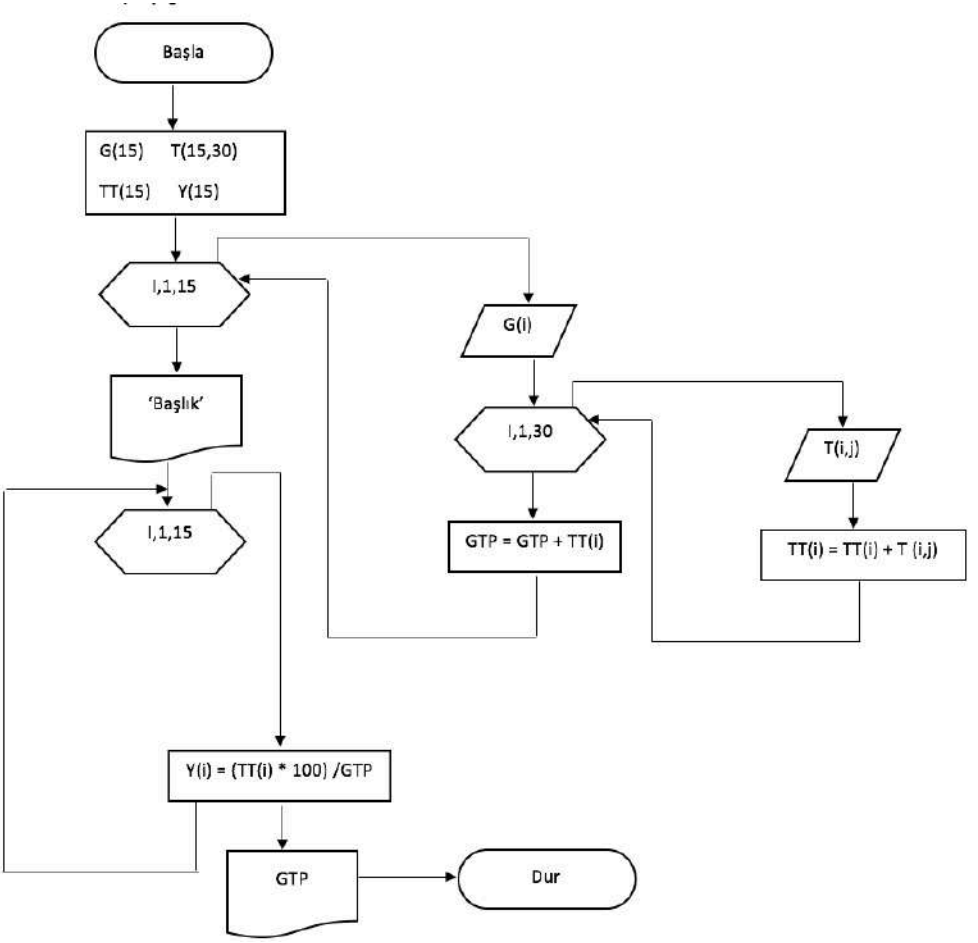
Akış Diyagramı



Örnek 3: 15 adet gazetenin adı, atanmış değerdir. Her bir gazetenin 1 aylık tirajlarını girerek ilgili dizileri oluşturarak akış diyagramını hazırlayınız. Akış diyagramında iki boyutlu dizi kullanılacaktır. Dizinin bir tanesi gazeteler için diğeri 30 günlük tirajlar için kullanılacaktır. **G=Gazete, T=Tiraj, TT=Toplam Tiraj, Y=Yüzde, GTP=Genel Toplam**

Çözüm 3:

Akış Diyagramı



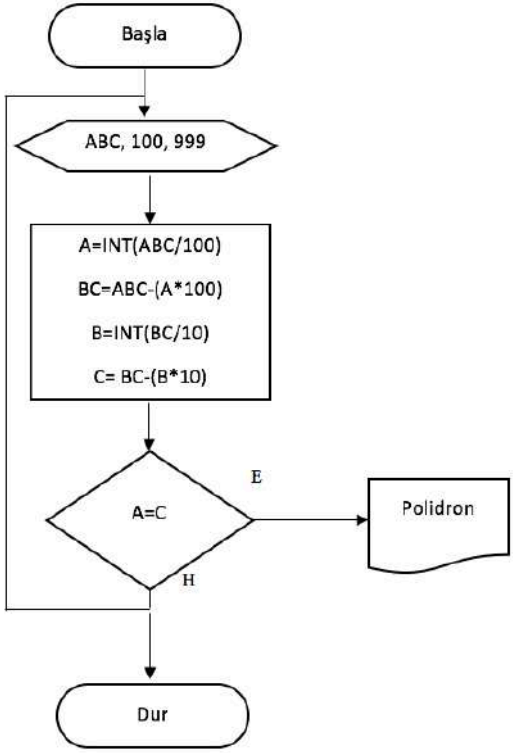
Örnek 4: İlk harfi ile son harfi veya ilk rakamı ile son rakamı aynı olan sayı veya karakterlere **polidron sayılar** veya **polidron karakterler** denir. Üç basamaklı sayılar üzerinde polidron olan sayıları bulan algoritma ve akış diyagramını hazırlayınız.

Çözüm 4:

Algoritma

1. Başla
2. ABC üç basamaklı sayılar döngüsünü kur
3. A=INT(ABC/100) işlemi ile A'yı ayır.
4. BC=ABC-(A*100) işlemi ile B'yi ayır.
5. B=INT(BC/10) işlemi ile C'yi ayır.
6. C=BC-(B*10)
7. A=C ise 'Polidron' değilse döngüye tekrar gir.
8. Bitir

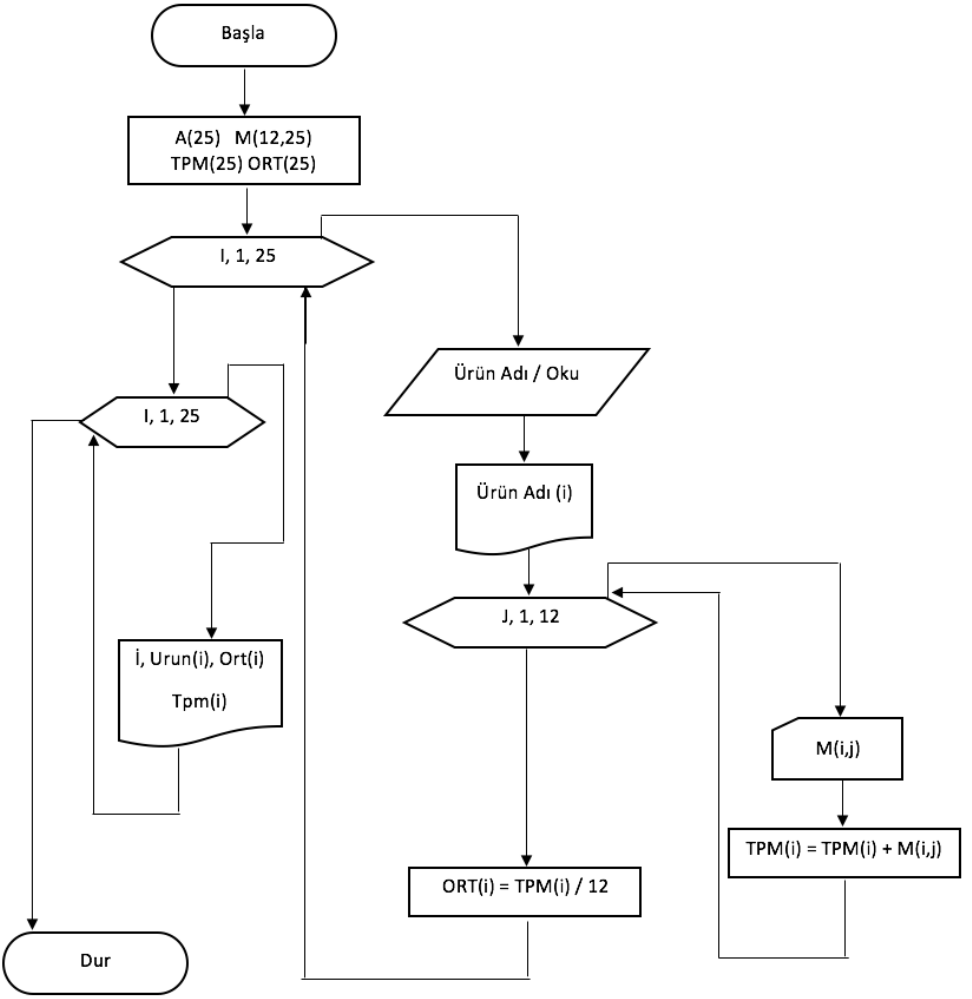
Akış Diyagramı



Örnek 5: 25 adet ürünün adı atanmış değerdir. Her ürünün 12 ay boyunca her ay için toplam üretim miktarlarının kullanıcı tarafından girilerek sonuçta Sıra, Ürün Adı, Yıllık Toplam ve Aylık Ortalama Üretim şeklinde liste verecek olan programın akış diyagramını hazırlayınız. **A=Ürün Ad, M=Miktar, TPM=Toplam Miktar, ORT=Ortalama**

Çözüm 5:

Akış Diyagramı



Programlama Dilleri

*Algoritma ve akış diyagramları*nı işlerken programlamaya geçmeden önceki süreç olduklarından bahsetmiştik. *Peki, programlama ve programlama dili nedir?* Biraz da bu konu üzerine konuşalım. Programlama aslında en temel tanım olarak **bilgisayar dünyasındaki problemlerin matematiksel olarak çözüme kavuşturulması işleminin bilgisayar aracılığı ile gerçekleştirilmesindeki arayüzdür**. Buradaki arayüz insan ile bilgisayar (*mikroişlemci*) arasındaki **iletişimi sağlayan araç** anlamına gelmektedir. *Peki, programlama hangi ortam üzerinden gerçekleştirilmekte?* Biraz da bu konuya açıklık getirecek olursak bir programın yazıldığı ortama **geliştirme ortamı**, bu ortamda uyulan standartlara ve metodolojilere göre kullanılan yapıya ise **programlama dili** diyebiliriz. Bu tanım tamamen *bana ait* olmakla birlikte taşları yerine koyduğumuzda oldukça basit ve akılda kalıcı bir tanımdır. Daha akademik bir tanım yapmak gerekirse de programlama dilleri geliştirme ortamında programların yazılmasına olanak sağlayan bir notasyondur. Programlama dillerinde **komutlar**, **döngüler**, **karar yapıları**, **fonksiyonlar** ve **diziler** gibi çeşitli kendine has yapı taşları bulunur. Komutlar genellikle İngilizcedir ve konuşma dilindeki terimlerden oluştuğu için **akademik bir İngilizce bilinmesine gerek yoktur**. Ancak yine de iyi bir İngilizceye sahip geliştirici, çıkabilecek sorunları daha iyi yorumlayacağından çözüme de daha hızlı ulaşacaktır.

Programlama dilleri temelde üç seviye grupta incelenmektedir:

1. **Düşük seviyeli diller**
2. **Orta seviyeli diller**
3. **Yüksek seviyeli diller**

Düşük seviyeli diller **makine dillerini içerir** ve **anlaşılması çok güçtür**. Makine dilinden biraz daha gelişmiş olan diller orta seviyeli diller olarak adlandırılırken günümüzde kullanılan popüler programlama dilleri yüksek seviyeli diller grubuna girmektedir.

Bilgisayarlar önceki konulardan hatırlayacağınız üzere **1** ve **0**'lardan oluşan sinyaller ile işlem gerçekleştirirler. Makine seviyesindeki diller tamamen bu yöntemle çalışırlar ve yazılan programın çalışması bu esasa dayanır. Yüksek seviyeli dillerde ise görsel programlamayı da düşünürsek yazılan programın **makine diline çevrilmesi gerekmektedir**. Bu çevrim işini, günümüzde **derleyiciler** (*compiler*) üstlenmektedir. Ayrıca kütüphaneler (*framework*) üzerinde de çeşitli **çevrim metotları** bulunur. Örneğin, .net framework platformunda **JIT**, **CLR** ve **MSIL** yordamları bu işi üstlenmektedirler.

.Net framework çatısı altındaki bir programlama dili ile geliştirilen uygulamanın **.exe** uzantılı dosyası, normal şartlarda doğrudan çalışabilir bir dosya değildir. Bu bir **çıktı**dır ve **MSIL** (*Microsoft Intermediate Language*) dosyası olarak bekler. Bu dosyaya makine diline göre **assembly** de diyebiliriz. **CLR** (*common language runtime*), yani **ortak dil çalışma zamanı** ise bilgisayarda bekleyen bu uygulama çalıştırıldığında **JIT** (*Just-In-Time*) derleyicisini çağırır ve işlemci JIT kodun çağrılan bölümlerini assembly kodlarına dönüştürür. Bu yapılar sayesinde uygulama .net framework çatısı altında **her yerde çalışır** ve **taşınabilir** özelliğe sahip olacaktır.

Programlama Dillerinin Tarihçesi

Bilgisayarların hayatımıza girdikleri günden bugüne pek çok komut icat edilmiştir. Bu komutlar, donanım seviyesindeki devrelerde çalışan komutlardan günümüz son teknolojisinde kullanılan yazılımlarda bulunanlara kadar hep **aynı mantık** üzerine kurulmuşlardır. Programlama konseptinin çıkmaya başladığı ilk yıllar olan 1960'lı yıllarda ihtiyaç olan komutlar, oldukça **basit** komutlardı. Fakat günümüzdeki komutlar, **nesneye dayalı programlama** dillerinde oldukça karmaşık hâller almakta. İlk programlama işlemi, fiziksel olarak **kartlar** ile gerçekleştiriliyordu. Buna örnek olarak "**Formula Translating**" kartları, yani **Fortran kartlarını** görebilirsiniz. Fiziksel programlamanın yerini zamanla **elektrik sinyalleri ile yapılan programlama** almıştır. Bilgisayarın komutlara karşı yaklaşımını belirleyen elektrik sinyalleri *düşük voltaj* ve *yüksek voltaj* olarak bugünün 1 ve 0'larının temelini atmıştı. **Düşük voltaj 0**'ı ifade ederken **+5 voltaj 1**'i ifade etmektedir ve makine dilinin temelini oluşturmaktadır. Elbette bu şekilde kodlama yapmak bugünün deyimi ile **hard-code** ile uğraşmak elektronik devrelerin her program için **yeniden**

kurulması anlamına geliyordu. Bu sorunu ortadan kaldırabilmek için geliştirilen konsept ile programlar birtakım kavramların etrafında geliştirilmeye başlandı. Her bir program için donanımı yeniden kurma maliyeti ve diğer sorunları ortadan kaldırmak için programların tek bir komut seti yerine **daha küçük parçalar** hâlinde yazılması önerildi. Bu küçük parçaların tekrar tekrar kullanımını amaçlayan konsepte **yordam** adı verildi. Yordam kavramının ortaya çıkmasıyla beraber *matematiksel işlemler, mantıksal işlemler, döngüler, değişkenler, nesneler* ve *karşılaştırma operatörleri* gibi kavramları meydana geldi. Yazılan kodun tekrar kullanılması konseptinde ise **kütüphane kavramı** ortaya çıkmıştır. Programlama dünyasına giriş yapılan bu yıllardan biri olan **1957** yılında IBM firması, düşük seviyeli bir dil olan yani makine diline yakın bir seviyede çalışabilecek **FORTRAN programlama dilini** ortaya çıkarmıştır. FORTRAN'ın çıkması ile birlikte mantıksal işlemler, döngüler, sayısal değerlerin rahat kullanımı gibi **olumlu sonuçlar** ortaya çıktı. 1959 yılında FORTRAN programlama dilinin iyi özelliklerinin alınarak geliştirilmesi sonucu, **temel giriş çıkış** (*Input-Output*) işlemlerini de destekleyen **COBOL programlama dili** ortaya çıkmıştır. Daha sonra sürekli olarak; **eski dilin üzerine koy yenisini çıkart** mantığı ile COBOL ve FORTRAN programlama dillerinin güçlü özellikleri alınarak **1969** yılında **PASCAL programlama dili** çıkmıştır. Pascal programlama dili bugünkü C++ .Net'ten alışkın olduğumuz **pointer** kavramını da beraberinde getirmiştir. Pointer kavramı, hafızadaki adresler üzerinde işlem yapmaya olanak veren işaretçilerdir. **1970-1972** yıllarında ise Pascal programlama dilindeki hataların büyük bir kısmını gidererek **C programlama dili** çıkmıştır. Ayrıca C dilinin günümüzde hâlen bu kadar popüler olmasının sebebi de **Unix işletim sistemini** yazmak için kullanılmış olmasıdır. Kronolojik olarak da gelişen programlama dillerinin her biri daha düşük seviyede ve daha anlaşılır olarak piyasada yerini almıştır. C dili çevresel işlemleri güçlü bir şekilde sağlaması sebebiyle genellikle **işletim sistemi** geliştirilmesinde kullanılmıştır. Programlama trendi, modüler bir yapıya doğru ilerlemesi sebebiyle C dilinden sonra **nesne tabanlı OOP** (*Object Oriented Programming*) veya **nesneye yönelik olarak bilinen programlama dilleri** çıkmıştır. OOP modeli ile C dilinin özelliklerinden doğan yeni programlama dilinin adı ise **C++** olmuştur. C++ programlama dilinden sonra ise bugünkü kullanım alanları yadsınamaz olan **Java** ve **C#** gibi diller ortaya çıkmıştır.

Teknolojinin geliştiği her platformda olduğu gibi programlamada da **performans** önemli bir yer aldığından Java ve C# dillerinde bulunan **Çöp Toplayıcı** (*Garbage Collection*) *performans arttırıcı* ve *gereksiz kullanımı temizleme* özelliği sayesinde **nesne tabanlı programlama dillerinin önemini bir kez daha vurguladı**. Ticari başarısını göz ardı edemeyeceğimiz Microsoft firması 2000 yılında **.Net Framework** geliştirme ortamını sundu ve yaklaşık **35 tane programlama dilini tek çatı altında topladı**. Microsoft, .Net Platformunda yer alan **C#.Net** ve **Java** programlama dilleri bugünkü yüksek seviyeli programlama dillerine birer örnektir.

Programlamada İşlemler

Bilgisayarlar, problemleri **matematiksel ifadeleri** kullanarak çözerler. Evet, bu tanımı algoritma ve akış diyagramları konularında da söylemiştik. Daha öncede belirttiğimiz gibi bilgisayarlar 1 ve 0 sinyalleri ile işlemleri gerçekleştiriyorlar. Lojik kapıları işlediğimiz bölümle de toplayıcı kapılarının aritmetik ve mantıksal işlemleri ne şekilde uyguladığı hakkında fikir sahibi olmuştuk. Bu bölümde ise programlamadaki ve dolayısıyla **programlama dillerindeki matematiksel ifadelere** yer verelim. Bilgisayar işlemcisini (CPU) insan beynine benzetmekle yakın bir yaklaşımda bulunabiliriz. Çünkü bilgisayar programları:

1. **Matematiksel (aritmetik) işlemleri**,
2. **Karşılaştırma (karar) işlemlerini**,
3. **Mantıksal (lojik) işlemleri gerçekleştirmektedir**.

Matematiksel İşlemler

Matematiksel işlemler altında **genel aritmetik işlemler** yer almaktadır. Bunlar bilindiği üzere *toplama, çıkarma, bölme, çarpma gibi dört işlemler ve logaritmik, üstel, trigonometrik* gibi fonksiyonel işlemlerdir. Aşağıdaki tabloda matematiksel işlemlerin programlamada ne şekilde gerçekleştirilmesi gerektiği gösterilmiştir.

Matematiksel İşlem	Normal Matematik	Programlamada Matematik
Üs Alma	a^2	a^2
Toplama	a+b	a+b
Çıkarma	a-b	a-b
Çarpma	a.b	a*b
Bölme	¼	1/4

Matematikten hatırlayacağımız bir diğer özellik ise **işlem önceliği** sırasıdır. Normal matematikteki gibi programlamada da **parantezlerin önceliği** bulunur. Eğer parantezlerin önceliği ile aynı özelliğe sahip işlemler söz konusu ise bu sefer programlama işlemlere **soldan sağa doğru öncelik** vermektedir. Aşağıdaki tabloda işlem **öncelik sıraları** gösterilmiştir.

Programlamada İşlem Öncelikleri		
Sıra	**İşlem**	**Programlama**
1	Parantez Önceliği	((…..))
2	Üs Alma	a^b
3	Çarpma ve Bölme	a*b ve a/b
4	Toplama ve Çıkarma	a+b ve a-b

Karşılaştırma İşlemleri

Programlama ve programlama dillerinin yapı taşlarından bir tanesi de **karşılaştırma işlemleri** ve **karşılaştırma** operatörleridir. **Matematiksel işlemler**, programlama dünyasının olmazsa olmaz bir bölümünü oluşturur fakat karşılaştırma işlemleri de yapılan çok fazla işlem, çözülen çok fazla problem vardır. Programlamada çeşitli karar modelleri bulunur. Bu karar **yapıları**/**modelleri** programlama dillerinde de farklı kullanım notasyonları hâlinde bulunur. Örneğin; öğrenci bilgi sistemine giriş yapacak olan bir öğrencinin üyelik rol bilgisinin öğrenci mi yoksa akademisyen mi olduğu programlama ve işlemci tarafında gerçekleştirilen bir karşılaştırma gerçekleştirilir. İki sayıdan hangisinin **daha büyük olduğundan** tutun alfa sayısal olarak adlandırılan, **sayısal olmayan metin değerleri** de karşılaştırılabilmektedir.

Aşağıdaki tabloda programlama dünyasındaki karşılaştırma işlemleri gösterilmektedir.

Karşılaştırma Operatörü	Anlamı
<	Küçüktür
>	Büyüktür
<=	Küçük Eşittir
>=	Büyük Eşittir
< >	Eşit Değildir
=	Eşittir

Bazı programlama dillerinde "**=>**" operatörü "**>=**" operatörü ile "**=<**" operatörü ise "**<=**" operatörü ile **aynı anlama gelmektedir**.

Operatörler ile ilgili **algoritma**, **akış diyagramı** ve **örnek kodlamasını** gerçekleştireceğimiz bir **örnek** yapalım. Örneğimizde kullanıcıdan tam sayı türünde iki değer alıp bu değerlerin birbirlerinden **küçük**, **büyük** veya **eşit olma durumlarını** kontrol edelim.

Algoritma:

1.Başla

2.A Sayısını Giriniz

3.B Sayısını Giriniz

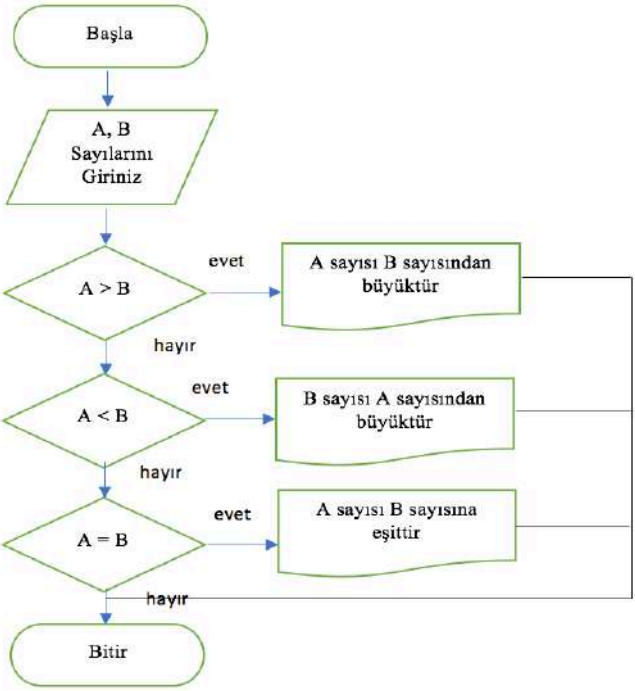
4.A>B ise Ekrana "A sayısı B Sayısından Büyüktür" Yaz

5.A<B ise Ekrana "B Sayısı A Sayısından Büyüktür" Yaz

6.A=B ise Ekrana "A sayısı B Sayısına Eşittir" Yaz

7.Bitir

Algoritması bu şekilde oluşturduğumuz örnek için bir de **akış diyagramı hazırlayalım**:



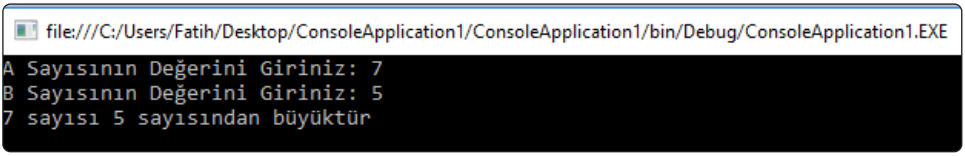
Örneğimizi **C# programlama dilini** kullanarak tamamlayalım:

```
static void Main(string[] args)
        {
            int a = 0;
            int b = 0;
            Console.Write("A Sayısının Değerini Giriniz: ");
            a = Convert.ToInt32(Console.ReadLine());
            Console.Write("B Sayısının Değerini Giriniz: ");
            b = Convert.ToInt32(Console.ReadLine());

            if (a > b)
            {
                Console.Write(a + " sayısı " + b + " sayısından
büyüktür");
            }
            else if (a < b)
            {
                Console.Write(b + " sayısı " + a + " sayısından
büyüktür");
            }
            else if (a == b)
            {
                Console.Write(a + " Sayısı " + b + " Sayısına
Eşittir");
            }

            Console.ReadLine();
        }
```

Yukarıdaki kodlara ait **ekran görüntüsünün çıktısı** ise aşağıdaki gibidir.



Şekil 7-1: Örnek Kodlamanın Ekran Çıktısı

Örnek kod ile ilgili bu bölümde detay vermeyeceğim. Programlama başlığı altında **değişken**, **döngü** gibi programlamaya ait terimleri işledikten sonra farklı başlıklarda programlama dilleri ile ilgili örnekler verileceğinden bu bölümde sadece **kodların ne şekilde göründüğünün incelenmesi** yeterli olacaktır.

Mantıksal İşlemler

Sayısal elektronik bölümünde yer alan *mantık kapılarını* hatırlayınız. AND, OR, XOR gibi çeşitli mantıksal durumlar için geliştirilmiş kapılar ile lojik seviyesinde işlemler gerçekleştirilmektedir. Doğruluk tablolarından da hatırlayacağınız üzere her bir **mantıksal ifadenin bir çalışma prensibi** bulunur. İşlemci seviyesinde gerçekleşen bu lojik işlemlerin programlama dünyasında da **karşılıkları** vardır. Bilgisayar ile insan arasındaki arayüz olan programlama dillerinde **iki** veya **daha fazla durumu** (*statement*) **karşılaştırmak için mantıksal işlemler** kullanılır. Tıpkı matematiksel ve karar işlemleri gibi mantıksal işlemlerin de operatörleri bulunur. Aşağıdaki tablo **mantıksal işlemlerin karşılıklarını** göstermektedir.

Mantıksal İşlem	Operatör Komut	Matematik Karşılığı
VE	AND	. (çarpma)
VEYA	OR	+ (toplama)
DEĞİL	NOT	‘ (değilini almak)

Mantıksal işlemlerde de matematiksel işlemlerde olduğu gibi **işlem önceliği hususu** vardır. Mantıksal işlemlerin **işlem önceliği tablosu** aşağıdaki gibidir.

Mantıksal İşlem Önceliği		
Sıra	**İşlem**	**Komut**
1	Parantez İçinde Bulunan İşlemler	(……)
2	DEĞİL	NOT
3	VE	AND
4	VEYA	OR

Mantıksal işlemlere bir çeşit doğruluk tablosu ile örnek vermek amacıyla aşağıdaki **tabloyu inceleyelim**.

Örnek: Bir üniversitede bulunan öğrencilerden durumu aktif olanlarının yaşı 24'ten küçük veya eşit olanların isimlerinin listelendiği programı kurgulayınız.

Cevap: `Eğer Durumu==Aktif VE Yas <= 24 ise listele AdSoyad`

Daha çok veri tabanı sorgusunu andıran örneğimizde **karşılaştırma operatörleri** ve **mantıksal işlemler** bir arada kullanılarak çözüme gidilmiştir. 4 tane öğrencinin olduğu rastgele bir listede sorgumuzu çalıştırdığımızı düşünürsek aşağıda bulunana benzer bir sonuç çıkacaktır.

5 öğrenci için sorgu doğruluk tablosu					
Durumu	**Yaş**	**1. Şart**	**2. Şart**	**Sonuç**	**Liste**
Aktif	18	1	1	1	Ali ÖZTÜRK
Pasif	23	0	1	0	-
Aktif	25	1	0	0	-
Pasif	21	0	1	0	-
Aktif	24	1	1	1	Şeyma AKINCI

Bu örneğe bir şart daha eklenirse mantıksal durumuna göre doğruluk tablosu **değişecektir.** Örneğin **VE KIZ** olan öğrenciler diye bir şart daha eklenirse yalnızca **Şeyma AKINCI** isimli öğrenci listelenir. Benzer şekilde *Aktif VEYA Yaşı 24'ten küçük* veya *eşit olanlar* denilseydi de yaşı 24'ten küçük veya eşit olan tüm öğrenciler aktif-pasif durumuna bakılmaksızın **listelenmiş olacaktı**.

Programlama Dili Çeşitleri

Günümüzde pek çok programlama dili **aktif** olarak kullanılmaktadır. Programlama dillerinin çeşitliliğinin artması hem *kullanıldığı platformlara* hem de *şirketlere* bağlıdır. Teknoloji çağında hemen her işlemin elektronik cihazlarla yapıldığını düşündüğümüzde **platforma göre programlama dillerinin çeşitlenmesi oldukça normal bir durumdur**. Örneğin, mobil programlama için android ve java programlama dilinin kullanılıyor oluşu gibi. Fakat günümüzde programlama dillerinin çeşitliliğinin artmasındaki bir diğer etken ise **yazılım şirketleri**dir. Büyük şirketlerin hemen hepsinin kendisine ait birer programlama dili bulunur. Bunu da **Apple için Swift** programlama dili ile örneklendirebiliriz. Geçmişten günümüze yaygın programlama dilleri aşağıdaki listede verilmiştir.

1. Assembly Programlama Dili

Alt seviye olarak sınıflandırılan dillerdendir. Oldukça *karmaşık* ve *makine diline yakın* olduğundan insanlar tarafından anlaşılması güçtür. Assembly diller, **makine dilinin kullandığı sayısal değerleri, programlanabilir alfa-sayısal ifadeler ile değiştirip alt seviyede çalışırlar**. Assembly dilini makine koduna çeviren programlara **Assembler** denilmektedir. Assembly programlama dilleri günümüzde doğrudan donanım ile haberleşecek ve performans bakımından önem arz eden projelerde hâlen kullanılabilir.

2. Basic Programlama Dili

Eğitim amacıyla geliştirilen Basic programlama dili, **1964** yılında John Kemeny ve arkadaşları tarafından geliştirilmiştir. Basic programlama dili, programlamayı öğrenmeyi teşvik ettiği için kısa süre yaygınlaşmış ve zamanla farklı şirketler tarafından geliştirilerek **QBasic**, **Turbo Basic**, **Visual Basi**c ve son olarak da Microsoft .net çatısı altında **Visual basic .net** olarak geliştirilmiştir.

3. C Programlama Dili

Efsane dil olarak da bilinen C programlama dili, **1969** yılında Dennis Ritchie tarafından geliştirilmiştir. C dilinin geliştirilme amacı **UNIX işletim sisteminin geliştirilmesi fikrine dayanmaktadır**. Bu yüzden Dennis Ritchie hem programlamanın hem de Linux platformlarının oluşturucusu gibi anılmaktadır. C programlama dili dünyanın **en çok kullanılan** programlama dilidir. Günümüzdeki görsel programlama veya elektronik programlama dillerindeki pek çok derleyici C programlama dilini temel alarak işlem yaptığından hâlen en çok kullanılan dil unvanını verebiliriz.

4. C++ Programlama Dili

1979 yılında Bjarne Stroustrup tarafından geliştirilen C++ (plus plus) programlama dili, C dilinin sınıflar ve diğer nesne temelli yapıların bir arada olduğu **C temelli fakat nesne yönelimli özelliklere sahip** olan hâlidir. C++, nesne yönelimli bir programlama dilidir.

5. C# Programlama Dili

C++ dilinin daha da geliştirilmesi sonucu ortaya çıkan yine **nesne yönelimli bir dildir**. Adından da anlaşılacağı üzere C++ gibi **C temelli bir dil**dir. 2000'li yıllarda Microsoft tarafından geliştirilmeye başlanan C# dili, nesne yönelimli programlama dünyasında oldukça ses getirmiş ve hâlen **nesne yönelimli programlama dilleri arasında en çok kullanılan dillerden**dir. İnsana yakınlığı konusunda oldukça başarılı olan C# programlama dili, Microsoft çatısı altında **C#.net** olarak geliştirilmektedir.

6. Java Programlama Dili

Sun Systems tarafından **1995** yılında geliştirilmiş **nesne yönelimli** ve **çok platformlu bir dil**dir. Günümüzde **Oracle** ve **Sun Systems tarafından ortak geliştirilmektedir**. Pek çok platformda bulunması sebebiyle dünyanın en yaygın programlama dilidir. Java programlama dili de C ve C++ dillerinden türetilerek geliştirilmiştir. Java programlama dili geliştirilmiş ortamları *cep telefonumuzdan bulaşık makinelerimize* kadar her yerde görebiliriz.

7. Python Programlama Dili

Eski bir programlama dili olmasına rağmen özellikle ülkemizde **son yıllarda rağbet görmeye başlamıştır**. **1991** yılında Guido Van Rossum tarafından geliştirilmeye başlanmıştır. Kolay öğrenilmesine olanak sağlayan syntax'ı ve derleyiciye ihtiyaç duymayan yapısı sayesinde günümüzdeki **en yaygın** programlama dilleri arasına girmeyi başarmıştır. **Nesne yönelimli** ve **çok amaçlı bir dil** olması sebebiyle *masaüstü uygulamalarından, web uygulamalarına ve elektronik programlamadan geliştirme kartlarına* kadar pek çok yerde kullanılmaktadır.

8. Swift

Apple tarafından geliştirilen **Objective-C tabanlı** yani yine **C tabanlı bir programlama dili**dir. *IOS işletim sistemlerinin geliştirilmesinde* kullanılmaktadır.

9. PHP

C tabanlı bir dil olan PHP, **web programlama dili** olarak bilinmektedir. Bazı kuruluşlar tarafından programlama dili olarak kabul edilmeyen php yalnızca *kodlama yığını* olarak da anılmaktadır.

10. Asp.Net – Asp.Net MVC

Web programlama dillerinden bir tanesi de **asp.net programlama dili**dir. Microsoft çatısı altında bulunan Asp.net ve Asp.net MVC yapıları **C#.net** veya **Vb.Net** ile geliştirilebilir.

Ayrıca günümüzde *masaüstü, web, mobil, derin öğrenme teknikleri, makine öğrenmesi, elektronik* ve *web script* platformlarında aşağıdaki diller de kullanılmaktadır.

Javascript

Go

Ruby

Perl

Groovy

Objective-C

Pascal–Delphi

Matlab

Pic Basic

R

Scala

Html

Cuda

Android

ActionScript

Jquery

8 NESNE TABANLI PROGRAMLAMA (OBJECT ORIENTED PROGRAMMING)

Nesne tabanlı programlama, programlama dünyasının **gerçek hayatta var olan yapıların programlamaya aktarılması** ile ortaya çıkan yaklaşımdır. Nesne tabanlı programlamanın yapısı incelendiğinde özellikle nesnelerin çok aktif rol aldığını düşünürsek gerçek hayattaki olguların **programlamayla entegre edildiğini** görebiliriz. Nesne tabanlı programlama, literatürde *nesneye dayalı programlama, nesneye yönelik programlama veya İngilizce karşılığı olan object oriented programming (OOP)* şeklinde karşımıza gelebilir. Nesne tabanlı programlama dillerinde vurgulanmak istenen, gerçek hayatta var olan olguların programlamaya aktırılması yaklaşımıdır. Gerçek hayatta var olan olguları programlamaya aktarırken ise **sınıflar** ve **nesnelerden** yararlanılır. Klasik programlama dillerinde fonksiyonlar ve fonksiyonları işleten kodlar bulunurken nesne tabanlı programlama dillerinde sınıflar, nesneler ve metotlar da bulunur. **Şekil 8-1**'de nesne tabanlı programlama kavramını gösteren ilişkili yapılar gösterilmiştir. Şekildeki kavramlardan; **Kalıtım**, **Kapsülleme**, **Soyutlama** ve **Çok Biçimlilik** nesne tabanlı programlamanın temel prensipleridir.



Şekil 8-1: Nesne Tabanlı Programlama Yapısı

Nesne Nedir?

Gerçek hayatla ilişkili olduğunu belirttiğimiz nesne tabanlı programlamada, nasıl günlük hayatımızda etrafımızdaki her şey birer nesne ise nesne tabanlı programlamada da **kullanılan her şey nesnelerden ibarettir** diyebiliriz. Nesneler, bildiğimiz üzere **somut** kavramlardır. Nesne tabanlı programlama da *soyut kavramlar da bulunur*. Nesne tabanlı programlama dillerindeki nesnelerin iki önemli özelliği bulunur ve bu özellikler **Durum** ve **Davranış** olarak bilinir. Programlama esnasında nesneleri kullanırken durum ve davranış özelliklerinden faydalanır ve bu özelliklere göre işlemleri gerçekleştirilir. Kitabımızın ilk konularında, örneklerimizi genellikle otomobillerden veriyorduk. Bu bölümün de daha iyi anlaşılabilmesi için *otomobillerden* örnek verebiliriz. Örneğin bir otomobilin *markası, modeli, rengi, fiyatı, türü, motor kapasitesi* gibi özellikleri otomobilin **durumunu** belirtmektedir. Programlama tarafında otomobilin özelliklerini belirten durum yapısına **attribute** (*öznitelik, özellik*) denilmektedir. Otomobilin yapabildiği *hız, taşıyabildiği yük, yavaşlaması, vites değiştirebilmesi* veya *yolcu kapasitesi bu kapasitenin değiştirilmesi* gibi eylemleri

ise **davranışını** belirtmektedir. Buradan hareketle, nesne tabanlı programlama dünyasında **duruma ve davranışa sahip olan her şeye nesnedir diyebiliriz**.

Otomobil örneğindeki durumunu belirtilen özellikler, nesne tabanlı programlama tarafında **değişkenleri** temsil etmektedir. Çünkü bu değerler, değişkenler aracılığı ile **bellekte tutulurlar**. Buradan da nesnelerin durumlarının nesne tabanlı programlamada **değişkenler ile tanımlanacağı** sonucuna ulaşabiliriz. Nesnelerin işletileceği **davranışların karşılığı** ise nesne tabanlı programlama da **metot** olarak adlandırılırlar. Prosedürel dillerdeki **fonksiyonlar** olarak da düşünebiliriz.

Ayrıca bir nesnenin kendi içerisinde barındırdığı birçok özelliği olabileceğini günlük hayattaki nesnelerden de bilmekteyiz. Bir otomobilin barındırdığı **birçok özellik** olduğu gibi bu özelliklerinin de barındırıldığı **ayrı bölümler** bulunur. Nesne tabanlı programlamada ise nesneye ait tüm özellikler **RAM** üzerinde bir yerde saklanmaktadır.

Sınıf Nedir?

Üniversite öğrencilerinin bir araya gelerek oluşturdukları *sınıflara*, nesne tabanlı programlama dilinde de **sınıf** diyebiliriz. Sınıfın üyesi olan *öğrencilere* ise **nesne** diyebiliriz. Bu tanımdan hareketle **sınıfların nesnelerden oluşan yapılar** olduğunu söyleyebiliriz. Nesnelerin tanımını yaparken bahsetmiş olduğumuz **durum** ve **davranışlarının tutulduğu yer sınıf**lardır. İlk örneğimizdeki otomobillerin durumları; *markası, modeli, fiyatı, rengi, ağırlığı* gibi özellikleridir. Bu özellikler otomobil nesnesinin durumlarını, yani programlama tarafındaki **değişkenleri** temsil etmektedir. Otomobil nesnemizin *hızlanması, yük kapasitesi, yavaşlaması* gibi özellikleri ise **davranışları** temsil ettiğini söylemiştik. Programlama tarafında ise bu özellikler **metotları** temsil etmektedir. Sınıflar ise **durumlar ve davranışların bir arada barındırıldığı yerler**dir. Otomobiller ile ilgili özelliklerin tutulduğu sınıfa otomobil adını verirsek otomobil sınıfından çeşitli özelliklere göre istediğimiz kadar otomobil üretebiliriz. Üretilen **her bir nesneye programlama tarafında instance** (*örnek*) denilmektedir. Yani otomobil sınıfının örnek üyeleri. Her bir örneğin sahip olduğu değerler davranış ve durumda belirtildiği gibi olacaktır. Örnekten tanıma doğru geri gidecek olursak da bütünü oluşturacak şekilde ortak özelliklere sahip olan **örneklerin** (*üyelerin*) **bir araya getirdikleri yapılara sınıf** denilmektedir.

Nesne ve Sınıf İlişkisi

Sınıflar (*class*), nesne tabanlı programlama dillerinin **soyut veri** türleridir. Nesneler (*object*) ise sınıfların **somutlaşan birer varlığı**dır. Soyut sınıf tiplerinde, nesnelerde olması gereken özellikler barındırılır. Bu özellikler **davranışlar** ve **durumlardan** oluşmaktadır. Soyut sınıf tipinden bir örnek (*instance*) oluşturulması işleminde **ortaya nesne çıkar** ve **bir sınıftan istenilen kadar nesne türetilir**. Oluşturulan bu nesnelerin tipleri sınıflardaki ile aynıdır. Bu yüzden **aynı sınıflardan üretilen nesneler aynı tipte** olmakta, **farklı sınıflardan üretilen nesnelerin tipleri de farklı** olabilmektedir. Örneğimiz üzerinden gidecek olursak otomobil sınıfından türeyen nesnelerin tipi ile elektronik cihazlar sınıfından türeyen nesnelerin tipi, yani **özellikleri** farklı olacaktır. Türetilen nesneler ile ilgili karıştırılmaması gereken bir diğer nokta ise *aynı sınıftan türeyen nesnelerin özellikleri aynı olsa bile değerlerinin farklı olabileceğidir*. Otomobil sınıfındaki nesnelerin renk özelliği düşünüldüğünde her biri farklı renge sahip olabilir.

Özetle, nesne tabanlı programlama dillerinde kullanılan yordamlar **doğrudan uygulama koduna yazılmaz**, **sınıflar içine yazılırlar**. Gerektiğinde de bu sınıflardan türetilen nesnelerin üzerinden çağrılarak nesne tabanlı programlama konsepti ile geliştirme yapılmış olur.

Soyutlama (Abstraction)

Nesne tabanlı programlamada nesneler, bir varlığın **temsil biçimi**dir. Bir nesnenin bilgisayar dünyasında temsil edilebilmesi için **özellikleri temel rol** oynamaktadır. Nesnelerin karakteristik özellikleri ve gerçekleştirecekleri eylemlere ise **metot** denilmektedir. Soyutlaştırma gereksiz işlemlerin gizlenmesi ve oluşturulan bileşenlerin yalnızca alakalı kısımlarının işletilerek diğer kısımlara daha **öz** veya **soyut** bir şekilde sunulması işlemidir. Özelliklerin içerisinde **üye değerleri** ve **değişkenler** yer alabiliyorken metotların içerisinde de **metotlar** temsil edilir. Nesne tabanlı programlama dillerinde soyutlama işlemi, **sınıflar** (*class*) **tarafından gerçekleştirilir**. Nesneye ait özellikler ve metotlar sınıflar içerisinde tanımlanmaktadır. Sınıflardan üretilen örnekler sınıfa ait tüm özelliklere ve metotlara sahip olur ve program içerisinde doğrudan kullanılabiliyorken **sınıf soyut bir yapıda olduğu için doğrudan kullanılamaz**.

Kapsülleme (Encapsulation)

Oluşturduğumuz sınıf içindeki verilerin, yani durum ve davranışların **dışarıdan değiştirilmesini engellemeye yarayan olaya kapsülleme** denilmektedir. Aslında kapsülleme işlemini sınıfı oluşturan metot ve özelliklerin gerçekleşme biçiminin, bu sınıfı kullanacak olan kullanıcılardan kapsüllenerek gizlenmesidir. Sınıfa erişebilen üyelerin gizlenmesi erişim belirleyicileri tarafından da yapılabilir. **Erişim belirteçleri** (*Access modifier*); public, private, protected, internal sayesinde kapsülleme işlemleri daha kolay yapılmaktadır.

Public: Uygulama ortamındaki sınıflara diğer projelerde dâhil olmak üzere her yerden erişebilir.

Private: Oluşturulan sınıflara sadece tanımlandığı sınıftan erişebilmesini sağlar. Geliştirici tarafından oluşturulan sınıflara ya da yapılara erişim belirteci belirtilmez ise derleyici otomatik olarak private olarak tanım yapacaktır.

Protected: Sadece tanımlandığı sınıfın içerisinden ve o sınıftan türetilmiş diğer sınıfların içerisinden erişim sağlanır.

Internal: Aynı derleyici içerisinde bulunan tüm sınıflardan erişim sağlanabilir.

Kalıtım/Miras Alma (Inheritance)

Gerçek hayatta da olduğu gibi soyut sınıfın özelliklerini bir sonrakine aktarabilmesi olayıdır. Kalıtım, bir nesnenin **özelliklerinin farklı nesneler tarafından da kullanabilmesini sağlayan** nesne tabanlı programlama özelliğidir. Yani bir çocuğun göz rengini babasından alması **kalıtım yolu** ile geçmiş olabilir ve **bu aktarım soyut**tur. Nesne tabanlı programlama dillerinde de oluşturulan sınıflardan bir başka sınıfa, özellikleri olduğu gibi **miras bırakılabilir**. Alt sınıflara özelliklerini miras bırakan sınıfa **temel sınıf** veya **parent sınıf** gibi isimler verilebilir. Oluşturulan yani miras alan sınıflara ise **Child Class** denilmektedir. Örneğin otomobil örneğimizde otomobil sınıfında bulunan motor hacmi, renk, tür, marka, model özellikleri kendisinden türetilecek diğer sınıflarda da olması gereken temel özelliklerdir, bu yüzden bu özellikler ve nitelikleri aynen miras edilir.

Çok Biçimlilik (Polymorphism)

Çok biçimlilik, nesne tabanlı programlama dillerinin temel özelliklerinden **üçüncüsü**dür ve bir nesne elemanının **farklı ortamlardaki farklı sonuçlar doğurabilme olayı**dır. Otomobil üst sınıfından türetilen diğer sınıflar, otomobil sınıfının aynı özelliklerine taşımalarına rağmen otomobil türü özelliğinden ötürü **SUV**, **HB** veya **SEDAN** gibi türler çıktı verilebilir. Bir başka örnek ise insan sınıfına ait bir özellik olan konuşma eyleminin farklı milletlerde farklı konuşma (*speech*) olarak çıktı vermesi olayıdır. Bu olay nesne tabanlı programlama dilinde **çok biçimlilik** olarak adlandırılır.

Sonuç olarak nesne tabanlı programlama yapmanın avantajları arasında kod tekrarını en aza indirme, projenin herhangi bir bölümünde meydana gelecek güncelleme işleminde kod hakimiyetinin ve yönetiminin kolay bir şekilde gerçekleştirilmesi, yapılan işlemlerde her tanımın bir nesne olması sebebiyle farklı projelere kolay entegre edilebilmesi yer almaktadır. Nesne tabanlı programlama yapılarak programcıların **yığma** diye de tabir edilen **dağınık ve sadece kendileri tarafından anlaşılabilecek kod yapıları inşa etmeleri engellenir**. OOP sayesinde yapılan işler parçalara ayrılıp planlı bir şekilde hareket edilmesini sağlar. Nesne tabanlı programlama sayesinde kod okunurluğu artar ve projeye sonradan dâhil olan birisi de kodları kolaylıkla okuyabilir ve ekleme, çıkarma yapacağı zaman başka kod bloklarında vakit kaybetmeden doğrudan ilgili sınıfı tespit edebilir.

Nesne tabanlı programlamada kalıtım özelliğinin pekiştirilebilmesi için **C#.net dili** ile geliştirdiğim örnek kodlamayı incelediğimizde geometrik şekillerin alanı ve çevresinin hesaplanmasında kullanılan dört işlem için genel bir hesaplama sınıfı yazılmış ve bu sınıftan miras alarak **çember**, **kare**, **faktöriyel** gibi özel işlemlerin yapılabileceği diğer sınıflar yazılmıştır. **Diğer sınıflar genel dört işlem sınıfından miras alınarak türetilmiştir**.

Aşağıdaki kod bloğunda **dört işlemin yapıldığı** soyut sınıf hazırlanmıştır. Miras alma işleminin **sonraki hesaplama metotlarında ":" ile gerçekleştirildiğine dikkat ediniz**.

```
public abstract class Calculation
    {
        public int Sum(int a, int b)
        { return a + b; }

        public int Subtract(int a, int b)
        {
            return a - b;
        }

        public int Multiply(int a, int b)
        {

            return a * b;
        }
        public double Multiply2(double x, double y, double z)
        {
            return x * y * z;
        }

        public int Divide(int a, int b)
        { return a / b; }
}
```

Yukarıdaki sınıfı miras alarak aşağıdaki kod bloğunda **çemberin alanı** ve **çevresi** hesaplanmıştır. **Miras alma işleminin ":" ile gerçekleştirildiğine dikkat ediniz**.

```
public class CircleCalculation : Calculation
    {
        double pi = 3.14;
        double deger = 2;
        public double CircleArea(double R)
        {
```

```
            double result = Multiply2(R, R, pi);
            return result;
        }

        public double CirclePerimeter( double R)
        {
            double result = Multiply2(pi, R, deger);
            return result;
        }
    }
```

Aşağıdaki kod bloğunda ise yine genel hesaplama sınıfından miras alınarak **karenin alanı** ve **çevresi** hesaplanmıştır. **Miras alma işleminin ":" ile gerçekleştirildiğine dikkat ediniz**.

```
public class SquareCalculate : Calculation
    {
        public int SquareArea(int line1, int line2)
        {

            int result = Multiply(line1,line2);
            return result;

        }

        public int SquarePerimeter(int line1, int CountLine)
        {
            int result = Multiply(line1, CountLine);
            return result;
        }
    }
```

Aşağıdaki kod bloğunda ise yine genel hesaplama sınıfından miras alınarak t**oplama serisinin hesaplaması** yapılmıştır. **Miras alma işleminin ":" ile gerçekleştirildiğine dikkat ediniz**.

```
class SumOfSeriesCalculation : ExponentialCalculation
    {
        int donus;
```

```
        int Sonuc1, Sonuc2;
        public int SumOfSeries(int n)
        {

            int result;
            int result1;
            int x = 4;
            int y = 2;
            result = Subtract(n, x);
            result1 = Sum(2, y);
            for (int i = 1; i <= x; i++)
            {

                Sonuc1 = ExponentialCalculate(result, 4);
            }

            for (int i = 1; i <= y; i++)
            {
                Sonuc2 = ExponentialCalculate(result1, 2);
            }
            donus = Multiply(Sonuc1, Sonuc2);
            return donus;
        }
    }
```

Aşağıdaki kod bloğunda ise yine genel hesaplama sınıfından miras alınarak **faktoriyel hesaplaması** yapılmıştır. **Miras alma işleminin ":" ile gerçekleştirildiğine dikkat ediniz**.

```
class FactorialCalculation : Calculation
    {

        int result = 1;

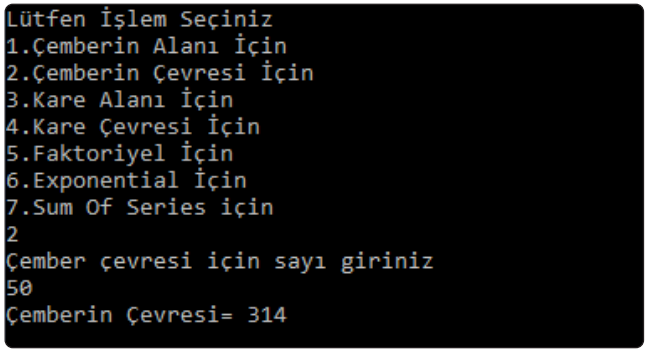
        public int CalculateFactorial(int alinan)
        {

            for (int i = 1; i <= alinan; i++)
            {
```

```
                result = Multiply(result, i);
            }
            return result;

        }
    }
```

Şekil 8-2'de kaynak, yukarıdaki kodların konsol ortamında çalıştırılması sonucu gösterilmiştir. Proje, konsol ortamında çalıştığında oluşturulan hesaplama işlemlerinden hangisinin yapılacağını soruyor ve işlem numarasının girilmesini bekliyor. Örnek görüntüde **2 numaralı** işlem seçilmiştir ve daha sonra işlemin türüne göre örneğimizde **çemberin çevresini hesaplayacağı** için kullanıcıdan **değer istiyor** ve girilen değere göre kendi sınıfından işlemleri gerçekleştirip kalıtım aldığı sınıftan da temel işlemleri refere edip **işlem sonucunu ekrana döndürüyor**.



Şekil 8-2: Kalıtım ile Hesaplama Projesi Ekran Görüntüsü

9

VERİ TABANI

Bilgi teknolojilerinin gelişmesi ile birlikte **veri** teriminin önemi oldukça artmıştır. Bilgisayarlarda yapılan tüm işlemler elektronik ortamda depolanmakta ve ihtiyaca göre yeniden kullanıcıya çeşitli programlar vasıtasıyla sunulmaktadır. Veri tabanı, eski bilişim kaynaklarında **veri kütüğü** olarak da anılmaktadır. Kütük denilmesindeki sebep ise **verinin sabit olarak belirli bir lokasyonda depolanmasından** ileri gelmektedir. Yazılan programlardan ihtiyaç hâlinde platformdan bağımsız bir şekilde veriye erişmek ve işlemek amacıyla kullanılan arabirim yazılımlara ise **veri tabanı yönetim sistemi** denilmektedir. Veri tabanı kullanmak günümüzde artık olmazsa olmazdır fakat yine de avantajlarını düşündüğümüzde veri tekrarını ortadan kaldırdığını, bellek yönetimini ve veri bütünlüğünü sağladığını görebiliyoruz.

Veri Tabanı Nedir?

Veri tabanı, belirli bir düzene göre verileri kaydeden **dijital bir yazılım**dır. Veri kelimesi, içerisinde düzenlenmemiş bilgileri tutan bir kümeyi ifade etmektedir. **Verilerin dağınık bir şekilde kayıt altında tutulduğu sistemlere veri tabanı** diyebiliriz. Bu verilerin anlamlandırılması amacıyla düzenli ve belirli bir standarda göre kayıt altında tutulduğu sistemlere de veri tabanı denilmektedir. Bir veri kümesinin işe yarayabilmesi için **anlamlı sonuçlar** verecek durumda olması gerekmektedir. Veri tabanı sistemleri de burada devreye girer ve her bir veri tabanı sisteminin veriyi barındırma algoritması **farklılık** gösterir. Buradan hareketle veri tabanının verilerin bir araya gelerek ihtiyaç hâlinde tekrar çağrılabilmesini, yönetilebilmesini ve kullanıcıya sunulabilmesini sağlayan bir platform olduğunu söyleyebiliriz.

Veri tabanı içerisinde **tablolar**, **prosedürler**, **sorgular**, **formlar** ve bunlardan elde edilen **raporlar** bulunur.

Veri tabanları yapılarına göre çeşitlenebilmektedir. Örneğin birden fazla lokasyonda depolanan veri tabanlarına **paylaşımlı veri tabanı** denilmektedir. Paylaşımlı veri tabanlarında bağlantı kuracak istemcilere yetkilerine göre kullanıcılar tanımlanır ve istemciler bu yetkiler doğrultusunda veri tabanına bağlanır ve diğer işlemleri yaparlar. İçerisinde tutulan verinin türüne göre de farklılık gösteren veri tabanlarında özellikle medya sektöründeki grafik, ses veya harita gibi verilerin tutulduğu veri tabanları **nesne tabanlı veri tabanı** olarak bilinmektedir. Veri tabanı içerisindeki farklı hizmet modüllerine rahatlıkla geçiş yapabilen veri tabanı türüne ise **hiper medya veri tabanı** denilmektedir.

Veri tabanı oluşturmadan önce bazı ön çalışmaların yapılması gerekmektedir. İlk önce, **hangi amaca hizmet edecek** bir veri tabanına ihtiyacımız olduğunu belirlemeli buna bağlı olarak da **problemi iyi tanımlamamız** gerekmektedir. Akabinde veri tabanının veya kullanılacağı programı kullanacak kişilerin **yetenekleri, eğitim seviyesi, bilgisayar bilgisi** gibi etkenler düşünülmelidir. Bu kişilerin de veri tabanından beklentilerini iyi kavrayıp gereksinimleri ona göre planlamak gerekir. Veri tabanı tasarımı işleminde, veri tabanındaki verilerin ihtiyaç hâlinde çeşitli kriterler ile kullanıcıya rapor şeklinde yeniden sunulacağı **raporların türleri** de düşünülmeli buna göre bir **tablo yapısı** inşa edilmelidir. Eğer tablolar arasında **bire bir** veya **bire çok** ilişki kurulacak ise ilişkisel veri tabanı yöntemi benimsenerek tablolar oluşturulmalı, ilişkiler kurulmalıdır. Günümüzdeki modern veri tabanı sistemleri genellikle **ilişkisel veri tabanlarıdır**. Tablo içerisindeki bir alanın başka tablolarda da yer alabilmesi veya oradan referansla işlem yapması durumunda **veri** ve **tablo alanı** (*field*) **tekrarını önleyen ilişkiler kurulur**. Örneğin müşteriler tablosunda da müşteri adı, siparişler tablosunda da sipariş yapan müşterinin adı bulunur. Böyle bir durumda bu bilgiyi her iki yerde de tutmak yerine bir tanesinde (*master*) tutarak diğer tabloda ilişki kurulur. Veri tabanlarının özellikleri oldukça önemlidir. Tasarım aşamasında **UML** diyagramlarında da ilişkiler belirtilir.

Veri Tabanı Yönetim Sistemi

Yeni bir veri tabanı oluşabileceğimiz, veri tabanı üzerinde yönetimsel işlemleri gerçekleştirebileceğimiz ve veri tabanı içerisine tablolar oluşturup yönetebileceğimiz arayüze **veri tabanı yönetim sistemi** denilmektedir. Veri tabanı yönetim sistemleri (**VTYS**) sayesinde veriye uzaktan veya doğrudan erişebilme, CRUD işlemleri olarak adlandırılan Create, Read, Update, Delete gibi işlemler yapılabilmektedir. En yaygın veri tabanı yönetim sistemlerinden bir tanesi **Microsoft Sql Server Management Studio** yazılımıdır. Bu yazılım ile uzaktaki veya yereldeki veri tabanına bağlanılır ve üzerinde işlemler yapılır.

Veri tabanı yönetim sistemi (**VTYS**) ile *sorgular yazılabilir, prosedürler oluşturulabilir* veya *agent ismi verilen belirli bir zamanda çalıştırılmak üzere görevler* oluşturulur. Veri tabanı içerisinde yer alan tablolara programlama ile erişilip işlemler kodlar vasıtasıyla geliştirilen programlarda yapılabiliyor olsa da üzerinde çalışılan veri tabanı ile ilgili işlemlerin **doğrudan bir arayüz vasıtasıyla** yapılması daha **kullanışlı** olmaktadır. Ayrıca Oracle gibi büyük sistemleri kullanan kişilerin, veri tabanı işlemleri için kesinlikle bir veri tabanı yönetim sistemine ihtiyaçları bulunur. **MY SQL** ve **MS SQL** için yaygın veri tabanı yönetim sistemi yazılımlarından bir tanesi de **Toad** isimli yazılımdır.

Yapısal Sorgulama Dili

Structured Query Language olarak terimleşen SQL, Türkçeye **yapısal sorgulama dili** olarak çevrilmiştir. **SQL bir programlama dili değildir**. Bir programlama aracı da değildir. **Yalnızca bir sorgulama dilidir** ve bu işi yapısal olarak gerçekleştirir. SQL denilince Microsoft çağırışım yapıyor olsa da aslında SQL **ortak bir platform**dur. ORACLA'da olsa MY SQL'de olsa içerisinde yer alan sorgu dili SQL'dir ve sorguları diğer platformlarda da aynıdır. Bu yüzden SQL bilen birisi, Microsoft SQL Server Management Studio arayüzü ile ve aynı bilgisi ile Microsoft Office Access programı ile de işlemlerini gerçekleştirebilir.

SQL Server

SQL Server, Microsoft firmasına ait **veri tabanı sistemi**dir. İşlemlerin kullanıcı ara birimi vasıtasıyla gerçekleştirilebilmesine olanak sağlayan veri tabanı yönetim sistemi yazılımına ise **SQL Server Management Studio** denilmektedir. SQL Server denilince komutlar üç başlık altında incelenmektedir.

1. DDL Deyimleri

DDL (*Data Definition Language*), veri tabanı **tanımlama dili** anlamına gelir. Veri tabanı yapısını, tabloları ve şemaları tanımlamak için kullanılmaktadır. Bazı DDL Deyimleri;

CREATE: Yeni nesne oluşturur.

ALTER: Yapısal Değişiklik yapar.

DROP: Nesne Silme işlemi yapar.

TRUNCATE: Kayıtları, içerikleri alanlar ile birlikte silme işlemi yapar.

RENAME: Nesne adını değiştirir.

2. DML Deyimleri

DML (*Data Manipulation Language*) veri tabanında **nesneleri yönetmek için** kullanılır. Veri tabanından veri eklemek, silmek veya değiştirmek için kullanılan deyimleri içerir. Bazı DML deyimleri;

SELECT: Veri tabanından kayıtları listeler.

INSERT: Tablolara kayıt ekleme işlemini gerçekleştirir.

UPDATE: Tablolarda yer alan kayıtları günceller.

DELETE: Tablolardan kayıt silme işlemini gerçekleştirir.

3. DCL Deyimleri

DCL (*Data Control Language*), veri tabanındaki **verilere olan erişimi kontrol etmek için** kullanılan deyimleri içerir. Bazı DCL deyimleri;

GRANT: Kullanıcılara veri tabanı erişim yetkisi vermek için kullanılır.

REVOKE: Verilen yetkinin geri alınması için kullanılır.

Kısa Kısa SQL Server Management Studio

Bu bölümde, SQL Server Management Studio veri tabanı yönetim sistemi yazılımı ve SQL Server veri tabanı motoru yazılımlarının **ne şekilde kurulabildiği** ile ilgili internette sayısız kaynak olduğu için kurulumların nasıl yapıldığına değinmiyor **SQL Server** ve **Management Studio ile ilgili bilinmesi gereken hususlara** geçiyorum.

SQL Server ve Management Studio ile ilgili özet bilgiler aşağıdaki gibidir.

- Yedekli çalışan veri tabanı sunucularında ana sunucunun başına bir iş geldiğinde yedek sunucuda kopma olmaksızın işlemlere kaldığı yerden devam edebilmesini sağlayan sistemlere "**Failover Cluster**" denilmektedir.
- SQL Server 2017 ile birlikte **LINUX–DOCKER** desteği gelmiştir.
- Mixed Mode olarak kurulan bir sistemde varsayılan şifre "**sapass**" olarak belirlenir.
- SQL Server ve Management Studio kurulumu sonrasında çalışma durumu **Başlat/Çalıştır/Services** bağlantısından incelenebilir.
- Management Studio yazılımını **ssms.exe** kısayolu ile çalıştırabilirsiniz.
- SQL Server, **1433** portunu kullanmaktadır. Portu değiştirmek isterseniz **SQL Server Configuration Manager/Network Configuration/MSSQL/TCP/IP** özellik penceresinden değiştirebilirsiniz.
- Eğer sistemdeki sistem admin (sa) şifresi bilinmiyor fakat **sa şifresinin değiştirilmesi gerekiyorsa** Management Studio'ya erişim olduğu takdirde aşağıdaki kod;

```
ALTER TABLE SA WITH PASSWORD ='SAPASS'
```

Eğer Management Studio üzerinde TSQL arayüzüne **erişim söz konusu değilse** komut terminalinden;

CMD Sağ Tık/Yönetici Olarak Çalıştır ve **aşağıdaki komut çalıştırılır**.

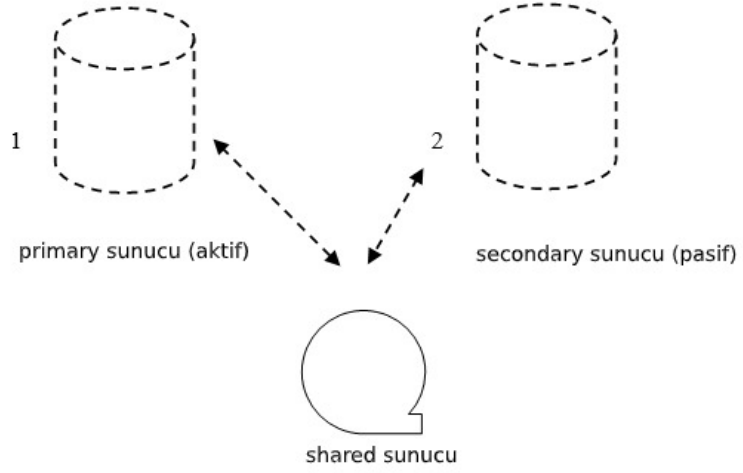
```
osql -L / osql -S BİLGİSAYAR_ADI -E
EXEC sp_password NULL, 'sapass', 'sa'
GO EXIT
```

`osql -L` komutu sunucuları listeler.

- SQL Server Management Studio' da T-SQL kodları yazarken her bir sorgu **farklı satırda** yazılmaktadır. Alt alta birden çok satır bulunan bir sorgu ekranında yalnızca istenilen sorguyu çalıştırmak için **ilgili sorgu satırı seçilir** ve **seçili hâldeyken F5 ile sorgu çalıştırılır**. Eğer seçim yapmadan F5 ile tüm sorgular çalıştırılırsa çalıştırılmasında sakınca olan bir sorguyu da çalıştırma riski bulunur.
- Veri tabanı ve içeriğini farklı bir sunucuya anlık olarak kopyalama işlemine **Replication** denilmektedir. SQL Server Management Studio'da ile üç çeşit kurtarma (*disaster recovery*) senaryosu bulunur. Bunlar; **Full Recovery Model**, **Simple Recovery Model** ve **Bulk Recovery Model** olarak bilinir. Kurtarma yöntemini management studio ile belirlemek için **Veri Tabanı Adına Sağ Tık/ Properties/Options** menüsü **Recovery Model** alanından seçim yapılır. Bu işlemi komut ile gerçekleştirebilmek için New Query ile yeni bir sorgu açıp (*veri tabanı seçili iken*) aşağıda yer alan komut çalıştırılır;

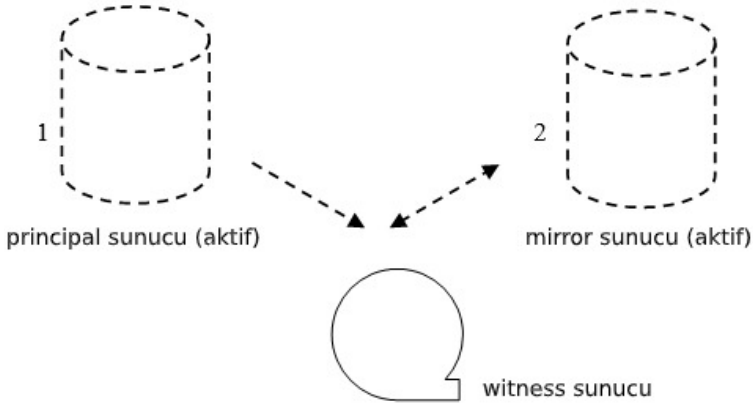
```
ALTER DATABASE VERİ_TABANI_ADI SET RECOVERY FULL
```

- Yedekleme senaryolarında Failover Cluster'da Primary yani birincil sunucu **aktif** iken Secondary yani ikincil sunucu **pasif** olmaktadır. Veriler ise aradaki **Shared** yani paylaşım sunucusunda bulunur.



Şekil 9-1: Failover Cluster Yedekleme Senaryosu

Mirroring senaryosunda ise iki sunucu da **sürekli olarak aktif**tir. Aradaki bağı **Witness** sunucu yürütür.



Şekil 9-2: Mirroring Yedekleme Senaryosu

- SQL Server Management Studio üzerindeki tüm veri tabanlarını veya ilgili veri tabanını kontrol etmek (*check*) etmek için (*New Query*) **yeni sorgu** ekranında aşağıdaki komut çalıştırılır. Bu komut çalıştırıldıktan sonra sonuç penceresinde veri tabanı ile ilgili **check durum bilgisi** görüntülenecektir.

```
dbcc checkdb('VeriTabanı_Adi')
```

Tüm veri tabanları için: **`dbcc checkalloc`**

- **DB Check** işlemi faydalı bir işlemdir, ara sıra yapılmasında yarar bulunur. **Hataları tespit etmek** ve **fiziksel olarak gösterilmesini sağlamak için** ise aşağıdaki komut kullanılır.

```
dbcc checkfilegroup
```

- **Tablo bazlı hataları tespit etmek için** aşağıdaki komut kullanılır.

```
dbcc checkconstraints('tablo_adi')
dbcc checktable('tablo_adi')
```

- Hata tespit işleminden sonra **bozulan verileri onarmak için** aşağıdaki komut kullanılır.

```
dbcc checkdb('VeriTabaniAdi',Repair_fast)
```

- Data hatalarında onarım yapıp **IO hatalarında onarım yapamayan komut** aşağıdaki gibidir.

```
dbcc checkdb('VeriTabaniAdi', Repair_Rebuild)
```

- **IO hatalarını da çözerek onarım yapan komut** aşağıdaki gibidir.

```
dbcc checkdb('VeriTabaniAdi',Repair_Allow_Data_Loss)
```

- SQL Server Management Studio ile bakım planı (*backup*) işlemi oluşturmak için **Object Explorer** penceresinde bulunan **Management** klasörü altındaki **Maintenance Plans** dizinine **sağ tıklanır** ve **yeni bakım planı oluşturulur**. Bakım planı oluştururken (*yedekleme işlemi vb*.) Toolbox yani araç kutusundaki **Backup Database Task**, **Maintenance Cleanup Task** ve **Rebuild Index Task** gibi bileşenlere **çift** tıklanarak bakım-yedekleme planı oluşturulur.
- Veri tabanlarının içerisinde, zamanla kullanılmayan alanlar oluşur. Bu alanlar veri tabanı dosya boyutunun **büyümesine** neden olmaktadır. Fiziksel olarak fazlalıkların küçültülmesi disk kazanımı ve veri bütünlüğü sağlayacağından veri tabanlarında **kullanılmayan alanların silinmesi gerekir**. Bu işlemi, SQL Server Management Studio'da **Shrink Task** ile gerçekleştiririz. Fiziksel küçültme işleminde index'lerde yer değişikliği yapıldığından normal datalar yerine **Log Data**ları üzerinde yapılması tavsiye edilir. Toolbox üzerinde bulunan **Shrink Database Task** nesnesi ile bu işlem gerçekleştirilir.
- SQL Server Management Studio'da arka tarafta hemen hemen her şeyin tutulduğu istatistiklere ulaşmak için **Update Statistics Task** nesnesi kullanılır.
- SQL Server'a ait çeşitli log kayıtları (*iz kayıtları*) bulunmaktadır. Bunlardan normal geçmiş log kaydı veya bakım planı olan Maintenance Plan geçmişi gibi log'larda yer açabilmek için **History Cleanup** isimli araç kullanılmaktadır.
- Aktif veri tabanlarını gözlemek için başlat menüsünden **SQL Server Profiler** uygulaması çalıştırılarak gözlem yapılabilir.

SQL Server' da incelememiz gereken bir diğer konu başlığı ise **CONTRAINTS** yani **sınırlandırmalar** konusudur. Tablolar içerisinde bulunan alanlara girilecek olan verinin tutarlılığı ve güvenliği ile ilgili çeşitli kısıtlama işlemi yapılır. Tablolardaki alanlara girilen verinin kontrol edilmesi belli bir kurala göre kısıtlanması son kullanıcı bazlı işlem yapan kimseler tarafından veri girişini kolaylaştırır ve bilginin güvenliğini de arttırmaktadır. Bir tabloda kısıtlama işlemini yapabilmek için **komut yöntemi** veya **Management Studio'nun arayüz desteğinden** faydalanılabilir. Arayüz ile gerçekleştirmek için ilgili tablo, Design görünümde açıldıktan sonra ilgili alanların başına **ters tık** yapılarak veya **ilgili alanlar değiştirilerek** kısıt verilebilir. Komut ile gerçekleştirmek için ise **ALTER TABLE**, eğer tablo ilk defa oluşturuluyor ise **CREATE TABLE** komutları ile istenen kısıt verilebilir. Kısıtlamalar ve açıklamaları aşağıdaki gibidir.

1. **Default**: Alanların herhangi birinde **INSERT** yordamından sonra girilmeyen bir değer olması durumunda varsayılan olarak önceden belirlenmiş olan değerin atanması sağlanır.
2. **Primary Key**: Tablolarda **birincil anahtar** tanımlamak için kullanılır. Primary Key, yani birincil anahtar olarak tanımlanan alandaki değerlerin her biri birbirinden farklı olmak zorundadır ve bir tabloda **en fazla 1 tane birincil anahtar** olabilir.
3. **Check**: Tablolardaki alanlara girilecek olan değerleri sınırlandırır. Yani girilen değerleri **kontrol eder** ve **belirlecek olan bir ifadenin girilmesini engeller**.
4. **Unique**: Tablolardaki alanlara aynı değerden **tekrar girilmesini engeller**. Benzersiz alan olmasını sağlar.
5. **Foreign Key**: Yabancıl anahtarlar başka bir tablodaki **birincil anahtar ile ilişki kurmak için** kullanılırlar.

T-SQL

Transact-SQL ile veri tabanında bulunan bilgilerin üzerinde **analiz işlemleri** ve **çeşitli raporlama** işlemleri kolaylıkla yapılabilmektedir. Microsoft SQL Server üzerinde yapılan T-SQL işlemleri ile ORACLE veya benzeri başka bir sistemde yapılan T-SQL kodları **komut bazlı farklılık** gösterebilmektedir. Önceki bölümde de bahsettiğimiz gibi SQL komutları kullanım amaçlarına göre **üç ayrı başlık altında** incelenmektedir.

1. SELECT

Select komutu, veri tabanındaki **bilgileri listelemeye yarayan** komuttur. Bir tablomuz olsun ve tabloda öğrenci bilgilerinin yer aldığını düşünelim. Herhangi bir kriter koymadan tüm öğrencileri listelemek istediğimizde **`select`** komutunu aşağıdaki gibi kullanabiliriz.

```
SELECT * FROM TABLO_ADI
SELECT * FROM OGRENCI_TABLOSU
```

Eğer tablo içerisinden belirli sütunlar listelenmek istenirse **`*`** deyimi yerine **sütun isimleri** yazılmalıdır. Aşağıdaki komut, **belirli sütunların** listelenmesini sağlar.

```
SELECT OGR_AD, OGR_NO FROM OGRENCI_TABLOSU
```

Listeleme işlemine kriter konulmak istenir ve sonuç biraz daha özelleştirilmek istenirse bu sefer **`WHERE`** deyimi ile kriter belirtilmelidir. Memleketi **`KONYA`** olan öğrencileri listelemek istediğimiz **SQL komutu** aşağıdaki gibidir.

WHERE

```
SELECT * FROM OGRENCI_TABLOSU WHERE OGR_MEMLEKET='KONYA'
```

SQL SERVER Managament Studio üzerinden T-SQL kodlarını New Query ile açtığımız sorgu yazma penceresi altında çalışırken **metinsel ifadeleri tek tırnak içerisine almalıyız**. Veri tabanı içerisindeki örnek olarak bahsettiğimiz öğrenci tablosunda aynı ad ve soyada sahip yüzlerce öğrenci olabileceğinden benzersiz kayıtları listelemek isteyebiliriz.

Örneğin tabloda birden fazla sayıda ALİ KAYA isimli öğrenci olsun eğer sorgu mantığımızda **bize bir adet ALİ KAYA dönmesini istiyorsak** ve **tekrarlı verileri görmek istemiyorsak** `DISTINCT` deyimini kullanabiliriz. Bu senaryoyu farklı durumlar içinde düşünebilirsiniz. Aşağıdaki komut ile **benzersiz kayıtlar listelenmektedir**.

DISTINCT

```
SELECT DISTINCT OGR_AD FROM OGRENCI_TABLOSU
```

Bu komuta memleket veya yaş şartı eklenmek istenirse **`WHERE`** deyimi ile aşağıdaki gibi yazılır.

```
SELECT DISTINCT OGR_AD FROM OGRENCI_TABLOSU WHERE OGR_
MEMLEKET='KONYA' AND OGR_YAS='20'
```

Bir sorguda **birden fazla kriter ile filtre yapılmak isteniyorsa** yukarıdaki komutta olduğu gibi **`WHERE`** deyiminden sonra **`AND`** ifadesi kullanılır. Eğer şartların ikisi de **aynı anda sağlanmak zorunda değilse** bu sefer **`AND`** ifadesi yerine **`OR`** ifadesi kullanılır. Aslında kitabımızın ilk bölümlerinde yer alan *Sayısal Elektronik* konularındaki *mantık kapıları* başlığında işlediğimiz **`AND`** ve **`OR`** ile aynı mantık ile çalışmaktadır. Eğer şartların ikisi de **aynı anda sağlanmak zorundaysa** **`AND`**, şartlardan **bir tanesinin sağlanmasının yeterli olduğu durumda** ise **`OR`** ifadesi SQL komutlarında **`WHERE`** deyiminden sonra kullanılmaktadır. Aşağıdaki komut örneklerinde **`AND`** ve **`OR`** ifadeleri yer almaktadır.

AND/OR

```
SELECT * FROM OGRENCI_TABLOSU WHERE OGR_MEMLEKET='İZMİR' AND OGR_
YAS='20'
```

Yukarıdaki komut, öğrenci tablosundaki öğrencilerden memleketi yalnızca **`İzmir`** olan ve yaşı yalnızca **`20`** olan öğrencileri listelemektedir. Yaş alanını tablo yapısında **`string`** yani metinsel ifade olarak tutulduğunu varsaydığımız için yaş değerini de **tek tırnak** içine alıyoruz. Eğer yaş değeri tablodaki alanında **tam sayı** yani **`int`** olarak tutulduğunu varsayarsak yaş değerini **tek tırnak içine almamıza gerek yoktur**.

```
SELECT * FROM OGRENCI_TABLOSU WHERE OGR_MEMLEKET='İZMİR' OR OGR_
YAS='20'
```

Yukarıdaki komut ise öğrenci tablosundaki memleketi **İzmir** olan öğrencileri veya yaşı **20** olan öğrencileri listelemeye yaramaktadır.

BETWEEN/NOT BETWEEN

Listelenen veriler belirli iki değer arasında ise **görüntülenmesinin istenmesi durumunda** **BETWEEN** deyimi kullanılmaktadır. **Between arasında ise** anlamına gelirken **NOT BETWEEN** ise **aralık dışında ise getir** anlamına gelmektedir. Aşağıdaki örnek komutlarda **Between** ve **Not Between** komutları yer almaktadır.

```
SELECT * FROM OGRENCI_TABLOSU WHERE OGR_YAS BETWEEN 20 AND 30
```

Bu komut ile öğrenciler tablosunda listelenen **öğrencilerden yaşları 20 ile 30 arasındakilerin getirilmesi sağlanır**.

```
SELECT * FROM OGRENCI_TABLOSU WHERE OGR_YAS NOT BETWEEN '20 AND '30
```

Bu komut ise öğrenciler tablosunda listelenen **öğrencilerden yaşları 20 ile 30 arasında olmayanların getirilmesi sağlanır**.

LIKE

Yapılacak bir sorguda **sütuna ait değer tam olarak bilinmiyor** veya **yazılmak istenmiyorsa** bunun yerine örneğin adı '**ah**…' ile başlayanları sorgula gibi bir sorgunun yazılması gerektiği durumlarda **LIKE** deyimi kullanılır. **LIKE** kullanılırken tablodaki sütun adından sonra **eşittir işleci kullanılmaz**. *Eğer şununla başlayıp herhangi bir şey ile devam edenleri getir* sorgusu yazılacak ise **LIKE** deyiminden sonra aranacak karakterler ve **%** işleci kullanılır. Ama tam tersi *herhangi bir şey ile başlayıp şu karakter ile devam ediyorsa* şeklinde yazılacak bir sorguda **LIKE %** işleci ve aranan karakterler şeklinde yazılır. Aşağıda her iki durumu da **örnekleyen komutlar** bulunmaktadır.

```
SELECT * FROM OGRENCI_TABLOSU WHERE OGR_AD LIKE 'ah%'
```

Bu komut adı '**AH**' ile başlayan öğrencileri listeleyecektir. Örneğin **Ahmet**, **Ahu** gibi.

```
SELECT * FROM OGRENCI_TABLOSU WHERE OGR_AD LIKE '%li'
```

Bu komut ise adı herhangi bir değer ile başlayan fakat kesinlikle '**li**' karakterleri ile biten öğrencileri listelemektedir. Örneğin, **Ali**, **Veli** gibi.

ORDER BY

Yapılan listeleme komutundan dönen verilerin alfabetik olarak **A-Z** veya **Z-A** şeklinde ya da **sayısal olarak küçükten büyüğe** veya **büyükten küçüğe** sıralanarak getirilmesi istendiği durumlarda **ORDER BY** ifadesi kullanılır. Sayısal, alfa sayısal değerlerde olduğu gibi tarih verilerini de küçük tarihten büyük tarihe ya da tam tersi olacak şekilde sıralayabilir. Kullanımına ilişkin **örnek komut** aşağıdaki gibidir.

```
SELECT * FROM OGRENCI_TABLOSU ORDER BY OGR_IL_PLAKA_KODU ASC
```

Yukarıdaki komut herhangi bir **filtre olmaksızın** tüm öğrencileri listelemektedir. Fakat eklenme sırasına göre veya veri tabanının sunacağı varsayılan sıralama kriteri yerine, öğrencilerin memleketlerine göre belirlenen il plaka koduna göre küçükten büyüğe sıralanması sağlanmıştır. **ORDER BY** ifadesinden sonra yer alan **ASC**, **küçükten büyüğe sıralamayı** sağlamaktadır. Eğer **tersten sıralama yapılmak istenirse** aşağıdaki komut kullanılmalıdır.

```
SELECT * FROM OGRENCI_TABLOSU ORDER BY OGR_IL_PLAKA_KODU DESC
```

GROUP BY

Listelenen verilerin **belirli bir sütun değerine göre gruplanarak** getirilmesi için kullanılan ifade **Group By** ifadesidir. Kullanımında dikkat edilecek husus, gruplama değerinin adının **SELECT** ifadesinden **hemen sonra yazılması gerektiğidir**. **Group By** genellikle **Count**, **Sum**, **Avg**, **Max** ve **Min** gibi kümeleme fonksiyonları ile birlikte kullanılır. Aşağıdaki komutlarda **Group By** deyimi örneklendirilmiştir.

```
SELECT OGR_BOLUM FROM OGRENCI_TABLOSU WHERE OGR_AKTIF=1 GROUP BY
OGR_BOLUM
```

Yukarıdaki komut öğrenciler tablosundaki öğrencilerden **aktif olanları bölüme göre gruplandırarak** getirmektedir. Eğer belirlenen sütundaki değerlerin **toplamını alarak gruplandırılmak istenirse** aşağıdaki gibi bir kullanım olacaktır.

SUM

```
SELECT SUM(OGR_BOLUM) FROM OGRENCI_TABLOSU WHERE OGR_BOLUM='İŞLETME'
AND OGR_AKTIF=1 GROUP BY OGR_BOLUM
```

Yukarıdaki komut **aktif** ve **bölümü işletme olan öğrencilerin sayısının toplanarak bir kolonda gruplanmasını** sağlamaktadır.

COUNT

Kritere göre dönen sonuçların sayısını, yani **dönen satır sayısını** veren deyimdir. Sonuçların belirli bir sütun adı altında toplanarak gösterilmesi yaygın bir alışkanlık olduğundan **Group By** ile birlikte de kullanımı yaygındır. Aşağıdaki örnekte maaşı 5.000'ine eşit ve üzerinde olan akademik personel sayısını döndüren sorgu yer almaktadır.

```
SELECT COUNT(AKADEMIK_MAAS) FROM AKADEMIK_CARI_TABLOSU WHERE
CALISMA_DURUMU='AKTİF' AND AKADEMIK_MAAS>=5000
```

HAVING

Group By ile gruplandırılarak **dönen kayıtları sınırlandırmak** amacıyla kullanılan deyimdir. Aşağıda kullanımına ilişkin örnek bulunmaktadır.

```
SELECT SUM(AKADEMIK_MAAS) FROM AKADEMIK_CARI_TABLOSU WHERE CALISMA_
DURUMU='AKTİF' GROUP BY AKADEMIK_MAAS HAVING AKADEMIK_MAAS>=5000
```

IS NULL

Yapılan sorgulama da aranan değerin **null olması durumunda sonuç** döndürür. **Null** sıfır demek değildir. **Henüz değer oluşmamış alanları döndürüyor** gibi düşünebiliriz.

```
SELECT * FROM OGRENCI_TABLOSU WHERE OGRENCI_BORC IS NULL
```

UPDATE

Tablolarda yer alan verilerde **güncelleme işlemini yapabilmek için** kullanılan deyimdir.

```
UPDATE OGRENCI_TABLOSU SET OGRENCI_DONEM='2019/2020 GÜZ' WHERE
OGRENCI_DONEM='2018/2019 BAHAR'
```

Yukarıdaki komut ile dönemi **2018/2019 YAZ** olan öğrencilerin dönem bilgileri **2019/2020 GÜZ** olarak değiştirilir. Aynı tablo için aynı anda birden fazla alanın değişmesi gerekiyor ise **SET** ifadesinden sonra güncellenecek alanlar aralarına **virgül** konularak belirtilebilir. Çoklu **update** işlemine aşağıdaki komutu örnek gösterebiliriz.

```
UPDATE OGRENCI_TABLOSU SET OGRENCI_DONEM='2019/2020 GÜZ', SINIF=3,
SOMESTER=5 WHERE OGRENCI_DONEM='2018/2019 BAHAR' AND SINIF=2
```

2. DELETE

Veri tabanından **kayıtları silmek için** `DELETE` deyimi kullanılır. Kullanımına ilişkin örnek komut aşağıdaki gibidir.

```
DELETE FROM OGRENCI_TABLOSU WHERE ID=1254
```

3. INSERT

Veri tabanına T-SQL komutları ile **veri kaydetmek için** `INSERT INTO` deyimi kullanılır. Kullanımına ilişkin örnek komut aşağıdaki gibidir.

```
INSERT INTO OGRENCI_TABLOSU (AdSoyad,Numara,Memleket) VALUES ('ali
veli','115036000',Ankara)
```

4. CREATE DATABASE

Yeni veri tabanı oluşturma işlemini **komutlar ile gerçekleştirmek istediğimizde** kullanabileceğimiz deyimdir.

```
CREATE DATABASE Yeni_Veri_Tabani_Adi
```

5. DROP DATABASE

Veri tabanının **silinmesi gerektiği durumlarda** bu işlemi komut ile gerçekleştirmek istediğimizde kullanılacak deyimdir.

```
DROP DATABASE Silinecek_Veri_Tabani_Adi
```

6. ALTER DATABASE

Veri tabanında tablolarda yer alan alanlarda **değişiklik yapılacağı zaman** kullanılan komuttur. Yeni bir alan **eklemek**, var olan alanın özelliklerini **değiştirmek** gibi işlemlerde kullanılır. Veri tabanı yönetim sisteminde arayüzden yapılabilecek olan işlemlerin komut ile nasıl yapıldığını aşağıdaki örnekten inceleyebiliriz.

```
ALTER TABLE OGRENCI_TABLOSU ADD Yerlestirme_Turu NVARCHAR(50)
```

7. STORED PROCEDURE

Veri tabanı ile yapılacak olan **bir veya birden fazla işlevi yerine getirmek amacıyla** önceden yapılandırılmış bir veya birden fazla tablo ile işlem gerçekleştiren içerisinde kodlar barındıran yaptığı işi anımsatıcı bir isimlendirme ile **Stored Procedure (SP)** olarak kaydedilen sql cümleciklerinin bir araya geldiği yapılardır. Yapısal yordamlar olarak bilinirler. **Stored Procedure**'er veri tabanında çalıştıkları için oldukça **verimli**dirler.

Veri tabanına yeni bir öğrenci eklemek için kurgulanan **stored procedure** örneği aşağıdaki gibidir.

```
CREATE PROCEDURE OGR_EKLE(
@yeniOgrenciID INT,
@YeniOgrenciAd NVARCHAR(50),
@YeniOgrenciFakulte NVARCHAR(70)
) AS
INSERT INTO OGRENCILER_TABLOSU (ID, Ad, Fakulte) VALUES
(yeniOgrenciID, YeniOgrenciAd, YeniOgrenciFakulte)
DECLARE yeniOgrenciID INT SET yeniOgrenciID = SCOPE_IDENTITY()
Prosedürü çalıştırmak için ise aşağıdaki komut kullanılır.
EXECUTE OGR_EKLE 5,'Ahmet Kara','Spor Bilimleri Yüksekokulu'
```

8. KARAR YAPILARI VE DÖNGÜ

Programlamada olduğu gibi SQL üzerinde de karar yapıları ve döngüler kullanılmaktadır. Veri tabanında bir **karar verilmesi gerektiği durumlarda**; **IF ELSE**, **IF EXIST**, **CASE** karar yapıları kullanılmaktadır. Döngü olarak ise **WHILE** döngüsü yaygın bir şekilde kullanılmaktadır.

IF ELSE

Karar ifadelerinden programlama dillerinden de aşina olduğumuz en yaygın karar ifadesi **eğer komutu**dur. **IF** komutu sonucu dönen değer **TRUE** olduğu zaman sorgu çalıştırılır eğer **IF** sonucu dönen değer **FALSE** olursa bu sefer **ELSE** içerisindeki işlemler gerçekleştirilir. **IF ELSE** için örnek kullanım aşağıdaki gibidir.

```
DECLARE @maas INT
SELECT @maas = COUNT(*) FROM UNIVERSITE_TABLOSU
IF @maas > 10000 BEGIN
PRINT '10 binden büyük değerler var'
END
ELSE BEGIN
PRINT '10 binden daha küçük değerler var'
END
```

IF EXIST

Veri tabanında **bir kaydın yer alıp almadığını sorgulamak için** kullanılan bir komuttur.

```
IF EXISTS(SELECT * FROM sys.tables WHERE Name='OGRENCI_TABLOSU')
BEGIN
PRINT 'OGRENCI_TABLOSU VERİ TABANINDA MEVCUT' END
ELSE BEGIN PRINT 'OGRENCI_TABLOSU adında bir tablo veri tabanı
içerisinde mevcut değil' END
```

CASE

`Case` bir **komut değil, işlev**dir. Bu yüzden `IF` gibi tek başına kullanılabilir bir yapıda değildir. `SELECT`, `UPDATE` gibi **deyimler ile birlikte kullanılır**. Amacı **birden fazla mantıksal karşılaştırma yapılması gerektiği durumlarda tek bir ifade içerisinde işlemi gerçekleştirebilmektir**. Kullanımına ilişkin örnek aşağıdaki gibidir. Örnekte İç Anadolu ve Akdeniz bölgesinden gelen öğrencileri `İç Anadolu` ve `Akdeniz` takma adı ile şehri tanımlı olmayan öğrencileri de diğer takma adı ile getirebileceğimiz `case` sorgusu ele alınmıştır.

```
SELECT AdSoyad, Memleket,
       CASE
       WHEN Memleket = 'ANKARA' THEN 'İÇ ANADOLU'
       WHEN Memleket = 'SİVAS' THEN 'İÇ ANADOLU'
       WHEN Memleket = 'KAYSERİ' THEN 'İÇ ANADOLU'
       WHEN Memleket = 'ADANA' THEN 'AKDENİZ'
       WHEN Memleket = 'MERSİN' THEN 'AKDENİZ'
       ELSE 'DİĞER BÖLGE'
       END AS 'BÖLGE ADI'
FROM OGRENCI_TABLOSU WHERE Memleket IN ('ANKARA','SİVAS','KAYSERİ','
ADANA','MERSİN','İSTANBUL','İZMİR')
```

WHILE

`WHILE` bir **döngü**dür. **Karar yapısı ile karıştırılmamalıdır**. Programlama dillerinde de yer alan döngüler ile aynı mantıkta çalışmaktadır. C# programlama dilindeki `foreach` döngüsü ile çok benzerdir. **Bir şart sağlandığı sürece şu işlemleri gerçekleştir mantığı ile çalışmaktadır**. Döngünün sonlanması için içerisinde işletilen şartlardan `TRUE` ya da `FALSE` değer dönmesi gerekmektedir. `IF ELSE` ve `WHILE` beraber kullanılabilir. `WHILE` kullanımına ilişkin en temel örnek olan sayıların bir sayaç vasıtasıyla oluşturulup 1'den 100'e kadar olanların ekrana yazdırılmasını sağlayan örnek aşağıdaki gibidir.

```
DECLARE @sayac INT = 1;
WHILE @sayac < 101
BEGIN
PRINT @sayac
SET @sayac+=1;
END; GO
```

9. TRIGGERS (TETİKLEYİCİLER)

Tetikleyici anlamına gelen **TRIGGERS**, **bir işlem gerçekleştirildiğinde farklı bir işlemin daha tetiklenerek yapılmasını sağlar**. Örneğin öğrenci ekleme işleminde öğrencinin kayıt türü YKS ise her seferinde ekleyen operatörün YKS seçeneğini seçmesi yerine **otomatik** olarak veri tabanına YKS bilgisinin kaydedilmesini sağlamak için kaydetme işleminde bir **trigger** yazılabilir.

Tetikleyiciler, veri tabanı yapısında **DML işlemleri üzerinde çalışmaktadırlar**. Dolayısıyla **UPDATE**, **INSERT** ve **DELETE** işlemleri ile içli dışlı olunacağı anlaşılmaktadır. Tetikleyicilerin nasıl oluşturulduğu ve detaylandırıldığı ile ilgili aşağıdaki örnekleri incelemeliyiz.

CREATE TRIGGER ilk_ornek On OGRENCI_TABLOSU bu komut **ilk_ornek** adında bir tetikleyici oluşturur ve **OGRENCILER_TABLOSU** adındaki tablo ile çalışılacağını belirtir.

Akabinde **FOR** veya **AFTER** deyimlerinden bir tanesi seçilir. **FOR** deyimi **trigger**'in anında çalışması için kullanılırken **AFTER** tablo üzerinde yapılan işlemler sona erdiği zaman çalışmasını sağlar. **Trigger**'lar çalışması esnasında sanal tablolar kullanırlar, bunlara **Virtual Table** denilmektedir. Tablo üzerindeki ekleme işlemlerinde **INSERTED**, silme işlemlerinde ise **DELETED** sanal tabloları kullanılır.

Öğrenci silme işlemini bir **Trigger** ile gerçekleştirmek istediğimizde aşağıdaki örneği incelemeliyiz.

```
CREATE TRIGGER Sil
ON OGRENCI_TABLOSU
FOR DELETE
AS
PRINT('Bu tablo silinemez')
ROLLBACK TRANSACTION
```

Silme işlemini yapmaya çalışırken **FOR** kullanıldığından **uyarı verilir** ve **sildirmez**. Sonrasında ise yapılan işlemleri **rollback transaction** ile geri alır.

Bir başka örnek ise aşağıdaki gibidir.

```
CREATE TRIGGER YKS_DOLDUR  YKS bilgisini otomatik dolduracak trigger
adı
ON OGRENCI_TABLOSU   öğrenci tablosu üzerinde işlemler yapılacak
AFTER INSERT   ekleme işleminden sonra çalışacak olan bir trigger
AS
BEGIN
    DECLARE @ID int   sanal ID
    SELECT @ID = ID FROM INSERTED   sanal ID işlemi
    UPDATE OGRENCI_TABLOSU SET KAYIT_TURU='YKS' WHERE ID=@ID
END
```

Kayıt yapılırken operatör yalnızca öğrencinin temel bilgilerini sisteme kaydediyor olmasına rağmen öğrencinin kayıt türü alanına **otomatik** olarak '**YKS**' yazılmaktadır.

VERİ YAPILARI (DATA STRUCTERS)

10

Veri yapılarını anlamaya literatürde de olduğu gibi *verinin tanımı* ile başlayalım. **Veri**: Kurguladığımız algoritmalarca ele alınan ve üzerinde yapılan işlemlere göre farklı farklı sonuçlar veren en temel elemandır. Veriler sayısal, alfasayısal, resim veya ses gibi türler barındırmaktadırlar. Bu yüzden her bir tür için yapılabilecek onlarca farklı veri işleme modeli olması olağandır. Kurguladığımız algoritmaların işe yarar olabilmesi verileri **doğru işleyebilmekten** geçer. Bir problemi çok uzun veya çok kısa kod ve algoritmalar ile çözebiliriz ama burada önemli olan yöntemin kısa veya olması değil **işlevsel** ve **içeriğine uygun** olarak yapılmasıdır.

Veri yapıları ise verinin **bellek üzerindeki düzenini** göstermektedir. Tam sayı, karakter, kesirli sayı gibi değişkenleri içerisinde barındıran programlama dillerinin bu elemanları bellekte saklayabilmek için kullandıkları bir veri yapısı söz konusudur. Bilgisayar ve özellikle programlama dünyasında problemlerin çözümü için bir **veri işleme modeline dayandırılarak** çözüme gitme ilkesi benimsenmiştir. Bu yüzden verinin türüne göre **işleyeceği sürece** ve **vereceği çıktıya göre** farklı veri yapıları kalıplaşmıştır. Bu kalıplaşma, veri yapılarının kullanılması sayesinde **doğru program geliştirmeye** de olanak sağlamaktadır. Bir problemin giderilmesinde veri yapılarının kullanılma ve hatta doğru veri yapısının kullanılması işlemlerin daha hızlı ve daha stabil olmasını da sağlayacaktır. Algoritmaların analiz edilmesi bölümünde de anlattığımız gibi *doğru algoritmanın kullanılması hız olarak olumlu yönde fayda sağlamaktadır*.

Veri yapıları eğer programlama dilleri tarafından sabit bildirerek veya değişken tanımlayarak kullanıldı ise bunlara **temel veri yapıları** denilmektedir. Kendisinden önceki veri yapıları üzerinde kurgulanan önceden geçerli veri yapılarının kullanıldığı veri yapılarına ise **tanımlamalı veri yapıları** denilmektedir.

Geliştirilecek olan program için en baştan çalışma hızı konusunda fikir verecek olan yapılardan bir tanesi **kullanılacak veri yapı**larıdır. Programın kod tasarımı, etkinliği ve bellek gereksinimi gibi konularında seçim yapılan veri yapısına göre daha **en başından bilgi verir**. Veri modelleri **Linked List**, **Tree**, **Graph**, **State Machine** vb. başlıklar altında incelenmektedir. Her bir başlığın altında ise çeşitli veri yapıları bulunur.

Linked List (Bağlı Liste)

Düğümlerden (*Node*) oluşan yapılara, **bağlı listeler** denilmektedir. Bağlı liste yapısı, verileri özel bir veri saklama yöntemi kullanarak RAM üzerinde tutmaktadır. Bağlı liste denilmesinin sebebini düğümlerin tutmuş oldukları **iki ayrı bilgiden ötürü** söyleyebiliriz. Bir tanesi diğer düğümün **adresi** diğeri ise düğümde **tutulan veri**dir. Diğer düğümün adres bilgisinin tutulması bağlı liste yapısı için oldukça önemlidir. Bilgilerin RAM'de tutuluyor oluşu bilgiye ulaşırken adresi üzerinden referans alarak ulaşılması anlamına gelmektedir. Düğümlerin adres bilgilerinin doğru bir şekilde tutuluyor olması sayesinde veriler dizi değişkenlerde tutuluyormuşçasına **RAM üzerinde** tutulmaktadır. Bağlı listeler **tek yönlü** listeler (*singly linked list*), **çift yönlü** bağlı listeler (*doubly linked list*) ve **dairesel bağlı** listeler (*circular linked list*) şeklinde üç çeşittir.

Bağlı listeler ile aşağıdaki işlemler gerçekleştirilebilir.

- **Eleman Ekleme İşlemleri**
  - En başa eleman eklenmesi
  - Araya eleman eklenmesi
  - Sona eleman eklenmesi
- **Eleman Silme İşlemleri**
  - En baştan eleman silinmesi
  - Aradan eleman silinmesi
  - Sondan eleman silinmesi
  - Ters çevirme işlemleri

Linked list, yani bağlı listelerde sadeleşmiş bağlı listelere **tek yönlü bağlı listeler** denilmektedir. Tek yönlü denilmesinin altında her bir düğümün kendisinden sonra gelen düğümü **referans etmesi fikri** yatmaktadır. Sürekli ileriye doğru bir refere etme olayı olduğundan tek yönlü denilmektedir. C dili ile yazılan tek yönlü bağlı liste yapı örneği aşağıdaki gibidir.

```
Struct dugum{
int degisken;
struct dugum *next; };
```

Yapısal anlamda tek yönlü bağlı listelere oldukça benzeyen fakat **fazladan işaretçi değişken barındıran listelere ise çift yönlü bağlı liste** denilmektedir. Çift yönlü bağlı listelerde bulunan işaretçi değişken sayesinde kendinden önce gelen düğüm işaret edilir, böylelikle geriye doğru hareket edilebilir. Çift yönlü bağlı listelerde unutulmaması gereken detay, **fazladan barındırılan işaret değişkeninin de bellekte bir yer tuttuğu**dur.

```
Struct dugum{
int degisken;
struct dugum* önceki;
struct dugum* sonraki; };
```

Bağlı listeler ile dizileri **karşılaştırmak** gerekirse dizilerde;

- Eleman silme işlemi daha zordur
- Yeni bir eleman ekleme işlemi daha zordur
- Dizi boyutunu değiştirme işlemi zordur
- Ve dizideki her eleman için hafızada yer ayrılır

Bağlı liste kullanımı ile bu problemler yaşanmaz.

Dairesel bağlı listelerden tek yönlü dairesel bağlı listeler, listedeki son elemanın işaretçisi **ilk listenin ilk elemanının adresini** gösterir. Böylelikle listedeki elemanlardan bir tanesinin bilinmesi **tüm elemanlara erişilmesini** sağlar. İki yönlü dairesel listelerde ise ilk düğümden önceki **düğüm son**, son düğümden sonraki düğüm de **ilk düğüm**dür. Bu da dizilimde yer alan elemanların dairesel biçimde sıralanmasını ve elemanların kolaylıkla bulunmasını sağlamaktadır.

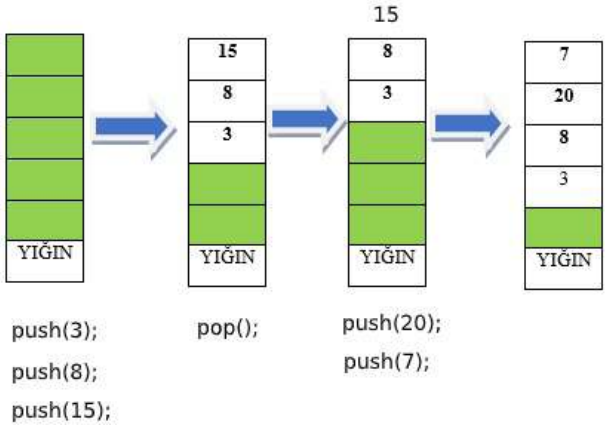
Stack (Yığın)

Yığınlar (*yığıt olarak da bilinir*) LIFO veya FIFO mantığı ile çalışır. **LIFO Last In First Out** yani *son giren ilk çıkar* anlamına gelirken **FIFO First In First Out** ise *ilk giren ilk çıkar* mantığı ile çalışmaktadır. Eleman ekleme çıkarma işlemleri yığının en üstünden yapılır. Yani yeni bir eleman ekleneceğinde veya silineceğinde bu işlem yığının **en üstünden** yapılır. Bu sayede eklenen bir eleman yığında en üstteki eleman olacağından işlemler **LIFO mantığı** ile gerçekleşmiş olur. Çoğu yazılım firmasının işe alım mülakatlarında soru olarak yönelteceği konulardan bir tanesidir. Yığın yapısını kullanabilmek için dizi veya bağlantılı liste kullanılabilir. Programlarda karşılaştığımız **geri al** (UNDO) **işlemleri yığın (stack)** ile yapılmaktadır. Yine *web tarayıcıları, mobil uygulamalardaki geriye alma işlemleri* de yığınlar ile yapılmaktadır.

Yığınlar ile yapılan temel işlemler aşağıdaki gibidir.

- `Empty stack`: Boş yığın.
- `Push` **(koymak)**: Yığına eleman eklemek.
- `Pop` **(almak)**: Yığından eleman çıkarmak.
- `Top`: En üstteki elemanın ne olduğunu gösterir.
- `Size`: Yığında barındırılan nesne sayısını verir.
- `isEmpty`: Yığın boş mu dolu mu bilgisini verir.

Aşağıdaki şekilde boş bir yığına `Push()` ile üç adet veri ekleniyor. Verilerin üstten eklendiğini gösteren şekil `Pop()` ile **en üstteki verinin alındığını** gösteriyor. Daha sonra `Push()` ile iki adet daha veri ekleniyor ve verilerin yine en üstten başlanarak eklendiğine dikkat çekiliyor.



Şekil 10-1: Stack Çalışma Yapısı

Queue (Kuyruk)

Kuyruklar, **doğrusal veri yapıları**dır. Kuyruğa eleman ekleme işlemi sondan yapılırken eleman çıkarma işlemleri **baştan** yapılmaktadır. Dolayısıyla kuyruğa yeni bir eleman ekleneceği zaman bu eleman **kuyruğun sonuna eklenmiş olur**. Kuyruktan eleman çıkarılacağı zaman ise kuyrukta bulunan **ilk eleman** çıkartılır. Kuyruklar bu çalışma prensibi sebebiyle FIFO ve LIFO listesi olarak da bilinirler. Bilgisayar dünyasında örneklendirecek olursak işletim sistemlerindeki işlem kuyruğunu ve yazıcı yazdırma işlem sırası gibi uygulamalardan bahsedebiliriz. Kuyruklar ile yapılan temel işlemler aşağıdaki gibidir.

- **`enqueue (object)`**: Kuyruğa ekleme işlemi yapar yani nesneyi kuyruğun en sonuna ekler.
- **`Object dequeue()`**: Kuyruktan çıkarma işlemi yapar yani baştaki nesneyi getirir.
- **`Object front() (getHead/getFront)`**: Kuyruğun başındaki nesneyi kuyruktan çıkarmadan geri döndürür.
- **`Integer size()`**: Kuyrukta saklanan nesne sayısını geri döndürür.
- **`Boolean isEmpty()`**: Kuyruk içerisinde bir veri olup olmadığına, yani nesne olup olmadığına bakar.

Recursion (Özyineleme)

Kendi kendini çağıran fonksiyonlara **özyineli** (*recursive*) fonksiyonlar denilmektedir. Özyineleme işlemi, **bir işlemdeki sonucunu kendinden önceki bir veya iki işlem sonucuna bağlı olarak bulunması prensibine** göre işletir. Problem çözme tekniği olarak ise özyineleme veri yapısı ile kurgulanan bir sistemde, bir problem benzer şekle sahip daha basit alt problemlere bölünür ve bu şekilde çözülmesi sağlanır. Sorun çözülene kadar alt problemler de başka alt problemlere bölünmektedir. **İteratif yapılara karşı özyinelemeli yapılar kullanılabilir**. Bu çalışma yapısıyla özyinelemeli mekanizmalar güçlü bir problem çözme yöntemi hâline gelmektedir.

Hashing

Hashing, **mevcut veriyi kullanarak o veriden mümkün olduğu kadar benzersiz bir tam sayı elde edilmesi** işlemidir. Elde edilecek olan tam sayı, verilerin indisi gibi kullanılır ve böylelikle verilere doğrudan erişilmesi mümkün kılınır.

Hashing fonksiyonları aşağıdaki gibidir.

- `RadixTransformation` (*Taban dönüşümü*)
- `Extraction` (*Çıkarma/çıkarım*)
- `Division` (*Bölme*)
- `Mid-Square` (*Kare*)
- `Folding` (*Katlama*)
- `SelectingDigits` (*Digit Seçme*)

Hashing Çakışmaları aşağıdaki gibidir.

- **Chaining** (*Zincirleme*)
- **Quadratic Probing** (*Kuadratik Doldurma*)
- **Linear Probing** (*Doğrusal Doldurma*)
- **Double Hashing** (*Çift Kırpma*)

- Hashing veri yapılarında arama işlemleri temel işlem anahtarları ile karşılaştırılır.
- Anahtarın tablo içerisinde bulunduğu pozisyona ulaşıncaya kadar arama işlemine devam edilir.
- **`Hash`** fonksiyonu ile aranan anahtar elemanına doğrudan erişebilmektedir.
- **`Hash`** fonksiyonları, anahtarların tablolarda tutulduğu **`index`** sıralarını vermektedirler.
- Her bir anahtara yalnızca bir pozisyon eşleyen fonksiyonlara **`perfect hash`** denilmektedir.
- Tabloda boş yer olmaması durumundaki her bir anahtarın bir pozisyona eşlendiği fonksiyonlara **`simple perfect hash`** denilmektedir.

Tree (Ağaç Veri Modeli)

Ağaç veri yapıları belirli bir sayıda düğüm içeren ve onları birbirine ulaştıran dalları olan bir veri modelidir. Önceki veri modelleri gibi **doğrusal bir yapıya sahip değildir**. Genellikle yukarıdan aşağıya bir ağaç yapısını düşündüğümüzde hiyerarşik ilişkileri göstermek için kullanılırlar. Düğüm ve kenar denilen parçalardan oluşurlar. Her **bir düğüm (node) bir nesneyi gösterir**. **Her bir kenar (edge) ise iki düğüm arasındaki ilişkiyi**, bağlantıyı göstermektedir. Ağaç veri yapılarını genellikle dosya sistemlerinde görebiliriz. Ağaç modellerinde ağacın sağ altı ile sol altı arasındaki yükseklik farkının en fazla bir düğüm olduğu modellere **AVL ağacı** denilmektedir. Elbette bu kural ağacın tüm düğümleri için geçerli olmak durumundadır.

Ağaç veri yapılarında her düğümün değeri **`Child`** düğümlerden küçük değil ise bu ağaç için **maksimum yığın** (*max heap*) ağaç veri yapısıdır denilebilir. Tam tersi ağaçtaki her düğümün değeri **`child`** düğümlerden büyük değilse bu sefer ağaç veri yapısında **minimum yığın** (*min heap*) denilmektedir. Bir başka ağaç veri yapısı türü çok yollu veya çok kollu olarak bilinen **Multi-Way** ağaç veri yapılarıdır. Bu veri yapılarında bir **m-way arama** ağacındaki her bir **`node`** için **m-1 adet** anahtar (*key*) ve **m tane child**'a sahiptir. Bir düğümdeki anahtar (*key*), ağacın sol altında bulunan tüm anahtarlarından **büyük** ve sağ altında bulunan tüm anahtarlardan **küçüktür** olarak tanımlıdır. Çok yollu ağaç veri yapıları genellikle diskler üzerindeki bilgilerin elde edilmesi durumunda kullanılır. Çok yollu ağaçlar **B-Trees** ve **B*-Trees** olarak çeşitlenir.

Bubble Sort (Kabarcık Sıralaması)

Kabarcık sıralaması olarak bilinen veri yapısı modeli, **yaygın modellerden bir tanesidir**. Bir dizi düşünün ve bu dizinin elemanları üzerindeki ilk indeks değerine sahip olandan başlayarak birbirinin yanı başında bulunan **iki tane elemanın** kendi aralarında sıralanması ile yapılır. Dizinin en başından en sonuna kadar tüm elemanlar için bu işlemin gerçekleştirildiği düşünüldüğünden dizinin son elemanı en büyük eleman konumuna gelmiş olacaktır. Yani **küçükten büyüğe** doğru sıralandığını da düşünebiliriz. Bu sıralamadan sonra yapılacak bir diğer sıralama ile **en sağda bulunan elemanı dışarıda bırakarak sıralama yapılır**. Dışarıda bırakma işlemi ile döngüdeki sayaç değişkeninin değerinin her seferinde bir eksilmesini sağlanır. Sayaç değişkeninin değeri 1'e ulaştığında ise dizinin en solunda kalan son iki eleman da kendi aralarında sıralanırlar ve sıralama işlemi bu şekilde sonlanmış olur. Örnek kod bloğu aşağıda verilmiştir. Örnek kod bloğu **anonim kodlardan** oluşmaktadır. İnternet üzerindeki pek çok kaynakta benzer kodları görebilirsiniz.

```
void KabarcikSiralamasi(int [] x)
{ int n = x.Length; int temp, j, del;
for (del=0; del<n-1; del++)
{ for(j=0; j<n-del-1; j++)
{ if (x[j] > x[j+1])
{
temp = x[j];
x[j] = x[j+1];
x[j+1] = temp;
} }
} }
```

Quick Sort (Hizli Siralama)

Hızlı sıralamada, dizideki elemanların **en başından veya en sonundan eleman seçmek diye bir işlem bulunmaz**. Bunun yerine sıralanacak diziden **rastgele** bir eleman seçilir. Seçilen bu elemana A diyecek olursak A elemanı B. yere konumlandırıldığında dizilim sıfır ile B-1'inci yerler arasındaki elemanlar A'dan küçük, B+1'den (n-1)'e kadar olan elemanlar A'dan büyük olacaktır şeklinde yorumlanabilir. Bu işlemler **A[0]-A[B-1]** ve **A[B+1]-A[n-1]** alt dizilimleri için de tekrarlandığında dizideki veriler sıralanmış olacaktır. En küçük hangisi en büyük hangisi olan tespit edildikten sonra büyük olan ile kıyas yapılsın ve bu sayede sıralansın gibi kaygıları olmadığından **hızlı sıralama** olarak adlandırılır. Örnek kod bloğu aşağıda verilmiştir. Örnek kod bloğu anonim kodlardan oluşmaktadır. İnternet üzerindeki pek çok kaynakta benzer kodları görebilirsiniz.

```
void HizliSiralama (double *left, double *right){
 double *p = left, *q = right, w, x=*(left+(right-left>>1));
printf("%f ",*(left+(right-left>>1)));
printf("r %d\n",right); printf("l %d\n",left);*/
 do {
 while(*p<x) p++;
 while(*q>x) q--;
 if(p>q) break;
 w = *p; *p = *q; *q = w;
 } while(++p <= --q);
 if(left<q) qsort2(left,q);
 if(p<right) qsort2(p,right); }
void main(){
 double dizi[10] = { 15, 45, 23, 17, 88, 35, 13, 28, 75, 98 };
 HizliSiralama ( &dizi[0], &dizi[7] );
 for(int i=0;i<8;i++) printf("%f ",dizi[i]);}
```

Selection Sort (Seçmeli Sıralama)

Seçmeli sıralama algoritması, öncelikle dizideki elemanlardan **en küçüğünü tespit eder** ve **bu elemana odaklanır**. Daha sonra dizideki konum olarak son sırada yer alan eleman ile bulunan en küçük elemanın yer değiştirilmesi sağlanır. Aynı şekilde dizideki ikinci en küçük eleman da bulunur ve bu eleman dizideki konum olarak ikinci sırada bulunan eleman ile yer değiştirilir. Kısacası bu işlem dizideki tüm elemanlar **sıralanana dek** bu şekilde yapılmaya devam eder. Elemanların seçildikten sonra uygun yere konumlanması ile sıralanma işlemi tamamlandığından seçmeli sıralama adını almıştır. Örnek kod bloğu aşağıda verilmiştir. Örnek kod bloğu **anonim kodlardan** oluşmaktadır. İnternet üzerindeki pek çok kaynakta benzer kodları görebilirsiniz.

```
void SecmeliSiralama(int x[], int n) {
 int i, indis, j, uzn;
 for(i=0; i<n; i++) {
 uzn = x[i]; indis = i;
 for(j=i+1; j<n; j++)
 if (x[j] < uzn) { uzn = x[j]; indis = j; printf("a=%d \n ",x[j]); }

 x[indis] = x[i];
 x[i] = uzn; } }
void main() {
 int dizi[10] = {15, 45, 23, 17, 88, 35, 13, 28, 75, 98};
 SecmeliSiralama (&dizi[0], 10);
 for(int i=0;i<10;i++) printf("%d\n ",dizi[i]);
}
```

Insertion Sort (Eklemeli Sıralama)

Eklemeli sıralama veri yapısında sıralama işlemi, **elemanların sırasına uygun olarak listeye tek tek eklenmesi** ile gerçekleşmektedir. Dizideki elemanlar tek tek yerleştirilerek sıralama işlemi belli bir anda dizinin belli bir kısmını **sıralı tutar** ve **bu işlemi her adımda biraz daha genişleterek çalıştırır**. Sıralı kısımlar işlem son bulunca dizinin tamamını sıralamış olur ve işlem tamamlanmış olur. Örnek kod bloğu aşağıda verilmiştir. Örnek kod bloğu **anonim kodlardan** oluşmaktadır. İnternet üzerindeki pek çok kaynakta benzer kodları görebilirsiniz.

```
void eklemeliSiralama(int x[], int n)
{
int i,k,y;
for(k=1; k<n; k++)
{
y=x[k];
for(i=k-1; i>=0 && y<x[i]; i--)
x[i+1]=x[i];
x[i+1]=y;
}; }
```

Merge Sort (Birleştirmeli Sıralama)

Birleştirmeli sıralama veri yapısı, **verinin hafızada sıralı olarak tutulması için geliştirilmiş sıralama modellerinden bir tanesi**dir. Sıralanacak olan diziye ikişer elemandan oluşan parçalara indirgeyene dek **sürekli olarak ikiye böler** en son bu parçaları da kendi aralarında sıraladıktan sonra **birleştirir**. Parçalı olarak sıralanan ve en sonunda birleşen dizi sıralanmış bir dizi hâline gelmiş olur. Birleştirme veri yapısına ait örnek kod bloğu aşağıda verilmiştir. Örnek kod bloğu anonim kodlardan oluşmaktadır. İnternet üzerindeki pek çok kaynakta benzer kodları görebilirsiniz.

```
#include <conio.h>
#define Buyukluk 10
void BirlestirmeSiralamasi(int x[], int n) {
 int aux[Buyukluk], i,j,k,L1,L2,size,u1,u2,islem=0; size = 1;
 while(size<n) {
 L1 = 0; islem++; k = 0; printf("İşlem Sayısı:%d\n",islem); getch();
 while(L1+size<n) {
 L2 = L1+size; u1 = L2-1; u2 = (L2+size-1<n) ? L2+size-1 : n-1;
 for(i=L1,j=L2; i<=u1 && j<=u2; k++){
 if(x[i]<=x[j]) aux[k]=x[i++]; else aux[k]=x[j++];
printf("aux[]=%d\n",aux[k]);
 }
 for(;i<=u1;k++) { aux[k] = x[i++]; printf("aux[]=%d\n",aux[k]);}
 for(;j<=u2;k++) {aux[k] = x[j++]; printf("aux[]=%d\n",aux[k]);}
 L1 = u2+1;
 }
 for(i=L1;k<n;i++){ aux[k++] = x[i]; printf("aux[]=%d\n",aux[k]);}
 for(i=0;i<n;i++) x[i]=aux[i];
 size*=2;
 }
}
void main() {
 int dizi[10] = { 15,45, 23, 17, 88, 35, 13,28, 75, 98 };
BirlestirmeSiralamasi( &dizi[0], 8);
for(int i=0;i<8;i++) printf("%d\n ",dizi[i]);}
```

Sıralama algoritmalarında **`Shell Sort`**, **`Radix Sort`** vb. daha birçok sıralama veri modeli görmek mümkündür. Bu bölümde amaç; veri yapılarının ne oldukları ve sıralama veri modellerinden bazılarını örnek kodları ile birlikte incelenmesidir. Konu ile ilgili yazılmış *pek çok kaynak* ve *kitap* bulunmaktadır. Kitabımızın bu bölümü **veri yapılarının ne oldukları hakkında fikir verme amacı** taşıdığından detaylı bilgilere çok fazla girilmediğini söylememizde sakınca yoktur. Ayrıca örnek kodları da internet taraması yapan okurlarımızın kafalarının çok fazla karışmaması için genellik her kaynakta, kitapta veya blog sayfalarında veri yapıları denilince hazırlanan basit kod bloklarından seçtiğim için çok rahat **üzerinde geliştirme değişiklik yapabilirsiniz**. Veri yapılarını mühendislik fakülteleri veya bilişim alanında eğitim veren bölümlerde **veri yapıları** (*data structers*) dersi içerisinde, **otomata teorisi** (*automata theory*) dersinde veya **discrete matematik** (*discrete mathematics*) derslerinde grafları hatta dijkstra algoritması gibi konuları da içeren versiyonları ile üniversite öğreniminde karşımıza gelmektedir.

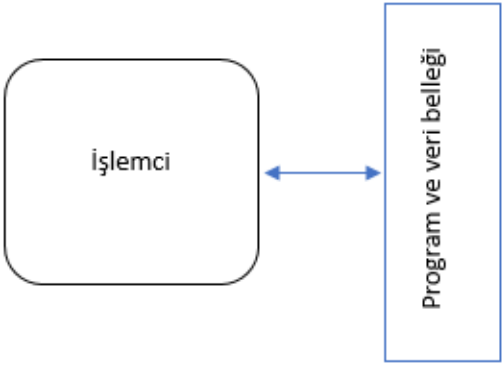
BİLGİSAYAR MİMARİSİ

11

Bilgisayarların; işlemci saat frekansları, bellek kapasiteleri, veri yolları gibi bilgisayarları kullanabilmemiz için gerekli olan detaylı özelliklerini inceleyen alana **bilgisayar mimarisi** denir. Bilgisayar sistem tasarımcılarının bir bilgisayar sistemi tasarlarken uyacağı kuralları, yöntemleri ve seçilen mimari gibi konular ise **bilgisayar organizasyonunu** ilgilendirmektedir. Kitabımızın ilk bölümlerinde *temel kavramlarda anakart, işlemci, ekran kartı nedir ve nasıl çalışır* gibi sorulara cevap bulmuştuk. Bu bölümde ise bilgisayarları **mimari seviyede** inceleceğiz. Örneğin bilgisayar tasarımcıları olarak üretici firmalarda çalışan mühendisleri baz aldığımızda, mühendisler bilgisayarın yazılım ve donanım bileşenlerinden oluşan **elektromanyetik bir cihaz** olduğunu bilmek zorunda ve tüm tasarımı buna göre yapmak durumundadır. Bu yüzden bilgisayarların mimarileri ve organizasyonları da önem teşkil etmektedir. Öte yandan son kullanıcı için bilgisayarın çalışması ve işlemlerini gerçekleştirebiliyor olması önemlidir. Son kullanıcı hangi bileşenlerden oluştuğuna veya mimarisinin ne olduğunu bilmek zorunda değildir. Bilgisayar mühendisleri veya bu işe gönül koymuş kimseler olarak elbette üretici seviyesinde her şeyi bilmek zorunda değiliz ama Apple markalı cihazların neden diğer muadil markalara göre daha stabil çalıştığına dair bir cevap vermek durumundayız. Bu yüzden *bu bölümde anlatılacak olanlar, günlük hayatta karşımıza çıkacak sorular için önemlidir*.

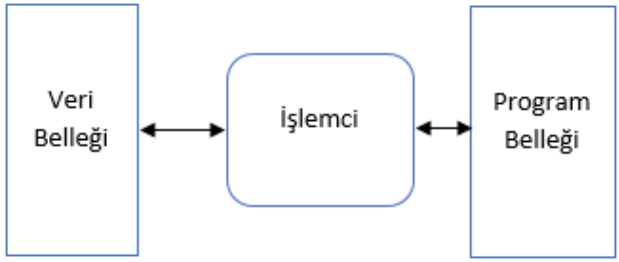
Bir problemin çözümü hem **yazılım** ile hem de **donanım** ile gerçekleştirilebilir. Çözümde dikkat edilmesi gereken kriter maliyet ve işin tamamlanma süresi olduğunda **donanım ile yapılacak çözümler pahalı** olabilir. Diğer yandan **yazılımla yapılan çözümde ekleme, silme veya düzeltme işlemleri çok daha kolay** bir şekilde gerçekleştirilebilir. Donanım ile yapılan çözüm, doğrudan donanım kullanılması sebebiyle daha *hızlı* olurken yazılım ile yapılan çözüm biraz daha *yavaş* kalabilir. Bu çözümlerden yerine ve şartlarına göre uygun olan bir tanesi seçilmelidir fakat bu iki çözümün de arasında kalan başka bir çözüm **mikroprogram yöntemidir**. Bu yöntem yazılım ile yapılan çözümdeki esneklik ile donanım ile yapılan çözümdeki hız etkenini bir arada tutarak oldukça **kullanışlı bir yöntem** sunmaktadır. Bilginin bilgisayarlar üzerinde depolanarak mikroprogram mimarisi tarzıyla işletildiği iki tür mimari çeşitlenmiştir. Bilgisayar tarihinde çok önemli bir yeri olan bu mimariler **1. Von-Neuman mimarisi** ve **2. Harvard mimarisi**dir.

Von-Neuman mimarisi ilk önce **ENIAC** bilgisayarında uygulanmış ve günümüz bilgisayarlarının temelini oluşturmaktadır. Bilgisayarda işleme alınacak olan verilerin **tek bir ana bellek** üzerinde tutulduğu mimaridir. Bu mimaride bilgisayarlar beş ayrı bloktan oluşan bir yapı şeklinde tasarlanmıştır. Bunlar **Giriş Bloğu**, **Çıkış Bloğu**, **Bellek Bloğu**, **Aritmetik-Lojik Bloğu** ve son olarak **Kontrol Bloğu** şeklindedir. İşlemcinin yapısı ile ilgili olan bu kavramları ilk bölümlerimizde de işlemiştik. **Von-Neuman** mimarisinde işletilen komutlar *ardışıklık ilkesi* ile çalışmaktadırlar. Bu bir komutun yürütülme işlemi bitmeden sıradaki diğer komutun yürütülmemesi anlamına gelmektedir. Yani bu mimari veri işleme işini tam bir **ardışıklık sistemi** ile yürütmektedir.



Şekil 11-1: Von-Neuman Mimarisi

Harvard mimarisi ise tek ana bellek değil **iki ana bellek** kullanmaktadır. Belleklerin bir tanesinde *programları* diğerinde ise *verileri* tutmaktadır. Tanıma göre oldukça verimli çalışacağını düşündüğümüz bu mimari dönemin, donanım üretimi pahalılığı nedeniyle son kullanıcı seviyesinde bilgisayarların çok fazla **üretilmemesi** ile sonuçlanmıştır.



Şekil 11-2: Harvard Mimarisi

Günümüz işlemci mimarilerini düşündüğümüzde ise aslında Von-Neuman ve Harvard mimarilerinin hybrid/karma bir şekilde çalıştığını söyleyebiliriz. Ana bellek sanal bellek olarak iki kısma bölünür ve her ikisi de ayrı belleklermiş gibi kullanılabilir. Böylece verilerin ve programların depolanması farklı yerlerde yapılıyormuş gibi bir sonuç olur. Elbette burada bahsedilen bellek **sabit disk veya geçici depolama birimi değil**dir. İşlemcinin çalışma mekanizmasını anlattığımız bu bölümde, komut kümesindeki komutların sayısına göre işlemciler karmaşıklaşabilmektedir. Örneğin bir işlemcide ne kadar çok komut kümesi varsa işlemci o kadar karmaşık bir yapıya sahip olacaktır. Aynı anda *çok sayılı komutları çalıştıran işlemci mimarisine* ise **Very Long Instruction Word** kelimelerinin baş harflerinin kısaltması ile oluşan **VLIW** işlemci mimarisi denilmektedir.

RISC Mimarisi

Komut kümesindeki sayılarına göre işlemciler iki tür olarak sınıflandırılmaktadırlar. İşlemci mimarisindeki **temel komutların sayısı 250 civarındaysa** (*az ise*) bu tür işlemcilere **azaltılmış komut kümeli bilgisayar** denilmektedir. İngilizce karşılığı **Reduced Instruction Set Computer** kelimelerinin baş harflerinin kısaltması ile **RISC** olarak bilinmektedir.

RISC mimarisinde komutlar ve veriler için ayrı önbellekler bulunmaktadır. İşlemcinin yapısı kısmen basit kabul edilebilir fakat yazılım **CISC mimarisine göre daha zordur**. Bu mimari, hesaplama işlemlerinde hızı arttırmaya yönelik olarak özellikle Motorola işlemcilerinde çok kullanılmıştır. RISC mimariler yukarıda da bahsettiğimiz gibi **komut sayısı az olan mimarilerdir**. Bu komutların uzunluklarının az olması, adresleme yapılırken adresleme modları sayısının da az olmasını sağlamaktadır. Komut sayısının az oluşu karmaşık işlemlerde birden fazla komut kullanılmasına yol açtığından ortaya çıkan **programlar uzun** ve **bellekte fazla yer tutmaktadır**. Bu yüzden yerden tasarruf sağlayan bir mimari olduğunu söyleyemeyiz. Ayrıca donanımsal olarak tasarlandığı için çalışma **hızı oldukça iyi fakat güncellenmesi oldukça zor** olan bir mimaridir.

Avantajları;

- Hızlıdır.
- Donanımı oldukça yalındır.
- Tasarlaması kolay ve çok zaman almaz.
- Genel kaydedicilerin sayısının çokluğu işlemler esnasında önbelleğe erişim ziyaretini aza indirmektedir.

Dezavantajları;

- İşlemcide yürütülen programlar bellek üzerinde çok yer tutarlar.
- İşletim sistemi ve derleyiciye CISC mimarisine oranla daha çok yük düşer.
- Kontrol birimi donanımsal olduğu için komut kümesine eklemelerin yapılması mümkün olmaz.

CISC Mimarisi

İşlemcide kullanılan komut sayısı **500 civarında ise** (*çok ise*) işlemcinin yapısının karmaşık bir hâl alacağını bildiğimizden dolayı böyle işlemcilere **karmaşık komut kümeli bilgisayarlar** denilmektedir. **Complex Instruction Set Computer** kelimelerinin baş harflerinden oluşan bu mimariye **CISC** mimarisi denilmektedir.

CISC mimarisine sahip işlemcilere örnek vermek gerekirse günümüz Intel ve AMD işlemcilerini, RISC mimarisine örnek olarak Apple, Power, Motorola, SunSparc işlemcinilerini VLIW işlemcilere ise günümüz çok çekirdekli işlemcileri, AMD Zen veya Ryzen mimarili işlemcileri örneklendirebiliriz. **İşlemcilerin karmaşık fakat yazılımın kolay olduğu bir** mimaridir. CISC mimarisi yüksek seviyeli programlama dillerini de desteklemek amacıyla geliştirilmiştir. CISC mimarisinde işlemci çok sayıda komut barındırdığından programlama **oldukça kolaylaşmaktadır** ayrıca bellek kullanımı da etkin olduğundan işlemci karmaşık bir mimariye sahip olsa da **yazılım ile basitleşmekte** ve **eklemeler kolaylıkla yapılabilmektedir**.

Avantajları;

- Komut sayısı çok olduğundan programlama kolaydır.
- Komutların uzunlukları değişkenlerin uzunluğuna bağlı olduğu için adresleme modları sayısı stabildir.
- Genel kaydedicilerin sayısı RISC mimarisinde olduğu gibi 32 değil 8 adet civarındadır.
- Komut sayısı çok olduğundan karmaşık işlemleri tek bir komut ile yürütebilir bu da yerden kazanım sağlar.
- İşlemci yazılım ile yapıldığından mikroprogram tekniği kullanılır ve programlar ROM'da tutulur. Bu da programlanabilmesini sağlar.

Dezavantajları;

- Yazılım ile yapıldığından RISC mimarili işlemcilere göre daha yavaş kalır.
- Tasarım süresi yine yazılım olduğundan daha yavaşlar.
- Genel kaydedicilerin sayısının az olması sebebiyle sık sık önbellek erişimi istenebilir bu da hız kaybına yol açacaktır.

İşlemci Bellek Hiyerarşisi

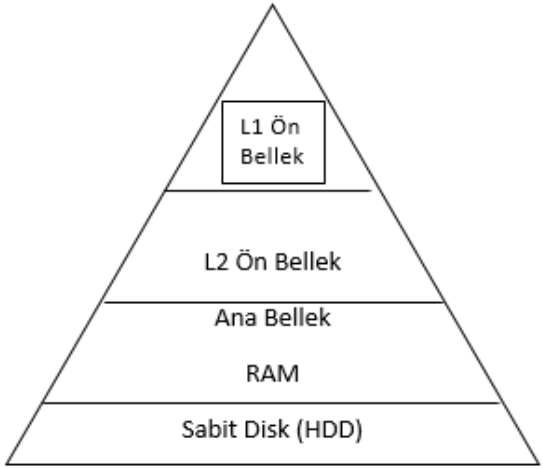
Kitabımızın ilk bölümlerinde işlemiş olduğumuz işlemci ve bellek konularından da hatırlayacağımız üzere işlemcilerde yer alan **önbellekler** bulunmaktaydı. İşlemcinin üzerinde bulunan önbelleklere **Cache bellek** de denilmekteyken işlemci ile doğrudan iletişim kurabilen ve adreslenebilen belleklere **ana bellek** denilmektedir. Ayrıca sabit diskleri işlerken bahsettiğimiz diğer bir bellek türü de **sanal bellekler**dir. Sanal bellekler, fiziksel belleğin kapasitesinin yetersiz kaldığı durumlarda **sabit diskin bir bölümünün ana bellek gibi davranmasına** verilen isimdir.

Günümüzdeki işlemcilerin saat frekansının, yani hızının çok olması sabit disk veya geçici depoma birimi olan ana belleğin **yavaş** kalmasına neden olmaktadır. İşlemci hızına yetişecek ana belleğin dışında en az işlemci kadar hızlı ara bir bellek ihtiyacı daha hasıl olduğundan işlemcilerdeki önbellekler tasarlanmıştır. İlk tanımda da kısaca bahsedildiği gibi önbellekler, **ara bellek** veya **tampon bellek** olarak da adlandırılır. İngilizce karşılığı ise **temporary** veya **cache bellek** olarak bilinir. İlk bölümlerde işlemci başlığında anlattığımız gibi bir işlemci, bir veriyi sabit disk üzerinden alıp okurken **veri öncelikle ana belleğe yazılır** buradan da **önbelleklenir**. İşlemci veriyi okuyacağı zaman öncelikle önbelleğe bakar eğer orada adreslenmiş bir veri var ve değişikliğe uğramamış ise direkt oradan okur ama daha önce okunmamış veya değişikliğe uğramış ise tekrar ilk çalışma sistemini tekrarlar ve veriyi yükler.



Önbellek, geçici bellek ve sabit disk üçlüsünden oluşan bu mekanizmayı bir de programlamadaki **Heap**, **Stack** ve **Pointer** konularını tekrar ederek yorumlayınız.

Verinin işletilmesi ile ilgili olan senaryoda sabit diskten geçici belleğe oradan da işlemci üzerindeki önbelleklere yüklenerek işlemciye verilen yapıda bellek hiyerarşisinden söz edecek olursak işlemciye en yakın olan önbellekler **en üstte** ve işlemciye en uzak dolayısıyla en yavaş olan sabit diskte **grafiğin tabanında** yer alacaktır. **Şekil 11-3** bellek hiyerarşisini göstermektedir.



Şekil 11-3: Bellek Hiyerarşisi

ALGORİTMALARIN ANALİZİ (ANALYSIS OF ALGORITHMS)

12

Algoritma ve programlamaya giriş bölümünde algoritmanın *temel tanımını*, *nasıl kurgulandığını* ve *akış şemaları ile gösterimini* işlemiştik. Programlamaya giriş bölümünde ise oluşturulan algoritmaya uygun programlamanın yapılabilmesi için **temel bilgileri** işlemiştik. Bu bölüm, biraz daha ileri seviye bir bölüm olmakla birlikte lisans veya yüksek lisans seviyelerinde ders olarak tam da bizim başlığımızdaki gibi yer almaktadır. Çeşitli algoritmaların, özellikle sıralama algoritmalarının analiz edildiği bu konu başlığında, belli başlı **algoritmalar analiz edilecek** ve **şekiller ile çalışma mantıkları** gösterilecektir. Çok yaygın bir konu olan algoritmaların analizi soyut olması sebebiyle biraz anlaşılması güç gibi durabilir fakat bu konu ile ilgili özellikle internette değerli yazarların çeşitli kaynaklarına rastlayabilirsiniz.

Kitabımızın birinci bölümden itibaren sürekli bilgi teknolojilerinin yeni araştırma konularının **optimizasyon**, **hız** ve dolayısıyla **zaman** olduğundan bahsetmiştik. Bu bölümde de yine altını çizerek bilgi teknolojilerinin önemli iki araştırma alanlarından **Hafıza** ve **Zaman** kaynaklarına değinilmektedir. İlk bilgisayarın bir oda büyüklüğünde olduğunu hatırladığımızda yıllar boyunca süre gelen hız ve hafıza problemleri sebebiyle boyut olarak küçülen bilgisayarlar hız artışı bakımından da epeyce yol almıştır. Buradan hareketle bilgisayarlar içerisinde çalışan programların da optimize edilebilmesi gerekliliğinden ötürü algoritmalar ele alınmış ve analiz edilen algoritmalar **doğru yerde doğru şekilde kullanılarak** hızlı çalışan ve çok daha az yer tutan programların ortaya çıkması kaçınılmaz olmuştur.

Algoritmaların analiz edilmesi ile ilgili ihtiyacın nereden çıktığını sorguluyor olabilirsiniz. Çok basit bir **örnek** ile konuyu biraz daha detaylandıralım. Örneğin bir elektronik üretim tesisinde, bir yıl içerisinde üretilen beyaz eşya ve bilgisayar grubundaki ürünlerin en çok satış değerine sahip olanlarından başlayarak sıralanması gereksin. Büyük bir veri olması muhtemeldir, bu yüzden bu sıralamanın hangi algoritma ile yapılacağı bilinmelidir. Doğrudan ilk akla gelen sıralama yöntemi veriyi sıralayacaktır fakat süre olarak belki çok uzun zaman alacaktır. Bu da hem *sonuç odaklılık açısından* hem de *bilgisayarın bu sürede kapanma, bozulma vb*. risklerinden tercih edilmez. Bu tür bir sıralamayı yaklaşık olarak yirmi farklı yöntemle gerçekleştirebiliriz ama bilgisayar mühendisi olarak bu işi **deneme yanılma ile yapamayız**. Bilgisayar mühendislerini belki de geleneksel programcılardan ayıran önemli özelliklerden bir tanesi bu konu başlığını **okullarda ders olarak alıyor olmaları**dır. Bir bilgisayar mühendisi olarak sıralama algoritmalarını çok iyi biliyor olabiliriz fakat bilmemiz gereken bir diğer işlem de seçeceğimiz sıralama algoritmasının kullanmak zorunda olduğumuz bilgisayarda **performanslı** bir şekilde çalışması gerektiğidir. Çok kısa sürede sıralama işlemini tamamlayacak olan algoritmayı eğer uygun olmayan kapasiteye sahip bir bilgisayarda çalıştırırsak **zaman donanımsal olarak hafıza problemi yaşarız**. Bu sebeple uygun algoritmayı uygun donanım seviyesine göre seçmeliyiz. Konuyu mevcut örneğimiz ile detaylandırmaya devam ettiğimizde uygun algoritmanın ne kadar sürede işlemi tamamlayacağının ne şekilde bulunacağıdır. Algoritmaları tek tek denemenin yersiz olacağını bildiğimize göre bildiğimiz sıralama algoritmalarından hangisinin **en iyisi** olduğunu çok daha kolay bir şekilde tespit etmemiz gerekecektir. Bilgisayar mühendisliğinde **algoritmaların analizi** bu tür durumlar için cevap niteliğindedir. Daha açık bir ifade ile algoritmaların analizi, bizlere bir algoritmayı çalıştırmadan da analiz edilmesinin ne kadar sürede işlemi gerçekleştirebileceği hakkında bilgiler sunar. Algoritma analizi bilgisayar mühendisliği ders müfredatlarında ayrık matematik dersinde de görebileceğiniz gibi matematikteki **zaman karmaşıklığı** (*time complexity*) teoreminin bir uygulama alanıdır. Algoritmaların analiz edilmesinde algoritmanın performansı ve zaman ilişkisi hakkında matematiksel yöntemlerle bize bilgi verir. Bu yüzden bir mühendis tarafından geliştirilen kod ile yaptıkları işe son derece saygı duyduğum geleneksel programcıların geliştirdikleri kod arasında çok fark bulunur. Baştan **iyi kurgulanmış bir yazılım** hem **performans** hem de **hafıza sorunun ortadan kaldırır**.

Bu kodlamanın içerisinde **tasarım** ve **güvenli tasarım** örüntüleri de olduğu zaman bir mühendise yakışır biçimde kodlama yapılmış olacaktır.

Verilerin aranması ve sıralanması ile ilgili vermiş olduğum bu örnek için algoritmanın analiz edilmesi **doğru algoritma** ile **sıralanması** çok önemlidir ancak optimizasyon yapılmak isteniyorsa veri tabanı ve veri erişim yöntemlerinin de **doğru seçilmesi** gerekmektedir. Bu yüzden bu bölümde alakası olmadığı için **Dapper ve Elastic Search** kavramlarını araştırmalarını yeri geldiğinde kullanmaları için bilmeyen okurlarımıza öneririm.

Bilgisayar programcılığında özyinelemeli, yani **kendi kendini çağırabilen fonksiyon yapılarına** `recursive` fonksiyonlar denilmektedir. Çağırma işlemi (*call*) bir şart yerine gelene kadar yapılır ve şart sağlanınca `return` edilir. Genellikle döngüler içerisinde gördüğümüz `recursive` fonksiyonlar kullanılırken döngülerde olduğu gibi **başlangıç** ve **çıkış** noktaları bulunur. Aşağıda yaygın faktöriyel örneği içerisinde `recursive` fonksiyonda göreceğimiz işlemin **zaman karmaşıklığını** (*time complexity*) hesaplayalım.

Örnek: `Recursive T(n);`

```
int Faktoriyel(int n){
if (n<=1)
 return 1;
else
return (n* Faktoriyel (n-1));
```

Sonuç: `T(n) = 2n` olacaktır.

Algoritmaların analiz edilmesinin anlatıldığı bu bölümde, çeşitli algoritmaları analiz edeceğiz. Seçmiş olduğum algoritmalar **sıralama** veya **analiz konusunda yaygın olan algoritmalardan** oluşmaktadır.

Karmaşıklık Sınıfları (Complexity Classes)

Bilgisayar sistemlerinde kullanılan algoritmaların çözüme ulaşabilmelerindeki karmaşıklığı ölçmek için kullanılan matematikten kalıtım edilen teoriye **karmaşıklık teorisi** (*complexity theory*) denilmektedir. *İşletim sistemi algoritmaları, sıralama algoritmaları, arama bulma algoritmaları* gibi bilgisayar sistemlerinde kullanılan algoritmaların veya problemlerin çözümü için her birinin fonksiyon gibi düşünülebilmesi ve çözüm yolu olarak da matematikteki fonksiyonların karşılıklarını kullanarak **sınıflandırılması** ve bu karmaşıklık sınıflarının **algoritmalara uyarlanması** esasına dayanır.

Otomata dersinde, işletim sistemleri algoritmalarının sınıflandırılması dersinde veya algoritmaların analiz edilmesi dersinde karşılaşabileceğiniz karmaşıklık sınıfları; **Küçük-o**, **Büyük-O**, **Teta** ve **Büyük omega** notasyonu şeklindedir. Bu notasyonlar, aslında bir adet fonksiyon için uygulanan farklı özelliklerdir. Bu notasyonlar bir fonksiyondaki algoritma hakkında bilgiler verirler. Örneğin algoritmanın zaman ve hafızayla olan ilişkisini, yani *ne kadar yer kaplayacağını* ve *işlemi ne kadar zamanda yapabileceği* gibi durumlara **time and memory efficiency** denilmektedir. Yine bir diğer bilgi, **algoritmanın ölçeği**dir, **Scalability** olarak bilinir. Bir algoritmanın işlediği veri miktarına göre zaman ve ne kadar yere ihtiyaç duyduğu bilgisine ulaşılır. Aşağıdaki konu başlıklarında çeşitli sıralama ve arama algoritmalarına değineceğiz fakat burada bir örnek vermek gerekirse en temel arama algoritması olan **doğrusal arama algoritması** ile **karışık olarak verilen bir dizideki sayılardan bir tanesinin bulunması problemi** verilir. Bulma işleminde aranan değer, dizinin **ilk elemanı** olabilir. Böyle bir durumda aramanın en iyi ihtimalle kaç adımda bulunduğu tespit edilmiş olur ve buna `best case` denilmektedir. Tersini düşündüğümüzde ise aranan değer, dizinin **en son elemanı** olabilir. Bu durumda dizi, aranan değeri bulabilmek için **tüm elemanlara bakacaktır** ve **en kötü ihtimalle kaç adımda bulduğu da ortaya çıkar**. Buna `worst case` denilmektedir. Buradan hareketle algoritmanın yer ihtiyacına da değinmiş oluyoruz. Tüm elemanlara tek tek bakarak tüm diziyi işleten algoritmanın yer ihtiyacı da **dizinin boyutu kadardır** diyebiliriz. Dizinin eleman sayısına `n` denirse arama işlemi `n` sayı içerisinden yapılır ve `n` adet sayının hafızada tutulması gerekir.

Algoritmaların hafıza karmaşıklığı, hafızada tutmaları gereken değer olan `n`'dir. Çalışma hızının en iyi ihtimalle **kaç adımda bulabileceği** (`best case`) ve **en kötü durumda kaç adımda bulabileceğinden** (`worst case`) bahsetmiştik. *Peki, ortalama bir değer elde etmek istersek nasıl bir yorum yapabiliriz?* Bu durumda ise ortalama kaç adımda bulduğu tespit edilir, algoritma defalarca çalıştırılır ve tamamen istatiksel olarak ortalama kaç elemana bakıldığı tespit edilir. Buna da `avarage case` denilmektedir.

Çalışma hızını veren durum analizleri **karmaşıklık sınıflarını** veren analizlerdir.

O – **En kötü durum analizini**

Θ – **Ortalama durum analizini**

Ω – **En iyi durum analizini gösterir**

Birleştirme Sıralaması Algoritması (Merge Sort)

Sıralama algoritmalarından bir tanesidir. Oldukça yaygın olan bu algoritmanın *yapısını* ve *çalışma şeklini* bilgisayar mühendisliği ders müfredatlarındaki veri yapıları dersinde de görebilirsiniz. Birleştirme sıralaması algoritmasının (`merge sort`) veriyi sıralama mantığı, dizi içerisindeki elemanları **ikişer elemana indirgeyene kadar sürekli olarak ikiye bölünmesi esasına göre** çalışır. Ortaya çıkan parça elemanlar da kendi içlerinde sıralanır ve en son birleştirilir. Böylelikle nihai dizimiz `merge sort` ile sıralanmış bir dizi hâline gelecektir. Bu işlem literatürde **böl** ve **tamamla** yaklaşımı olarak da bilinir. Bir örnek ile birleştirme sıralama algoritmasını analiz edelim.

Örnek: 2,1,7,4,3,4,0,7 verisini birleştirme algoritması kullanarak sıralayınız.

Çözüm: pseudo kod ve algoritma oluşturma şekli ile çözelim.

1. Başla
2. Diziyi ikiye böl:

   2,1,7,4 ve 3,4,0,7
3. Yukarıdaki adımda bölünen dizileri tekrar ikiye böl:

   2,1: 7,4; 3,4:0,7
4. Üçüncü adımda yapılan bölme işlemi ile dizi 2 elemana ulaştığı için daha fazla bölme işlemi yapılmaz.
5. Her bir parçayı kendi içlerinde sırala

   1,2 : 4,7 ; 3,4 : 0,7

6. Bölünen bu parçaları kendi içlerinde birleştir yani bölünme gruplarına göre birleştir.
   1,2,4,7 ve 0,3,4,7
7. İki ayrı parça şeklinde olduğu için tekrar tek parça olacak şekilde sıralayarak birleştir.
   0,1,2,3,4,4,7,7
8. Birleştirilmiş tek parça bir dizi elde edildiği için sıralamayı durdur.

Elde edilen sonuç dizinin sıralanmış hâlidir.

`Merge Sort` algoritmasının pekiştirilmesi için aşağıdaki **C#.net** dili ile hazırlamış olduğum örnek kodlamayı inceleyebilirsiniz.

```
public void QuickSort(int[] intdizi, int left, int right,  int dizi)
        {
            int count = intdizi.Length;
            int i = left, j = right;
            int tmp;
            int pivot = intdizi[(left + right) / 2];
            while (i <= j)
            {
                while (intdizi[i] < pivot)
                    i++;
                while (intdizi[j] > pivot)
                    j--;
                if (i <= j)
                {
                    tmp = intdizi[i];
                    intdizi[i] = intdizi[j];
                    intdizi[j] = tmp;
                    i++;
                    j--;
                    dizi += 1;
                }
           }
            if (left < j)
                QuickSort(intdizi, left, j, dizi);
            if (i < right)
                QuickSort(intdizi, i, right, dizi);
        }
```

Aynı işlemi **`Recursive` Metot** kullanarak aşağıdaki örnek kod ile tekrar inceleyebiliriz.

```
public void MergeSort_Recursive(int [] numbers, int left,int right)
        {
          int mid;
          if (right > left)
          {
            mid = (right + left) / 2;
            MergeSort_Recursive(numbers, left, mid);
            MergeSort_Recursive(numbers, (mid + 1), right);
            DoMerge(numbers, left, (mid+1), right);
          }
        }
```

Taban Sıralaması Algoritması (Radix Sort)

Kök sıralama olarak da bilinen **`Radix Sort`** algoritmasının en güzel Türkçe karşılığı **Taban Sıralaması**dır diyebiliriz. Çünkü **`radix sort`** ile sıralama işlemi yapılırken verilerin taban değerleri, yani **sayı değerlerine göre kendi içlerinde bir sıralama yapılır**. Yani sıralanacak olan değerler sayı değerlerine göre (*digit*) **en küçükten en büyüğe olacak şekilde sıralanır**. Bu sıralamadan sonra elde edilen sayı sıralanmış olacaktır. Sıralama algoritmaları içerisinde en yaygın ve belki en güzel algoritmalardan bir tanesi olan sıralama algoritması konusunu *veri yapıları dersi* veya *algoritmaların analizi* dersinde görebilirsiniz.

Örnek: **`450, 90, 32, 530, 162`** sayılarını **`Radix Sort`** algoritması ile sıralayınız.

1. Başla
2. Sayıları ilk önce birler basamağına göre küçükten büyüğe sırala: 450, 90, 530, 32, 162
3. Şimdi sayıları onlar basamağına göre sırala: 32, 530, 450, 162, 90
4. Değerler arasındaki en büyük basamak değeri yüzler basamağına sahip sayılar olduğu için son olarak yüzler basamağına göre sayıları sırala: 32, 90, 162, 450, 530

`Radix Sort` yani **Taban Sıralaması** algoritması ile sayılar örnekte olduğu gibi çok kolay bir şekilde sıralanır. **`Radix Sort`** algoritmasının pekiştirilmesi için aşağıdaki **C#.net** dili ile hazırlamış olduğum **örnek** kodlamayı inceleyebilirsiniz.

```
public void RadixSort(int[] a)
        {
            int[] t = new int[a.Length];
            int r = 4;
            int b = 32;
            int[] count = new int[1 << r];
            int[] pref = new int[1 << r];
            int groups = (int)Math.Ceiling((double)b / (double)r);
            int mask = (1 << r) - 1;
            for (int c = 0, shift = 0; c < groups; c++, shift += r)
            {
               for (int j = 0; j < count.Length; j++)
                    count[j] = 0;
                for (int i = 0; i < a.Length; i++)
                    count[(a[i] >> shift) & mask]++;
```

```
        pref[0] = 0;
        for (int i = 1; i < count.Length; i++)
            pref[i] = pref[i - 1] + count[i - 1];
        for (int i = 0; i < a.Length; i++)
            t[pref[(a[i] >> shift) & mask]++] = a[i];
        t.CopyTo(a, 0);
    }       }
```

Seçerek Sıralama Algoritması (Selection Sort)

Sıralama algoritmalarından analiz edeceğimiz bir diğer algoritma ise **`Selection Sort`** olarak bilinen **seçim yaparak sıralama algoritmasıdır**. Tür olarak verinin bellekte sıralı bir şekilde tutulabilmesi için geliştirilen sıralama algoritmasıdır. Dizi içerisindeki her bir adımda en küçük sayının nerede olduğu tespit edilir ve bu sayı dizinin başındaki sayı ile sürekli olarak yer değiştirilir böylelikle küçük sayılar **başa tutturulmuş olur** ve **sıralama bu şekilde tamamlanır**.

Örnek: **`9, 5, 1, 3, 6, 2, 8, 4`** sayılarını **`Selection Sort`** algoritması ile sıralayınız.

Çözüm:

1. Başla ve i=0 döngüsü kur.
2. Dizi içerisinde yer alan elemanlardan i'nin sırasından sonraki en küçük sayının yerini tespit et. Yani örnek dizideki en küçük sayı 1'dır ve 2. sıradır. Dizilerin değerinin sıfırdan başladığını hatırlayalım.
3. Dizinin i. inci sırasındaki sayıyı 2.adımdaki en küçük ile değiştir. 1,5,9,3,6,2,8,4
4. Döngüdeki i'yi 1 arttır ve tekrar 2. adıma git.
5. 4. adımdan sonra i=1 olacaktır ve sonrasında gelen ilk sayı 1 artarak kalan sayılar 5,9,3,6,2,8,4 arasından en küçük sayının bulunması sağlanmış olacaktır. Bu sayı ise 2'dir ve sırası 4'tür. Sayılar tekrar sıralanır ve dizi 1,2,9,3,6,5,8,4 halini alır.
6. Döngüdeki i değeri 2 olmuştur ve bu sayıdan itibaren en küçük sayı 9,3,6,5,8,4 sayılarının arasında aranır. İşlemlere aynı şekilde devam edildiğinde bulunan sayı 3'tür. Sırası ise 3'tür. Bu işlemden sonraki sıralanmış hali ise 1,2,3,9,6,5,8,4 olur.
7. İşlem yukarıdaki maddelerde olduğu gibi devam eder.

   1,2,3,4,6,5,8,9

   1,2,3,4,5,6,8,9 sonuç bu şekilde sıralanmış olur.

Seçerek sıralama algoritmasının `time complexity`, yani zaman karmaşıklığına bakacak olursak her adımda `n` adet sayı işlendiği için işlemler `n` kez tekrar edecektir. Bu yüzden **zaman karmaşıklığı** `O(n^2)` olur.

Hızlı Sıralama Algoritması (Quick Sort)

Sıralama algoritmalarından bir tanesi de `Quick Sort` olarak bilinen **hızlı sıralama** algoritmasıdır. Hızlı bir işlem gerçekleştirir çünkü diğer sıralama algoritmalarından farklı olarak eldeki dizinin elemanlarından **orta noktada bulunan bir sayıyı seçer** ve **orta noktasının önündeki ve arkasındaki değerleri bu sayıdan küçük veya büyüktür şeklinde sıralayarak sınıflandırır**. *Linear denklemler* dersinden hatırlayacağınız üzere matrislerde sıralama işlemi yaparken bir `pivot` değerimiz bulunurdu ve işlemleri bu `pivot` yani **eksen değerine** göre gerçekleştirirdik. Hızlı sıralama algoritmasında da orta noktadaki sayı `pivot` olarak belirlenmiş olur ve sıraya konan diğer sayılar `pivot` ekseninde sıralanmış olurlar.

Örnek: `1, 7, 0, 6, 5, 3, 2, 4` dizisini **hızlı sıralama algoritması** ile sıralayınız.

Çözüm:

1. Dizinin ortasında bulunan eleman tespit edilir. Örneğimize göre ortada bulunan eleman 6'dır.
2. Seçilen elemana göre sayıyı sınıflandır. 1,7,0,5,3,2,4 (6)
3. Elde edilen değere göre tekrar aynı sınıflandırmayı uygula bu sefer ortadaki sayı 5 olacaktır.
4. 0,1,2 (5) 4,7
5. Bu şekilde iki grup elde edildikten sonra da bu gruplar tekrar hızlı sıralama algoritmasına tabii edilir. 0,1,2,3 ve 4,7 sayıları ayrı ayrı sıralanmalıdır.
6. 0,1,2,3 sayılarının orta değeri 1'dir. 0 (1) 2,3 elde edilir. Diğer dizi sınıflandırıldığında ise 4,7 gelir.
7. 0 sıralanırsa 0 ve 4,7 sıralanırsa 4,7 gelir.
8. Birleştirme adımına geçilir ve bu adımda sayı 0,1,2,3,4,5,6,7 olarak sıralanmış olur.

Quick Sort algoritmasının pekiştirilebilmesi için aşağıdaki **C#.net** dili ile hazırlamış olduğum örnek kodlamayı inceleyebilirsiniz.

```
public void QuickSort(int[] intdizi, int left, int right,  int dizi)
        {
            int count = intdizi.Length;
            int i = left, j = right;
            int tmp;
            int pivot = intdizi[(left + right) / 2];
            while (i <= j)
            {
                while (intdizi[i] < pivot)
                    i++;
                while (intdizi[j] > pivot)
                    j--;
                if (i <= j)
                {
                    tmp = intdizi[i];
                    intdizi[i] = intdizi[j];
                    intdizi[j] = tmp;
                    i++;
                    j--;
                    dizi += 1;
                }
            }
            if (left < j)
             QuickSort(intdizi, left, j, dizi);
            if (i < right)
                QuickSort(intdizi, i, right, dizi);        }
```

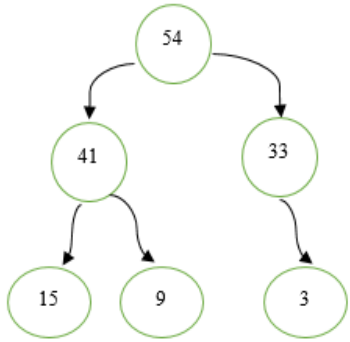
Yığınlama Sıralaması Algoritması (Heap Sort)

`Heap Sort`, yığınlama algoritması bellekte bir **yığın ağacı** oluşturur ve ağaç üzerindeki değerlerin **en üstündeki sayıyı işleme alarak** sıralar. Oldukça basit kurgulanan bu algoritmayı bir örnek ile inceleyelim.

Örnek: `54, 3, 15, 41, 9, 33` dizisini `heap sort` algoritması kullanarak sıralayınız.

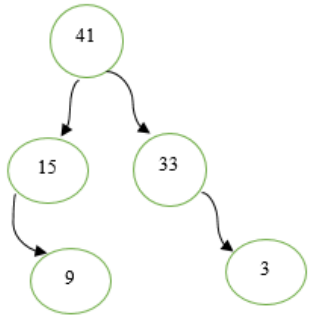
Çözüm:

1. Başla
2. Büyük sayılar en üste küçük sayılar ise onların altında olacak şekilde sayılardan bir yığın ağacı oluşturulur.



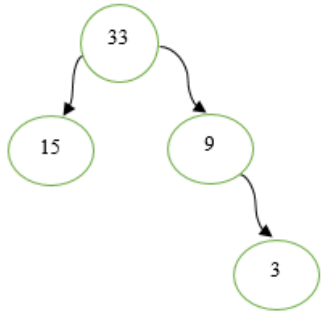
Şekil 12-1: Diziye Göre Yığın Ağacının Oluşturulması

3. Tepe değeri 54 olduğundan sonuç değeri {54} olarak alınır ve ağaç yapısından silinir. Daha sonra elde kalan değerler ile aşağıdaki gibi yeni bir ağaç oluşturulur.



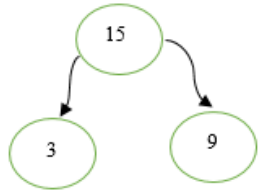
Şekil 12-2: Yeni Tepe Değeri 41 Olduktan Sonra Yığın Ağacı

4. Yeni ağaç dizisindeki sonuç değeri {41} de ağaçtan silinir ve elde kalan sayılarla ağaç yeniden çizilir. Mevcut sonuç dizisi {54,41} olur.



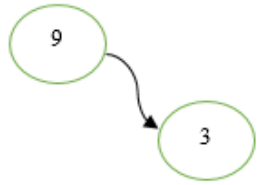
Şekil 12-3: Yeni Tepe Değeri 33 Olduktan Sonra Yığın Ağacı

5. Yeni oluşan bu ağacın en üstündeki değer ise 33'tür ve bu değer de ağaçtan silinince sonuç dizisi {54,41,33} olur. Kalan sayılar ile dizi tekrar yığınlama sıralaması algoritmasına tabi tutulur.



Şekil 12-4: Yeni Tepe Değeri 15 Olduktan Sonra Yığın Ağacı

6. Bu ağacın tepe değeri ise 15'dir ve bu sayıda silindikten sonra sonuç dizisi {54,41,33,15} olur. Kalan sayılar ile yığınlama algoritması bir kez daha çalıştırılır.



Şekil 12-5: Yeni Tepe Değeri 9 olduktan Sonra Yığın Ağacı

7. En son olarak sonuç dizisi {54,41,33,15,9} olarak bulunur. Aynı işlem kalan iki değer için de gerçekleştirilince sonuç değeri {54,41,33,15,9,3} olarak bulunur ve bu sayılar örnekte verilen sayıların büyükten küçüğe sıralanmış hâlini almıştır.
8. Dur.

İkili Arama Algoritması (Binary Search)

Binary Search, yani **ikili arama algoritması** isminden de anlaşılacağı üzere sıralama değil **arama algoritmasıdır**. Bu algoritmayla *veri yapıları* dersinde karşılaşmış veya karşılaşacak olmanız muhtemeldir. Binary Search algoritması verileri adımların her birinde **iki parçaya bölerek arama yapılması** esasıyla çalışır. Bir örnek ile arama algoritmasının nasıl çalıştığını inceleyelim.

Örnek: `8, 3, 15, 20, 75, 4, 40, 48, 56, 121.` Yandaki 10 elemanlı diziden `15` sayısını arayıp bulunuz.

Çözüm:

1. Başla
2. Dizide arama işlemi yapılırken dizi elemanlarının ortasındaki değere göre bakılır. Örnek dizideki ortadaki değer yani 5. değer 4 sayısıdır.

8	3	15	20	75	**4**	40	48	56	121

3. Ortadaki değere bakılır ve 4 aranılan değerden küçük olduğu için dizinin kalan elemanlarından solundaki elemanların ortasındaki değer olan ikinci elemana bakılır.

8	3	**15**	20	75	4	40	48	56	121

4. Daha ilk işlemde aradığımız sayıyı buluyor ve ikili arama algoritmasını örneklemiş oluyoruz. Benzer şekilde eğer sayıyı sol tarafta bulamasaydık bu sefer de sağındaki değerlerin ikili ikili ortasındaki değerlere bakarak aranılan sayıyı tespit edebiliriz. Eğer aranan sayı dizi elemanları içerisinde yoksa büyük ve küçük değerlerin hangisinin arasında kalıyor ise o değeri aranılan değer olarak kabul ediyoruz.
5. Dur.

Doğrusal Arama Algoritması (Linear Search)

Sequential Search (*Sıralı arama*) ismi de verilen bu arama, tam anlamıya bir veriyi **arananlara teker teker bakarak aramaktır**.

Yani aşağıdaki sayılar arasında `15` sayısını aramak için:

4 6 12 8 5 15 25 34

Teker teker bütün sayılara bakılır. Örneğin *baştan başlanarak 4 aranan sayı mı?* **değil** *6 aranan sayı mı* **değil** 1*2 aranan sayı mı* **değil** şeklinde bütün sayılar alınır ve aranan sayı bulunana kadar devam eder.

Örnekten de anlaşılacağı üzere `n` sayılık bir kümede arama işlemi yapılırken aranan sayının bu kümede hiç bulunmaması durumu ancak `n` sayının **tamamına bakılarak olabilir**. Dolayısıyla algoritmanın karmaşıklığı **(complexity, growth rate, big-oh) O(n)** olmaktadır.

Açgözlü Yaklaşımı Algoritması (Greedy Algorithm)

Açgözlü yaklaşımı olarak bilinen **Greedy Algoritması**, *sonuca en yakın olan seçimin yapılabilmesi için mümkün olacak işlemi gerçekleştirir*. Daha açık bir ifade ile bir seçim işlemi söz konusu olduğunda bizi sonuca en çok yaklaştıracak olan doğru seçimin yapılmasında **önerilerde bulunan** bir algoritmadır. Burada dikkat edilmesi gereken şey, **en yakın seçimin doğru seçim olup olmayacağıdır**. Greedy algoritması anlatılırken örnekler genelde *bozuk para üzeri* ile ilgilidir. Biz de bu bölümde gerekli açıklamaları yaptıktan sonra **C#.net** ile bir örnek yaparak konuyu pekiştirelim. Diyelim ki bir satıcının satmakta olduğu ürün bedeli 12 TL olsun müşteri de satıcıya 15 TL ile ödeme yapsın. Bu durumda satıcı müşterisine 3 TL bozukluk ile para üzerini verecektir. Şimdi, örnek bir proje geliştirelim ve *greedy algoritmasını pekiştirelim*. Örnek projemizde form alanında kullanıcının para miktarını girmesi isteniyor ve para miktarına göre yapacağı bir harcama da olası en yakın bozukluk ihtiyacını kullanıcıya belirtiyor. Örnekte 6 adet para miktarı **sabit kabul ediliyor**. Bu birimler 1'lik, 5'lik, 10'luk, 25'lik, 100'lük ve 200'lük banknot olarak da düşünülebilir.

Kullanıcıdan alınan değer **para** isimli tam sayı bir değişkende tutulmaktadır. **MinimumCoinHesapla** isminde oluşturulan metot ise **hesaplama işlemini** gerçekleştirmektedir.

Hesaplama işlemi için gereken metot aşağıdaki gibidir:

```
public void MinimumCoinHesapla(int c)
  {
     int[] coin = new int[] { 200, 100, 25, 10, 5, 1 };
     int CoinNum, count = 0;
     for (int i = 0; i < coin.Length; i++)
       {
          if (coin[i] <= c)
             {
               CoinNum = c / coin[i];
               lbl_Sonuc.Text += " " + CoinNum + "x " + coin[i] + "
lik\n";
               c -= CoinNum * coin[i];
               count += CoinNum;

                if (coin[i] == 1)
                    {
                        txt_Coin1.Text = CoinNum.ToString();

                    }
                    if (coin[i] == 5)
                    {
                        txt_Coin2.Text = CoinNum.ToString();

                    }
                    if (coin[i] == 10)
                    {
                        txt_Coin3.Text = CoinNum.ToString();

                    }
                    if (coin[i] == 25)
                    {
                        txt_Coin4.Text = CoinNum.ToString();

                    }
```

```
                    if (coin[i] == 100)
                    {
                        txt_Coin5.Text = CoinNum.ToString();

                    }
                    if (coin[i] == 200)
                    {
                        txt_Coin6.Text = CoinNum.ToString();

                    }
                }
            }
            lbl_Sonuc.Text += "Toplam : " + count + " Bozukluk
Gerekli";

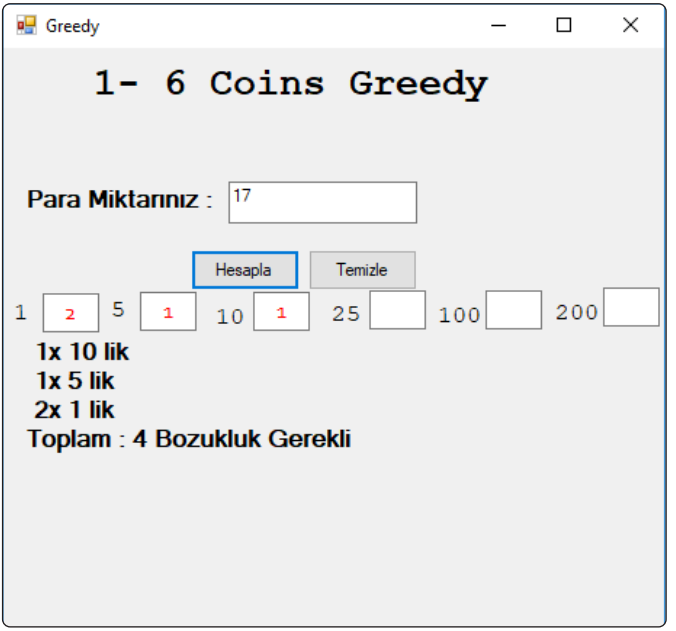
        }
```

Hesapla butonu altında işletilen kodlar aşağıdaki gibidir:

```
if (txt_ParaMiktari.Text != "")
            {
                lbl_Sonuc.Text = "";
                int para = Convert.ToInt32(txt_ParaMiktari.Text);
                MinimumCoinHesapla(para);

            }
            else
                MessageBox.Show("Para Miktarı Giriniz.", "UYARI!",
MessageBoxButtons.OK, MessageBoxIcon.Error);
```

C#.Net ile gerçekleştirdiğimiz örnek projenin ekran görüntüsü **Şekil 12-6**'da gösterildiği gibidir.



Şekil 12-6: Greedy Algoritması Proje Ekran Görüntüsü

OTOMATA TEORİSİ (AUTOMATA THEORY)

13

Bilgisayar mühendisliği, bilgisayar bilimleri lisans veya yüksek lisans müfredatlarında yer alan oldukça önemli konulardan bir tanesidir. Otomat teorisi **özdevinirler** veya **özdevinim kuramı** olarak da bilinir. Özdevinimler, *soyut matematiksel sistemleri* ve *bu sistemleri kullanarak hesaplama problemlerinin çözümlerini araştırırlar*. Bahsi geçen soyut sistemlere ise **otomat** denilmektedir. Günümüzde bir işi kendi kendine yapabilen çay, kahve vb. otomatları görmekteyiz. Kelime anlamı da tam olarak buradan geliyor aslında. Otomat, Yunanca bir kelimeden gelmekte ve **kendi kendine hareket eden** anlamına gelmektedir. Özdevinirler dil kavramları ile oldukça yakın bir araştırma alanı olduğundan derleyici tasarımında, hesaplama teorilerinde ve çözümlemede çok önemli rol oynarlar.

Otomatlar, durum veya konfigürasyonda ilerleyerek bir girdi üzerinde hesaplamalar yapabilen **soyut makine**lerdir. Hesaplamaların yapılan her durumda bir geçiş fonksiyonu bulunur. Bu mevcut fonksiyon konfigürasyonunun temelinde bir sonraki fonksiyonda yapılacak **konfigürasyon** belirlenir. Bu işlem bu şekilde devam ederken sonuç olarak kabul edilebilecek bir yapılandırma söz konusu olduğunda bu aşamadaki giriş kabul edilir kendi kendine yapılan bu işlem tamamlanır. Bilinen en güçlü otomat **Turing makinesidir** (*Turing Machine*).

Otomata teorisinin temel amacı bilgisayar bilimcilerinin, ayrık sistemlerin dinamik davranışlarını tanımlayıp analiz edebilecekleri yöntemler geliştirmektir. Bir otomata makinesi **girişler**, **çıktılar**, **geçiş fonksiyonları** ve **durumlardan** oluşur. Otomata ailelerini incelediğimizde ise temel olarak **dört** tane otomata ailesi bulunur.

Bunlar; **sonlu durum makineleri** (*Finite Automata*), **aşağı itme otomatları** (*Pushdown Automata*), **doğrusal sınırlı otomatlar** (*Linear Limited Automata*) ve **Turing makineleri**dir (*Turing Machine*). Automata konusu çok derin ve uygulama ile anlaşılması mümkün bir konu olduğu için kitabımızın bu bölümünü sadece temel kavramları ve tanımları vererek tamamlayacağım. Gramer yapısının nasıl çalıştığı, dil bilgisinin nasıl olduğu, kodların ne şekilde okunabildiği ve yorumlandığı, bağımsız içerikler, pompalama ön savı ve daha birçok konu hakkında giriş seviyesinde bilgi vereceğimiz bu bölümün sonunda konuyu anlaşır hâle getirsin diye **Lex** ve **Yacc** ile örnek bir **compiler tasarımı** yer almaktadır. Automata konusu hakkında detaylı bilgileri özümsemek için öncelikle **Programlama Dilleri** (*Programming Languages*) konularını iyi anlamak gerekmektedir.

İçerikten Bağımsız (CFG-CFL)

Bilgisayar programlama dünyasındaki **dil tasarımı** ve mimarisinde kullanılan **gramer tipleri** vardır. Bu gramerler, dilin kurallarını ifade ettiğinden dil bilgisini vermektedirler. İçerikten **bağımsız dil bilgisi** (*CFG*) olarak bilinir. S ile ifade edilen devamlı sembolünün a ile gösterilen sonluya dönüşümü **S→a** ile gösterilir. Yani bir dildeki kuralları ifade ederken **büyük harf ile gösterilen semboller devamlıları**, **küçük harflerle gösterilenler ise sonluları** gösterir. İçerik bağımsız diller çoğu programlama dilinin temelinde yer almaktadır ve dört temel özellik barındırmaktadırlar. Bunlar, *sonlular, bağlantılar, başlangıç* ve *devamlılar* şeklindedir. Bilgisayar programlama dünyasındaki dillerin tasarımı ve mimarilerinde kullanılan **içerik**, bağımsız gramer oluşturma şeklidir. İçerikten bağımsız dil veya orijinal ismiyle **context free language** (*CFL*) olarak bilinir.

Backus Normal Form (BNF)

Bilgisayar programlama dünyasında ve mimaride bir **dili tanımlamak** için ve **dilin gramerini gösterebilmek** için kullanılan gösterim şeklidir. BNF, aslında iki farklı açılıma sahiptir, bunlardan bir tanesi **Backus Naur Form** iken diğeri **Backus Normal Form**'dur. İlk başta ALGOL'un söz diziminde kullanılmak amacıyla Backus Normal Form ortaya çıkmışken daha sonra Peter Naur tarafından yapılan birtakım değişiklik sonrası **Backus Naue Form** olarak da anılmaya başlanmıştır. Gösterimi örnek olarak: **<etiket> ::= __ifade__** şeklindedir. Etiket her zaman için **::=** operatörlerinin **solunda** yer almak durumundadır. Sağ tarafında ise pek çok etiket tanımlanabilir.

Sonlu Durum (Finite State)

Sonlu durumlu makine kavramını geliştirenler arasında yalnızca bilgisayar bilimciler değil, matematikçiler, mühendisler, psikologlar ve hatta biyologlardan oluşan bir ekip vardı. Bu ekibin amacı **bilgisayarda yapılan işlemler** veya **beyinde yapılan düşünce sürecini modellemekti**. Dönemin (*1950'li yıllar*) nörobilimcilerinin yayınladıkları "**Sinir Aktivitesinde Bir Mantıksal Hesaplama**" konu başlıklı makaleleri ile sinir ağları ile ilgili çalışmalara otomatlar, hesaplama ve sibernetik teorisi gibi önemli teoriler kattılar. Bu teorilerden hareketle ilerleyen yıllarda iki bilgisayar bilimcisi **Mealy** ve **Moore** teorilerini makineler üzerinde yaygınlaştırdılar. Sonlu durumlu makineler Mealy makinesi ve Moore makinesi olarak Mealy makinesi çıktılarını **mevcut durumlardan** ve **girdiler** üzerinden hesaplarken Moore makinesi çıkışını **yalnızca mevcut duruma göre** dayandırarak hesaplamaktadır. Sonlu durum makinelerinde durum geçiş fonksiyonu **I → Z** olarak tanımlıdır.

Sonlu durum makineleri oldukça ideal hesaplama modelleridir. Az miktarda bellek için **belleği yormadan çalışırlar**. Sonlu durum makinelerinin belleği ne şekilde çalıştırdığını ve ideal hesaplama modeli olduğunu anlayabilmek için bu konuda yaygınca verilen **yürüyen merdiven** örneğini verebiliriz. Bir yürüyen merdiven harekete iniş veya çıkış yönlü olarak başlayacaktır. Eğer iniş yönünden bir insan binerse iniş yönünde bu işlem bittikten sonra da çıkış yönünden bir insan bindiğinde bu sefer de çıkış yönünde hareket edebilecektir. (*Tek bir merdiven*) Önceki iniş veya çıkış yönleri hafızada tutulmayıp sadece o anki durumu çalıştırdığından **sonlu durum makinesi gibi** çalışmaktadır diyebiliriz.

Sonlu durum makinelerinde;

Durumlar: Gelen isteklerin geçmişini yansıtacak sınırlı durumlardır.

Girişler: Sonlu giriş setleri, merdivenin çıkabileceği veya inebileceği yüksekliğe bağlıdır. Yüksekliği metre cinsinden düşündüğümüzde **I** setini kullanabiliriz.

Çıkışlar: Merdivenin kullanım ihtiyacına göre yukarı ya da aşağıya inme hareketine bağlı olarak sonlu çıkış setidir.

Bu durumda bir sonlu durum makinesi aşağıdaki şekilde tanımlanır.

Q = sonlu durum kümesi

I = sonlu giriş sembolü kümesi

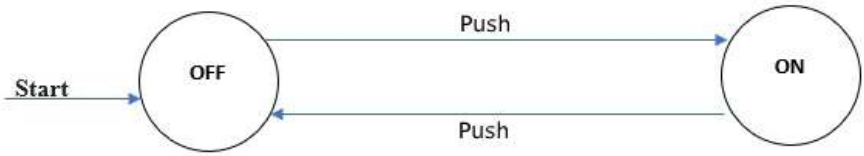
Z = sonlu çıkış sembol seti

∂ = Durum geçiş fonksiyonu olarak adlandırılan I x Q'nun Q ile eşlenmesi, yani I x Q→Q

W = I x Q'nun W ile Z'ye eşleştirilmesi yani çıkış işlevi

A = F' nin Q'nun bir alt kümesi olduğunun kabul durumları kümesi

Matematiksel ifadeleri dikkatle incelediğimizde sonlu durum makinelerinin sonlu sayıda durum içerdiği söylenebilir. Her durum **sınırlı sayıda girdi kabul eder** ve **her durum geçiş haritalama işlevinde gösterilen makinenin sürekli giriş eylemlerini tanımlayan kuralları içerir**. Bir giriş makinenin durum değiştirmesine neden olacağından her giriş sembolü için her durumdan tam olarak bir geçiş bulunur.



Şekil 13-1: Sonlu Durum Makinesi

Turing

Hesaplamada kullanılan en basit otomatların **sonlu otomatlar** olduğunu artık biliyoruz. Fakat yapısındaki yalınlık sebebiyle sonlu otomatlar **çok basit işlemleri hesaplayabilmektedir**. Buna nazaran daha büyük bir hesaplama yapabilmek için daha güçlü otomatlara ihtiyaç duyulmuştur. Turing makinesi **daha büyük** hesaplamaları yapabilmektedir. Turing makineleri daha yüksek seviyeli ve sonsuz otomatlar ile çalışmaktadır. Bilgisayar biliminin ünlü araştırmacısı Alan Turing, **Turing makinesi** adını verdiği hesaplama modeli ile **ilk sonsuz hesaplama otomatını tasarlamıştır**. Bir Turing makinesi **[Q, Σ, Γ, δ, q0, B, F]** seti ile tanımlanır.

Q = bir durum q0 başlangıç durumu olan sonlu durum kümesi

Σ = B içermeyen sub bir altküme, giriş sembollerinin setidir

Γ = izin verilen bant sembollerinin sonlu seti

δ = sonraki hareket işlevi, Q x Γ ile Q x Γ x {L, R} arasındaki bir eşleme işlevi, burada L ve R sırasıyla sol ve sağ yönleri gösterir

q0 = başlangıç durumu olarak Q

B = boşluk olarak Γ sembolü

F ⊆ Q final durumları seti

Chomsky (CNF)

Programlama dilleri hakkında yapılan çalışmalarda **düzenli dilleri belirtmek için** kullanılan yapılardan bir tanesidir. İçerikten bağımsız diller diye ifade edilen şey aslında **Context Free Grammer (CFG)**, yani içerikten bağımsız dillerin de temelini oluşturmaktadır. CFG yapısında da dört temel özellik bulunmaktadır. Bunlar; **sonlular**, **devamlılar**, **geçiş kuralları** *ve* **başlangıç** şeklindedir. Sonlular **{a,b}**, devamlılar **{S,A,B}**, geçiş kuralları sembol ile gösterilen **S,A,B** açılımları ve başlangıç ise **Start** kelimesinden gelen **S** ile gösterilmektedir. Bu açılım düzenli ifadeler cinsinden gösterileceğinde **a^nb** şeklinde gösterilir. N yerine sonsuz değer alabiliyorken sonunda kesinlikle **b sonlusu** ile bitmek zorundadır.

Dil tasarımında çoklukla kullanılan yapılardan bir tanesi olan içerikten bağımsız dil bilgisine **Context Free Grammer (CFG)** denildiğini öğrendik. Bu konularda yapılmış çalışmalardan bir tanesi **Chomsky Normal Form (CNF) kurallarıdır**. Bir dil CNF kurallarına uyuyor ise **CFG** kurallarına da uyuyor demektir.

Pompalama (Pumping Lemma)

Bilgisayar programlama dünyasında mimariler içerisinde tasarlanan bir dil içerikten bağımsız dil grubuna veya herhangi başka bir gruba dâhil olması durumunda bu dil istenildiği kadar pompalanırsa pompalansın yine kendi **dil grubuna dâhil olacaktır**. Burada bahsedilen pompalama işlevi, **pompalama ön savı** (*pumping lemma*), **gramerin arttırılması** anlamına gelmektedir. Pompalama ön savı bir **ispat yöntemidir**. Bir dil pompalanarak, bahsi geçen dil grubunda olup olmadığı ile kesin sonuca varılacağı durumlarda kullanılır.

Benzer bir şekilde bir dilin düzenli ifade grubunda olup olmadığını tespit etmek için de **pompalama ön savı** kullanılmaktadır. Dil, gramer bakımından incelendiğinde terimler parçalara ayrıldığında her bir parçanın **tekrarlanabilir olması** gerekmektedir. Düzenli ifade grubunda olup olmadığı tespit edilebildiği gibi pompalama ön savında bir dilin içerik bağımsız olmadığı ispatlanırken içerik bağımsız olabileceği hakkında bir yorum yapılması da **pek mümkün olmayacaktır**.

YACC

YACC, dil tasarımında dillerin ortaya çıkarılması, kurallarının tanımlanması gibi **derleyici teorisi uygulamalarında** kullanılan bir programdır. YACC ile dildeki grameri ve söz dizimini (*sytanx*) belirlenebilir. Tasarlanan dildeki terimlerin neyi ifade edeceği, parantez kapsamlarının (*scope*) ne şekilde işletileceği gibi istenilen şekilde **dil tasarlamaya** olanak sağlar. Terimlerin neyi ifade edeceği de bu tasarım esnasında tanımlanacağından `IF` terimini kontrol şartı değil de döngü olarak, `FOR` terimini de döngü olarak değil de **değişken** olarak tanıtırsak ortaya çıkacak dil ile geliştirme yapılırken çalışma mantığı bu şekilde olacaktır. Yine dil tasarımı esnasında kendi dilimize ait `reserved terms` yani rezerve edilmiş terimleri de belirleyebiliriz. (`Not, Int, String`, … vb.)

YACC dokümanında `%token` ve `%type` gibi tanım değerleri yer almaktadır. Örnek projemizde nasıl kullanıldıklarını detaylı bir şekilde görme fırsatımız olacaktır.

LEX

LEX, programlama dili tasarımında kullanılan kelimelerin analizini yapan programdır. **Lexical analysis** kelimelerinin kısaltmasında meydana geldiği için kısaca **LEX** denilmektedir. **YACC ile tanımlanan özellikleri okuyup yorumlayan program da diyebiliriz**.

Bir LEX dokümanında üç bölüm bulunur. İlk bölüm **değişken tanımları**, **fonksiyonlar**, **döngüler** gibi kütüphaneler vasıtasıyla tanımlanacak yapıları içerirken ikinci bölüm **düzenli deyimleri**, **ihtimallere** göre tanımlanacak bilgileri barındırır. Son bölümde ise **fonksiyonların içerikleri** yer alacaktır.

YACC-LEX Uygulaması

Lex ve Yacc tanımlarını yaptıktan sonra bu bölümde örnek bir **compiler** (*derleyici*) yapalım. Derleyicimizde programlama dilindeki gramer ve söz dizimini **Yacc dosyasında**, burada yapılan tanımlamaları, yorumlama işlemini ise **Lex dosyasında** gerçekleştireceğiz. Her iki dosya da ubuntu üzerinde **nano** veya başka bir **text editör** ile derlenmelidir. Derlenen compiler kendine has özelliklerine göre **döngü**, **değişken tanımlama** veya **dört işlem** gibi işlemler gerçekleştirecek sonuç yine bizim gramerimize göre hesaplanmış olacaktır.

Yacc.y uzantılı dosyamızın içeriği aşağıdaki gibidir. Burada dilimiz ile ilgili **tüm gramer yapısını tanımlıyoruz**.

```
%{
#include <stdio.h>
#define RESET   "\x1b[0m"
int knt;
int i;
int jnt;
int snt;
int vnt;
int zaq;
%}
%start start
%token atama carpma_operator toplama_operator cikarma_operator
bolme_operator int_value identifier int_values program end begin
WHILE
```

```
%token esitDegil buyuk kucuk kucukEsit buyukEsit IF THEN dot Newline
whitespace END_OF_FILE
%%
start :
{
extern FILE *yyin;
yyin = fopen("run.txt", "r");
printf("\n");}
{printf("\n\nParse Tree:\n\n" RESET);}
input;
input:
{printf("<input>(");
knt=knt+1;}
         |
        input stmt_list '\n'
     |
    input main '\n'
        ;
main:
program identifier {printf("<Main>(program)<identifier>");jnt=1;
knt=knt+1;} stmt_list {printf("END"); zaq=1;}end;
stmt_list:
|stmt_list stmt
;
stmt:
{printf("<stmt>("); knt=knt+1;}assign
|
{printf("<stmt>("); knt=knt+1;}if_statement
|
{printf("<stmt>("); knt=knt+1;}while_statement
;
while_statement:
{printf("<while_statement>");}
WHILE cond while_block
;
while_block:
begin{printf("begin");} stmt_list {printf("end");
for(i=0;i<=knt;i++)
printf(")");
knt=0;
```

```
}end
;
if_statement:
{printf("<if_statement>("); snt=1; knt=knt+1;}
IF cond THEN {printf("<stmt>("); snt=1; knt=knt+1;}stmt
;
cond:
{printf("<cond>("); knt=knt+1;}identifier{printf("<identifier>");}
op primary_value
;
op:
kucuk{printf("'<'");}| buyuk{printf("'>'");}|
buyukEsit{printf("'>='");}| kucukEsit{printf("'<='");}|
esitDegil{printf("'<>'");}
;
primary_value:
int_value{printf("<int_value>)");}
|
identifier{printf("<identifier>)");}
;
assign:
{printf("<assign>(");
knt=knt+1;}
{printf("<identifier>:=<expr>(");
knt=knt+1;}
identifier atama expr
;
expr:
primary
|
carp
|
bol
|
cikar
|
topla
;
carp:
identifier carpma_operator {printf("<expr>(<identifier>)*<expr>(");
knt=knt+1;}term
```

```
;
bol:
identifier bolme_operator {printf("<expr>(<identifier>)/<expr>(");
knt=knt+1;} term
;
cikar:
identifier cikarma_operator {printf("<expr>(<identifier>)-<expr>(");
knt=knt+1;} term
;
topla:
identifier toplama_operator {printf("<expr>(<identifier>)+<expr>(");
knt=knt+1;} term
;
term
     : primary
     | term carpma_operator primary
     | term bolme_operator primary
     | term toplama_operator primary
     | term cikarma_operator primary
     ;
primary:
identifier
|
int_values
{
printf("<int_value>");
if(jnt==0)
for(i=0;i<knt;i++)
printf(")");
else if(jnt==1 && vnt==0) {
for(i=0;i<knt-2;i++)
printf(")");
knt=knt-(knt-2);
vnt=1;}
else if(jnt==1 && vnt==1) {
for(i=0;i<knt-2;i++)
printf(")");
knt=0;
vnt=1;}}
;
```

```
%%
main()
{
     if(yyparse() && zaq==0)
            printf("\n!!! ERROR(S) OCCURED DURING PARSING PROCESS!\
n\n");
     else
            printf("\n\nPARSE BAŞARILI\n\n" RESET);
     return 0;
 printf("\n");
}
yyerror(s)
char *s;
{
if(zaq==0)
  fprintf(stderr, "%s\n",s);
}
yywrap(){
     return -1;
}
```

Döngüleri, değişkenleri ve sabitleri tanımladığımız bu dokümanın elbette yorumlanması gerekecek. Bu işi de bizim için **Lex dosyasına** yazacağımız **kodlama** hâlledecektir. Sonuç itibariyle **C benzeri bir dil** oluşmuş olacaktır.

Lex.l uzantılı dosyanın içeriği aşağıdaki gibidir.

```
%{
#include <stdio.h>
#include <string.h>
#include "y.tab.h"
int a;
int b;
extern int yylval;
int take(int);
long taken(long);
int Line,MyCol;
extern char *yytext;
%}
```

```
%%
" "       ;
">" {return(buyuk);}
"<" {return(kucuk);}
">=" {return(buyukEsit);}
"<=" {return(kucukEsit);}
"<>" {return(esitDegil);}
"*" {return(carpma_operator);}
"/" {return(bolme_operator);}
"+" {return(toplama_operator);}
"-" {return(cikarma_operator);}
":=" {return(atama);}
"program" {return(program);}
"end" {return(end);}
"if" {return(IF);}
"then" {return(THEN);}
"while" {return(WHILE);}
"begin" {return(begin);}
<<EOF>>     return END_OF_FILE;
[a-z]    {
            a = yytext[0];
            yylval = a - 'a';
            return(take(identifier));
          }
[0-9]*     {
            a = yytext[0];
            yylval = a - '0';
            return(int_values);
          }
[a-z][a-z0-9]* return(take(identifier));

"." {printf(".");}
(\r\n|\r\n) {printf("\n");}
[ \t\v\f]+  {printf("      ");}
%%
int take(int type){
            b = strlen(yytext);

    if(b>10){
            yyerror("10 Karakterden Uzun Tanımlama Degeri Girdiniz.
```

```
Hatalı Deger.");
     }
     else{
     Line++;
     return type;}
}
```

Oluşturduğumuz gramere göre çalışacak olan derleyicinin **syntax**'ı ise aşağıdaki şekildedir.

```
program sample
     abc:=a/4
     b:=20
     while a<b
     begin
     a:=c/4
     end end
```

BİLGİSAYAR AĞLARI

14

Bilgisayarların birbirleriyle haberleşmesine olanak sağlayan yapıya **AĞ** denir. Uzun ve elektronik haberleşme alanına giren tanımlara girmeden en kısa hâliyle **bilgisayar ağı** denilince bilmemiz gereken tanım budur. Tarihçesi ve gelişimi internet üzerinde pek çok kaynakta uzun uzun anlatılmaktadır. Biz bu bölümde, bilgisayar ağlarının ne olduklarını tanım olarak öğrenip, **sektördeki önemine** ve **kullanılan cihazlara** değineceğiz. Bilgisayarların haberleşmeleri için *Amerika DARPA kuruluşunda ortaya çıkan...* diye başlayan tanımları görmüşsünüzdür. Biz bu tanım ile değil ama ortaya çıkışı ile ilgilenebiliriz. Veri paylaşımı ihtiyacının duyulmaya başlandığı o günlerden bugünlere bilgisayar ağları hayatımızın aslında **tam merkezindedir**. Kısaca düşündüğümüzde zihnimizde yer edinen onca teknoloji, soket ile haberleşen medikal cihazlar, araçlar, TV yayınlarının günümüzdeki hâlleri, şirket içi iletişim gibi pek çok yerde ağlar **olmazsa olmazımızdır**.

Geçmişten günümüze servis edilen internet hizmetleri de gelişimine paralel olarak çeşitlilik göstermiştir. Örneğin çevirmeli ağ (*dial-up*), ADSL, VDSL, Fiber vb. gibi altyapılar. Bu altyapılara da bağlı olarak çeşitli hızlarda kullanabildiğimiz internet sayesinde **lokasyon fark etmeksizin** bilgisayarlar arasında haberleşmeyi gerçekleştirebilmekteyiz.

İnsan iletişiminde nasıl yazılı kurallar var ise devletler arasındaki protokoller vb. uyulması gereken onca kural var ise bilgisayarlar arasındaki haberleşmede de **uyulması zorunlu kurallar vardır**.

Bu kurallar çeşitli kuruluşlar tarafından ortaya konmuş (*genellikle Amerika*) ve bugünün dünyasında **protokoller** olarak standartlaşmıştır. Örneğin en temel internet kuralı **internet protokolüdür** (*IP*).

Bu bölümdeki okuyucu hedef kitlem, tamamen **öğrenci düzeyidir** diyebilirim. Karşınıza network ile ilgili pek çok kaynak çıkmış olabilir. Türkçe ve dünya dillerinde bu konuyla ilgili çok değerli kaynakların olduğu tartışılmazdır. Ancak benim bu bölümdeki amacım, network konusunda **tecrübesi olmayan** (*öğrenci*) okurlarımızı çok kavrama boğmadan **networkün ne olduğu ve sahada nasıl çalıştığı** ile ilgili adım adım bilgiler vermek olacaktır.

Haberleşme ile ilgili kuralların olduğunu ve bu kuralların artık standartlaştığını söylemiştik. Daha iyi anlaşılması için burayı biraz açmak gerekirse; bu kurallar, insanların bilgisayar kullanırken uyması gereken kurallar değildir. Bu kurallar zaten üreticiler tarafından işletim sistemlerine veya NIC kartlarına, yani **Ethernet kartlarına** tanımlanmış kurallardır. Yani istesek bile bu kuralları **değiştiremeyiz**. Değiştirmek istediğimizde de zaten iletişim sağlanmayacaktır. IP adresini düşünelim, mesela bir IP adresinin formatını değiştirdiğimizde **haberleşme sağlanmayacaktır**. Veya A, B, C, D, E sınıfı IP blokları dediğimiz blokların dışına çıktığımızda da **yine haberleşme sağlanmayacaktır**. O hâlde network konusunda çalışma yapmak istiyor ve henüz temel bilgiye sahipsek kuralların olduğu ve bu kurallara uymamız gerektiğini bilmeliyiz.

Tanımlanan bu kurallar veya standartların her birinin birer adı olup, aslında günlük hayatta detaylı olarak bilmesek bile karşımıza gelmektedir. Örneğin güvenlik kamerası sistemi kuran bir işletmenin, sabit IP adresi alması gerektiğini çevremizden duyarız. Burada **IP** (*internet protocol*) ve sabit-dinamik IP kavramlarını duymaktayız. Benzer şekilde bilgisayarımız internete bağlanmadığı zaman *acaba IP almış mı* diye ağ bağlantılarından kontrolünü sağlarız.

Ağ Çeşitleri ve Topoloji

Bilgisayar ağları hizmet vereceği potansiyele göre çeşitli **isimler** ve **özellikler** almaktadır. Genelde büyüklüklerine göre sınıflandırılan ağlar **en sade hâliyle**;

- **LAN** (*Local Ares Network*) **Yerel Alan Ağı**
- **WAN** (*Wide Area Network*) **Geniş Alan Ağı**
- **MAN** (*Metropolitan Area Network*) **Metropol Ağı**

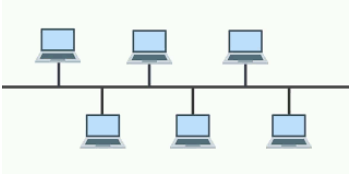
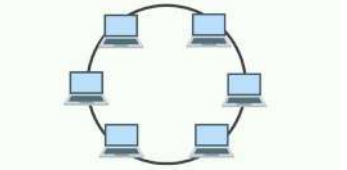
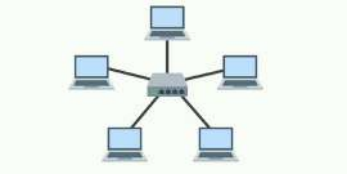
Olarak bilinirler. Bunlara ilaveten **CAN** (*Campus Area Network*), **PAN** (*Personal Area Network*), **SAN** (*Storage Area Network*) ve **VPN** (*Virtual Private Network*) ağları da literatürde ve uygulamada karşımıza çıkacaktır.

İsimlerinden çıkarım yapabileceğimiz gibi bu ağların **kullanım yerleri farklıdır**. Kurgulanan onca standart ve topoloji her ihtiyaca yönelik aynı tutulsaydı örneğin ev kullanıcısı ve üniversite ağı aynı şekilde kurgulansaydı bu ev kullanıcısı için gereksiz maliyet ve zor bir yönetim anlamına gelecekti. Bu yüzden bu şekilde bir sınıflandırma yapılır ve bu sınıflandırmanın da içerisinde **topoloji** denilen uygun kurgu yapılır.

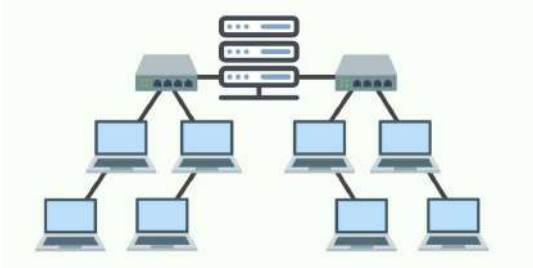
Topoloji: Network üzerindeki her bir bilgisayarın nasıl konumlandırılacağını, veri iletişimin hangi standartlarla sağlanacağını belirten yapıdır. Bu yapı, fiziksel olarak bilgisayarın nasıl yerleşime sahip olacağı ve mantıksal olarak da veri iletişiminin ne şekilde gerçekleşeceği hakkında yönetilebilirlik sağlar.

Topoloji Türleri

Yol (**BUS**) **Topolojisi**, **Halka** (**Token Ring**) **Topolojisi**, **Yıldız** (**Star**) **Topolojisi**, **Ağaç** (**Tree**) **Topolojisi**, **Karmaşık** (**Mesh**) **Topolojisi** ve bunların başına "**gelişmiş**" ön adı alarak genişletilmiş versiyonları bulunur. Bu topolojilerden haberleşme senaryosuna göre uygun olanı seçilir. Aşağıdaki tablo üzerinden *topolojilerin özelliklerini inceleyelim*.

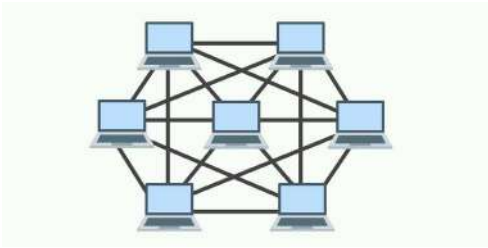
Topoloji Adı	Tanımı	Avantajları	Dezavantajları
Yol	Bilgisayarların tek bir hat üzerinde bağlı olarak çalıştığı topolojidir. Sonunda bir sonlandırıcı bulunur.	Ağa bir bilgisayar dâhil etmek çok kolaydır.	Hata tespiti zordur. Bir bilgisayarda sorun olursa tüm sistemde ağ kesintisi yaşanır.
 Şekil 14-1: Yol Topolojisi			
Halka	Bilgisayarların mantıksal olarak halka şeklinde birbirlerine bağlı oldukları topolojidir.	Çakışma olasılığı düşüktür.	Bir bilgisayar bozulduğundan tüm ağ bağlantısı etkilenir.
 Şekil 14-2: Halka Topolojisi			
Yıldız	Tüm bilgisayarların ortak bir NODE üzerine bağlanması şeklidir (*Switch, Hub gibi*).	Arızalı bilgisayar tespiti çok kolaydır. Kurgulamak kolaydır. Arıza olduğunda ilgili bilgisayarı etkiler. Çakışma olmaz.	Hub kullanımı trafiğin artmasına yol açar. Daha çok kablolama gereklidir. Hub veya Switch arızasında tüm ağ durur.
 Şekil 14-3: Yıldız Topolojisi			

Topoloji Adı	Tanımı	Avantajları	Dezavantajları
Ağaç	Yıldız topolojisindeki ağları birbirlerine bağlamak için kullanılır. Amaç ağları büyütmektir.	Farklı interface grupları bir araya getirilmiş olur.	Kurulumu zordur. Omurga bozulursa tüm sistem etkilenir



Şekil 14-4: Ağaç Topolojisi

Karmaşık	Node'ların birbirlerine bağlandığı güvenli ve geniş bir sistemdir.	Bir yerde arıza olması sadece kendisini etkiler. Hız oldukça yüksektir.	Çok fazla bağlantı olabilir. Maliyeti yüksektir



Şekil 14-5: Karmaşık Topoloji

Kablo Çeşitleri

Topolojilerde görmekte olduğumuz bağlantılar hep **kablolarla** sağlanmaktadır. Kablolarda, iletişimde standart hâle gelmiş ve uygun topoloji veya altyapı için **uygun olan kablonun** seçilmesi gerekmektedir. Kablolar internetin gelişimiyle beraber internet hızlandıkça ve daha çok veri iletebilir hâle geldikçe bunu mantıksal katmandaki haberleşme metotlarıyla birlikte altyapıda bulunan **kablolar** sayesinde gerçekleştirmiştir. Çevirmeli internet kullanıldığı zamanki kablolama ile ADSL kullanımındaki kablolama farklılık gösterirken **günümüz fiber teknolojisindeki kablo farklıdır**.

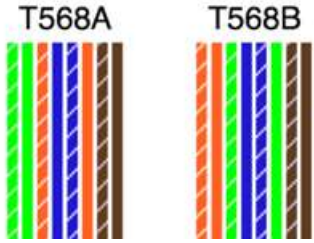
Bu kablo türlerinin kendilerine kıyasla **bağlantı mesafesi** gibi özellikleri bulunur. Kimi kablolar 100 metreye kadar hizmet ederken kimi kablolar 1000 metreye kadar hizmet verebilmektedir. *Kablo çeşitlerini* aşağıdaki tablo üzerinden inceleyelim.

Kablo Adı	Özelliği	Detay	Görsel
Koaksiyel (Coaxial) Kablo	Ses ve video iletebilir.	Televizyon anten kablosu ile aynıdır. Uydu üzerinden internet hizmeti alınan sistemlerde de kullanılır.	 Şekil 14-6: Koaksiyel Kablo
Çift Bükümlü (Twisted Pair) Kablo	Oldukça yaygındır. CAT1, CAT2, CAT3, CAT4, CAT5, CAT5e, CAT6, CAT6a, CAT7 gibi kategorileri vardır.	Günümüzde en çok CAT5 kullanılır. 100 MHz'lik frekansa sahiptir ve 100 Mbit/Saniye veri taşır. CAT6 ise 250 MHz'lik bir frekansa sahiptir. 1 GB/Saniye veri taşıyabilir.	 Şekil 14-7: Çift Bükümlü Kablo
Fiber Optik Kablo	Elektrik sinyali yerine ışık hızıyla veri iletimi sağlayan kablolardır. Çok yüksek veri iletimi yapabilirler.	İnce fiber cam üzerinden ışık dalgası geçirerek iletişim sağlanır. Ekleme ve arıza bulma için özel cihazlar kullanılır.	 Şekil 14-8: Fiber Optik Kablo

Bu kabloların bilgisayarlarda ve internet cihazlarında kullanılabilmesi için **uçlara** ihtiyaçları vardır. Bu uçlar **her bir kablo için farklıdır**. Örneğin Koaksiyel kablo için BNS konnektör kullanılırken eş bükümlü, yani **tipik CAT5 kabloları için RJ-45 konnektörleri** kullanılmaktadır. Fiber optik kablolar için de yine ona uygun konnektör kullanılmak zorundadır. Bu konnektörlerin kablolara çakılması için **ekipman** gerekmektedir. Özellikle fiber optik yapısındaki hemen her iş için **özel donanımlara ihtiyaç vardır**. Klasik elektrik ileten kablolardaki gibi *kesip eklemek, bantlamak* gibi işlemler **fiber optik kablolara yapılamaz**.

CAT5 gibi kablolama sistemlerinde kabloların içerisindeki **renkli sarmallar** iletişimin doğru sağlanabilmesi için **belli bir standarda göre sıralanmaktadır**.

A ve B tipi haberleşme standardı için kablolara uç basılırken **renk sıralamasına dikkat edilmelidir**. **Şekil 14-7**'de görülen renkli ikililer B standardına göre sıralanacağı zaman iki ucu da sırasıyla şu şekilde sıralanmalıdır; **Turuncu Beyaz Turuncu**, **Yeşil Beyaz Mavi**, **Mavi Beyaz Yeşil**, **Kahverengi Beyaz Kahverengi**. A standardına göre sıralanacağı zaman ise; **Yeşil Beyaz Yeşil**, **Turuncu Beyaz Mavi**, **Mavi Beyaz Turuncu**, **Kahverengi Beyaz Kahverengi** şeklinde sıralanmaktadır.



Şekil 14-9: Renk Sıralaması

Kablosuz Bağlantılar

Kablosuz bağlantı teknolojisi, hayatımızda oldukça yaygındır. Wifi sistemleri her geçen gün yaygınlaştırılmış ve artık evlerimizde, restoranlarda, otobüs, tren ve uçaklarda, kamu kurumlarında ve hatta park bahçelerde bile insanların kullanımına sunulmuştur. Wifi hayatımıza **IEEE 802.11** standardıyla karşımıza çıkan bir **yerel ağdır**. Wifi bağlantıları **radyo sinyallerini** kullanırlar. **Sinyal dağıtıcı** ve **sinyali alıcı** birer cihaz vasıtasıyla iletişim sağlanır. Kablosuz ağlar **2.4 GHz** ile **5 Ghz** arasındaki frekansı kullanırlar. Aşağıdaki tablodan *IEEE standartlarına göre WIFI özellikleri* incelenebilir.

Frekans	IEEE Standart Adı	Hız
2.4 GHz–2.5 GHz	802.11n	100 Mbps
2.4 GHz	802.11g	54 Mbps
2.4 GHz	802.11b	11 Mbps
5 GHz	802.11a	Max 54 Mbps/Ort 6-24 Mbps

OSI Referans Modeli

Open System Interconnection kelimelerinin baş harflerinden kısaltılan **OSI** modeli, **yedi katmandan oluşan bir model**dir. Bahsi geçen tüm protokol ve standartların uygulandığı model de diyebiliriz. OSI ile ilgili kaynaklarda genellikle çok detaylı bilgiler verilmektedir. Bu bilgilere ihtiyacımız olduğunda kesinlikle başvurmalıyız ama kitabımızın konsepti gereği OSI ile ilgili *işimize yarar kısma* tanım olarak değinecek, sahada kullanılan metotların *OSI karşılığının ne olduğu* gibi daha akılda kalıcı bir anlatıma yer vereceğiz.

Tüm haberleşmenin bir dize **standart** ve **protokol** ile sağlandığı ve dünyanın çeşitli yerlerindeki standart oluşturucu kuruluşlar tarafından oluşturulduğu ve geliştirilmelerinin devam ettiğini söylemiştik. Bu tür kuruluşlara genellikle sektörün dünya lideri pozisyonundaki firmalar da katılım sağlamaktadır. **OSI referans** modelini **International Organization for Standardization** yani **ISO** geliştirmiştir.

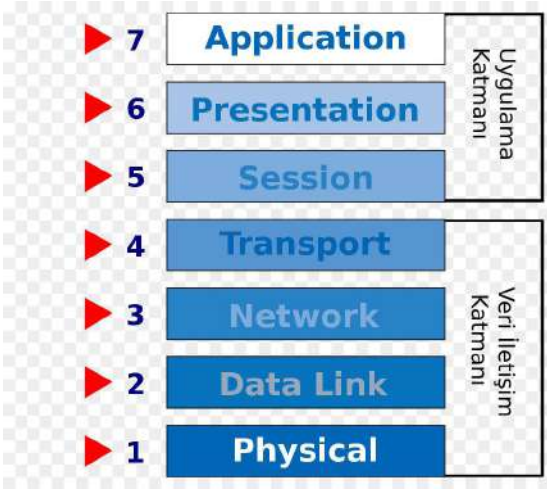
Standart olmasaydı bilgisayar iletişimi sağlanamaz mıydı? **Kesinlikle sağlanırdı**. Soru bu olduğunda standartlara ihtiyacımızın olmadığını söyleyebiliriz. Ancak ortak dili konuşmak ve modüler bir yapıya geçebilmek için **ortak dili** konuşuyor olmamız kaçınılmaz bir hâl almıştır. Eğer bu tür standart olmasaydı bilgisayarlar yalnızca aynı üreticinin donanımlarının **birbirlerini tanıması sebebiyle iletişim kurulabilirdi**. Ancak farklı markalara sahip donanımlar birbirleriyle haberleşemezlerdi. Aslında tarihinde bu şekilde farklı markalarla gelişen internet haberleşmesi teknolojisine OSI referans modeli **DUR** diyerek belirli standartlar ortaya koymuş ve marka bağımsız olarak ortak bir haberleşme yapısını geliştirmiştir. Buna göre OSI referans modeli sayesinde protokoller ve mimarilerdeki standartlar **sanki aynı ağ ürününün bileşenleriymiş gibi kullanılabilmektedir**. OSI tüm bu standardı ortaya koyarken ağ mimarisindeki soyut **seviyeleri 7 katmana ayırmıştır**. Bu katmanlarla birlikte OSI referans modeli 7 katmanlı OSI olarak da ifade edilmektedir.

OSI katmanları için buraya kadar anladıklarımızla yorum yapacak olursak *uygulama, fiziksel, iletişim, transfer* gibi temel network aksiyonlarının her birinin katmanlara ayrıştırıldığını ve bu katmanlarda yazılan standart kurallara göre haberleştiğini üreticilerin de örneğin bir **switch** üretirken bu katmanlı mimariyi esas aldıklarını söyleyebiliriz.

OSI referans modelindeki **ilk 4 katman**, **veri iletişim katmanı** olarak gruplandırılırken sonraki **üç katman da uygulama katmanı** olarak gruplandırılmaktadır. Aşağıdaki görsellerde OSI referans modelinin *7 katmanlı hâli ve gruplandırılmış hâlini* inceleyebilirsiniz.

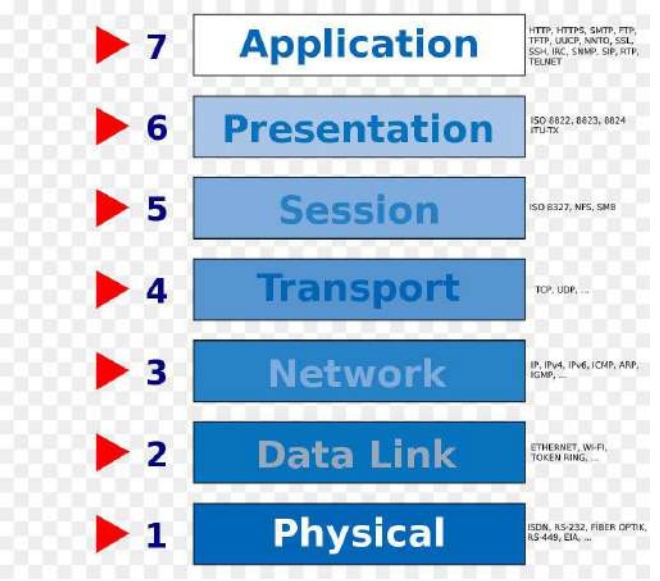


Şekil 14-10: OSI Referans Modeli



Şekil 14-11: Gruplandırılmış OSI Referans Modeli

Bu katmanların her birinde çalışan protokoller de **farklıdır**. Aşağıdaki şekilde *OSI referans modelinin katmanlarında çalışan protokolleri* inceleyebilirsiniz.



Şekil 14-12: Standart ve Protokoller

Peki, bu katmanlarda neler dönüyor bir de bunları inceleyelim.

Fiziksel Katman–Physical Layer (1. Katman)

OSI referans modelinin **donanım katmanı**dır. OSI arasındaki haberleşmeyi bir bilgisayara benzettiğimizde ise OSI modelinin **donanım kartı bu katman**dır. Elektrik, ışık, radyo gibi sinyallerin **iletiminden sorumlu**dur. Donanım kelimesi bir kez geçtiği için diğer ağ cihazları olan router, hub gibi cihazlarından da bu katman üzerinde çalıştığını söyleyebiliriz. Ayrıca yukarıdaki görselde de görüldüğü gibi **RS232** gibi protokoller bu katmanda çalışmaktadır.

Veri İletim Katmanı–Data Link Layer (2. Katman)

OSI referans modelinin **ikinci katmanı**dır. Veri bağı, veri bağlantısı gibi isimlerle de çağrılır. Bu katmanda fiziksel katmana erişmek için gereken kurallar belirlenir. Bu katmandaki işlemlerin bir kısmı **Ethernet kartında** gerçekleşmektedir. Bu katman; **bilgisayarların tanınması**, **hattın o an kim tarafından kullanıldığı gibi bilgileri kontrol etme görevini** de yürütür. Veri iletim katmanındaki işlemlerde MAC adresine dayalı bir kontrol yapılır ve iletişimdeki olası bir hata durumu için hata kontrol kodu **CRC**, alıcı ve gönderen **MAC** adresleri ile birlikte paketler **fiziksel katmana iletilir**. Aynı şekilde alıcıdan gönderen yönündeki iletişimde de aynı işlemlerin tersi yapılır. Tersten yapılan iletişimde de **LLC** alt katmanı devreye girer ve bozuk giden verilerin tekrar gönderilmesi gibi işlemlerden sorumludur. MAC adreslerinden bilgisayarların tanınması işlemini **akıllı switch** cihazları yapabildiği için switch cihazları ikinci katmanda yani veri iletim katmanında çalışan cihazlardır.

Ağ Katmanı–Network Layer (3. Katman)

Veriler farklı bir ağa da gönderilmesi gerektiğinde **ağ katmanı** devreye girer. Yönlendiricinin kullanacağı bilgi bu katmanda eklenir. Bu katman sayesinde veriler yönlendirici **router cihazı** aracılığı ile yönlendirilebilir. Kısacası switching, routing mekanizmaları bu katmanda çalışmaktadır. Buradan da anlaşılacağı üzere Router cihazı bu katmanda çalışmaktadır. Ayrıca **TCP/IP** protokolü bu katmanda çalışmaktadır.

Transfer Katmanı–Transport Layer (4. Katman)

Alt katmanlar ile üst katmanlar arasındaki iletişimde **köprü görevi gören** bir katmandır. Veriyi ağ paketleri boyutunca parçalara bölmektedir. Bu yüzden bu katmandaki veriler segmentler hâlinde taşınırlar. Verinin transferinden sorumlu olduğu için **uçtan uca iletimini ve ulaşıp ulaşamadığını** kontrol eder. Söz konusu transfer olunca **TCP** ve **UDP** protokollerinin burada çalıştığını söylemeden geçmeyelim.

Oturum Katmanı–Session Layer (5.Katman)

İletişim hâlindeki bilgisayarların aynı anda **bire çok bağlantı** kurmaları durumunda doğru bilgisayar ile eşleşmesi için **gereken oturum yönetimini** sağlar. Bir üst katmana yani sunum katmanına iletilecek olan verilerde farklı oturumlara sahip verilerin **ayrıştırılması sağlanır**. Aynı anda bilgisayar, disk ve yazıcı kullanımı olduğu senaryoyu düşünebiliriz. Bu ayrımı yaparken **NetBios** gibi protokollerden faydalanır.

Sunum Katmanı–Presentaion Layer (6. Katman)

Bilgisayar dilinde gerçekleşen alt katmanlardaki tüm işlemlerin artık sunum katmanı sayesinde kullanıcıya anlaşılır şekle dönüştürülmesi yani **sunulması işleminden sorumlu**dur. Örneğin mimaride gerçekleşen tüm **kriptolama vb. işlemler** artık bu katmanda sunulacak hâle getirilirler.

Uygulama Katmanı–Application Layer (7.Katman)

Soyut seviyeden somut seviyeye doğru yükselen OSI referans modeli yapısında **kullanıcıya en yakın olan katman**dır. Adı üzerinde, bu katmanda çeşitli uygulamalar kullanabiliriz. Son kullanıcı ile uygulamalar arasındaki prosesler, örneğin *web browser* gibi bir uygulama ile bu katmanda gerçekleştirilmektedir. Dosya aktarımı gibi hizmetler de bu katmanda çalışmaktadır. **FTP**, **http**, **Telnet** gibi protokoller bu katmanda çalışmaktadır.

Protokoller (Portlar)

İnternet dünyasındaki protokolleri IP protokolü, taşıma protokolü ve uygulama protokolü olarak OSI modelinde ve **Şekil 14-12**'de incelemiş olduk. Şimdi, bu kategoriler ışığında biraz detaylarına inelim.

DHCP: Bir bilgisayara otomatik olarak IP tanımı atanmasını sağlar.

ICMP: Çeşitli bilgi mesajlarını içeren protokoldür.

RIP: Yönlendiricilerin route table'larını otomatik türetebilmelerini sağlayan protokoldür.

OSPF: RIP'ten daha gelişmiş bir protokoldür. Aynı iş için üretilmiştir.

IGMP: Multicast yayınlara abone olunması veya durdurulması işlemine yarar.

ARP: Herhangi bir IP adresinin hangi MAC adresine ait olduğunu tespit eder.

RARP: Hangi MAC adresinin hangi IP adresini kullandığını tespit eder.

UDP: Veri transferini IP üzerinden gerçekleştirir. Verinin teslim edilip edilmediğinin garantisini vermez. Basit bir yapıya sahiptir. Hızlıdır fakat güvenli diyemeyiz.

TCP: Yine IP üzerinden veri transferini gerçekleştirir verinin iletilip iletilmediğini kontrol eder. Bu kontrol mekanizması için UDP'den farklı olarak TCP haberleşen iki sistemin birbiriyle iletişim hâlinde olması beklenir.

DNS: Alan adlarını IP adreslerine, IP adreslerini ise alan adlarına dönüştüren sistemdir.

HTTP: TCP üzerinden çalışır ve Html veya herhangi bir türden veri gönderimi için kullanılır.

HTTPS: Veri gönderimi işlemini RSA ile şifreler ve güvenli iletişim kurulmasını sağlar.

POP3: E-posta sistemlerinde e-posta alabilmek için kullanılan protokoldür.

SMTP: E-posta sistemlerindeki e-posta gönderebilmek için kullanılan protokoldür.

FTP: Temel dosya transfer işlemlerinde kullanılan protokoldür.

ADI	Port Numarası
FTP	20,21
SFTP	115
SSH	22
TELNET	23
SMTP	25
DNS	53
HTTP	80
HTTPS	443
KERBEROS	88
POP	109
POP3	110
IMAP4	143
SNMP	161
BGP	179
LDAP	389
IMAP4 SSL	993
POP3 SSL	995
NETBIOS	139
MYSQL	3306
RDP	3389

VLAN, Statik-Dinamik VLAN

Yerel alan ağımızı sanal olarak ayrıştırma yapısına **VLAN** denir. **Virtual LAN** olarak bilinir. VLAN yaparak; ağ üzerindeki kaynakları mantıksal olarak ayırmış oluruz. Bu ayrım, switch üzerindeki portlara atanarak yapılır. VLAN yapısı sayesinde her VLAN kendi broadcast'ini kullanacağı için trafik azaltılmış olur dolayısıyla **bant genişliği** sağlandığından söz edebiliriz. Network'ü VLAN'lara ayırmak daha **yönetilebilir** ve kontrolün **elde tutulabildiği bir network oluşmasını** sağlar. VLAN'lar arasındaki haberleşmelerde de ağ güvenli hâle getirilmiş olur. VLAN yapısı kullanmak aslında iç network güvenliğinin **temel adımlarından** biridir.

Bir iş yerinde; muhasebe, insan kaynakları, depo ve stajyer birimler olsun. Muhasebe biriminde faturalar bizim için gizli ve önemli bilgi olurken insan kaynaklarında da bordro ve maaşlar bu şekilde gizli ve önemli bilgiler olsun.

Eğer VLAN yapısı olmasaydı bu birimlerin her biri aynı networkten internete çıkıyor olacaklardı. Hem güvensiz hem de olası bir port arızası vb. durumda tüm ağ etkilenmiş olacaktı. Ancak VLAN yapısı ile bu dört birimin her biri için **ayrı VLAN tanımı yaparak** insan kaynaklarının eriştiği verilere muhasebenin erişememesi, stajyerlerin hiçbir yere erişmeden sadece internete çıkabilmeleri gibi ayarlar yapılabilir. **Saldırganlar için de önlem niteliğinde olan VLAN yapısı** özellikle büyük networklerde **kesinlikle kullanılmalıdır**. Kablosuz internet hizmeti verdiğiniz Wifi VLAN yapısı ile mali işlerin yürütüldüğü muhasebe VLAN yapısının niçin birbirinden ayrı tutulması gerektiğini şimdi daha iyi benimsemiş olmalıyız.

VLAN'ların arasındaki ilişki ise iki şekilde yapılmaktadır. Bunlardan ilki **Statik VLAN** ikincisi ise Dinamik VLAN yapısıdır.

Statik VLAN: Her bir VLAN için sistem yöneticisi tarafından bir **port ataması** yapılır. Bu atama switch üzerine yapılan bir atamadır. Örneğin stajyer VLAN ID-50 portunda çalışsın, muhasebe VLAN ID-30'da çalışsın gibi. Bu portlar sistem yöneticisi değiştirmediği sürece **sabit** kalırlar. Bu sayede müdahale oldukça yalın bir şekilde yapılabilir. Böyle bir yapıda her bir birimin kendisine ait VLAN ID'leri olur ve bu ID'ler switch tarafında bir porta refere edilir. Ancak bir de ağdaki diğer network cihazlarının kendi aralarında **haberleşirken kullanacakları** bir VLAN daha olmak zorundadır. Bu VLAN'da **Management** yani **yönetim VLAN**'ı olarak bilinir ve VLAN-100 olarak tanımlanır. Tüm ağ cihazları Management VLAN üyesi olmak durumundadır.

Dinamik VLAN: Statik VLAN yapısında olan VLAN port eşleşmesi işleminin **dinamik olarak yapıldığı** yöntemdir. Merkezî bir MAC adresi yönetim uygulaması üzerinden sistemdeki MAC adreslerinin işlendiğini düşünelim. Buraya kayıtlı cihazlardan bir tanesinin VLAN tanımı yapılmamış olan bir portla ağa çıkması durumunda, VLAN yönetim sistemindeki kayıtlı MAC adresleri içerisinden bu cihazın **MAC adresi sorgulanır** ve karşılık gelen VLAN değeri ilgili switche, ilgili porta **dinamik olarak atanır**. Çalışma zamanında gerçekleşen bu atama işleminden dolayı dinamik adını almaktadır.



ACCESS ve TRUNK LINK VLAN yapılarını araştırınız. Bir network yönetiminde ne anlama geldiklerini pekiştiriniz.

IP Adresi Çeşitleri ve Sınıfları

Bir Windows işletim sistemine sahip bilgisayarın, **Ağ Bağlantılarım Ayar** penceresinde sıklıkla gördüğünüzü düşündüğüm **IP** adresi çeşitleri **IPv4** ve **IPv6** şeklindedir. Günümüzde IPV4 kullanılırken interneti kullanan cihazların sayısının her geçen gün artması sebebiyle güvenli iletişimin sağlanabilmesi için **IPv6 protokolü** de geliştirilmiştir. IPv6'ya geçiş için henüz net bir tarih bulunmamaktadır. IPv6 128 bitlik yapısı sayesinde şu an için bitmeyecek bir IP havuzu anlamına gelmektedir.

IPv4, **32 bit** adresleri kullanmaktadır. Bu adresleme ile **4 milyar 294 milyon 967 bin 196** adet cihaz adreslenebilmektedir. Evet, *artık yetmiyor* dediğimiz rakam tam olarak budur. 2^32 cihazın adreslenebildiği IPv4 yapısında adresler birbirlerinden sekiz bitlik parçalar hâlinde ayrılan 4'lüklerden oluşmaktadır. Bu değerler **0-255** arasındadır. Örneğin: **193.168.10.5**

Ağ üzerinde iletişimde bulunan her cihaz ki bu cihazlar sadece bilgisayar olmak zorunda değildir günümüzde **endüstri 4.0** kavramı ile birlikte birçok cihaz internete çıkabilmektedir. Bu cihazlar evlerimizdeki fırınlardan otomobillerimize kadar çok geniş yelpazede yer almaktadırlar. Bu cihazların her birinin IP adresine sahip olmaları gerekmektedir. IP bir cihazın internet dünyasındaki **kimliğidir**. Hâl böyle olunca yine bir sınıflandırma mekanizması gelişmiş ve IP adresleri **A, B, C, D, E** şeklinde sınıflara ayrılmıştır.

Bir IPv4 adresi **32 bitten** oluşan ve **4 oktetten** meydana gelen bir dizidir. Aralarında nokta ile ayrılan bu oktetlerden ilk oktet grubuna bakarak hangi sınıfta olduğuna karar verebiliriz.

A Sınıfı: İlk oktetin 0-127 arasında olduğu sınıftır.

B Sınıfı: İlk oktetin 128-191 arasında olduğu sınıftır.

C Sınıfı: İlk oktetin 192-223 arasında olduğu sınıftır.

D Sınıfı: İlk oktetin 224-239 arasında olduğu sınıftır.

E Sınıfı: İlk oktetin 240-255 arasında olduğu sınıftır.

Bazı IP adresleri önceden rezerve edilmiştir veya bilimsel amaçla kullanmak vb. gibi sebeplerden dolayı kullanılamaz.

Ağ Cihazları

Bu başlığa gelene kadar çeşitli ağ cihazlarının isimlerine değindik. Ancak bunların ne olduklarına değinmedik. Bu bölümde ise temel network **aktif cihazlarının** neler olduklarına değinelim.

Switch: Üzerinde bağlı olan bilgisayarların veya cihazların haberleşmesini kendi içerisindeki konfigürasyona bağlı kalarak gerçekleştiren **akıllı** (*yönetilebilir*) **ağ donanımıdır**. ISO referans modelinin Data Link ve Network katmanlarında (*2 ve 3*) yer alır.

Hub: Üzerinde bağlı olan bilgisayar veya cihazların haberleşmesini sağlar ancak yönetilemezdir bu yüzden **Switch'lerden farklıdır**.

Router: Gelen ağ paketini inceler ve farklı bir ağa yönlendirebilir. Birden çok ağ arasındaki bağlantıyı sağlayan ağ donanımıdır.

Backbone: Birden fazla switch arasındaki iletişimin yönetildiği **ana switch**'dir. *Omurga switch* olarak da bilinir.

Firewall: Bilgisayar sistemlerini korumak için üretilmiş **güvenlik duvarı** sistemleridir. Bu sistemlerin bir donanım ile paketlenerek ürün hâline getirilmiş şekline de *firewall cihazı* denmektedir.

Analyzer: Network'teki tüm hareketleri analiz eden, özellikle firewall'dan aldığı verileri işleyen ve anlamlı sonuçlar çıkacak şekilde **parse eden** sistemlerdir. Donanım birimleriyle paketlenerek ürün hâline getirilince *analyzer cihazı* denilmektedir.

DDOS Attack Rejecter: Network üzerinde DDOS ataklarına karşı koymak için kullanılan bir cihazdır.

WAF: Web Application Firewall, network'te bulunan **web sunucunun önüne** konumlandırılır. Web sunucusunun güvenliğinden (*firewalldan sonra*) sorumludur. Üzerine SSL sertifikası kurulumu yapılabilir. IP engelleme, kısıtlama, atak durumunda uygulamayı kapatma gibi aksiyonlar alınabilir.

Access Point: Kablosuz internet hizmetinde **kapsama alanı genişliği** sağlayan cihazlardır. Evlerimizde de prizlere takılan versiyonları mevcuttur. Büyük sitemlerde yönetilebilir olması tercih edilirken evlerimizde sadece modemden aldığı sinyalleri güçlendirmesi yeterlidir.

Access Point Controller: Büyük network'lerde Access pointlerin de yönetilebilmesi, belirli konfigürasyonlara göre çalışabilmesi farklı binalarda farklı SSID yayınlanması vb. özellikler için tüm Access pointlerin **tek bir yerden** yönetilebilmesini sağlayan cihazlardır.

Modem: Modülasyon ve Demodülasyon kelimelerinin birleşiminden oluşmuştur. Sayısal verileri telefon hattından **geçebilecek hâle getirmek** için bir tür modülasyon ve gelen verileri bilgisayarlarımızın anlayacağı hâle dönüştürme işlemi için de bir tür demodülasyon işlemi yapmaktadır.

Örnek Bir Kurgu

İnternet mühendisliği veya network yönetimi kavramları için anlatacaklarımız elbette burayla sınırlı değildir. Günümüzde elektronik işlemlerin hepsinin internete olan bağımlılıkları bu elektronik haberleşmeyi de geniş bir çalışma alanı hâline getirmiştir. Kitabımızın bu bölümünde *ilgili konulara özet niteliğinde değinmiş* ve *okuyucuları bir sonraki adıma taşıyabilmek için gerekli altyapıyı verdiğimizi düşünüyorum*. Bu bölümü tamamen okuyan bir aday, ileri seviye network kitap veya kaynaklarından çok daha fazla istifade edebilecektir.

Bu başlık altında ise **bir işletme için kurulacak network altyapısını hayal edeceğiz** ve **yapılması gereken işlemleri kurgusal olarak maddelendireceğiz**.

1. İnternet hizmeti aboneliği alınmalıdır. Bu abonelik sonrası bize **dış IP adreslerimiz** verilmelidir.
2. Sistem odası kurulmalı ve bu odanın içerisine bir adet **omurga switch**, **omurga switch yedeği**, **Firewall**, programların çalışabilmesi için sunucu bilgisayar ve DMZ port yapılandırması yapılmalıdır.
3. Kurum içi telefon hizmetinin de IP telefonlar ile sağlanacağını düşünürsek IP santral yazılımı da **sistem odasında** konumlandırılmalıdır.
4. DMZ **Switch sistem odasına** konumlandırılmalıdır.
5. WAF ve Analyzer **duruma göre sistem odasına** konumlandırılmalıdır.
6. Kurum içerisindeki tüm birimler için **VLAN yapılandırması** yapılmalıdır.
7. Kamera, fotokopi cihazı, çeşitli otomasyon makineleri için **VLAN tanımları** yapılmalıdır.

8. Tüm sunucu, santral ve kenar switch'lerin kurulumu yapılmalı, omurga switch'e **gerekli olan konfigürasyon** yapılmalıdır.

9. Firewall üzerinden **gerekli kurallar** (*Policy*) yazılmalıdır.

10.İnternete kullanıcılar NAT yapılarak çıkartılmalıdır. **NAT kuralları da Firewall'a yazılmalıdır**.

11. Kurum içerisinde çalışan tüm bilgisayarların **RDP portları kapatılmalı** ve uzaktan bağlantı gerekiyorsa **Firewall'dan SSL VPN ayarlanmalıdır**.

12. Web sunucusunda barındırılan tüm web sitelerinde **SSL sertifikası** yüklenmelidir.

13. Birimlere internet erişimi **VLAN üzerinden verilmeli** ve **IP-MAC eşleştirmesi** yapılmalıdır.

14. Kablosuz internet erişiminde **Captive Portal yapısı** kullanılmalı ve erişimler LOG'lanmalıdır. Kablosuz internet yine ayrı bir VLAN ile servis edilmelidir.

15.IPS/IDS sistemleri ile birlikte güncel veri tabanına sahip bir **anti-virüs yazılımı da bir sunucuya yüklenmeli** ve **yönetilmelidir**.

16.KVKK hükümleri gereğince bir sunucuda **DLP** yazılımı çalıştırılmalıdır.

17. Bilgisayarların merkezî yönetimi için **Active Directory** kurulabilir.

18. Prizlerdeki internet portları için **port-security** yapılmalıdır.

19. Kullanıcıların yazılım yükleme işlemleri kontrol altına alınmalı, gerekmiyorsa izin verilmemelidir.

20. Oyun vb. zararlı veya mesai saatleri içerisinde erişilmemesi gereken web siteleri **IP bazlı** veya **kategori bazlı** olarak engellenebilir, filtrelenebilir.

Yukarıdaki maddelerin tamamı kesinlikle yapılması gereken işlemler diyemeyiz. Ancak kurumsal bir yapı hayal ettiğimizde en az bu maddelerde belirtilen işlemler yapılmalıdır. Elbette bu işlemlerin üzerinde çok daha yapılacak eylem mevcuttur. Bu bölümümüzde, işlemleri yukarıda olduğu gibi 20 madde ile listelememdeki amaç; *henüz üniversite öğrencisi olan okurlarımızın* veya *teorik bilgilerin pratikte nerelere karşılık geldiğini merak eden araştırmacılarımızın*, bilişim işine gönül vermiş değerli okurlarımızın öğrendikleri bilgilerin sahada ne şekilde hangi cihazlarla veya hangi yöntemlerle uygulanabildiği hakkında kısacası soyut birçok kavram hakkında daha net düşünebilmesini sağlamaktır.

Üniversitelerdeki birçok öğrenci arkadaşımızdan gelen soruları da değerlendirerek hazırlamış olduğum bu bölümde, temel olarak **network kavramının anlaşılması**, uygulamada **ne tür aksiyonların alınabildiği** ve **işin nerelere doğru yöneldiğinin** farkındalığının oluşması amaçlanmıştır.

YAPAY SİNİR AĞLARI-MAKİNE ÖĞRENMESİ VE DERİN ÖĞRENME KAVRAMLARI

15

Bu konu karşımıza; makine öğrenmesi, yapay zekâ, derin öğrenme ve hatta bulanık mantık derslerinde bile çıkabilecek bir konu başlığıdır. Yapay zekâ kavramı yeni bir kavrammış gibi görülse de aslında makinelerin **insan gibi düşünebilmesi** kavramının 40 yılı aşkın bir serüveni bulunmaktadır. Bu bölümde yapay sinir ağları, makine öğrenmesi ve derin öğrenme konularına değiniyor olacağız.

Yapay Sinir Ağları (Ysa)

Yapay sinir ağları, tıpkı insan beynindeki davranışlar gibi sinirlerin davranışlarının yorumlanarak **çeşitli amaçlar doğrultusunda sonuçlar elde edilmesi amacıyla** kullanılan yapılardır. Yapay sinir ağları öncelikle çeşitli konularda **örnekler toplar** ve bu örnekler üzerinde **öğrenme gerçekleştirilir**. Daha önce bilinmeyen örnekler öğretildikten sonra artık tanıdığı örneklerle tekrar karşılaştığında o örnekleri kullanarak çeşitli karar mekanizmalarının çalışması bu modelden beklenen bir sonuçtur. Buradan hareketle çalışma prensibinin bir kez daha insan beyni gibi insan zekâsı gibi olduğundan yapay zekâ adını aldığını söyleyebiliriz. Karmaşık ve büyük problemleri çözmek için bilgisayara bir kez konu hakkında tüm örnekler öğretildikten sonra bu örneklerden yararlanarak problemleri çözme veya çeşitli yorumlar katma mekanizmasında **yapay sinir ağları algoritmaları** önemli rol oynamaktadır.

Yapay sinir ağları yukarıda da bahsedildiği gibi **insan beynindeki sinir ağlarını taklit eden** yapay sinir ağlarıdır. Bu sinir ağları birbirlerine **ağırlıklı bağlantı** denilen metotlarla bağlantılıdır ve her bir bağlantısının kendisine ait belleği bulunmaktadır. Yapay sinir ağlarının kullanım yerleri, aslında tanım bir kez daha okunursa biraz daha güncel örnekler ile herkesin zihninde canlanacaktır. Günümüzde *insansı robotlar, çeşitli sanayi kolundaki iş robotları, görüntü veya ses tanıma operasyonları, tahmin işlemleri, sistem tanımlama işlemleri, hastalık tahmin teşhis işlemleri, firewall gibi network cihazlarının kural tahmin modülleri* vs. pek çok yerde kullanım örneğine rastlamak mümkündür. Üniversitelerimizde veya araştırmacıların **Yapay Sinir Ağlarını** (**YSA**) incelemeye başladıkları ilk deneme projelerinden bir tanesi de bir ilin son üç yıllık hava tahmin raporlarına göre bir sonraki yıl için hava tahminini yapan projelerdir. Bu tür projelerde algoritmaları **MATLAB** gibi programlarda hazır olarak görebilmekteyiz. (*Neural Networking Toolbox*) Örneğin böyle bir projede eski yıllara ait verilerden bir kısmını **eğitim verisi**, bir kısmını **test verisi** olarak tanımlayabiliriz.

YSA modellerini ve algoritmalarını daha iyi kavrayabilmek için öncesinde **makine öğrenmesi**, **derin öğrenme derslerini** ve hatta **discrete mathematics** gibi içerisinde en iyi sonuç algoritmalarını barındıran dersleri almış olmak önem arz etmektedir. Bu derslerin konularını iyice hazmetmeden yapay sinir ağları veya yapay zekâ uygulamalarına göz atmak yalnızca bu konu hakkında bir kültür edinilmesini sağlayacaktır. Yapay sinir ağlarının birtakım işlevleri gerçekleştirmesi beklenmektedir. Bu işlevler şöyledir;

- **Öğrenme**
- **İlişki Kurma**
- **Sınıflandırma**
- **Genelleştirme**
- **Tahmin Yürütme**
- **Özelliklendirme**

Bir YSA modelini klasik bir algoritmadan ayrı tutmak gerekir. Çünkü YSA modelleri doğrusal olmayışı ve paralel oluşu ile **biyolojik sinirler gibi davranır**. Ayrıca hata toleransı, öğrenebilirliği ve analiz edilebilme kolaylığı ile farklılık gösterir. Bu farklılıkları olumlu hatta **avantaj** olarak nitelendirebiliriz. Elbette avantaj olarak nitelendirdiğimiz her modülün **dezavantajı** da olacaktır. YSA dezavantajı olarak; *eğitim süresini, eğitimde meydana gelen hataları* ve *başlangıçta iyi bir YSA modeli hesaplaması yapılmaması* gibi olumsuzlukları da göz ardı etmemek gerekmektedir.

YSA yapısını incelediğimizde temel yapı taşının tıpkı biyolojik sinir ağlarında olduğu gibi **sinir hücreleri** olduğundan söz edebiliriz. YSA yapısında bu sinir hücrelerini **nöron** olarak adlandırmaktayız. Bu nöronların temelde beş bölümü bulunmaktadır. Bu bölümler aşağıdaki gibidir;

- **Ağırlıklar**
- **Girdiler**
- **Birleşim Yerleri** (*fonksiyonlar*)
- **İşlem Yerleri**

Sonuç olarak **çıktılar** şeklindedir.

YSA modellerini yapılarına göre inceleyebileceğimiz gibi **algoritma** ve **öğrenme zamanına göre** de incelemek mümkündür. Aşağıdaki tabloda YSA modellerinin *kategorileştirilmiş hâlini* inceleyebilirsiniz.

Yapay Sinir Ağları Modelleri	
Yapılarına Göre	İleri Beslemeli (next propagation)
	Geriye Beslemeli (back propagation)
Eğitim Algoritmasına Göre	Danışmanlı
	Danışmansız
	Destekleyici
Eğitim Zamanına Göre	Dinamik
	Statik

İleri Beslemeli YSA Modeli: Katmanlar üzerinden veri iletiminin söz konusu olduğu bir algoritma düşünelim. Bu algoritmada nöronlar arasında bir **sıra hiyerarşisi** bulunmaktadır. Belirli bir katmandaki nöron yalnızca kendinsen bir sonraki katmana veri iletir ve sistem bu şekilde **sürekli ileri besleme** yaparak çalışmaktadır. Bu modelde nöron gerideki veya üzerindeki katmana **kesinlikle veri iletiminde bulunamaz**. Bunu en kısa yol algoritmalarında kendi üzerinden geçemez veya geldiği yerden geri dönemez kurallarındaki gibi düşünebiliriz.

Geri Beslemeli YSA Modeli: Bu modelde ileri beslemeli modelden farklı olarak nöron **kendisinden önceki** katmana, **sonraki katmana** ve hatta **üzerindeki kendi katmanına** bile veri iletimi yapabilir. YSA modellerinde sıklıkla kullanılan öğrenme algoritmalarından bir tanesidir. Her bir iterasyon sonrasında çıkış katmanındaki hatalar hesaplanmakta ve bu hatalar geriye doğru, yani çıkış katmanından **giriş katmanına doğru** tüm nöronlara iletilerek ağırlıkların hata paylarına göre bu şekilde yeniden hesaplanması sağlanmış olur. Bu algoritmada hata payının hesaplandığı formül yandaki gibidir. $\partial=(z_k-y_k)*f'(y_k\ NET)$

Danışmanlı Model: Yapay sinir ağında algoritmaya verilen her bir giriş verisi için aynı şekilde **çıkış verisi de verilmektedir**.

Danışmansız Model: Algoritmaya giriş için verilen giriş verisi ile birlikte **çıkış verisi verilmeyen** modeldir. Model burada kendisi birbirine benzeyen çıkışları gruplamaktadır.

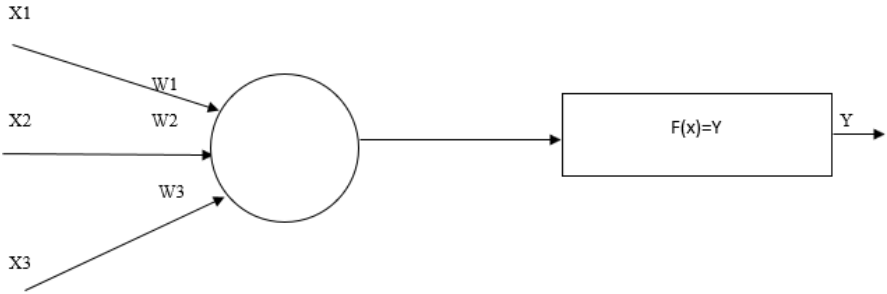
Destekleyici Model: Algoritmada giriş verileriyle beraber ayrıca bir de çıkışlarına **token**'lar verilir. Puan gibi düşüneceğimiz bu token'lardan en yüksek değere ulaşacak şekilde **bir düzenlemeye girerek destekleyen model**dir.

Statik Model: Yapay sinir ağı önceden de bahsettiğimiz şekilde **eğitim verileri ile eğitilmektedir**. Eğitim verisiyle eğitilen yapay sinir ağının yapısı kaydedilir ve algoritma kaydedildiği şekilde çalışmaya devam eder. Herhangi bir **değişikliğe ise uğramaz**. İlk çalışmalar arasında yer alan yapay zekâ ile çalışan sabit suçlara göre cezalar mahkemesi örneklerinde statik modelleri çok görmekteyiz.

Dinamik Model: Bu modelde ise tahmin edeceğimiz üzere yapay sinir ağı bir kez eğitildikten sonra kullanım esnasında sürekli bir **iyileştirme benimser** ve **düzenlemeler devam edebilir**. Öğrenen ve sürekli kendini geliştiren YSA modellerinden burada söz edebiliriz.

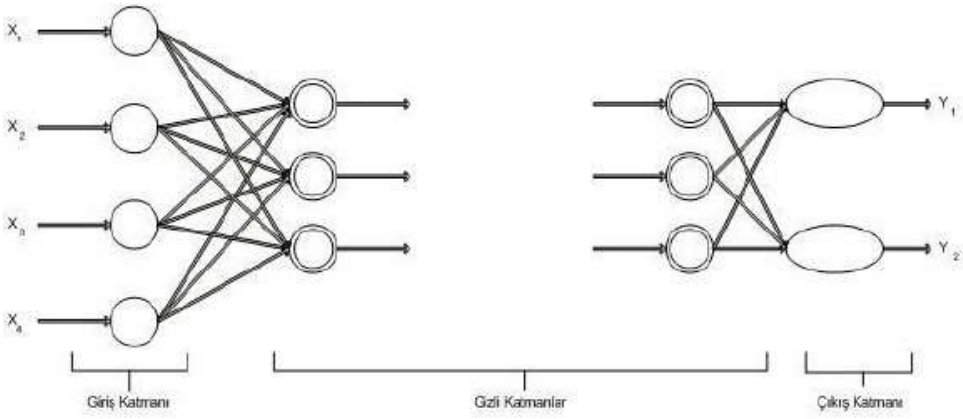
YSA Çalışma Prensibi

Yapay sinir ağlarındaki nöronlar kendilerine girdi olarak gelen verilere ait **ağırlıkları inceler** ve **bu ağırlığa göre bir değerlendirme yaparak çıkış verisini üretir**.

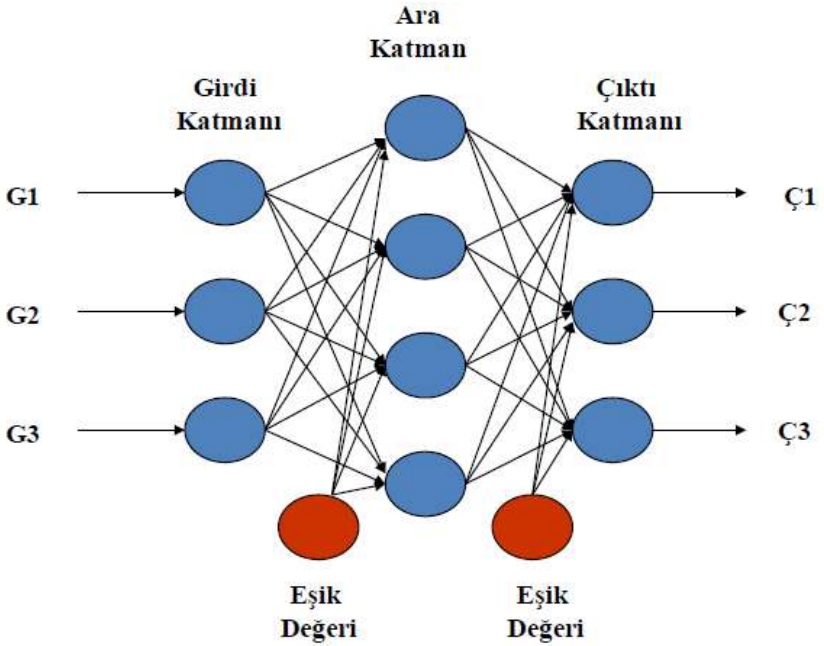


Şekil 15-1: YSA Çalışma Sistemi

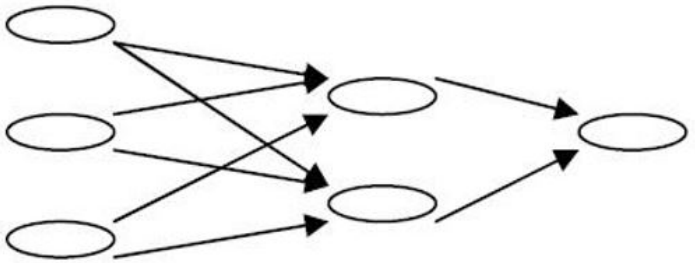
Giriş katmanından alınan verilerin, **doğrudan çıkış katmanına iletilmeden ara bir katmanda işlenerek** çıkış katmanına iletildiği bir diğer çalışma şekli daha vardır. **Şekil 15-1**'de giriş katmanından alınan verilerin çıkış katmanına iletildiğini görmekteyiz. **Şekil 15-2**'de ise ara katmanın devreye girdiğini görüyoruz.



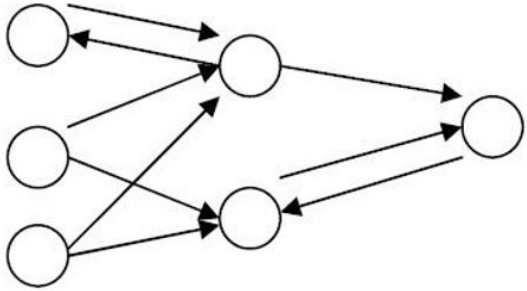
Şekil 15-2: Ara Katmanlı YSA



Şekil 15-3: Çok Katmanlı Örnek YSA Modeli



Şekil 15-4: İleri Beslemeli YSA



Şekil 15-5: Geri Beslemeli YSA

Aşağıdaki tabloda yapay sinir ağı modellerindeki kullanılan **terminoloji** ile **biyolojik sinir ağlarındaki karşılıkları** verilmiştir. Bu tabloyu YSA ile ilgili bilgi veren hemen her dokümanda görebiliriz. İlk bakışta önemsiz gibi gelebilir ancak YSA üzerinde çalışmalar yapmaya başlandığında biyolojik sinir ağlarının yapısını **iyi bir şekilde biliyor olmanın artıları** görülmektedir.

Biyolojik Sinir Ağı	Yapay Sinir Ağı
Nöron	İşlem Elemanı
Dentrit	Girişler
Hücre Gövdesi	Aktivasyon Fonksiyonu
Akson	Çıkış
Sinaps	Ağırlıklar, Bağlantılar

Peki, biyolojik sinir ağlarındaki terimleri de bilmemizin iyi olacağından bahsediyoruz ama bu terimler ne anlama geliyor denildiğini duyar gibiyim. Bu yüzden kısa kısa bizi ilgilendirdiği kadar, medikal projelerde veya ileride karşımıza çıktığında yorum yapabilecek kadar yeterli düşüncesi ile **kısa tanımlar** verelim.

Nöron: Yapay sinir ağının en küçük parçası olan *sinir hücresine karşılık geldiğini* önceden de söylemiştik.

Dentrit: Farklı hücre kanallarından gelen verileri toplayan kollardır. Yani *veri girişidir*.

Sinaps: Hücrelerin çıkış kollarının *diğer giriş kollarıyla* yani dentrit'lerle olan bağlantılarıdır.

Akson: İletimin *tek yönlü* gerçekleştirildi sistemin çıkışıdır.

Makine Öğrenmesi

Uzun ve akılda kalmayan öğrenme yerine ezbere yönelten tanımları bir kenara bırakalım ve makine öğrenmesini çok kolay bir tanımla ele alalım. Bilgisayarın tıpkı YSA'da olduğu gibi **eğitim verilerinden faydalanarak eğitilmesi** ve sonuçta bize bu eğitim verisine dayanarak, kendi mantığını oluşturarak çıktılar vermesine olanak sağlayan çağımız teknolojisine **makine öğrenmesi** diyebiliriz.

Makine öğrenmesi konusunu konuşurken sıklıkla **algoritmalardan** bahsediyor olacağız. İlkokul öğrencilerinin hemen hepsinin aynı zekâ seviyesinde olduğunu ve bir sınıfta toplandığını düşünelim. Bu öğrencileri, öğretmen hangi yönde eğitirse öğrenciler toplumda o yönde çalışmalar yapacaktır. Ve her bir öğretmenin de öğrencilerini farklı yönlerde ama kendi içlerinde öğrencilerin aynı konular üzerine çalışmalar yaptığını düşünürsek her bir öğretmen, öğrencilerini farklı konularda sınıflandırmış olur. İşte makine öğrenmesinde karşımıza sıklıkla çıkan **sınıflandırma algoritması** da aslında bu şekilde çalışıyor diyebiliriz. Algoritma aynı olup farklı şekilde eğitirsek bize farklı sonuçlar yorumlayacaktır. Bu algoritmayı çiçek yapraklarına göre üreme şekillerini veren çalışmayla eğitirsek bu şekilde bir sınıflandırma yapacak veya araç karbon salınımına göre verileri işleyip ilgili tekniğe uygun bir şekilde eğitirsek karbon salınımına göre araçları *dizel*, *benzinli* vb. sınıflandıracaktır.

Şimdi, biraz daha büyük bir pencereden bakalım ve dünya üzerindeki herhangi bir probleme yönelik geliştirilmiş algoritmaların ve problemlerin bir yerde olduğunu düşünelim. Tüm bu algoritmaların tek bir yerden yönetilmesi farklı farklı algoritmaların ortak bir amaç için kullanılması işinde **makine öğrenmesinden** bahsedebiliriz.

Makine öğrenmesi ikiye ayrılır. Bunlara **Gözetimli Öğrenme** ve **Gözetimsiz Öğrenme** denilmektedir. **Gözetimli öğrenmede**; tahmini yapılacak olan veri grubu ile ilgili tüm parametreleri **sisteme öğretiriz** ve ondan bir sonraki işlemde **kendisinin tahmin etmesini bekleriz**. Yapılan her bir gözlem için bir **sınıflandırma yapılır** ve **bu sınıflar ayrıktır**. Ayrıca her bir gözlemde öğrenilenlere bakılarak **gerçek tahminler yürütme mekanizması** çalışmaktadır. Buna da **regresyon** denilmektedir. **Gözetimsiz öğrenmede** ise; veri grubu ile ilgili parametrelerden bir ya da birkaçını biz de **bilmeden sisteme öğretiriz** ve yine de **sistemin tahmin de bulunmasını bekleriz**.

Gözlemlerin etiketlenmediği bunun yerine algoritmanın kendisinin keşifler yaparak tahmin sürecine gittiği gözetimsiz öğrenmede gözlemler **homojen** olarak kümelenirler ve bu şekilde gruplandırılırlar. Bununla birlikte daha doğru tahmin sonucuna ulaşabilmek için kısaltma yani **reduction** yapılarak gözlemlerdeki özelliklerde **özetlemeye gidilir**.

Daha somut bir örnek vermek gerekirse; **Gözetimli Öğrenme** modelinde bir yemek yapmak için tüm yemeklerin içindeki malzemeleri tek tek sisteme öğretiriz ve başka bir yemek yapılmak istendiğinde ne tür malzemeler girilmesi gerektiği tahminini sistemden alırız. **Gözetimsiz Öğrenme** modelinde ise yemeklerin içindeki malzemeleri eksik gireriz ama yine de bir yemek yapılacağından eksik öğrenme olsa bile yemeği yapabilmemiz için gerekli olan içerikleri bize vermesini bekleriz. Gözetimsiz öğrenme modelinde, **belirsiz olan örüntülerin keşfedilmesi** beklenir.

Makine öğrenmesi çalışmalarında içerisinde algoritmaları **hazır bir şekilde barındıran uygulamalar** kullanılabilir. (*KİWİ gibi.*) Ayrıca MATLAB üzerindeki makine öğrenmesi araçları ile de çalışmalar yapılabilmektedir. Bu gibi uygulamalarda *veriyi programa verir, doğru sınıflandırma algoritmasını seçer ve sonucu görmek üzere çalıştırırız.*

Makine öğrenmesi ile ilgili olarak kafamızda yer edinecek kadar bilgi sahibi olduğumuza göre işin **terminolojisine** de hâkim olmak adına aşağıdaki tabloyu incelemek yerinde olacaktır.

Gözlemler (Observations)	Öğrenmek için kullanılan her türlü veri türü. (ses, yazı, koku vb.)
Özellikler (Features)	Öğrenilen veri türünün ifade biçimi (tonu, rengi, uzunluğu vb.)
Etiketler (Labels)	Öğrenilen veri türü için yapılan işaretleme (canlı, cansız)
Eğitim Verileri (Training Data)	Öğrenme gerçekleşsin diye verilen veri dizisi
Test Verileri (Test Data)	Öğrenilen verilerden ayrılarak tutulan tahmin için kullanılan verilerdir.

Makine öğrenmesinde iyi bir sonuç çıkarabilmek için **çalışma yapısını aşağıdaki sıralamada olduğu gibi kurgulamayız**.

1. Yapılan öğrenme işleminden özellik çıkarımı ve çıkarılan özelliklerden tahminde **yüksek değere sahip olacak özelliğin** seçilmesi gerekmektedir.

   Örneğin: trafik kazaların nedenlerinin araştırıldığı ham bir öğrenme verisinde kazazedelere evlerinin nerede olduğu sorusundan ziyade araçlarının markası, yolun durumu gibi daha çok kazayla ilgili sorular sorulmalıdır ve bu alakalı sorulardan da öğrenmede kullanılacak bir tanesi seçilmelidir.

2. Yukarıdaki örnekte olduğu gibi özellik çıkarımı yapıldıktan sonra trafik kazalarına **sebep olduğu düşünülen kriterlere göre** bir algoritma modeli oluşturulmalıdır.

   Örneğin: kaza olasılığı = araç yaşı > 20 && sürücü>65

3. Doğru özellik çıkarımı ve doğru modele ulaştıktan sonra öğretimdeki verilerden test verileri ile ilk iki maddede benimsediğimiz yöntemlerle veri **test edilmelidir**. Bu test sonucunda da verimizi ve modelimizi değerlendirmiş olmaktayız.

   Mühendisliğin hangi disiplininde olursa olsun bir değerlendirme yapılıyorsa burada da değerlendirmenin *belirliliğine*, *hassaslığına* ve *doğruluk* göstergesine dikkat edilmesi gerekmektedir.

4. Doğruluk, hassaslık ve belirlilik gibi kriterlerden de geçer not alan bir değerlendirme yaptıysak artık bu öğrenme işleminde **iyileştirme çalışmaları** yapabiliriz.

   Nasıl ki ham veriler üzerinden incelediğimiz LOG kayıtlarını en anlamlı ve işe yarar hâle getirebilmek için parse ediyorsak burada da **modelimizi iyileştirmeliyiz**.

5. İyileştirme esnasında fark edilen bir olumsuzluk, çalışmayan algoritma, değerlendirme gibi proses varsa **geriye dönüş işlemleri tekrar edebiliriz**.

6. Artık modeli toparlayabiliriz. Ham veriden özellikleri çıkartıp bu özellikler arasından seçim yapıp daha sonra test verisi ile süreci başlatıp gerekirse iyileştirme çalışmalarını da yaptıysak artık elimizde **test edilmiş ve iyi yönde tahminler veren bir algoritma var demektir**.

Makine öğrenmesinde bu altı maddede yer aldığı gibi süreç baştan sonra bu şekilde yönetilirse **yakın tahminler veren bir algoritmaya sahip olmak kaçınılmaz olacaktır**. Makine öğrenmesi ile ilgili güncel projeleri incelemek sizlere neler yapılabileceği konusunda **yol gösterici** olacaktır. Makine öğrenmesi diğer konularımızda da olduğu gibi **hazır bir sistem değildir**. Yani lazım olunca kullanılacak hazır bir yapısı yoktur. Öncesinde veri toplanmalı, yukarıda da bahsettiğimiz gibi çalışma modeli çıkartılmalı, uygun bir algoritma belirlenmeli ve üzerine çalışmalar yapılmalıdır. Günümüzde güvenlik duvarı cihazlarının içerisinde yazılımlarda artık **makine öğrenmesi** modüllerini görebilmekteyiz. Benzer şekilde nesnelerin interneti olarak adlandırılan **IoT** teknolojileri ile çalışan sistemlerde de makine öğrenmesi çalışmalarına rastlıyoruz. Örnek projeler yapmak üzere hazır veri setlerinden faydalanmak isteyenler için **Machine Learning Repository** olan *https://archive.ics.uci.edu/ml/datasets.php* web sitesini örnek olarak verebilirim.

Derin Öğrenme

Aslında her biri çok geniş araştırma konusu olan **yapay sinir ağları**, makine öğrenmesi ve derin öğrenme konularını iyi kavramak için konularına uygun kütüphaneler temin ederek araştırma örnekleri yapılması gerekmektedir. Kitabımızın bu bölümünde uygulama üzerinden anlatım yapma şansımız olmadığı için ve aynı zamanda kitabımızın bütününde bilgisayar mühendisliği konularının neler olduğuna değinmek ve gerektiği kadar örnekleme yapma akışına uyabilmek için bu bölümde de biraz **özet niteliğinde** ama konuyu kavramaya da yardımcı olacak nitelikte bir anlatım tarzını benimsiyor olacağız.

Çağımızda hâlen yirmi, otuz yıl öncesinin konularının bilgisayar derslerinde gündem olduğunu hepimiz bilmekteyiz. Hem ilgili konunun çıkış noktasına atıfta bulunmak hem de geçmişten günümüzde ne kadar gelişim gösterdiğini kavramak ve bundan sonraki süreçte çalışmaların hangi yöne doğru ilerleyeceğini bu sayede kestirebilmekteyiz. **Öğrenmeyi öğrenmek** isimli çalışma mekanizmamız da bu bölümü, bir projeyle örneklendirmek yerine **temelini atmayı** ve diğer bölümlerimizde de olduğu gibi ihtiyaç anında **bize yetecek kadar bilgiyi alıp** projelerimize uygulamalar üzerinden devam edebileceğimiz bir şekilde devam edecektir. Bu yüzden makine öğrenmesi, derin öğrenme veya YSA için kitabımızda örneklere uygulama olarak değil de **sözel olarak** değinmiş olmamızın bir kez daha altını çizmek istiyorum.

Kitabımızın proje kitabı değil, mühendislik disiplinindeki güncel konuları ele alan ve bu sayede bir aday öğrenciyi veya bu işe gönül vermiş aday araştırmacıyı **nelerin beklediğine ışık tutmak**, sektördeki araştırmacılara veya mevcut öğrencilere ve mühendislere gerektiğinde **rehberlik edecek bir el kitabı** olması yönünde ilerleyeceğini belirtmek isterim.

Peki, nedir bu derin öğrenme? **Deep Learning** olarak karşımıza çıkan bu disiplin, aslında bir **makine öğrenme tekniği**dir. Bu sefer üzerinde çalışılan veri kümesindeki çıktıları tahmin edebilmek üzere **yapay zekânın eğitilmesini sağlayan bir öğrenme modeli**dir. Derin öğrenme uygulamalarını kendi laboratuvar ortamımızda geliştirmek için **Pyhcharm/Python** platformu kullanabilirsiniz. **Pyhcharm**, Visual Studio benzeri bir arayüz sunacak ve hataların görsel olarak yakalanabileceği, çıktıların grafiksel olarak sunulabileceği bir ortam hazırlar.

Bir yazılım firmasında **full stack** uygulama geliştirenler ya da bir kuruluşta network sistem yöneten kimseler için derin öğrenme pek bir şey ifade etmeyebilir. Derin öğrenme konusu genellikle **akademik zümrelerde adından söz ettirir** ve her ne kadar bilgisayar mühendisliği veya veri bilimler için gündemde olsa da aslında veriyi sınıflandırmak isteyen her bilim dalında **yer edinmesi gereken** bir modeldir. Örneğin bir ziraat mühendisinin de bitkiler ile ilgili sınıflandırma ve tahmine ihtiyacı olduğu projelerde derin öğrenmeden bahsedebiliriz. Benzer bir şekilde YSA konusunda da örneklendirdiğimiz gibi hava tahmini yaparken de derin öğrenmeden söz edebiliriz. Bu tür çalışmaları yürüten veri bilimciler veriyi de biraz açacak olursak; *ses tanıma, video analizi, konuşma, görüntü sınıflandırma* gibi uygulamalarda derin öğrenme modelinden faydalanırlar.

Görsel çalışmalar üzerinde sınıflandırma ve analiz işlemleri yapılması gerektiğinde makine öğrenmesinde **grafik işlemcilerine oldukça büyük iş yükü düşmektedir**. GPU olarak ilk bölümlerde de tanıdığımız grafik işlemci ünitesi çok büyük miktardaki veri sınıflandırma işlemlerinde eğitim ve test verilerini saptarken **derin sinir ağlarının kullanımı sürecinde çok çalışırlar**.

Tanım olarak derin öğrenme, **elinde bulunan veri setinden yararlanarak sonuçları tahmin eden makine öğrenmesi yöntemidir**. Katmanlı bir mimari olan derin öğrenme, **makine öğrenmesinin alt katmanı**, makine öğrenmesi ise **yapay zekânın alt katmanı** olarak ifade edilir.



Şekil 15-6: Derin Öğrenme Mimarisi

Günümüzde derin öğrenme uygulamalarını en çok **yüz tanıma** sistemlerinde görmekteyiz. Bunun dışında biyomedikal mühendisliği alanının yaptığı çalışmalarda örneğin; kanserli hücre tespitinde ve tanı koyma aşamalarında derin öğrenme algoritmalarına sıklıkla rastlanmaktadır.

Bilgisayar sistemleri, donanım birimlerinde nasıl geçmişten günümüze sürekli bir **optimizasyonu** benimsemiş ve bu yüzden **iyileşmeye gitmişse** derin öğrenme algoritmaları da aynı şekilde **sürekli iyileştirmeyi hedefler** ve **doğru tahmin sonuçları üretebilmek için algoritmalar sürekli geliştirilir**. Öğrenci projelerinde yüz tanıma, el yazısı tanıma veya dilden ırk tanıma gibi öğretilerin sonuçlarını veren projeler olduğu gibi tarım ve sağlık sektöründe de çeşitli uygulamaları olduğunu söylemiştik. *Peki, ama endüstri 4.0 ile birlikte derin öğrenme neden yerini makinelerde de almasın?* Elbette bu tarafta da oldukça büyük gelişmeler mevcut ve çalışmalar hızla devam etmektedir. Örneğin bir **network switch** cihazının veya bir **firewall** hatta bir **waf** cihazının bile modülleri arasında artık **derin öğrenme algoritmaları ile çalışan makine öğretilerine rastlanmaktadır**. Network üzerinde bir arıza meydana geldiğinde veya topolojide bir aksaklık oluştuğunda geçmiş öğretileriyle belki de bizlere yeni kuralların ne şekilde yazılması gerektiği hakkında tahminler verecek olan uygulamalara cihazların içindeki **derin öğrenme** ve **makine öğrenmesi** modüllerinde rastlayabiliriz.

Uygulama yapmak isteyenler için **veri bulmakla uğraşmamak adına** veri seti sunan web siteleri bulunmaktadır. Buradaki veri setlerinden kendinize uygun olan bir tanesi ile gerek Pyhcharm ile gerekse MATLAB üzerindeki yapay zekâ araçlarını kullanarak projeler üretebilirsiniz. Bu web sitelerinden bir tanesine örnek olarak *www.kaggle.com/datasets* linkini verebilirim.

16 DESIGN PATTERNS ANTI-PATTERNS VE SECURE DESIGN PATTERNS KAVRAMLARI

Kitabımızın bu bölümü **yazılımcılara** hitap ediyor dersek yeridir. Bu bölümde bir yazılım geliştirirken benimsenmesi gereken en iyi yöntemler (*design patterns*) ve benimsendiğinde **sorun çıkaracak yöntemler** (*anti-patterns*) ele alınmıştır. Ayrıca bir yazılım geliştirme yaşam döngüsü içerisindeki güvenlik hususları göz önüne alınarak bunun içinde benimsenecek olan yöntemlere (*secure design patterns*) değinilmiştir.

Teknolojinin gelişim hızı, ihtiyaçların çeşitlenmesi ve yazılım geliştiren mühendislerin yeniliklere kolay adaptasyonları sayesinde yazılım geliştirme sektöründe her dönem yeni bir kavram ön plana çıkmıştır. Günümüzün önemli konularından bir tanesi ise yazılım geliştirme yaşam döngüsündeki uyulması gereken **tasarım şablonları** ve **güvenliktir**. Büyük sistemlerde açığa çıkan güvenlik zafiyetleri sebebiyle yazılım mühendisliğinde güvenlik konusu önemli bir araştırma alanı hâline gelmiştir. Bilgi teknolojilerinin kullanıcıya sunmuş olduğu yapılabilirlik ilkesi sebebiyle ihtiyaçların artması ve günümüzde problemlerin çoğunun programlama ile çözüme kavuşuyor olmasından dolayı modern işletmelerin çok büyük miktardaki kurumsal hafıza ve diğer gizli verileri **bilgi sistemlerinde depolanmaktadır**. Özellikle kişisel veriler başta olmak üzere her türlü verinin **gizlilik esası bulunur** ve bilginin değerinin güvence altına alınması için yetkisiz her türlü erişime kapalı olması ve izinsiz değişikliklere karşı da korumalı olması gerekmektedir.

Şirketlerin, eğitim kurumlarının, devletlerin ve son kullanıcıların kullanmış olduğu her türlü yazılımda **çeşitli veriler** tutulmaktadır. Bu verilerin kaybolması veya değiştirilmesi oldukça büyük bir risktir ve **mali kayıplara** neden olabilmektedir. Verilerin veya sistemlerin korunması için bilgi teknolojileri yöneticileri, sisteme kullanıcıların nasıl erişmeleri gerektiğini rol tabanlı olarak veya zaman kısıtlaması ile belirleyerek **çeşitli güvenlik politikaları** oluştururlar. Fakat bu politika veya dijital poliçeler bilgi sistemleri yöneticileri tarafından kullanışlı olmayan gereklilikler şekline dönüşürse güvenlik için alınan önlemleri içeren politikalar doğru uygulanmamış olacaktır. Mevcut sistemleri güvenlik ilkeleri ve kanunlarda yer alan **uygulamalara göre yönetmek**, olması gereken bilgi sistemleri yönetim şekli olarak kabul edilebilir. Fakat sistemleri bu şekilde yönetmekten çok daha önce yapılması gereken bilgi sistemlerinin geliştirilmesi esnasında güvenlik tasarım desenleri ve diğer tasarım desenlerinin benimsenmesi ve yazılım geliştirmede uygulanmasıdır. Güvenlik ile ilgili ortaya çıkan sorunları çözmek için yapı taşlarından bir tanesi yazılım mühendisliğindeki **tasarım desenleri** (*design patterns*)'dir. Bu desenler, güvenliği veya işlemleri tekrar belirleyen **genel çözümleri** içermektedir. Ek olarak hâlihazırda kullanılmakta olan yazılımların uygulama ve dağıtım aşamalarında güvenlik sorunlarının tespit edilmesi ve giderilmesi ile ilgili pek çok araç vardır. Ama güvenlik konusunun üstünde özellikle durarak anlatmaya çalıştığım **tasarım desenleri** konusunda, güvenliğin sağlanabilmesi için geliştirme esnasında tasarım desenleri gibi mimarileri benimsemek çok daha faydalı ve **best practise** bir durumdur. Güvenlik kusurlarının ve yazılım iş akışındaki sürekliliğin temel nedenlerinin henüz geliştirme aşamasında iyi bir şekilde anlaşılması, yazılım geliştirmenin tüm aşamalarında bu mimarilerin göz önünde bulundurulması ile geliştiricilere farkındalık da kazandıracaktır.

Yazılım geliştirmede tasarım desenlerinin uygulanması, problemlerin ele alınabilir **alt problemlere** dönüştürülebilmesini sağlamaktadır. Türkiye Cumhuriyeti, Cumhurbaşkanlığı Resmî Gazetesinde 2019 Temmuz ayında *Bilgi ve İletişim Güvenliği Tedbirleri* konu başlığı ile yayımlanan Genelde de kamu kurum ve kuruluşlarının **bilgi ve iletişim güvenliği konusunda uymaları gereken birtakım zorunluluklar** açıklanmıştır. Genelgede yer alan konulardan yazılım geliştirme sürecini ilgilendiren maddeler incelendiğinde yazılımların zararlı kimselerin yapacakları saldırılara karşı güvenli olması gerektiği ve işletim sistemi bazlı kullanıcı ilkelerinin belirli erişim protokolleri ile sağlanması gerektiğinden bahsedilir.

Tasarım deseni kavramındaki **tasarım**, bir sorun için belirlenmiş olan genel bir çözümdür ve yeniden kullanılabilir bir yapıdadır. Tasarım kavramı, algoritma veya doğrudan koda dönüşebilen **bitmiş bir tasarım değildir**. Algoritmalar, hesaplama problemlerine odaklanır ve onları çözerler. Tasarımlar ise pek çok farklı durumda kullanılabilecek bir problemin **nasıl çözüleceği ile ilgili açıklama niteliğindeki şablonları** ifade eder.

Design Patterns

Tasarım desenleri olarak da dilimizde karşılık bulan **design patterns** kavramını "**tasarım şablonları**" veya "**tasarım örüntüleri**" şeklinde de görebiliriz. Şablon şablon veya örüntü denildiği zaman bizlere ifade edeceği yaklaşım aynı olmalı, grafik tasarımı değil **yazılım mimarilerindeki şablonlar** olarak yorumlamalıyız. Tasarım kelimesini elbette **grafiksel bir tasarım olarak nitelendirmeyeceğiz**. İşin mimari tasarımıyla ilgili birtakım standart veya benimsenmesi gereken **kod yazma yaklaşımı** olarak içselleştirmemiz gerekiyor.

Tasarım Desenleri: Yazılım geliştirmede düzenli olarak ortaya çıkan sorunlara karşı geliştirilmiş çözümler bütünüdür. Yazılım geliştirme süreci için kolayca uygulanabilir ve zamandan tasarruf edilmesine olanak sağlayan stratejiler sunmaları amaçlanmıştır. Düzgün bir şekilde kurgulanmış ve tanımlanmış bir tasarım desenine eşlik eden bir dokümantasyon ile geliştiriciler verilen problemi **hızlı bir şekilde belirleyebilir** ve **uygulayabilirler**.

İlk tasarım deseni notasyonu, Christopher Alexander tarafından **uzay** ve **yapı** mimarisi için **yeniden kullanılabilir stratejiler** konusundaki çalışmalarında geliştirilmiştir. Beck ve Cunningham her desenin çevremizde **tekrar tekrar ortaya çıkan bir problemi anlattığına** değinmişlerdir. 1987 yılında Beck ve Cunningham Alexander'ın desen fikrini, nesne yönelimli bir programlama dili olan **Smalltalk**'taki grafiksel kullanıcı arayüzlerinin geliştirilmesinde uygulamışlardır. Cunningham ve Beck, bir Smalltalk kullanıcı arayüzü geliştirmek için çeşitli modeller geliştirerek elde etmiş oldukları 23 farklı sonucuyu **OOPSLA-87** atölyesinde yayınlamışlardır. 1990-1992 yılları arasında ise Gamma ve arkadaşları dörtlü çete olarak bilinen **GoF desen kataloğunu** derlemişlerdir. Bu modeller daha sonra **Gang of the Four (GoF)** kitabı olarak bilinen yazılım tasarım kalıplarının **ilk son derece sezgisel kitabını** yayınladılar. Bu kitabı, günümüzde yaklaşık 2.000 kalıptan oluşan Grady Booch'un yazılım mimarisi el kitabı gibi birçok desen koleksiyonunun yayınlanması izlemiştir.

Her biri birbirinden farklı sorunları çözmeye yönelik geliştirilen **tasarım şablonlarını** (*design patterns*), türlerine göre aşağıdaki gibi ele alabiliriz.

1) Creational Patterns

Yazılım geliştirme yaşam döngüsünde kullandığımız tüm nesnelerin oluşumu ile ilgili düzenleri ortaya koyan tasarım şablonlarıdır. **Factory Method**, **Singleton**, **Abstract Factory**, **Builder** gibi çeşitleri bulunur.

2) Structural Patterns

Adından da anlaşılacağı üzere yapısal olarak nesnelerin birbirleri ile olan ilişkilerini ortaya koyan tasarım şablonudur. **Adapter**, **Decoder**, **Facade**, **Bridge**, **Composite**, **Flyweight** ve **Proxy** gibi çeşitleri bulunur.

3) Behavioral Patterns

Nesneleri davranışsal olarak takip eden tasarım şablonlarıdır. **Observer**, **Chain of Responsibility**, **Command, State**, **Strategy**, **Template**, **Visitor**, **Iterator**, **Mediator** en yaygın türleridir.



Şekil 16-1: Design Patterns

Anti-Patterns

Anti pattern kavramı ise bir sorunun çözümünde kullanılan **yanlış çözümler** veya **faydasız** çözümlerdir. Yani yanlış geliştirilmiş çözümlere **anti pattern** denilmektedir. Anti-Pattern'ler geliştirilirken yapılan hataların ilk başta fark edilememesinden dolayı sorunlar projenin **ileri aşamalarında** veya **kullanımında** ortaya çıkmasına neden olmaktadır. Genellikle ortaya çıkma serüveni geliştirici ekibin tasarım desenleri üzerine yeni yeni tecrübe ediniyor olması ve ortaya tasarım desenlerine ait bir şeyler koyma çabasıdır.

En yaygın anti-pattern'ler aşağıdaki gibidir.

- Golden Hammer
- God Object
- Hard Coding
- Smoke and Mirrors
- Analysis Paralysis
- Overengineering
- Boat Anchor

Golden Hammer: Bir çözümün farklı birçok çözüme karşın uygulanmaya çalışmasıdır.

God Object: Çok fazla alt kapsayıcısı olan nesnelerdir.

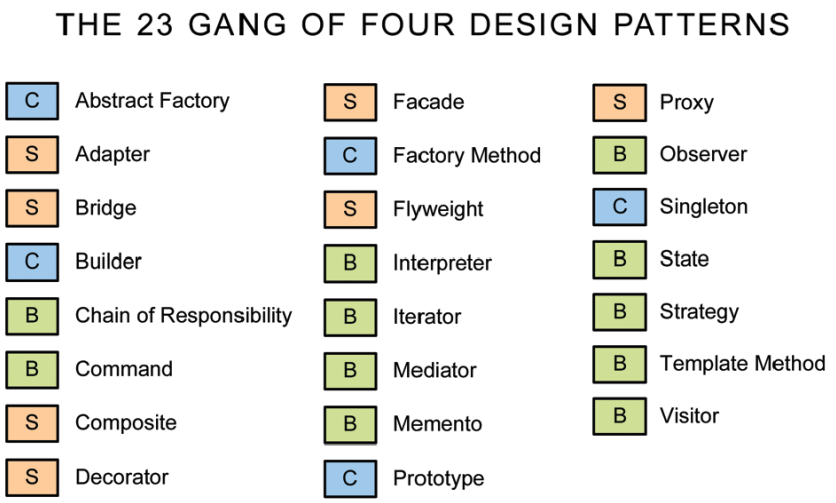
Hard Coding: Katmanlı bir mimari yerine kod içerisinde çok fazla tanımlama ve değişken kullanılarak yapılan geliştirmeye verilen isimdir. Spagetti kodlama gibi düşünülebilir.

Smoke and Mirrors: Yanıltıcı araçların kullanılması durumudur.

Analysis Paralysis: Proje analiz edilmesi sürecinde çok zaman harcanması genellikle tecrübesiz geliştiricilerde rastlanır.

Overengineering: Karşılaşılan problemi mevcut hâlinden çok daha karmaşık algılamak.

Boat Anchor: Proje süreçlerine dâhil olmayan ancak yine de tutulan kodlar.



Şekil 16-2: Anti-Patterns

Secure Design Patterns

Yazılımda güvenliği, bir kurumun verilerinin istenmeyen erişime veya değiştirilmeye karşı **korunması** olarak ifade edebiliriz. Barbacci isimli araştırmacı yazılım mühendisliği enstitüsünde belirtmiş olduğu gibi güvenlik, kaçak kullanıcı girdisinden hassas bilgilere ve yetkisiz erişime karşı korumaya kadar değişen uygulamaları içerir.

Güvenlik geleneksel olarak bir sistemin nasıl çalışması gerektiğine dair **işlevsel olmayan** gereksinimlerden biridir. Güvenlik ne yazık ki genellikle tasarım, yani uygulama geliştirme sürecinin sonlarında veya dağıtım aşamasından sonra düşünülür. Bu bir kurumun verilerini risk altına sokma açısından bakımı daha zor ve daha az güvenli bir sistem olmasına yol açar. İşlevsel olmayan gereklilikler çoğu zaman sistem geliştirilirken belirtilir. Çünkü gereksinimleri standart olarak formüle etmek zordur. Burada bahsedilen güvenlikle ilgili işlevsel olmayan gereksinim, (**İOG**)'lerin yönetilmesi stratejisi ilk önce güvenliği daha spesifik İOG'lerin hiyerarşisine indirgemektedir. Daha sonra ayrıştırılmış gereklilikleri, güvenlik gelişimi için en iyi uygulamalar kataloğu ile ilişkilendirmektedir. İOG güvenliğinin ayrışması, güvenliği seviyeler şeklinde özelliklere ayırmak için kullanılmıştır.

1. **Gizlilik**: Bir işletmenin hassas verilerinin yetkisiz erişime karşı korunmasıdır.
2. **Bütünlük**: Kuruluşun verilerinin yetkisi değiştirilmeden korunmasıdır.
3. **Erişebilirlik**: Bir kuruluşun verilerinin yetkili kullanıcılar tarafından kullanılmamasına karşı korunmasıdır.
4. **Hesap Verilebilirlik**: İşletmenin verilerini etkileyen eylemleri gerçekleştiren kişi veya kişilerle olan birliğidir.

Gizlilik ile ilgili gereklilikler; bir sistemin nasıl tasarlandığını, erişimin nasıl kontrol edildiğini ve süreçlerin nasıl kontrol edildiği ile kullanıcıların nasıl doğrulandığını düşünmektedir. Gizliliğin ihlali, genellikle kötü niyetli taraflarca tehlikeye atılan verilere **yetkisiz erişim** sağlanmak için istismar edilen sistemlerdeki **hatalara** bağlanabilir. Günümüzde verilerin gizliliği ve güvenliği ile ilgili alınan önlemler, yazılım geliştirmede güvenli tasarım desenlerinin etkin kullanımı ile büyük ölçüde önlem niteliği taşımasının yanı sıra, hükümetlerin artık bilgi güvenliği yönetim sistemi ve veri güvenliği politikaları ile kişisel verilerin korunması kanununun üzerinde yürütmekte oldukları çalışmalar ile önemli bir yer almıştır.

Güvenli tasarım desenleri, yazılım geliştirmede güvenlik açıklarının **yanlışlıkla geliştirilen** projeye eklenmesini ortadan kaldırmak ve olası güvenlik açıklarının sonuçlarının **etkisini azaltmak için** kullanılmalıdır. Güvenli tasarım desenleri, erişim kontrolü gibi güvenlik mekanizmalarını tanımlayan **yazılım tasarım kalıplarıdır**. Güvenli desenler teriminin ilk kullanımı 1997'de **Yoder** ve **Barcalow** tarafından yazılmıştır.

Güvenli tasarım desenleri, güvenlik açıklarını gerçekleştirilen **kodlamaya girmesini önlemek** veya güvenlik açıklarının sonuçlarının **etkisini azaltmak için genel bir tasarım rehberliği** sunar. Güvenli tasarım desenleri, nesne yönelimli tasarım yaklaşımlarıyla sınırlı değildir. Prosedürel programlama dillerine de uygulanabileceğini söyleyebiliriz. Bu modeller güvenli kodlama kılavuzlarından daha yüksek bir **soyutlama** seviyesindedir. Güvenli tasarım desenleri, erişim kontrolü, kimlik doğrulama ve yetkilendirme gibi belirli güvenlik mekanizmalarını tanımlamak yerine güvenli geliştirme süreçlerini tanımlar ve mevcut güvenlik sistemlerinin yapılandırılması hakkında **rehberlik** sağlaması sebebi ile güvenlik modellerinden farklıdır.

Kitabımızın bu bölümünde **Güvenli Tasarım Desenlerini** (*Secure Design Patterns*), soyutlama seviyelerine göre üç başlık altında inceleyeceğiz. Güvenli tasarım desenlerinin yazılım geliştirmeye iyi entegre edilmesi ve hiyerarşik olarak güvenlik ile ilgili olan tasarım desenlerini sınıflandırılarak yöntemlerin geliştirme sürecinde **iyi pekiştirilme** ile ilgili konulara da değineceğiz.

1. Mimari
2. Tasarım
3. Uygulama

Mimari Seviye Güvenli Tasarım Desenleri

Mimari seviye desenler, sistemin farklı bileşenleri arasında üst düzey sorumluluk dağılımına odaklanır ve bu üst seviye bileşenler arasındaki etkileşimi tanımlar. Mimari seviye desenleri:

- **Güvenilmez Ayrışma Tasarım Deseni** (*Distrustful Decomposition Design Pattern*)
- **Ayrıcalık Ayrıştırma Tasarım Deseni** (*Privilege Seperation Design Pattern*)
- **Çekirdeğe Erteleme Tasarım Deseni** (*Defer To Kernel Design Pattern*)
- **Hatalarla Tam Görünüm Tasarım Deseni** (*Full View With Errors Design Pattern*)

a) Güvenilmez Ayrışma Tasarım Deseni

Güvenli tasarım deseninin amacı, ayrı işlevleri **karşılıklı güvenilmeyen programlara** taşımaktır.

Birçok saldırı, yüksek izinlerle çalışan **hassas uygulamaları** hedeflemektedir. Bu durum saldırganın fazla bilgiye erişmesine izin vermekte veya saldırganın uygulamada daha kısıtlayıcı izinlerle çalıştığından daha fazla güvenlik açığı kullandıktan sonra **daha fazla zarar** vermesine neden olmaktadır. Bu tür saldırı sınıfının bazı örnekleri aşağıdaki gibidir.

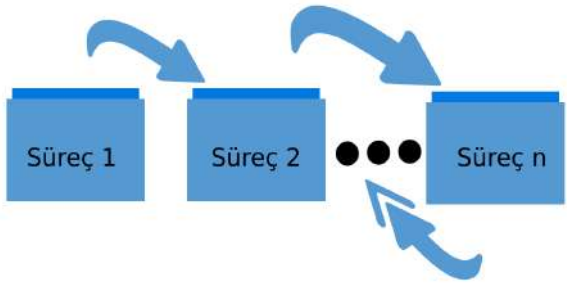
- Internet Explorer'ın yönetici ayrıcalıklarına sahip bir hesapta çalıştırıldığı çeşitli saldırılar
- Saldırganın, rastgele kodu **kök** (*root*) olarak çalıştırmasına izin veren telnet servis dosyalarında arabellek taşması sonucu meydana gelen güvenlik açığı

- Norton Anti Virüs firmasının 2005 yılında saldırganların yönetici ayrıcalıklarıyla çalışırken **keyfi senaryo** (*script*) komut dosyaları çalıştırmalarına izin veren güvenlik açıkları

  Bu saldırıların tümü, UNIX altındaki **kök kullanıcı** veya Windows altındaki yönetici yetkisine yükseltilmiş ayrıcalıklarla çalışan bir uygulamada bulunan **güvenlik kusurlarından** yararlanmaktadır ve ardından bilgisayarda çalışan diğer uygulamaları tehlikeye atmak için uygulamanın yükseltilmiş ayrıcalıklarını ve temel işlevlerini **hassas verilere ulaşmak için** kullanır. Güvenilmez ayrışma tasarım deseni, güvenlik açıklarını sistemin küçük bir kümesine indirgeyerek, sistemin tek bir bileşenini tehlikeye atarak sistemin tümünün tehlikeye girmesini engeller.

  Bu sayede kötü niyetli faaliyetlerde bulunan saldırganların elinde, tüm uygulamanın işlevselliğini ve verilerini değil yalnızca tehlikeye atılmış bileşenin **işlevselliğine** ve **verilerine** sahip olacaktır. Bir sistemin güvenilmez ayrışma tasarım desenini kullanabilmesi sistemin birden fazla üst düzey işlev gerçekleştirebildiği ve çeşitli işlevlerinin farklı ayrıcalık seviyesine sahip olması gerektiği durumlarda **yararlı** olacaktır.

  Güvenilmez ayrışma (*Distrustful Decomposition*) tasarım deseni **ayrıcalık azaltma** (*Privilege Reduction*) olarak da bilinmektedir. Bu modelin genel yapısı, sistemi **her biri potansiyel olarak farklı ayrıcalıklara sahip** olacak şekilde ayrı işlemler olarak çalışan **iki** veya **daha fazla** programa böler. Her işlem, sistem işlevselliğinin küçük ve iyi tanımlanmış olan bir alt kümesini işler. İşlemler arasındaki iletişim, soketler, RPC, SOAP veya paylaşılan dosyalar gibi işlemler arası bir iletişim mekanizması kullanarak gerçekleşir. **Şekil 16-3**'te **güvenilmez ayrışma tasarım** deseninin yapısı gösterilmektedir.



Şekil 16-3: Güvenilmez Ayrışma Tasarım Deseninin Yapısı

Güvenilmez ayrışma, tek bir bileşen programın başarıyla kullanılması durumunda, saldırganın **tüm sistemi tehlikeye atmasını önler**. Çünkü başka bir program, tehlikeye giren programın sonuçlarına güvenmeyecektir.

Uygulamada bu desen, aslında işletim sisteminde zaten bulunan standart işlem/ayrıcalık modelinin ötesinde bir şey kullanmaz. Her program potansiyel olarak farklı kullanıcı ayrıcalıklarına sahip **kendi süreç alanında** çalışır. Farklı programlar arasındaki iletişim, **tek yönlü** veya **iki yönlü** gerçekleşir. Tek yönlü iletişim kontrolünde, kontrolü çalıştırmak için veya süreci oluşturmak için bazı programlama metotları (*fonksiyon, metot*) kullanılabilir veya işletim sistemine özgü başka bir yöntem kullanılır. İki yönlü iletişim kontrolünde **TCP** veya **SOAP** gibi iki yönlü süreçler arası iletişim mekanizması kullanılır.

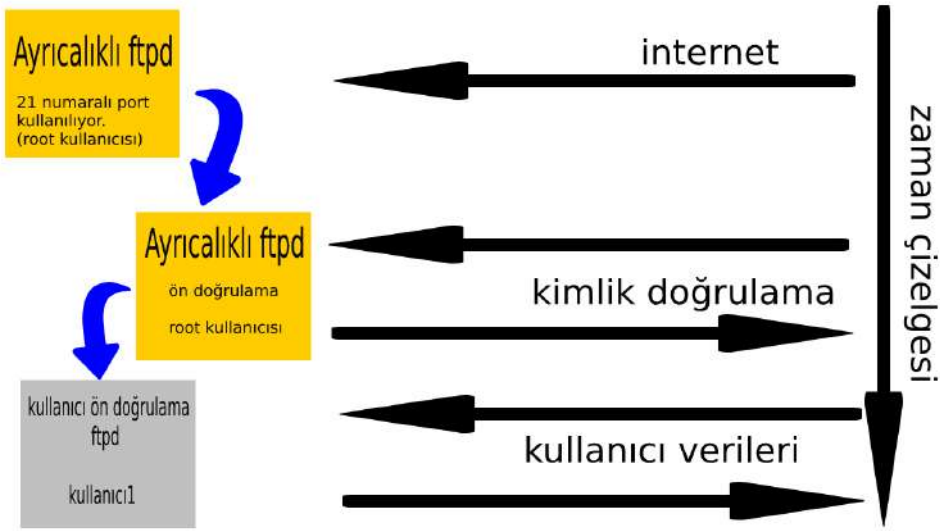
Güvenilmez ayrışma tasarım deseni ile ilgili bilinen uygulamalar aşağıdaki gibidir.

- Qmail posta sistemi
- Postfix posta sistemi
- Microsoft uygulama yönetici ayrıcalıkları
- Microsoft Windows Vista kullanıcı hesabı kontrolü

Bu desenin uygulandığı en iyi örnek kullanıcılar, sistemler ve yazılım bileşenleri arasındaki etkileşimin geniş bir kombinasyonunu içeren karmaşık bir sistem olan **Qmail posta sistemi**dir.

b) Ayrıcalık Ayrıştırma Tasarım Deseni

Bu desenin (*Privilege Separation*) amacı programın işlevselliğini **etkilemeden** veya **sınırlamadan**, özel ayrıcalıklı **kod miktarını azaltmaktır**. Bu desen güvensiz ayrışma deseninin daha *spesifik* bir örneğidir. Birçok uygulamada basit bir işlem kümesi daha yüksek ayrıcalıkları gerektirmektedir. Buna karşın çok daha büyük karmaşık ve güvenlik hatalarına açık işlem kümesi de normal ayrıcalıklı bir kullanıcı bağlamında çalıştırılabilir. **Şekil 16-4**'de ayrıcalık ayrıştırma deseninin uygulanabileceği bir **FTPD** sistemi ve bu desenin kullanılmaması durumunda ortaya çıkabilecek güvenlik sorunlarının ayrıntılı bir görünümü yer almaktadır.

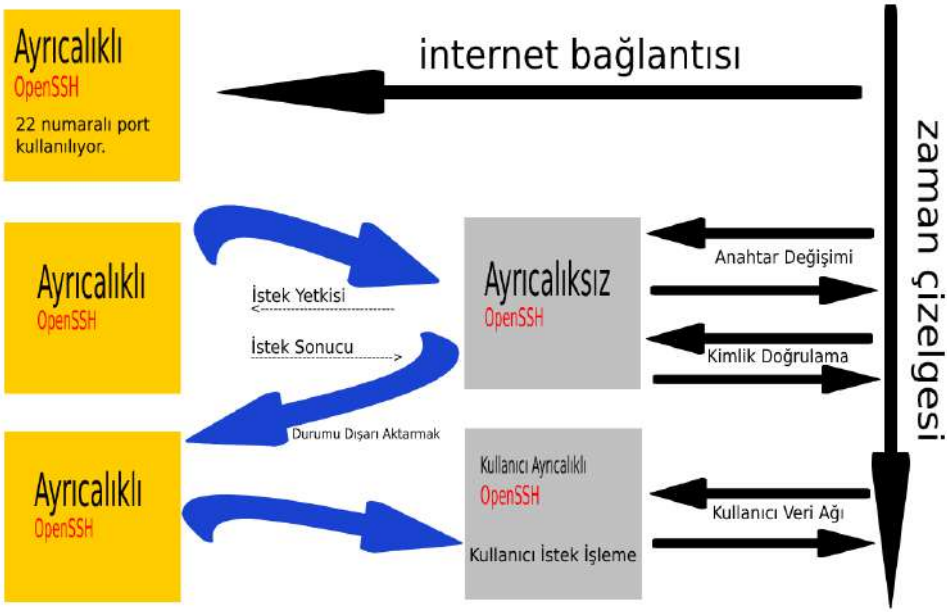


Şekil 16-4: Ayrıcalık Ayrıştırma Deseninin Uygulandığı FTPD Sistemi

Güvenlik açığı ayrıcalıklı sunucusu, henüz güvenilmeyen sistem kullanıcısıyla bir bağlantı kurduğunda ve kullanıcıyı sunucu ile aynı yükseltilmiş **ayrıcalıklara sahip bir yöntem** (*child*) ile **doğrulamaya çalıştığında** ortaya çıkar. Kötü niyetli bir kullanıcı bu noktada ayrıcalıklı yöntem (child) ile güvenlik açıklarından yararlanabilir ve ayrıcalıklı sürecin kontrolünü veya erişimini kontrol altında tutabilir. Bu bağlamda da **Ayrıcalık Ayrıştırma tasarım deseni**, Güvenilmez Ayrışma tasarım deseninin daha spesifik bir örneği hâline gelmiş olur.

Bu tasarım deseni özellikle kullanıcıların kimliğini doğrulaması ve ardından kullanıcıların normal kullanıcı düzeyinde ayrıcalıklarla etkileşimli programlar çalıştırmalarına izin vermesi gereken sistem hizmetleri için özellikle yararlıdır.

Şekil 16-5'de **ayrıcalık ayrıştırma** (*PrivSep*) **tasarım deseninin** yapısını ve davranışını gösterir. Bu şema alt işlemler oluşturmak için UNIX altında (*fork*) işlevine başvuruda bulunmaktadır. Desen Windows işletim sistemlerinde, `Süreci Oluştur ()` (*CreateProcess*) metodu ile alt işlem uygulamak için kullanılır.



Şekil 16-5: Ayrıcalık Ayrıştırma Tasarım Deseni Yapısı

Ayrıcalık ayrıştırma tasarım deseninde ayrıcalıklı sunucu işlemi, *sistem kullanıcısı, ayrıcalıklı müşteri süreci, kullanıcı imtiyazlı istemci süreci* gibi süreçler yer alır. Ayrıcalıklı sunucu işlemi, alt süreçler tarafından ele alınacak olan işlevsellik için **ilk isteklerin yerine getirilmesinden sorumlu**dur.

Sistem kullanıcısı, sistemden bir miktar işlem yapmasını ister ve bu ilk işlevsellik isteği kullanıcı tarafından **ayrıcalıklı sunucuya** yönlendirilir. Bahsedilen kullanıcı elbette yerel veya uzak kullanıcı olabilir.

Ayrıcalıklı müşteri süreci, kullanıcının isteğinin doğrulanmasından sorumludur. Bunun güvenilir bir kullanıcıdan gelen geçerli bir istek olup olmadığı henüz bilinmediğinden alt süreçler işleme **kimlik doğrulamasının ayrıcalıkları** ile sınırlandırılır.

Kullanıcı imtiyazlı istemci süreci, sistem kullanıcısı ve istekleri onaylandıktan sonra ayrıcalıklı sunucudan uygun kullanıcı düzeyinde ayrıcalıklara sahip bir **alt süreci** oluşturur. Kullanıcı imtiyazlı alt süreç, aslında sistem kullanıcısının istediğini yerine getirir.

Bu desen yükseltilmiş ayrıcalıklara sahip bir sürecin kontrolünü ele almaz. Böylece zararlı kimselerin neden olabileceği hasar **sınırlandırılmış** olur. Kod incelemeleri ve ek testler ile **resmi doğrulama teknikleri** gibi ek doğrulamalar izinsiz ayrıcalık artışını daha da azaltabilecek özel bir ayrıcalıkla yürütülen koda odaklanabilir.

Ayrıcalık ayrıştırma tasarım deseni, **ön kimlik doğrulama** ve **ön kimlik doğrulama sonrası** şeklinde iki aşamadan oluşmaktadır. Ön kimlik doğrulaması aşamasında kullanıcı bir sistem ile bağlantıya geçer ancak henüz kimliğini doğrulamaz. Dolayısıyla hiçbir erişim ve dosya sistemine **işlem imtiyazı bulunmaz**. Ön doğrulama sonrası kimlik doğrulama aşamasında ise kullanıcı sisteme başarıyla doğrulanır ve dosya sistemi erişimi gibi kullanıcı ayrıcalıklarına sahiptir.

Ayrıcalık ayrıştırma deseninde uygulanan genel işlem aşağıdaki gibidir:

- Ayrıcalıklı bir sunucu oluşturulur.
- İlk kullanıcı istekleri bu sunucuya yönlendirilir.
- Sunucuya bir kullanıcı isteği geldiğinde, sunucu kimlik doğrulama işlemi sırasında gereken kullanıcı etkileşimi için güvenilmeyen, ayrıcalıklı olmayan bir **yöntem** (*child*) ortaya çıkarır.
- Kullanıcının kimliği doğrulandıktan sonra, sunucu kullanıcının isteğini yerine getirmek için uygun bir **Kullanıcı Kimliği** (*UserID*) ile başka bir alt işlemden çıkar.

Linux altında `fork ()`, `chroot ()` ve `stuid ()` yöntemleri kullanılarak ayrıcalık ayrıştırma tasarım desenine uygun uygulamalar yapılabilir.

Ayrıcalık ayrıştırma tasarım deseni ile ilgili bilinen uygulamalar aşağıdaki gibidir.

- **OpenBSD** (*sshd, bgpd, ospfd, ripd, rtadvd, X sunucusu, snmpd, ntpd, dhclient, tcpdump*)
- Güvenli **XML-RPC** Sunucu kütüphanesi

c) Çekirdeğe Erteleme Tasarım Deseni

Çekirdeğe erteleme (*Defer to Kernel*) tasarım deseninin amacı, yükseltilmiş ayrıcalıklar gerektiren işlevleri, yükseltilmiş ayrıcalıklar gerektirmeyen işlevsellikten **açıkça ayırmak** ve **çekirdek düzeyinde mevcut olan kullanıcı doğrulama işlevinden yararlanmaktır**. Mevcut kullanıcı doğrulama çekirdeği işlevselliğini kullanmak, kullanıcı düzeyindeki güvenlik kararlarını uygulama yollarını yeniden icat etmek yerine çekirdeğin güvenlik kararlarını uygulama etmedeki rolünü güçlendirmektedir. Çekirdeğe erteleme tasarım deseni ile referans izleme tasarım deseni arasındaki temel fark, çekirdeğe erteleme deseninin işletim sistemi çekirdeği tarafından sağlanan **kullanıcı doğrulama işlevselliğinin kullanımına odaklanmasıdır**. Oysa referans izleme tasarım deseni yetkilendirme yöntemini belirtmemektedir.

Bu güvenli tasarım deseni için birincil motivasyon, yükseltilmiş ayrıcalıklarla çalışan ve dolayısıyla ayrıcalık saldırılarına karşı duyarlı olan kullanıcı programlarına olan ihtiyacı **azaltmak** ve **önlemektir**. UNIX tabanlı sistemlerde bu durum kullanıcı kimliğinin ayarlandığı (*setuid*) programların **azaltılması** veya **önlenmesi** anlamına gelir. Windows tabanlı sistemlerde ise bu durum, yönetici olarak çalışan kullanıcı programlarından kaçınılması anlamında gelir. Ek olarak bu tasarım deseni, işletim sistemi çekirdeği tarafından sağlanan kullanıcı doğrulama işlevselliğinin yeniden kullanılmasına odaklanmaktadır. Kullanıcıları doğrulamak için mevcut çekirdek işlevselliğinin yeniden kullanılması aşağıdaki avantajları beraberinde getirmektedir.

- Geliştiricilerin kendi kullanıcı kimliklerini ve doğrulama işlevlerini **yeniden yazmaları gerekmez**,
- Mevcut çekirdek kullanıcı kimliği ve doğrulama işlevselliği üzerinde zaten test ve doğrulama yapıldığından bu işlemlerin **tekrar yapılmasına gerek kalmaz**,
- Daha fazla taşınabilir bir çözümdür çünkü her bir işletim sisteminin kullanıcıları her platformla tutarlı bir şekilde **doğrulamasını** sağlayacaktır.

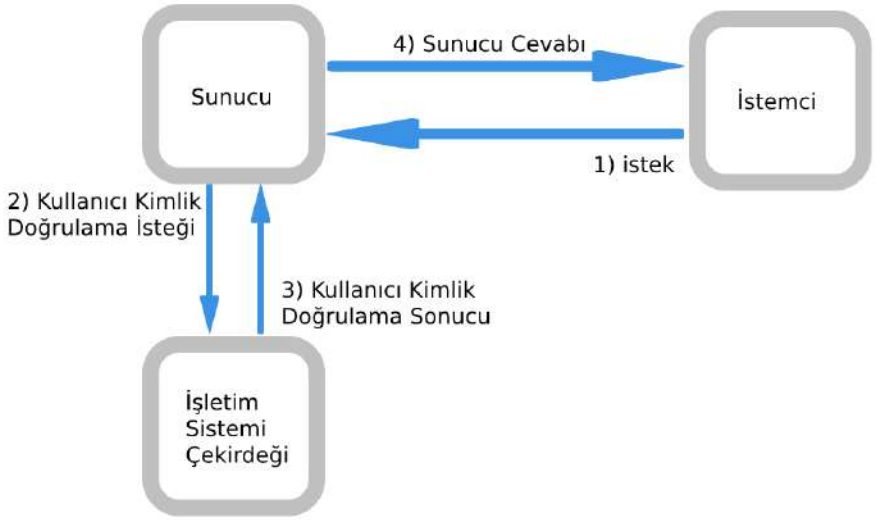
Eğer bir sistem aşağıdaki özelliklere sahip ise **çekirdeğe erteleme tasarım deseni** uygulanabilir.

- Sistem, **ayrıcalıklara sahip olmayan kullanıcılar** tarafından çalıştırılıyorsa,
- Sistemin işlevselliğinin bir kısmı **yüksek ayrıcalıklar gerektiriyorsa**,
- Yüksek ayrıcalıklar gerektiren işlevselliği yerine getirmeden önce, sistemin geçerli kullanıcının işlevini gerçekleştirmesine izin verildiğinin doğrulamasının gerektiği sistemlerde,

Özellikle UNIX tabanlı işletim sistemlerinde çalışan uygulamalar için sistemin aşağıdaki özelliklere sahip olması durumunda da **çekirdeğe erteleme tasarım deseni** uygulanabilir.

- Program, görevlerinin bir kısmını veya tamamını yerine getirmek için özel bir **Kullanıcı Kimliği** (*UID*) altında çalışıyorsa,
- Program, kullanıcılar tarafından gönderilen dosyaları veya iş taleplerini kabul edebiliyorsa,
- Yerel kullanıcılar için programın erişim kontrolünü veya muhasebe için her bir dosya veya iş talebini hangi **Kullanıcı Kimliği** (*UID*) veya **Grup Kimliğine** (*GID*) gönderdiğini biliyorsa,
- Yerel olmayan kullanıcılar için program başka bir kullanıcı **doğrulama** ve kayıt mekanizması kullanıyor ise,

Çekirdeğe erteleme tasarım deseni temel bir **istemci-sunucu mimarisini uygular**. Sunucu **yükseltilmiş** ayrıcalıklarla çalışır, istemcilerden gelen kullanıcı isteklerini kabul eder ve mümkün olduğunda kullanıcıları doğrulamak için **mevcut çekirdek işlevlerini** kullanır. Çekirdeğe erteleme tasarım deseninin genel yapısı **Şekil 16-6**'da gösterilmiştir.



Şekil 16-6: Çekirdeğe Erteleme Deseninin Genel Yapısı

Çekirdeğe erteleme tasarım deseninde **Müşteri Programı**, **Sistem Çekirdeği** ve **Sunucu Programı** bileşenleri gecikme düzenindeki katılımcıları belirtmektedir.

- **Müşteri Programı** (**Client Program**): İstemci programı, standart kullanıcı seviyesi yetkileriyle çalışır. İstemcinin yeterli ayrıcalıklara sahip olmadığı işleri yapmak için sunucuya iş istekleri gönderir.
- **İşletim Sistemi Çekirdeği** (**System Kernel**): İstemci ve sunucu arasındaki iletişim için kullanılan süreçler arası iletişim mekanizmasını yönetir, kullanıcı *kimliği doğrulama* ve *erişim işlevselliğini* sağlar. Çekirdeğe uygulanan önceden var olan işlevselliğin, sunucuda çalışmasına izin verilip verilmediğini kontrol etmek için kullanılır.
- **Sunucu Programı** (**Server Program**): Sunucu programı IPC mekanizmasının **tahsis edilmiş bir örneğini izler**, **istemcilerden gelen iş taleplerini okur** ve müşterilerinin gönderilen işlerin sunucuda **çalıştırılıp çalıştırılmayacağını** kontrol eder.

Daha önce tek bir çalıştırılabilir dosyaya dayanan uygulamalar, yeniden yapılandırılmalıdır. Bunlar UNIX tabanlı işletim sistemlerinde **kullanıcı kimliği ayarlanabilen** (*setuid*), Windows tabanlı işletim sistemlerinde ise **yönetici olarak çalıştırılabilen** dosyalardır.

Çekirdeğe erteleme tasarım deseninin bilinen uygulamaları aşağıdaki gibidir:

Sunucu bilinen bazı mekanizmalar aracılığı ile müşteri isteklerini kabul etmeye başlar.

İstemci sunucuya isteğe dâhil olan müşteriyi tanımlayan bilgileri içeren bir istek gönderir. Bu bilgi **kodlanmış** veya **mevcut bir kullanıcı kimliği** kullanılarak gönderilmiş olabilir.

Sunucu, kullanıcının isteğini alır ve kullanıcının isteğini karşılayıp karşılamadığını veya isteği reddetmek için bazı çekirdek düzeyi mekanizmaları kullanır.

Çekirdeğe erteleme tasarım deseni, Windows'ta **Güvenli Nesneler** (*Securable Objects in Windows*) olarak da bilinmektedir.

d) Hatalarla Tam Görünüm Tasarım Deseni

Hatalarla tam görünüm (*Full View With Errors*) tasarım deseni, geliştiricilerin kullanıcı arayüzü geliştirirken izinleri yok saymasına izin verir. Bu desen ile kullanıcıların **çağırmasına izin verilmeyen seçenekler de dâhil olmak üzere** bir programın tüm seçeneklerinin görülmesi sağlanır. Bir kullanıcı istenen seçeneği seçtiğinde, sistem kullanıcının bu işlemi **kullanıp kullanamayacağını** ve **buna göre cevap verip vermeyeceğini** belirler. Bu mimari desen, kullanıcıların yasa dışı işlem yapmasını önler ve neredeyse tüm ayrıcalıklara sahip kullanıcılar için çok faydalıdır.

Bu tasarım deseni aynı zamanda bir kayıt modülüne sahip olan hatalara cevap verebilecek bir **hata yönetimi çerçevesiyle** oluşturulmalıdır. Bu desenin kullanımı, eğitim materyallerinin tüm kullanıcılar için tutarlı olmasını sağlar. Bu desenin uygulanması kolaydır çünkü tek bir görünüm **tüm kullanıcılara** sunulabilir. Bununla birlikte, kullanıcıları **hata mesajları** ile bombardıman ederek karıştırabilir. Bu desenle iyi çalışan diğer desenler arasında, işlemlerde güvenlik kontrolleri gerçekleştirme mekanizmaları ve bir kullanıcının istenen işlemi gerçekleştirme izni olup olmadığına karar vermek için roller bulunur.

Tasarım Düzeyi Güvenli Tasarım Desenleri

Tasarım düzeyi desenleri, üst düzey bir sistem bileşeninin elemanlarının **nasıl tasarlanıp uygulandığını** açıklar. Üst seviye bileşenlerin kendi tanımlarını ve etkileşimlerini değil, tek bir üst seviye bileşenin **iç tasarımındaki sorunları** ele alır. **Tasarım düzeyi desenler**:

- Güvenli Fabrika Tasarım Deseni
- Güvenli Strateji Fabrikası Tasarım Deseni
- Güvenli Oluşturucu Fabrikası Tasarım Deseni
- Güvenli Sorumluluk Zinciri Tasarım Deseni
- Güvenli Durum Makinesi Tasarım Deseni
- Güvenli Ziyaretçi Tasarım Deseni
- Kontrollü Nesne Fabrikası Tasarım Deseni

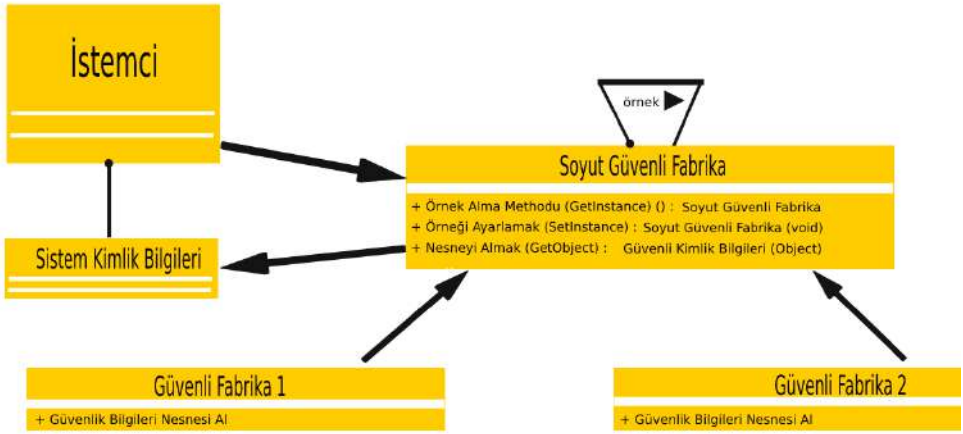
a) Güvenli Fabrika Tasarım Deseni

Güvenli Fabrika (*Secure Factory*) güvenli tasarım deseninin amacı, bir nesneyi oluşturma ya da seçmedeki güvenlik mantığını oluşturulan ya da seçilen nesnenin temel işlemlerinden ayırmaktır. Güvenli fabrika tasarım deseni, **soyut fabrika modelinin güvenliğe özgü bir uzantısı**dır.

Kullanıcı güven seviyesine göre doğru nesneyi seçmek için gerekli olan mantık, güvenli fabrika tasarım deseni kullanılarak nesneyi ve nesneyi oluşturan uygulama kodundan ayrılabilir. Güvenlik tabanlı nesne seçimi mantığı ile nesne uygulaması arasındaki kararlı olmayan bağlantı ve güvenlik tabanlı nesne seçimi mantığı ile güvenli uygulama arasındaki kararlı olmayan bağlantı güvenlik tabanlı nesne seçimi mantığının **doğrulanmasını**, **test edilmesini** ve **değiştirilmesini** kolaylaştırır.

Güvenli fabrika tasarım deseni aşağıdaki durumlarda **geçerli** olmaktadır:

- Sistem, bir kullanıcının güvenlik kimlik bilgilerine dayanarak bir nesnenin farklı versiyonlarının oluşturulduğu senaryolarda,
- Mevcut güvenlik bilgilerinin, doğru nesneyi seçmek ve oluşturmak için gereken **tüm bilgileri içerdiği durumlarda**.



Şekil 16-7: Güvenli Fabrika Tasarım Deseninin Yapısı

Çalışma zamanında güvenli fabrika örneğini değiştirme işlevi sunar ve güvenli fabrika sınıflarında uygulanacak bir nesneyi alma arayüzünü tanımlar. Somut Güvenli Fabrika, `Nesneyi Almak ()` yöntemini uygulayan soyut fabrikanın **somut uygulamalarını** temsil eder. İstemci sınıfı, kullanıcının güvenlik kimlik bilgilerini izler, fabrikanın somut bir örneğini almak için `Örnek Alma ()` yöntemini kullanır ve ardından somut fabrikanın `Nesneyi Almak ()` özelliğini kullanarak verilen güvenlik bilgileri için **bir nesne alır**. Güvenlik Kimlik Bilgileri, **geçerli kullanıcı kimlik bilgilerini** temsil eder. Güvenli fabrika modeli, güvenlik mantığını **istemciden ayırır** ve **gizler**; böylece test edilmesi daha kolay olan **daha kısa kod** elde edilir. Güvenli Fabrika, Müşterinin davranışını değiştirmek zorunda kalmadan güvenlik mantığının değişmesini sağlayan **kara kutu** görevi görür.

İstemci (**Client**): İstemci, bir kullanıcının veya sistemin çalıştığı ortamın güvenlik bilgilerini izler. İlgili güvenlik kimlik bilgileri göz önüne alındığında, müşteri güvenli fabrikanın somut bir örneğini almak için **Soyut Güvenli Fabrika** sınıfının `Örnek Alma ()` yöntemini kullanır ve ardından geçerli güvenlik verilen uygun nesneyi elde etmek için `Nesneyi Alma ()` yöntemini çağırır.

Sistem Kimlik Bilgileri sınıfı, bir kullanıcının **güvenlik kimlik bilgilerini** gösterir.

Soyut Güvenli Fabrika sınıfı birkaç amaca hizmet etmektedir. Bunlardan bir tanesi fabrikanın `Nesneyi alma ()` yöntemi ile güvenli bir fabrikanın **somut** bir örneğini sağlar. Diğeri sistemin, çalışma zamanında `Örneği Ayarlama ()` yöntemi ile varsayılan **somut güvenli fabrikayı** ayarlamasını sağlar. Bu çalışma zamanında farklı bir somut güvenli fabrika belirleyerek nesne seçim metodolojisini değiştirmeyi nispeten kolaylaştırır.

Soyut Güvenli Fabrika'nın tüm somut uygulamaları tarafından uygulanması gereken `soyut Nesneyi Alma ()` yöntemini tanımlar. `Nesneyi Alma ()` yöntemi, müşteri tarafından bazı güvenlik bilgileri verilen **uygun nesneyi elde etmek** için çağrılır.

Somut Güvenli Fabrika N: Soyut Güvenli Fabrika'nın çeşitli somut uygulamalarında farklı **nesne seçim metodolojileri** uygulanmaktadır. Her bir somut güvenli fabrika, `Nesneyi Alma ()` yönteminin bir uygulamasını sağlar.

Çeşitli Nesne sınıfı, güvenli fabrika tarafından döndürülen nesneler tarafından uygulanan temel arabirimi tanımlar.

Somut Nesne N: Geçerli güvenlik bilgileri tarafından belirtilen her nesne davranışı kümesi için bir **Çeşitli Nesne** somut uygulaması oluşturur.

Güvenli fabrika, arayan kişiye uygun nesneyi sağlayan **kara bir kutu** olarak çalışır. Bu, güvenlik bağımlı nesne seçim mantığını arayandan **gizler**. Güvenli fabrika tarafından oluşturulan nesnelerin yalnızca ilgili güven seviyelerine **uygun işlevsellik uygulaması** gerekir. Güvenli Fabrika tarafından oluşturulan nesnelerin, geçerli kullanıcı veya işletim ortamı hakkında bilgi verilen nesneye uygulanan bir eyleme **izin verilip verilmediğini kontrol etmek gerekmez**. Bu kontroller, nesneyi oluşturan **Güvenli Fabrika** tarafından zaten yapılmıştır.

Güvenli Fabrika tasarım deseninin kara kutu yapısı, sistemin **güvenlik bilgilerine bağlı davranışını değiştirmeyi kolaylaştırır**. Nesne seçim mantığındaki veya sağlanan nesnelerin kendisindeki değişiklikler, Güvenli Fabrikayı kullanan kodda **çok az değişiklik yapılmasını** gerektirecektir.

Güvenli fabrika tasarım deseninin uygulamadaki genel süreci şu şekildedir:

- Yapısı veya seçimi, kullanıcı veya işletim ortamıyla ilişkili **güven düzeyine bağlı olan bir nesne** tanımlanır.
- Nesne tarafından desteklenen genel işlevselliği tanımlayan **soyut bir sınıf** veya **arayüz** tanımlanır.
- Nesnenin güven düzeyine özgü davranışını uygulayan **somur sınıflar** uygulanır. Tanımlanan her güven nesnesi için bir somut uygulama oluşturulur.
- Soyut Güvenli Fabrika sınıfını tanımlamak için **temel soyut fabrika modeli** kullanılır.
- Bir kullanıcının veya ortamın **güven düzeyini belirlemek için gereken bilgileri** belirlenir. Bu bilgi Güvenlik Bilgileri sınıfını veya veri yapısını tanımlamak için kullanılır.
- İkinci adımda tanımlanan nesne ile dördüncü adımda tanımlanan güvenlik kimlik bilgilerine göre seçen **somut güvenli bir fabrika** uygulanır.
- Beşinci adımda tanımlanan soyut fabrika tarafından sağlanan, varsayılan fabrika olarak **beşinci adımda tanımlanan somut güvenli fabrika ayarlanır**.
- Yazılan uygulamanın, kullanılan güvenli fabrikayı kolayca değiştirmesini gerektirmiyorsa **soyut olmayan fabrika modeli kullanılarak** güvenli fabrika tasarım desenini uygulamak mümkün olacaktır.
- Beşinci adımda tanımlanan soyut fabrika tarafından sağlanan varsayılan fabrika olarak **beşinci adımda tanımlanan somut güvenli fabrika ayarlanır**.
- Yazılan uygulamanın, kullanılan güvenli fabrikayı kolayca değiştirmesini gerektirmiyorsa **soyut olmayan fabrika modeli kullanılarak** güvenli fabrika tasarım desenini uygulamak mümkün olacaktır.

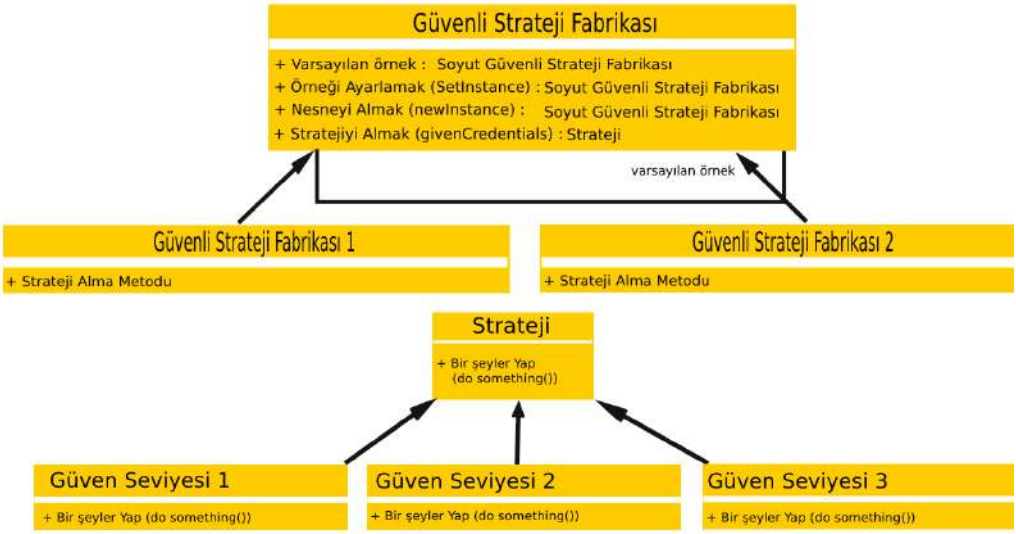
b) Güvenli Strateji Fabrikası Tasarım Deseni

Güvenli strateji fabrikası (*Secure Strategy Factory*) tasarım deseni bir kullanıcının güvenlik kimlik bilgilerine göre gerçekleştirmek istediği bir işlem için **uygun stratejide bir desen uygulayan** nesnedir. Uygun desenin seçilmesinde **güvenli strateji fabrikasının kullanımı kolaylık sağlamaktadır**.

Güvenli strateji fabrikası tasarım deseni, kullanıcının güvenlik bilgilerine dayanarak bir **strateji nesnesi seçmenin kolay yolunu** sunar. Bu tasarım deseni, sisteme giriş yapan kullanıcıya bağlı olarak değişiklik göstermesi gerektiğinde ve yalnızca **kullanıcının kimlik bilgileriyle belirlenebildiği zaman** kullanılmalıdır. Bu yüzden **strateji** adını da almıştır.

Güvenli olarak kabul edilen bir sistem içerisindeki fonksiyonlar, sistemdeki kullanıcıların veya sistemin çalıştığı ortamın güvenlik kimlik bilgilerine dayanarak **farklı davranışlar** gösterebilmektedir. Örnek vermek gerekirse desene uygun olarak çalışan bir sistemde, kullanıcının güven düzeyine bağlı olarak **değişkenlik gösteren çeşitli hata mesajları** yer alabilir. Güvenilmeyen kullanıcılar, güvenilen kullanıcılara göre daha fazla bilgi içermeyen **hata mesajı** alabilir. Güvenli strateji fabrikası deseni, genel bir sistem işlevi için güvenlikte kimlik bilgilerine dayanan bir strateji sergiler ve uygulanacak olan davranışların sistemden değiştirilmesi, test edilmesi ve doğrulanmasının daha kolay yapılması için davranışlar kodlama esnasında belirlenmelidir.

Şekil 16-8 Güvenli Strateji Fabrikası desen yapısını göstermektedir.



Şekil 16-8: Güvenli Strateji Fabrika Desen Yapısı

Şekilde yer alan sınıflardan **Güvenlik Bilgileri** sınıfı, bir kullanıcının güvenlik bilgilerini vermektedir.

Soyut Güvenli Strateji Fabrikası sınıfı birkaç amaca hizmet etmektedir. Bunlardan bir tanesi fabrikanın `Örneği Alma ()` yöntemi ile güvenli bir strateji fabrikasının **somut örneğini** sağlar diğeri ise sistemin çalışma esnasında `Örneği Ayarlama ()` yöntemi ile gerçek somut güvenli stratejiyi ayarlamasına olanak sağlar. **Soyut Güvenli Strateji Fabrikası** sınıfının tüm somut uygulamaları tarafından uygulanması gereken `soyut Stratejiyi Al ()` yöntemini tanımlamaktadır. **Somut Güvenli Strateji Fabrikası N sınıfı Soyut Güvenli Strateji Fabrikası** çeşitli somut uygulamalarında farklı strateji seçim yöntemleri uygulamaktadır. Strateji sınıfı, **soyut strateji sınıfıdır**. Genel bir sistem işlevini temsil eden bir yöntem tanımlamaktadır. Soyut strateji sınıfının tüm somut uygulamaları bu yöntemin uygulanmasında rol oynamaktadır.

Güven Seviyesi Strateji N sınıfı, sistemdeki genel işlevlerin her biri için güvenlik kimlik bilgisi temelli bir yöntem olarak her güven düzeyi için bir **somut Güven Seviyesi Strateji** uygulanmasını sağlamaktadır.

Güvenli strateji fabrikası deseninin uygulamadaki genel süreci aşağıdaki gibidi:.

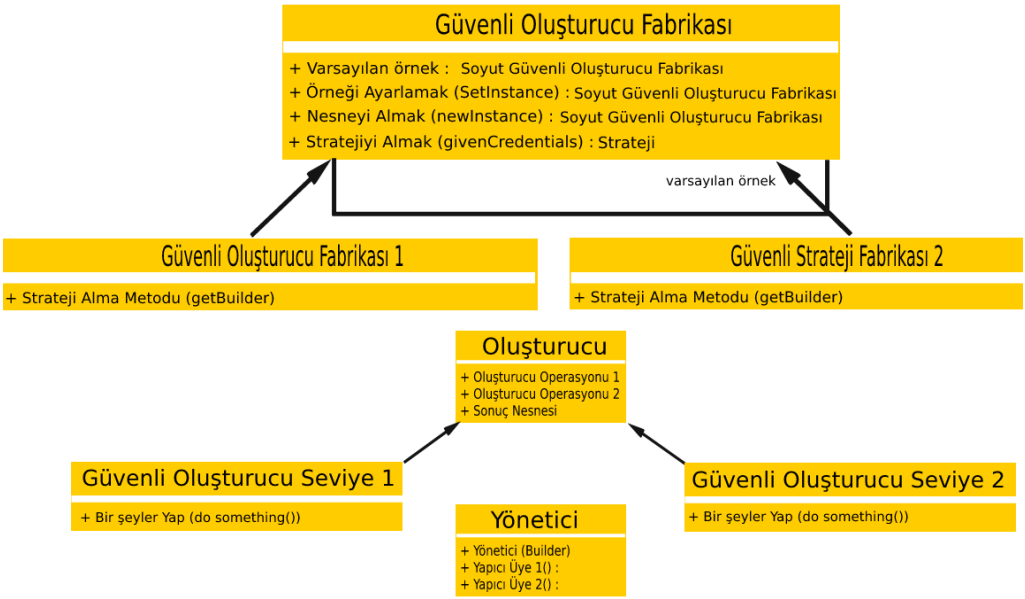
- Güvenli tasarım desenine uygun **genel bir sistem işlev**i tanımlanır.
- Tanımlanan genel sistem fonksiyonunu uygulayan sınıfların arayüzünü tanımlamak için **strateji deseni** kullanılır.
- İkinci adımda tanımlanan arabirimin **somut uygulamaları** yazılır. Bunu nesne tabanlı programlamadaki soyut sınıflardaki metotlarla iletişim kuran somut sınıflara benzetebiliriz. Bu yüzden genel sistem fonksiyonunun her bir güvenlik sürümü için **somut bir uygulama yazılmalıdır**.
- Soyut Güvenli Strateji Fabrikası sınıfını uygulamak için **temel soyut fabrika modeli** kullanılır.
- Kullanıcının güven düzeyini belirlemek için gereken bilgiler belirlenir ve bu bilgiler **Güvenlik Bilgileri** sınıfını veya **veri yapısını** tanımlamak için kullanılmaktadır.
- Dördüncü adımdaki güvenlik belgesi verildiğinde üçüncü adımda tanımlanan uygun stratejiyi seçen **somut bir güvenli strateji fabrikası** kurulur.
- Son olarak beşinci adımda tanımlanan somut güvenli strateji fabrikası, üçüncü adımda tanımlanan **soyut fabrika tarafından sağlanan varsayılan fabrika** olarak ayarlanır.

c) Güvenli Oluşturucu Fabrikası Tasarım Deseni

Güvenli oluşturucu fabrikası (**Secure Builder Factory**) tasarım deseni, güvenlik kimlik bilgilerine göre bir strateji seçmenin **kolay bir yolunu** sunar. Bu model, güvenli fabrika tasarım deseninden genişletilmiştir. Güvenli oluşturucu fabrikası, sisteme giriş yapan kullanıcıya bağlı olarak **değişiklik göstermesi gerektiğinde** ve **yalnızca kullanıcının kimlik bilgileriyle belirlenebildiği zaman kullanılmalıdır**.

Bu desenin amacı, karmaşık bir nesne oluşturulurken güvenliğe bağlı kuralları, nesneyi **gerçekten oluşturmaya dâhil olan temel adımlardan** ayırmaktır. Güvenli oluşturucu fabrikası tasarım deseni, güvenlik kuralları uyarınca karmaşık bir nesne oluşturacak olan **oluşturucu desenini uygulayan uygun nesneyi seçmek** ve **iade etmek için verilen güvenlik bilgilerini** kullanmaktadır.

Şekil 16-9 güvenli oluşturucu fabrikası tasarım deseninin yapısı gösterilmektedir.



Şekil 16-9: Güvenli Oluşturucu Fabrika Tasarım Deseni Yapısı

Şekil 16-9'da yer alan sınıflardan **Yönetici sınıfı**, oluşturucu deseninin **yönetmen ögesi**dir. Yönetmen oluşturucu örneği karmaşık nesneler oluşturur.

İstemci, güvenli oluşturucu fabrikası tarafından seçilen bir oluşturucu ile **yönetmeni harekete geçirir** ve daha sonra belirli karmaşık nesneler oluşturmak için **Yönetici** içerisindeki yöntemleri çağırır. Güvenlik Bilgileri sınıfı, bir kullanıcının **güvenlik kimlik bilgilerini** göstermektedir.

Soyut Güvenli Oluşturucu Fabrikası sınıfı, birkaç amaca hizmet eder. Bunlardan bir tanesi fabrikanın `Örneği Alma ()` yöntemi ile güvenli bir oluşturucu fabrikasının örneğini sağlar. Diğeri ise sistemin güvenli oluşturucu fabrikası çalışma zamanında `Örneği Ayarlama ()` yöntemi ile ayarlanmasını sağlar.

Bu çalışma süresinde oluşturucu seçim metodolojisini değiştirmeyi kolaylaştırmak amacıyla **farklı bir somut güvenli oluşturucu fabrikası** belirler. Soyut Güvenli Oluşturucu Fabrikası sınıfının tüm somut uygulamaları tarafından uygulanması gereken `soyut Oluşturucuyu Alma ()` yöntemini tanımlar. Somut Güvenli Oluşturucu Fabrikası N, Soyut Güvenli Oluşturucu Fabrikası sınıfının çeşitli somut uygulamalarında **farklı oluşturucu seçim yöntemleri** uygulamaktadır. Oluşturucu, soyut oluşturucu sınıfı karmaşık nesnenin tüm oluşturucuları için geçerli olan **temel oluşturucu yöntemlerini tanımlar**. Güvenli Oluşturucu Seviyesi N, **soyut oluşturucu sınıfının bir somut uygulamasıdır**.

Güvenli Oluşturucu Fabrikası tasarım deseninin uygulanmasındaki işlemler aşağıdaki gibidir:

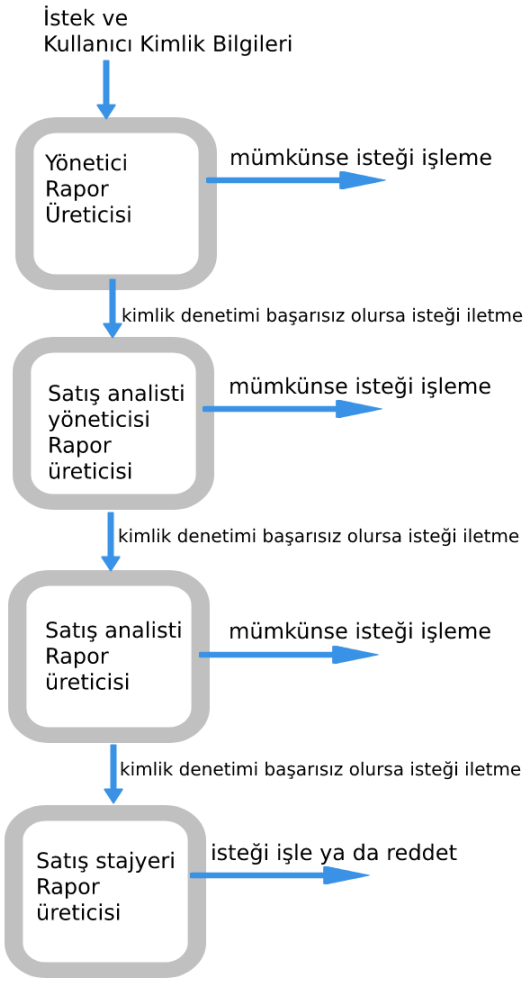
- Yapısı kullanıcı veya işletim sistemi ile ilişkili güven düzeyine bağlı olan **karmaşık bir nesne** tanımlanır.
- Bu tür nesneleri tanımlamak için oluşturucu deseninin kullanarak **oluşturucu arabirimi tanımlanabilir**.
- Karmaşık nesne için çeşitli güven düzeyine özgü yapı kurallarını uygulayan **somut oluşturucu sınıfları uygulanır**.
- Kullanıcıların güven düzeyini belirlemek için gereken bilgiler belirlenmelidir. Bu bilgiler **Güvenlik Bilgileri sınıfını** veya **veri yapısını tanımlamak üzere** kullanılacaktır.
- Güvenlik bilgileri verilen ve uygun oluşturucu seçen bir **güvenli oluşturucu fabrikası uygulanır**.

d) Güvenli Sorumluluk Zinciri Tasarım Deseni

Güvenli sorumluluk zinciri (*Secure Chain of Responsibility*) tasarım deseninin amacı, kullanıcı bağımlı işlemlerin gerçekleştirildiği senaryolarda, kullanıcıya bağımlı işlemleri belirleyen mantığı **basitleştirmektir**. Rol tabanlı erişim ve kontrol mekanizması kullanan bir uygulamada çeşitli sistem fonksiyonlarının davranışı mevcut kullanıcının rolüne bağlı olarak gerçekleşir.

Genel bir sistem işlevi için role dayalı davranışlar, kullanıcıya genel bir sistem işlevini gerçekleştirebilmesi ile ilgili olarak **düşük** veya **yüksek seviyede rolüne dayanarak erişim izni vermek** veya **vermemek** şeklinde gerçekleşir. Genel bir sistem işlevi için uygun olan davranışın rol temelli seçimini tek bir kritere bağlı uygulamak yerine, güvenli sorumluluk zinciri tasarım deseni ile **doğru davranışı seçme mantığını** kullanarak işletilmesi sağlanır.

Bir rapor oluşturma modülü düşünüldüğünde kullanıcıların raporları yalnızca **kendilerine tanımlanmış olan rollere göre oluşturabildikleri** söylenebilir. Örneğin genel yönetici rolüne sahip bir kullanıcı **mevcut tüm bilgileri içeren raporları** oluşturabilir. Ancak satış birimindeki yönetici, yalnızca satış verilerini içeren konularda tüm raporları oluşturabiliyorken satış uzmanı rolündeki kullanıcı yalnızca satış verilerinin bir alt kümesini içeren raporlar oluşturabilecektir. Bu roller göz önüne alındığında **Güvenli Sorumluluk Zinciri** güvenli tasarım deseni, bu örnekteki gibi rapor oluşturma işlevselliğini uygulamak için **Şekil 16-10**'daki gibi kurgulanmalıdır. Güvenli sorumluluk zinciri tasarım deseninin yapısı **Şekil 16-11**'de gösterilmiştir.



Şekil 16-10: Güvenli Sorumluluk Zinciri Tasarım Deseni Rapor Oluşturma Örneği



Şekil 16-11: Güvenli Sorumluluk Zinciri Tasarım Deseni Yapısı

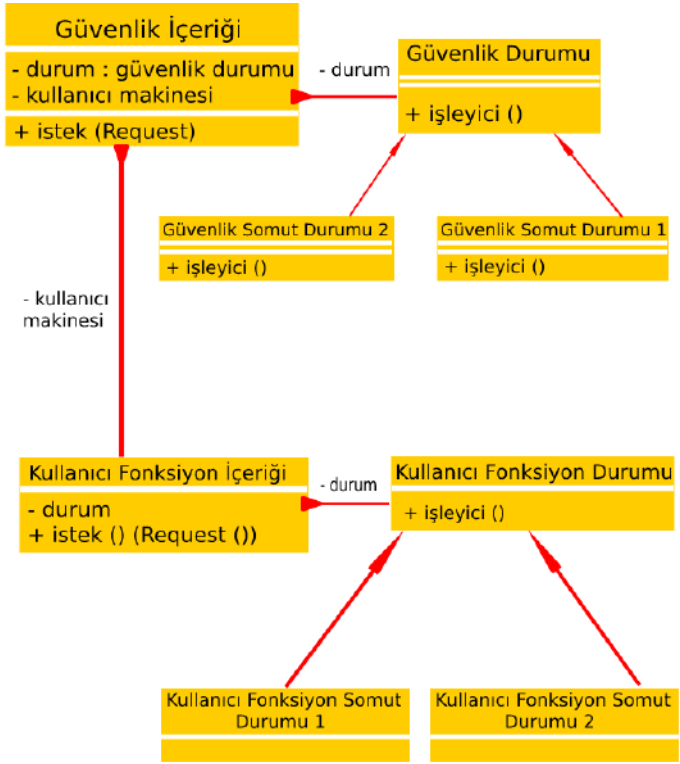
İşleyici sınıfların `İsteği İşle ()` yöntemi, kullanıcının güvenlik kimlik bilgilerini **argüman** olarak alır. Sistemdeki kimlik bilgilerinin güvenlik hususunda sınıfın bir isteği işleme izni olup olmadığını belirlemek için bir `Kimlik Bilgilerini Kontrol Et ()` yönteminin uygulanması gerekir. Eğer yapılan güvenlik kontrolü önemsiz ise `Kimlik Bilgilerini Kontrol Et ()` yöntemi **uygulanmayabilir**. Güvenlik Bilgileri sınıfı bir kullanıcının güvenlik kimlik bilgilerini gösterir. Somut İşleyici N, İşleyici sınıfının somut bir uygulaması olarak `İsteği İşlemek ()` yöntemini uygular.

e) Güvenli Durum Makinesi Tasarım Deseni

Güvenli durum makinesi (*Secure State Machine*) tasarım deseninin amacı, güvenlik ve kullanıcı düzeyinde işlevselliği uygulayarak **güvenlik mekanizmaları** ve **kullanıcı düzeyinde işlevsellik arasında net bir ayrım sağlar**.

Güvenli bir sistemin çalıştırılmasında iç içe geçmiş güvenlik ile ilgili tüm işlevsellik ve kullanıcı düzeyindeki diğer işlevsellikler karmaşıklığı arttırabilmektedir. Artan karmaşıklık uygulamanın güvenlik özelliklerini test etmeyi, incelemeyi ve doğrulamayı zorlaştırarak buradan da bir **güvenlik açığı** verilmesi olasılığını arttırır.

Ayrıca güvenlik işlevi ile kullanıcı düzeyindeki işlevsellik arasındaki sıkı bağlantı, sistemin güvenlik mekanizmalarının **değiştirilmesini zorlaştırmaktadır**. Desen aynı zamanda **güvenlik** ve **kullanıcı** düzeyinde işlevsellik arasındaki eşleşmeyi azaltarak sisteme gelecekteki değişikliklerin **kolayca uygulanabilmesini sağlar**. **Şekil 16-12**'de güvenli durum makinesi tasarım desenine ait yapı gösterilmiştir.



Şekil 16-12: Güvenli Durum Makinesi Tasarım Deseni Yapısı

Güvenlik İçeriği, kullanıcıların ilgilendiği arayüzü tanımlar. **Kullanıcı Fonksiyon İçeriği** sınıfını korur. Kullanıcı Fonksiyon İçeriği sınıfı için **Proxy** işlevi görür. Güvenlik Durumu, güvenlik durumu makinesi tarafından yönetilen olası işlemleri temsil eden arayüzü tanımlar.

Güvenlik Somut Durumu sınıfında her Güvenlik Durumu alt sınıfı her işlem için **güvenlik durumuna bağlı davranışı** uygular. Kullanıcı Fonksiyon İçeriği, güvenlik durumu makinesinin bileşenlerinin uygun olduğunda işlem isteklerini

kullanıcı düzeyinde durum makinesine iletebilmesi için **Güvenlik İçeriği** ile aynı işlemleri tanımlar.

Kullanıcı Fonksiyon Durumu, Güvenlik Durumu ile **aynı arayüze** sahiptir.

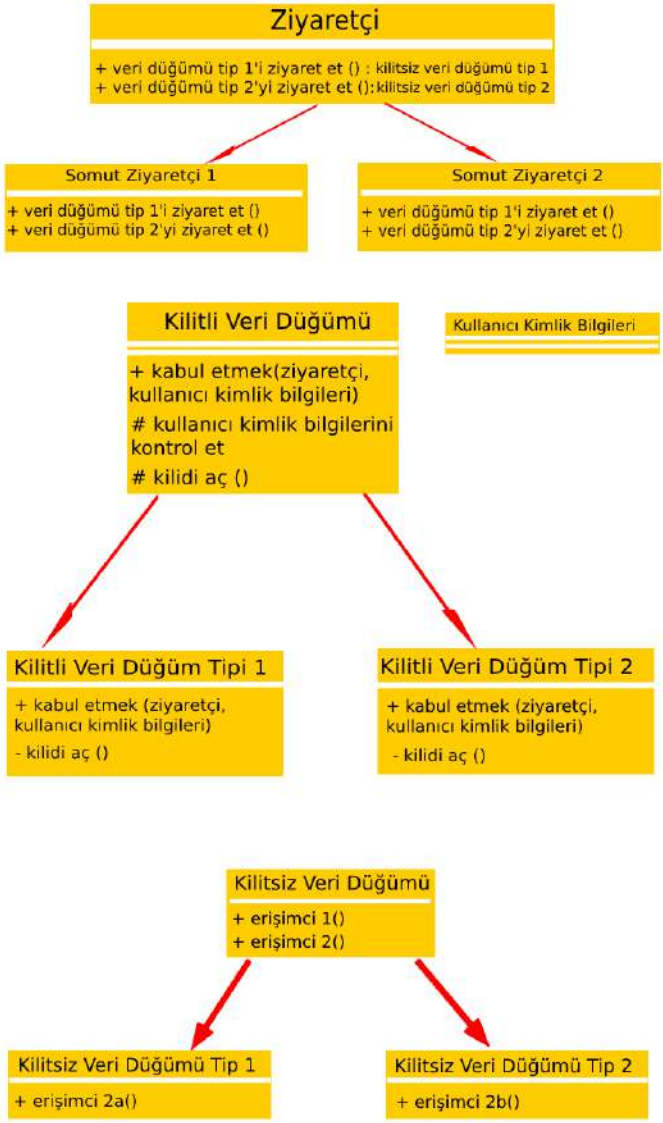
Kullanıcı Fonksiyonu Somut Durumu, Güvenlik Somut Durumuna benzer bir şekilde güvenlik durumunun her bir alt sınıfı, her işlem için kullanıcı düzeyinde duruma bağlı davranışını uygular.

f) Güvenli Ziyaretçi Tasarım Deseni

Güvenli Ziyaretçi (*Secure Visitor*) tasarım deseni, kullanıcı uygun yetkilerden yoksun olduğunda **düğüme erişimi engelleme** araçları sağlar. Güvenli sistemler, veri hiyerarşisindeki her bir düğümün farklı erişim kısıtlamalarına sahip olabileceği sistemlerdir.

Farklı düğümlerdeki verilere erişim, **verilere erişen kullanıcının rolüne bağlı** olabilir. Güvenli ziyaretçi tasarım deseninde ziyaretçi, düğümün kilidini açmak için uygun kimlik bilgilerini sağlamadıkça düğümlerin bir ziyaretçi tarafından okunmaya karşı kendilerini kilitlemelerine olanak tanır. Güvenli ziyaretçi tasarım deseni, kilitli bir düğüme erişmenin tek yolunun bir ziyaretçiyle olduğu ve veri yapısındaki düğümlere **yetkisiz erişimi önlemeye yardımcı olacak şekilde** tanımlanır.

Güvenli durum makinesi tasarım deseninde olduğu gibi güvenli ziyaretçi tasarım deseninin temel motivasyonu, güvenlikle ilgili **konular** ve **kullanıcı düzeyinde işlevsellik arasında** temiz bir ayrım sağlamaktır. Güvenli ziyaretçi tasarım deseni, tüm güvenlik hususlarını veri hiyerarşisindeki düğümlere tahsis eder ve geliştiricilerin yalnızca kullanıcı düzeyinde işlevsellik ile ilgilenen ziyaretçiler yazmasına izin vermez. Düğümleri veri hiyerarşisindeki güvenlik işlevlerinden yalnızca sorumlu kılmak, güvenlik işlevini kullanıcı düzeyindeki işlevsellikten daha titiz bir şekilde test etmeyi ve doğrulamayı **daha uygun hâle** getirmeyi sağlar. **Şekil 16.13**'te güvenli ziyaretçi tasarım deseninin yapısı gösterilmiştir.



Şekil 16-13: Güvenli Ziyaretçi Tasarım Deseni Yapısı

Ziyaretçi: Güvenli ziyaretçi modelindeki **ziyaretçi katılımcısı**, standart ziyaretçi şablonundaki ziyaretçi katılımcısıyla hemen hemen aynıdır. Modellerdeki birincil fark, çeşitli ziyaret yöntemlerinin Güvenli Ziyaretçi

deseninde **kilitsiz düğüm nesnelerini almasıdır**. Oysa Standart Ziyaretçi desenindeki ziyaret yöntemleri **basitçe bir düğüm nesnesi** alır. Yani standart desende **kilitli** ve **kilitsiz** veri düğümleri kavramı yoktur.

Somut Ziyaretçi: standart Ziyaretçi deseninde olduğu gibi somut ziyaretçi sınıfları, soyut ziyaretçi sınıfında **tanımlanan işlemleri uygular**.

Kilitli Veri Düğümü sınıfı, bir ziyaretçiyi kabul eden bir `Kabul ()` işlemini tanımlar. Ek olarak, Kilitli Veri Düğümü sınıfı bir kullanıcının kimlik bilgilerini kontrol etmek ve geçerli kilitli düğümün kilidini açmak için **bir işlem tanımlar**. Kilitli bir düğümün, düğümdeki verileri görüntülemek veya düğümdeki verileri değiştirmek için **ortak işlem yapmadığı unutulmamalıdır**. Düğüme olan tüm erişim, düğümün `Kabul ()` işlemi aracılığı ile yönlendirilmelidir.

Kabul () İşlemi: Kullanıcının kimlik bilgilerini kontrol eder. Kimlik bilgileri, kullanıcının geçerli düğümdeki verileri görüntülemesi için geçerliyse düğüm `Kilidi Aç ()` işlemini kullanarak **kendi kilidini açar** ve **kilidini açmış sürümünü ziyaretçinin ziyaret yöntemine geçirir**.

Kilitli Veri Düğümü Tipi N sınıfları: **Soyut Kilitli Veri Düğümü** sınıfında tanımlanan işlemleri uygular. Bu çeşitli kilitli düğüm nesnelerinin kilidini açmak ve düğümlerin kilitsiz sürümlerini döndürmek için `Kilidi Aç ()` işlemini içerir.

Kilitsiz Veri Düğümü sınıfı: Kilitli bir veri düğümünün **kilitsiz sürümünü** temsil eder.

Kilitsiz Veri Düğümü Tipi N: Kilitsiz Veri Düğümü ögesinin somut uygulamaları **soyut sınıfta tanımlanan işlemleri** uygular.

Kullanıcı Kimlik Bilgileri sistemin **mevcut kullanıcısını** veya mevcut kullanıcıya **atanan izinleri** temsil eder. Güvenli Ziyaretçi tasarım deseni, kullanıcı kimlik bilgilerinin uygulamasına çok fazla kısıtlama getirmez. Tek gereksinim, bir düğümün verilerine erişimi denetlemek için **kimlik bilgilerini kullanmasının mümkün olmasıdır**.

Güvenli ziyaretçi tasarım deseninin amacı, düğüm için uygun kimlik bilgilerini sağlamadan **verileri kilitli bir düğümde okumayı zorlaştırmaktır**. Desenin kendisi uygun kimlik bilgileri olmadan kilitli bir düğüm verisini programatik olarak okumayı zorlaştırsa da kilitli düğümü oluşturan **ham baytları okumak** ve böylece **kilitli düğümdeki verilere erişim sağlamak hâlâ mümkün olabilir**. Bu, kilitli düğümdeki verilerin aslında bir şekilde "**kilitli**" olması

gerektiği anlamına gelir. Veri şifreleme veya çevrimdışı depolama kullanılarak kilitli bir düğümde kilitlenebilir.

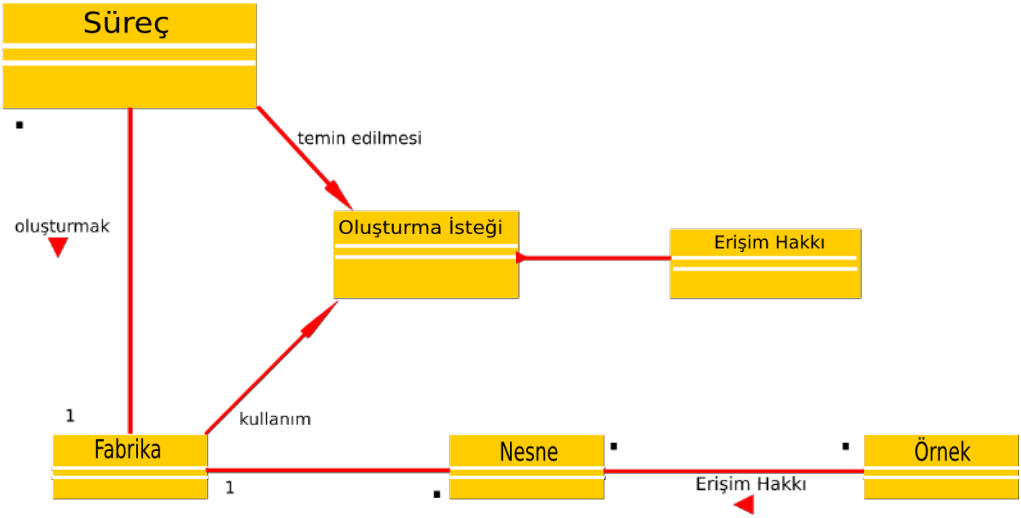
Şifreleme (Encryption): Kilitli bir düğümdeki veriler şifrelenebilir ve yalnızca bir ziyaretçinin kimlik bilgilerini kabul ettikten sonra düğümün kilitlenmemiş bir sürümünü yapma işleminin bir parçası olarak deşifre edilebilir.

Çevrimdışı Depolama: Kilitli bir düğümdeki gerçek veriler, bir veri tabanı gibi bir tür *hâricî*, *korumalı* **veri yönetim sisteminde** saklanabilir. Gerçek düğüm verileri yalnızca bir ziyaretçinin kimlik bilgilerini kabul ettikten sonra hâricî kaynaktan yüklenir.

g) Kontrollü Nesne Fabrikası Tasarım Deseni

Kontrollü nesne fabrikası (*Controlled Object Factory*) tasarım deseni, nesne oluşturma sırasında izinleri belirleyip ayarlayarak nesneler için işlem izinlerinin **yönetilmesine** yardımcı olur. Her bir nesne için izinler değişmez ise **en az ayrıcalık ilkesi uygulanamaz**. Dinamik izinlerin eksikliği, nesneleri kötüye kullanımlarına karşı savunmasız bırakabilir. Bu desen, nesneler oluşturulduğunda bir **konu listesi** ve **ayrıcalıklar** tanımlar.

Güvenlik politikaları, nesnelere kimlerin erişebileceğini açıklarken bu desen **kullanılmalıdır**. Bu deseni uygulamak için bir yöntem erişim kontrol listelerinin (**EKL-ACL**) nesnelerle **ilişkilendirilmesini** içerir. Sistemlerin kullanılabilir olması için nesnelerin izinlerinde dinamik değişikliklere izin vermeleri gerekir. Bu deseni kullanırken varsayılan izinler olmayacağından, tüm nesnelerin **tanımlanmış izinleri olmalıdır**. Kaynak havuzlarından alınan nesneler dinamik olarak ayarlanmış izinlere sahiptir.



Şekil 16-14: Kontrollü Nesne Fabrikası Tasarım Deseni Yapısı

Şekil 16-14, altı katılımcıyla Kontrollü Nesne Fabrikasını göstermektedir. *Süreç*, *Fabrika*, *Oluşturma İsteği*, *Nesne*, *Konu* ve *Erişim Hakkı* şekilde yer alan katılımcıları göstermektedir.

İşlem: Nesnenin oluşturulmasını ister.

Fabrika: Oluşturma isteğini kullanarak Nesneyi oluşturur. oluşturma İsteği, işlem tarafından gönderilir ve **Nesne** ile **Konu** arasında **Erişim Hakkı** içerir.

Konu: Nesneyi kullanmak isteyebilecek **diğer nesneleri** temsil eder. Kontrollü Nesne Fabrikası, nesnelerin onu kullanacak tüm konular için belirlenmiş **uygun yetkilere sahip olmasını** sağlar.

Uygulama Düzeyi Güvenli Tasarım Desenleri

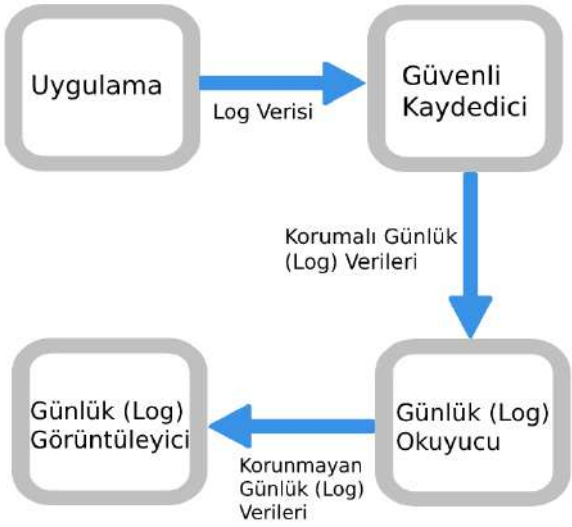
Uygulama düzeyindeki tasarım desenleri **düşük seviyeli güvenlik sorunlarına** yöneliktir. Bu sınıftaki desenler genellikle sistemdeki belirli işlevlerin veya yöntemlerin uygulanması için geçerlidir. Uygulama seviyesindeki modeller, **CERT** güvenli kodlama standartları tarafından belirlenen aynı problemi ele alır ve sıklıkla ilgili güvenli **kodlama kılavuzuna bağlanır**. Bu başlık altında incelenecek olan uygulama düzeyi tasarım desenleri aşağıdaki gibidir.

- Güvenli Kaydedici Tasarım Deseni
- Hassas Bilgileri Temizle Tasarım Deseni
- Güvenli Dizin Tasarım Deseni
- Yol Adı Kanonikleştirme Tasarım Deseni
- Kaynak Toplama Başlatma Tasarım Deseni
- Giriş Doğrulama Tasarım Deseni
- Kimlik Doğrulayıcı Tasarım Deseni
- Yetkilendirme Tasarım Deseni
- Kontrol Noktası Tasarım Deseni
- Bilgi Gizliliği Tasarım Deseni
- Sınırlı Görünüm Tasarım Deseni
- Çok Düzeyli Güvenlik Tasarım Deseni
- Rol Tabanlı Erişim Kontrolü Tasarım Deseni
- Rol Hakları Tanımı Tasarım Deseni
- Roller Tasarım Deseni
- Tek Erişim Noktası Tasarım Deseni
- Güvenli Oturum Tasarım Deseni
- Güvenli Erişim Katmanı Tasarım Deseni
- Güvenli Kanallar Tasarım Deseni

a) Güvenli Kaydedici Tasarım Deseni

Güvenli kaydedici (*Secure Logger*) tasarım deseninin amacı, bir saldırganın sistem hakkında potansiyel olarak **yararlı bilgiler toplayabilmesini** ve sistem günlüğünü düzenleyerek saldırganın **eylemlerini gizlemesini** önlemektir.

Sistem günlükleri genellikle bir sistemi yöneten veya yeni bir sistemde hata ayıklayan kişiler için yararlı olan çok fazla bilgi içerir. Ayrıca bir sistem günlüğünde yer alan bilgiler, sisteme saldırmak için yeni vektörler tasarlamak için oldukça elverişli bilgileri içerebilir. **Şekil 16-15**, güvenli kaydedici fabrikası tasarım deseninin yapısını göstermektedir.



Şekil 16-15: Güvenli Kaydedici Fabrikası Tasarım Deseni Yapısı

Uygulama: Güvenli kayıt sistemi tarafından işlenen ve depolanan kayıt verilerini oluşturan bir tür ana uygulamadır.

Güvenli Kayıt Cihazı: Güvenli kayıt sistemi, kayıt bilgilerini yetkisiz bir kullanıcının *kayıt verilerine erişmesini zorlaştıracak şekilde saklamaktan sorumlu*dur.

Günlük Okuyucu güvenli günlük kaydı sistemi: Günlük verilerini korumalı bir şekilde sakladığından, günlük kayıtlarının içeriğini okumak için güvenli günlük kaydı sistemine özgü bir okuma mekanizmasının kullanılması gerekecektir. Günlük dosyalarını okumak için standart mekanizmalar, örneğin

dosyayı standart bir metin editöründe okumak veya kelime işlem uygulaması, güvenli kayıt sistemi tarafından depolanan kayıt verileri için çalışmayacaktır.

Günlük Görüntüleyici: Yetkili bir kullanıcı bir tür günlük görüntüleme uygulamasını kullanarak **günlük verilerini okuyabilir**. Sonuç olarak güvenli günlük alt sistemi tarafından yönetilen günlük verilerine erişen saldırgan, gerçek günlük verileri içeriğini **sınırlı olarak görecek** veya **hiç göremeyecektir**. Bu sayede saldırgan, sistemde daha karmaşık saldırıları planlamak için günlük verilerindeki bilgileri kullanamayacaktır. Ayrıca bir saldırgan tarafından günlük verilerinde yapılacak olan saldırı amaçlı değişiklikler **yetkili kullanıcı** tarafından tespit edilebilir.

Bilinen güvenli **günlük kayıt sistemleri** aşağıdaki gibidir.

- **syslog-ng** (*http://www.balabit.com/network-security/syslog*)
- Syslog günlük kaydı uygulaması, bir Linux sunucusunda çalışan birden fazla uygulama tarafından kullanılabilecek güvenli bir merkezî kayıt sistemi sağlar.
- **smartInspect** (*http://www.gurock.com/smartinspect*)
- SmartInspect günlük kitaplığı, Java, Delphi ve .NET için AES şifreli günlük dosyalarını destekler.
- **CLoglt** (*http://www.codeproject.com/KB/files/logit.aspx?msg=686567*)

CLoglt uygulaması, şifreli günlük dosyalarının okunmasını ve yazılmasını sağlayan, **kaynak kodlu küçük bir kütüphanedir**. Windows için **C++** programlama dili ile yazılmıştır.

Güvenli günlük kaydını desteklemek için genel dosya sistemi **şifreleme yardımcı programlarını** kullanmak da mümkündür. Bu durumda, günlük dosyaları şifrelenmiş bir dosyaya veya disk birimine yazılır ve okunur. Bunlar aşağıdaki gibidir.

- **Windows XP Encrypting File System (EFS)**

(*http://support.microsoft.com/kb/307877*)

- Windows XP şifreleme dosya sistemi doğrudan Windows XP işletim sistemi tarafından desteklenen bir dosya veya klasör düzeyinde şifreleme özelliğidir.

- **TrueCrypt** (http://www.truecrypt.org)

  TrueCrypt, çeşitli işletim sistemleri için üçüncü taraf bir disk şifreleme aracıdır.

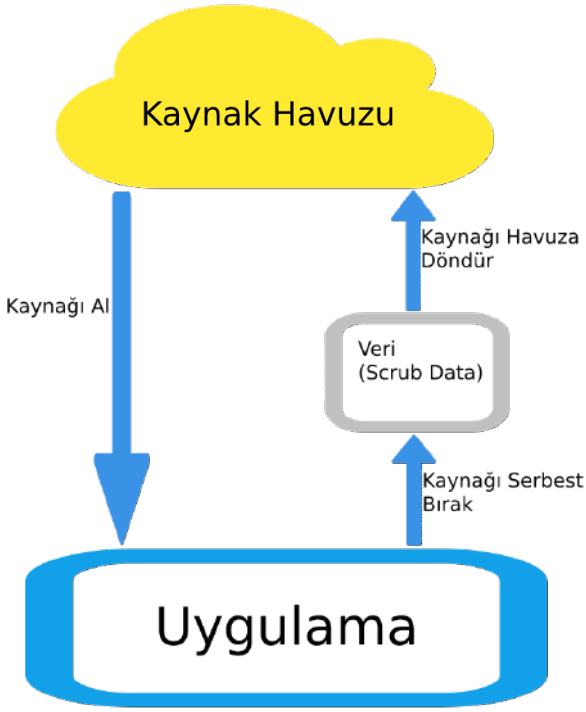
- **EncFS** (http://www.arg0.net/encfs)

b) Hassas Bilgileri Temizle Tasarım Deseni

Yeniden kullanılabilir bir kaynakta depolanan hassas bilgilere, yeniden kullanılabilir kaynağı serbest bırakmadan önce hassas bilgilerin silinmemesi durumunda **yetkisiz bir kullanıcı** veya **rakip tarafından erişilmesi mümkündür**. Bu desenin kullanımı, kaynak yeniden kullanmadan önce hassas bilgilerin yeniden kullanılabilir kaynaklardan **arındırılmasını** sağlar. Bu güvenli tasarım deseni "**yeniden kullanım için iade edilen yeniden kullanılabilir kaynaklarda depolanan hassas bilgileri temizle**" CERT Güvenli Kodlama önerisini temel alır.

Hassas bilgileri temizle (*Clear Sensitive Information*) tasarım deseni, temizlenmesi gereken yeniden kullanılabilir bir kaynak aracılığıyla hassas bilgilere erişimi önler. **Sistem**, **bellek**, **önbellek** ve **diskler** gibi kaynaklar aracılığıyla hassas verilere erişebiliyorsa bu desen kullanılmalıdır. Bir kaynak serbest bırakıldığında, hâlâ yetkisiz kullanıcılar tarafından erişilebilecek bilgilere sahip olabilir. Yeniden kullanılabilir kaynaklar **dinamik** olarak ayrılmış olan hafıza, **statik** olarak ayrılmış olan hafıza, **otomatik** olarak tahsis edilmiş bellek, hafıza önbellekleri, disk, disk önbellekleri gibidir.

Çoğu durumda, yeniden kullanılabilir bir kaynağı, dinamik olarak ayrılmış belleği **boşaltmak** veya bir **dosyayı silmek** gibi kullanılabilir kaynak havuzuna döndüren eylem, kaynağı **yeniden kullanım için** uygun olarak işaretler. Yeniden kullanılabilir kaynağın mevcut içeriği, kaynak yeniden kullanılıncaya ve kaynağa yeni veriler yazılana kadar bozulmadan bırakılır. Bu yetkisiz bir kullanıcının serbest bırakılmış bir kaynakta geride bırakılan eski bilgilere erişerek, potansiyel olarak **hassas bilgi sızıntısına yol açabileceği** anlamına gelir. Bu modelin ana odağı, yeniden kullanılabilir kaynağı mevcut kaynak havuzuna geri göndermeden önce tüm hassas verilerin silinmesini sağlamaktır. **Şekil 16-16**, hassas bilgileri temizle tasarım desenini göstermektedir.



Şekil 16-16: Hassas Bilgileri Temizle Tasarım Deseni Yapısı

Kaynak: Uygulama tarafından kullanılan **yeniden kullanılabilir** kaynak. Daha önce belirtildiği gibi kaynak dinamik olarak ayrılmış bir bellek parçası, diskteki bir dosya olabilir.

Uygulama: Ana uygulama, yeniden kullanılabilir kaynak havuzundan alınan bir kaynağa **hassas bilgileri** yazar.

Kaynak Havuzu: Uygulama tarafından kullanılan yeniden kullanılabilir kaynakların **kaynağını** ifade eder.

Fırçalama-Sıkıştırma Verileri: Uygulama, kaynağı kaynak havuzuna döndürmeden önce, yeniden kullanılabilir kaynaktaki **hassas bilgileri temizlemek için bazı mekanizmalardan faydalanmalıdır**.

Uygulama tarafından kaynak havuzuna döndürülen yeniden kullanılabilir bir kaynağa erişim hakkı kazanan yetkisiz bir kullanıcı, **hassas bilgileri okuyamaz**.

c) Güvenli Dizin Tasarım Deseni

Güvenli Dizin (*Secure Directory*) tasarım deseninin amacı, bir saldırganın, programın çalışması sırasında program tarafından kullanılan **dosyaları değiştirmesini önlemektir**. Bir program, program yürütme sırasında bir süre için bir dosyaya bağlı olabilir. Program geliştiricileri, genellikle program tarafından kullanılan dosyaların, programın yürütülmesi sırasında **dış kullanıcılar tarafından kullanılmayacağını varsaymaktadır**. Bununla birlikte, bu varsayımın yanlış olması durumunda, bir dosya birden fazla kullanıcı tarafından değiştirilebilir; bu kötü niyetli bir kullanıcının programın değiştirilmemiş hâlde kalmasına bağlı olarak programda bir yarış durumuna neden olması nedeniyle kritik bir süre boyunca dosyayı **değiştirebileceği** veya **silebileceği** anlamına gelir. Güvenli dizin tasarım deseni, program tarafından kullanılan dosyaların depolandığı dizinlerin, yalnızca **program kullanıcısı** tarafından yazılmasını sağlar.

Bu tasarım deseninin uygulanabilmesi için öncelikle dosyanın dizinindeki **kanonik yol** adı bulunur. Daha sonra kurallı yol adı tarafından referans verildiği şekilde dizinin **güvenli olup olmadığı** kontrol edilir. Güvenli dizin yapısı, dizinin kullanıcı ve süper kullanıcı ile sınırlı yazma izinlerine sahip olmasını sağlar. Diğer kullanıcılar güvenli dizindeki dosyaları değiştiremez. Ayrıca, ilgilenilen dizinden önce görünen tüm dizinlerin, diğer kullanıcıların güvenli dizindeki dosyaları **değiştiremez** ve **silemez**.

d) Yol Adı Kanonikleştirme Tasarım Deseni

Yol adı kanonikleştirme (*Pathname Canonization*) tasarım deseninin amacı, bir program tarafından okunan veya yazılmış tüm dosyaların herhangi bir sembolik bağlantı veya kısa yol içermeyen geçerli bir yoldan, yani **kanonik bir yoldan kaynaklanmasını** sağlamaktır. Verilen dizinlere giden yolun geçerli olmasını ve diğer konumlarla bağlantı kurmaması sağlanır. Geliştirilen bir uygulama eğer kullanıcıdan **yol adları kabul ediyorsa bu desen kullanılmalıdır**. Yol adı kanonikleştirme tasarım deseni dizinde gezinme güvenlik açıklarına karşı **koruma** sağlar. Sembolik bağlar ve diğer dosya sistemi özellikleri nedeniyle, bir dosya aslında yolla gösterilen dizinde bulunmayabilir. Bu nedenle, yol adında **dize temelli doğrulama yapılması** yanlış sonuçlara neden olabilir. **Doğru kurallı yol adının olması**, güvenli olup olmadığını anlamak için bir dizini denetlerken özellikle önemlidir.

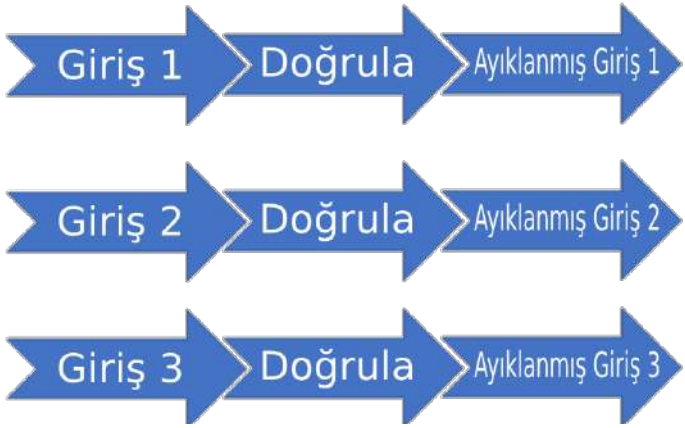
Programatik olarak yol adı kanonikleştirme tasarım deseninin yapısı, dosyayı açmadan önce verilen yoldaki işletim sistemi spesifik bir patika kanonikleştirme işlevinin çağrılmasını içerir. Kurallı yol adı **dosya üzerinde çalışırken** kullanılır. Kanonize edilmiş yol adının kendisi, sonuncu hariç kanonik yoldaki her öge, orijinal dizin olacak ve bir bağlantı veya kısayol olmayacak şekilde bir yapıya sahiptir. Son öge ise **orijinal dosyanın adı**dır.

e) Giriş Doğrulama Tasarım Deseni

Girdi verilerinin doğru şekilde doğrulanmasını sağlayarak birçok **güvenlik açığı önlenebilir**. Giriş Doğrulama, geliştiricilerin güvenilmeyen veri kaynaklarından gelen tüm hâricî girişleri doğru şekilde tanımlamasını ve doğrulamasını gerektirir. Giriş doğrulama (*Input Verification*) tasarım deseni, konunun girdiği verilerin geçerli olmasını ve kötü amaçlı metin içermemesini sağlar. Kullanıcı girişinin doğrulanamaması, sistemleri **SQL enjeksiyon** ve **taşma** (*overflow*) **saldırıları** gibi saldırılara maruz bırakabilir. Giriş Doğrulamayı uygulamak için geliştiricilerin hangi girişin güvenilmez olduğunu **tanımlamaları** ve kullanmadan önce bu girişi **doğrulamaları** gerekir. Giriş güvenilir sunucularda doğrulanmalıdır ancak sunucu ile iletişimi en aza indirmek için istemci tarafında da yapılabilir.

Bir uygulama tarafından doğrulanmamış kullanıcı girdisinin kullanılması, arabellek taşması saldırıları, SQL enjeksiyon saldırıları ve siteler arası komut dosyası çalıştırma saldırıları gibi birçok ciddi güvenlik açığının kök nedenidir. İstemci sunucu mimarisine sahip uygulamaların yaygınlığı göz önüne alındığında, sistem tasarımcılarının karşılaştığı sorunlardan biri, girdi doğrulamasının istemci tarafında veya sunucu tarafında gerçekleştirilmesidir. Giriş doğrulamadaki sorunlar **yalnızca istemci tarafında doğrulama yapıldığında** meydana gelir. İstemci tarafı doğrulamaları doğal olarak güvensizdir. Bir web sayfası gönderimini taklit etmek ve orijinal sayfadaki herhangi bir komut dosyasını atlamak genel olarak **kolaydır**. Ancak, istemci tarafındaki doğrulama kullanışlı olmasına rağmen güvenilmemelidir. Anında kullanıcı geri bildirimi yapmak, sunucuya tekrar bir **tur atılmasını önleyerek zamandan** ve **bant genişliğinden tasarruf sağlar**.

Giriş Doğrulama tasarım deseni, güvenilmeyen bir kaynaktan veri kabul eden herhangi bir yazılım için geçerlidir. Güvenlik sınırı boyunca bir program arabirimine ulaşan herhangi bir veri doğrulama gerektirir. Bu tür verilerin genel örnekleri *ortam, soketler, borular, dosyalar, sinyaller, paylaşılan hafıza* ve *cihazları* içerir. Web uygulamalarına özgü bazı girdi kaynakları, http formlarından `GET` ve `POST` parametreleridir. Giriş Doğrulama modelinin yapısı oldukça basittir ve yalnızca **Şekil 16-17**'de gösterildiği gibi güvenilmeyen her girişin **tanımlanması** ve **doğrulanması** gerekir.



Şekil 16-17: Giriş Doğrulama Tasarım Deseni Yapısı

f) Kaynak Toplama Başlatma Tasarım Deseni

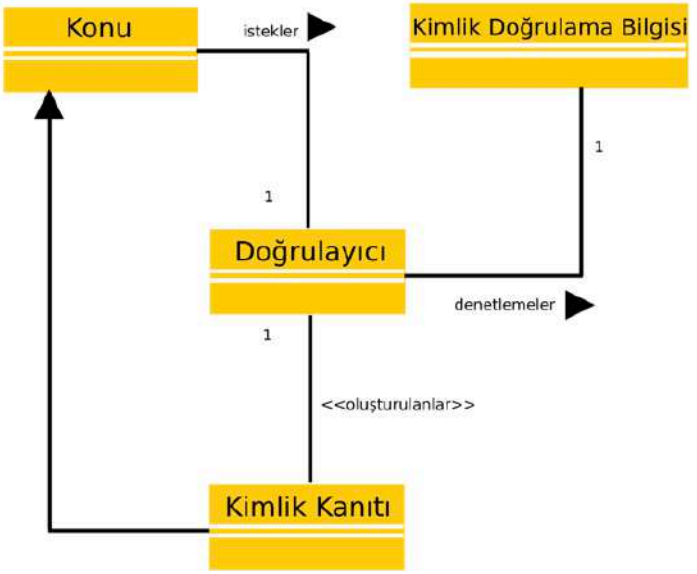
Kaynak toplama başlatma (*Resource Allocation is Initialize*) tasarım deseninin amacı, sistem kaynaklarının program yürütme yolları altında uygun şekilde **paylaştırılmasını** ve **dağıtılmasını** sağlamaktır. Kaynak edinme veya kaynak toplama başlatma tasarım deseni, bir nesnenin kaynaklarının istemciye güvenmek yerine yapıcı ve yıkıcı metotları kullanılarak doğru şekilde dağıtılmasını ve kaldırılması sağlar. Kaynak kullanımı, bellek yeniden tahsisini tetikleyecek kadar yoğun olduğu zamanlarda kullanılması gereken bir güvenlik tasarım desenidir. Bir sınıfı tekrar kullanmadan önce temizlenmesi gereken verileri ortaya çıkarabilen bu güvenlik tasarım deseni nesne tabanlı programlamada çöp toplayıcı (*literatürdeki bilinen adıyla Garbage Collection*) yöntemini çalıştırmaktadır.

Yazılım geliştirmede kaynak kullanımını **optimize edebilmek** ve **kaynak tüketimini önlemek için** kullanılan kaynakların zamanında serbest bırakılması gerekmektedir. Bu optimizasyon işleminde **henüz kullanılan kaynakların da bırakılmaması gerekmektedir**. Nesne tabanlı programlama konsepti ile güvenlik tasarım deseni yaklaşımının ortak paydası ise zamanından önce serbest bırakılan hafızanın kullanımının **güvenlik açığı** olarak kabul edilmekte oluşudur. Bellek tahsis sisteminde ne kadar çok hafıza talep edilirse sistem **boş olan hafızayı yeniden kullanıma açacaktır**. Bu kullanım ise orijinal hafıza serbest bırakıldıktan sonra yapıldığından, hafızaya kaydedilmiş verilerin üzerine yazılarak yapılmış olacaktır. Bu yüzden serbest kaynakların ne zaman kullanıma sunulacağı ile ilgili tasarım, yazılım geliştirilmesi esnasında planlanmalıdır. Yazılım geliştirme sürecinde bir kaynağın ne zaman gerekli olacağını ve ne zaman serbest bırakılacağını kestirmek zor olacağından kaynak toplama başlatma güvenlik tasarım deseni oldukça önemli bir desen durumundadır.

Kaynak toplama başlatma güvenlik tasarım deseninin kullanımına örnek vermek gerekirse program içerisindeki bir işlev hataya düşerse veya bir şekilde programı durdurursa program yine de **öncelikli olarak belleği boşaltacaktır**. Yani bir fonksiyonun başlangıcında hafızayı tahsis eden ve fonksiyon sona ermeden hafızayı serbest bırakan bir program çalışır.

g) Kimlik Doğrulayıcı Tasarım Deseni

Kimlik doğrulayıcı (*Authentication*) tasarım deseni, bir kişinin iddia ettiği kişi olduğunu doğrular. Bu güvenli tasarım deseni, farklı kullanıcıları barındırmak için farklı kimlik doğrulama algoritmalarının kullanılmasını desteklemektedir. Kimlik doğrulayıcı tasarım deseni, **güvenliği artırmak için kimlik doğrulama bilgilerini karmaşık bir yapı ile sağlar**. Doğrulayıcı, hassas bilgileri güvenli bir alanda saklarken farklı doğrulama algoritmaları gerektiren birçok kullanıcı türüne de cevap verebilmektedir. Algoritma karmaşıklığı, kullanıcının bildiklerinin kombinasyonlarına dayanan protokollerin kullanımına uyum sağlamak için değiştirilebilir. Kullanıcının ne türde olduğu, kullanıcının nerede olduğu ve sahip olduğu roller gibi kimlik doğrulama protokollerinin eklenmesi **işlemin yürütme süresini** ve **karmaşıklığı artıracaktır**.

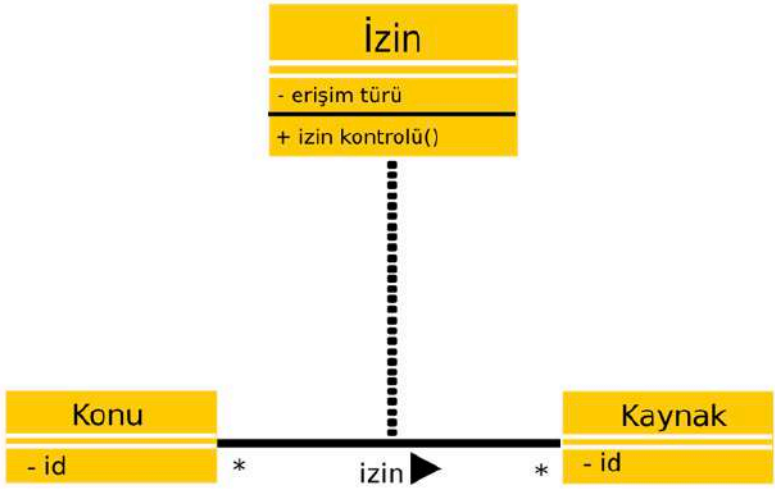


Şekil 16-18: Kimlik Doğrulayıcı Tasarım Deseni Yapısı

Şekil 16-18'de konu, kimlik kanıtı, doğrulayıcı ve doğrulama bilgisi katılımcıları ile kimlik doğrulayıcı tasarım deseni gösterilmektedir. **Konu**, istenen kaynaklara erişim verilmeden önce doğrulanması gereken varlıktır. **Kimlik kanıtı**, bir kez doğrulandıktan sonra konuya verilen nesne veya jetondur. **Kimlik Doğrulayıcı**, Konuyu ve Kimlik Kanıtını oluşturanın kimliğini doğrulamak için **Kimlik Doğrulama** bilgilerinin kullanan algoritmayı içeren nesnedir.

h) Yetkilendirme Tasarım Deseni

Yetkilendirme (*Delegation*) tasarım deseni, belirli bir kaynağa erişebilecek konuları ve aynı zamanda o kaynağa ait erişim haklarını tanımlar. Bu tasarım deseni, izinleri **konu** ve **kaynaklardan** ayırır. Çeşitli konular, kaynaklar ve ayrıcalıklarla ilgilenmek için yeterlidir. Bu yetkilendirme kuralları bir konu ve kaynakla ilişkilendirileceğinden güvenlik politikaları, yetkilendirme kurallarına dâhil edilecek konular için **imtiyaz belirleme ayarlarını** belirlemelidir.



Şekil 16-19: Yetkilendirme Tasarım Deseni Yapısı

Şekil 16-19'da Konu, Görev ve Kaynak katılımcılarıyla Yetkilendirme Güvenli Tasarım Deseni gösterilmektedir. Burada Konu, kaynağa erişmesi gereken varlıktır. Her bir İzin, **Konunun Kaynağa erişebildiği bir yolu** temsil eder ve erişimi kontrol etmek için mekanizmalar sağlar.

I) Kontrol Noktası Tasarım Deseni

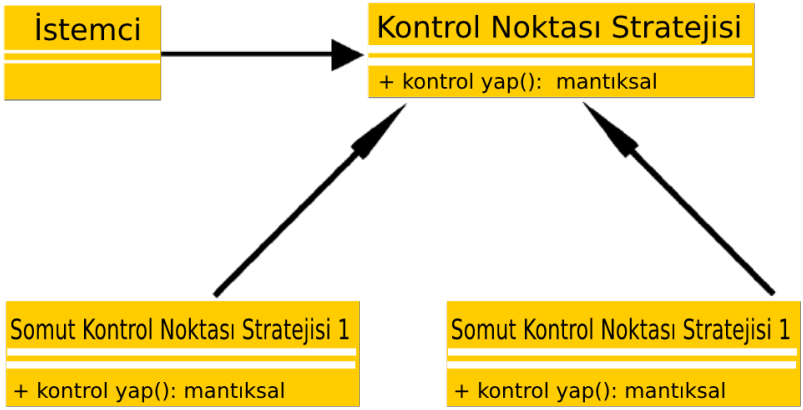
Kontrol noktası (*Check Point*) tasarım deseni, kullanıcıların **gizli bilgilere erişmesini önler** ve sistemin veri tabanına karşı **kötü niyetli davranışlara koruma sağlar**.

Kontrol noktaları, bir işletmenin güvenlik politikasını içine alan varlıklardır. Gamma'da açıklanan güvenli strateji fabrikası deseni kullanılarak **kontrol noktası deseni** uygulanabilir.

Güvenli strateji fabrikası tasarım deseninin bu kullanımı, bir sistemin çalışmasının farklı noktalarına farklı kontrol mekanizmalarının uygulanmasına izin vererek **güvenlik önlemlerinin müşteriye göre değişmesine izin verir**. Bir kontrol noktası örneği, sisteme giriş yapan bir kullanıcı olabilir.

Kontrol noktası güvenli tasarım deseni, bir kullanıcının *sisteme nasıl giriş yaptığı, kaç tane başarısız oturum açma girişiminin gerçekleştiğine* veya *günün saatlerine göre değiştirilebilir*.

Bu tasarım deseni, karmaşık güvenlik kontrollerini değiştirmeyi kolaylaştırmak ve uygulama durumuna bağlı olarak farklı stratejilerin kullanılmasına izin vermek için **tek bir sınıfa** ayırır. Bir kullanıcının kimliğini doğrulamak veya yetkilendirmek için kullanılabilir. Geçerli kullanıcı kimliği gibi genel bilgileri ayarlamak için **kontrol noktaları** kullanılabilir.



Şekil 16-20: Kontrol Noktası Güvenli Tasarım Deseni Yapısı

Şekil 16-20'de bir katılımcının kontrol noktası olduğu gösterilmektedir. Kontrol noktası, bir programın çalışma süresi boyunca herhangi bir zamanda kullanılmak üzere bir işletmenin güvenlik politikasını kapsar. Bu güvenli tasarım deseni, güvenlik mantığını normal **uygulama mantığından ayırır** ve **güvenlik kontrollerinin değiştirilebilir** ve **yeniden kullanılabilir olmasını sağlar**.

i) Bilgi Gizliliği Tasarım Deseni

Bilgi gizliliği (*information privacy*) tasarım deseni, verileri hırsızlıktan korumak amacıyla güvenli olmayan ortamlarda hassas verileri şifreler. Bu desen, veriler **iç veya dış sistemler arasında sık sık geçirildiğinde kullanılmalıdır**. Güvenlik politikaları, hangi verilerin şifreleneceğini ve bu verilerin ne kadar karmaşık bir şekilde şifreleneceğini belirlemek için verilerin **hassasiyet düzeyini** tanımlamalıdır. Şifreleme ve şifre çözme zaman alabildiğinden, mümkün olan yerlerde **kullanımlarını sınırlamak** en iyisidir. Bu örüntü, şifreleme ile nelerin gizlenmesi gerektiğini belirlemek için hassasiyet seviyesini kullanarak bu sorunu giderir.

Bu güvenli tasarım deseni, anahtar saklama mekanizmaları, şifreleme mekanizmaları ve şifreleme anahtarları ile algoritmalarını depolamak için **güvenli bir konum** gerektirir. Bu deseni kullanırken **hassasiyetini belirlemek** için **veriler sınıflandırılmalıdır**. Bu sınıflandırma, hangi içeriğin şifrelenmesi gerektiğini belirlemek amacıyla kullanılmalıdır.

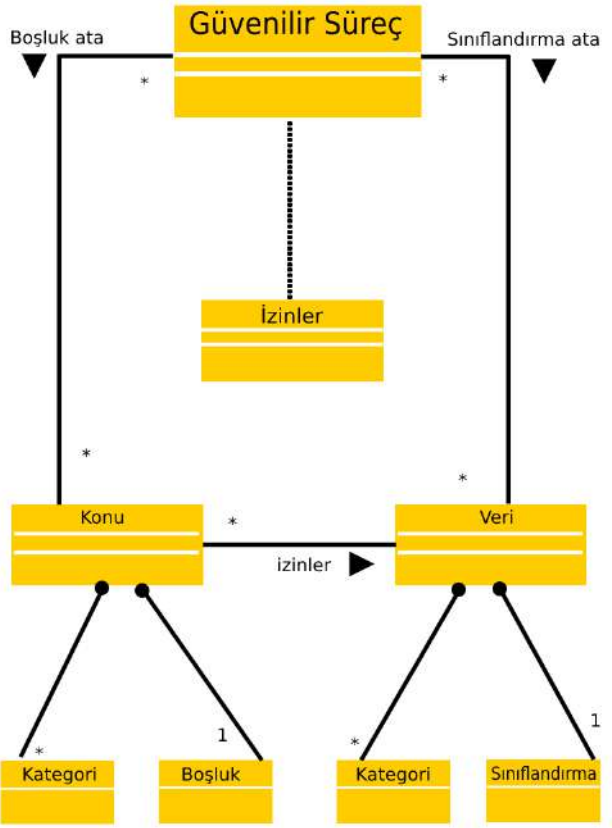
j) Sınırlı Görünüm Tasarım Deseni

Sınırlı görünüm (*Limited View*) tasarım deseni, yalnızca geçerli kullanıcının erişim için ayrıcalıklı olduğu işlevleri etkinleştirir. Bu desen, çoğu kullanıcı sınırlı sayıda işleme izin verdiğinde kullanılmalıdır. Bu desen kullanılırken, görünümü oluşturmadan önce **güvenlik kontrolleri** yapılır. Sınırlı görünüm, mevcut oturumun kullanıcı izinlerini kullanır ve mevcut kullanıcıya özel bir kullanıcı arayüzü oluşturur. Sınırlı görünüm tasarım deseni, kullanıcı arayüzünü oluşturmak için **yapıcı metotlar** kullanarak veya farklı görünümleri temsil etmek için bir durum makinesi kullanarak gerçekleştirilebilir. Sınırlı görünüm tasarım deseninin kullanımı, daha az güvenlik kontrolü ve daha az karmaşık bir kullanıcı arayüzü ile **daha temiz bir tasarımla** sonuçlanır.

k) Çok Düzeyli Güvenlik Tasarım Deseni

Çok düzeyli güvenlik (*Multi-level Security*) tasarım deseni, kullanıcıları ve verileri kabul edilebilir kullanım modellerine göre kategorilere ayırarak veri nesneleri için erişim izinlerini belirler. İhtiyaç duyulan bu sınıflamaları değiştirmek için **güvenilir süreçleri** kullanır. Kullanıcılar için sınıflandırmalar **boşluklar**, veriler için sınıflandırmalar ise **duyarlılık seviyeleri** olarak adlandırılır.

Sınıflandırmaların kullanıcılar ve veriler için kullanılması **sistemin gizliliğini** ve **bütünlüğünü** korur. Bu desenin kullanımı, izinlerin bir kuruluş içindeki rütbesine veya konumuna dayalı olduğu kuruluşlarla sınırlı olmalıdır.



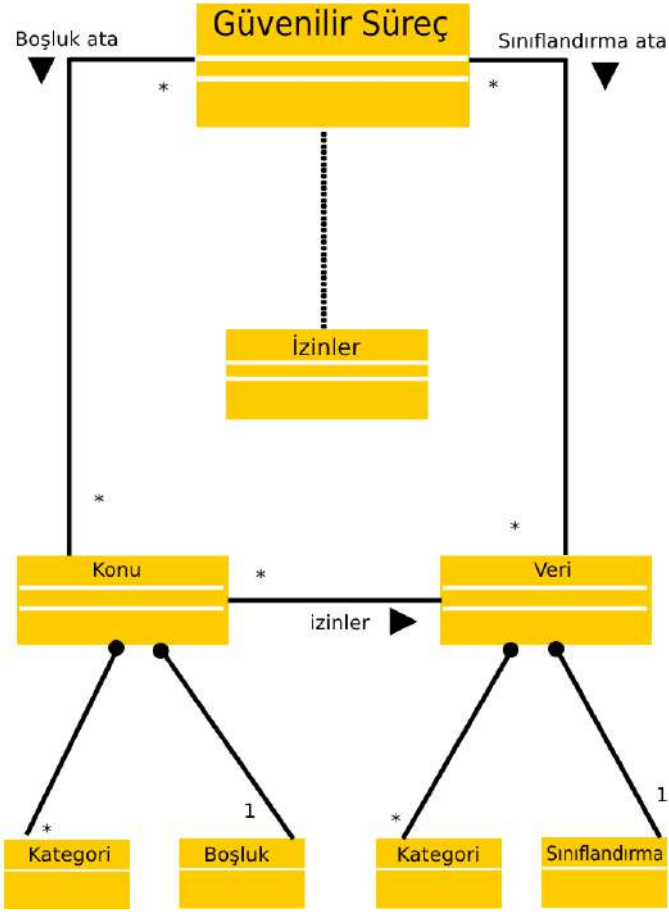
Şekil 16-21: Çok Düzeyli Güvenlik Tasarım Deseni Yapısı

Şekil 16-21'de yedi katılımlı ile Çok Düzeyli Güvenlik deseni gösterilmektedir. Katılımcılar, *Konu, Veri, Güvenilir Süreç, Kategori, Gümrükleme, Kategori* ve *Sınıflandırma* şeklindedir. **Konu**, ait olduğu organizasyon birimi için **Kategori** ve **Gümrükleme**'yi içerir. **Açıklık**, Konunun açıklık seviyesini temsil eder ve erişim izinlerini belirlemek için kullanılır.

Veriler ayrıca ait olduğu organizasyon birimi için **Kategoriyi** ve **Sınıflandırmayı** da içerir. Sınıflandırma, **Verilerin** hassasiyet seviyesini gösterir ve erişim izinlerini belirlemek için kullanılır. **Güvenilir Süreç**, Verilerin Konu ve Sınıflandırma düzeyinin Açıklık seviyesini değiştirebilen varlıktır.

l) Rol Tabanlı Erişim Kontrolü Tasarım Deseni

Role tabanlı erişim kontrolü (*Role Based*) tasarım deseni, sisteme atanan rollere dayanarak izinleri kullanıcılarla ilişkilendirir. Kullanıcıyı rollerle ve izinlerle ilişkilendirme, bireysel kullanıcıları **ayrı izin gruplarıyla ilişkilendirme çalışmasını ortadan** kaldırır. Bu desen, çok sayıda kullanıcı ile çok sayıda kaynağın ilgili erişim ayrıcalıklarını paylaştığında uygundur.



Şekil 16-22: Rol Tabanlı Erişim Kontrolü Tasarım Deseni Yapısı

Şekil 16-22'de dört adet katılımcı ile Rol Tabanlı Erişim Güvenlik Tasarım Deseni gösterilmektedir. Desenin yapısındaki katılımcılar **Kullanıcı**, **Rol**, **İzinler** ve **Veri Nesnesi** şeklindedir.

Kullanıcı, sistemin **geçerli kullanıcısını** temsil eden sınıftır. Rol, kullanıcının üstlendiği **mevcut rolü** temsil eder. **Veri Nesnesi**, kullanıcın erişmek istediği verileri temsil eder. Bir ilişkilendirme sınıfı, İzinler, rolün Veri Nesnesi için izinlerini belirler. Kullanıcının, Rolle **çoktan çoğa** bir ilişkisi olduğunu ve Rolün **Veri Nesnesi ile çoktan çoğa ilişkisi olduğu** unutulmamalıdır.

Bu desen, kullanıcıların yönetimi ve iş güvenliği için **iş yükünü azaltır** çünkü rollerden çok daha fazla kullanıcı vardır. Bu desen aynı zamanda kullanıcıların roller arasında **geçiş yapmasını** ve **uygulamanın kullanımını** kolaylaştırır.

m) Rol Hakları Tanımı Tasarım Deseni

Rol hakları tanımı (*Role Definition*) tasarım deseni, kullanım durumlarına ilişkin izinlerin verilmesine dayanarak "**en az imtiyaz**" ilkesini benimsemektedir. Bu desen, roller işlevsel görevlere karşılık geldiğinde kullanılmalı, "**en az ayrıcalık**" prensibini kullanarak izinler verilmelidir. Bu desende, bir rolün bir sistemde hangi işlemleri gerçekleştirebileceğini belirlemek için **sıra diyagramları** kullanılır. Bu işlemler daha sonra bir rolün **izinlerini belirlemek için** kullanılır.

Rol hakları tanımı, rol tabanlı erişim kontrolü gibi diğer rol tabanlı izin desenleriyle birlikte kullanılır. Bu desen, sorumluluklarına göre rollerin asgari izinlerini belirlemek için **kullanım durumunun geliştirilmesini** ve **dizi diyagramlarının analizini** gerektirir.

n) Roller Tasarım Deseni

Roller (*Roles*) tasarım deseni, birden fazla kullanıcı için geçerli olan güvenlik politikalarını tek bir varlıkta birleştirir. Bireysel kullanıcılar yerine ayrıcalıkların rollerle ilişkilendirilmesi, ortak ayrıcalıklara sahip kullanıcı gruplarını yönetmeyi kolaylaştırır. Rolleri uygulama stratejisi, kullanıcının birden fazla rolü olmasına ve birden fazla ayrıcalıklara sahip olması için bir role izin vermesidir. Bu uygulama, her rolün bir paydaşı **temsil etmesine** ve paydaşın **uygun ayrıcalıklarını yakalamasına** izin verir. Rol tabanlı kalıtım ilişkileri ayrıca rol hiyerarşisi oluşturmak için de kullanılabilir.

Roller güvenli tasarım deseninin bir sonucu, yöneticilerin, kullanıcı ayrıcalık ilişkileri yerine kullanıcı rolü ve rol ayrıcalık ilişkilerini yönetmeleridir. Birden çok kullanıcıyı bir rol ile ilişkilendirmek, kullanıcıları kişiselleştirilmiş **ayrıcalık kümeleriyle ilişkilendirmekten daha kolay**dır. Diğer sonuçlar arasında, sistem ayrıcalıklarını **gruplandırma** ve **yönetme** konularına uygundur. Bu durum sisteme fazladan karmaşıklık katma pahasına fayda getirmektedir. Rollerin iyi bilinen bir uygulaması UNIX'in sahip grup-diğer rol sistemidir. Roller ayrıca **aktörler**, **gruplar** ve **profesyoneller** olarak da bilinir.



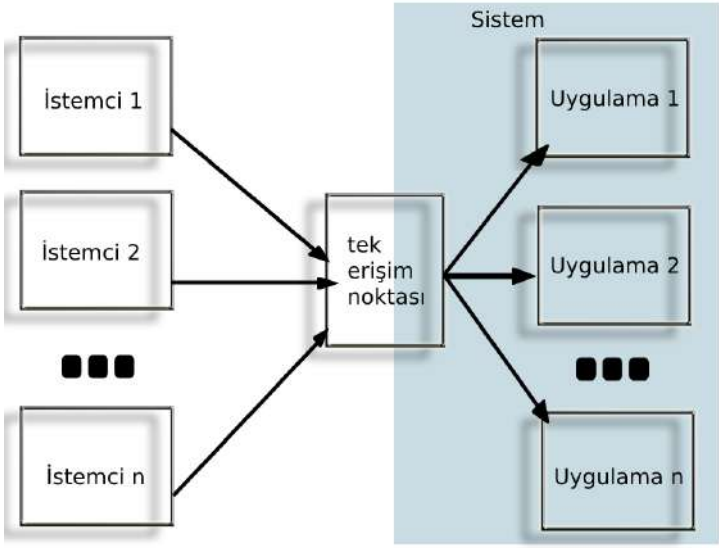
Şekil 16-23: Roller Tasarım Deseni Yapısı

Şekil 16-23'te üç adet katılımcı ile Roller Güvenli Tasarım Deseni gösterilmektedir. Katılımcılar; **Kullanıcı**, **Rol** ve **Ayrıcalık** şeklindedir. Bir Kullanıcı, Rolle ilişkilidir. Kullanıcı bir Rolle ilişkilendirildiğinde, bu kullanıcı **Rolle ilişkili Ayrıcalıklar kazanır**.

o) Tek Erişim Noktası Tasarım Deseni

Tek erişim noktası (*Single Access Point*) tasarım deseni, bir uygulamaya erişimi bir **giriş noktasına** sınırlar. Güvenlik gereklilikleri, çoklu arayüzleri destekleme ihtiyacı nedeniyle karmaşıklaştırmaktadır. Bu desen, kullanıcıları birden fazla giriş noktasında **doğrulama** ve bir uygulama boyunca **güvenlikle ilgili kodu çoğaltma ihtiyacını ortadan kaldırır**. Desenin yalnız giriş noktası, sistemde ihtiyaç duyulan bir kullanıcı hakkında tüm bilgileri toplamaktadır. Giriş noktası ayrıca tüm alt uygulamaları başlatmalı ve kullanıcıyı mevcut oturum için **gerekli olmayan bilgileri** girmesi için zorlayabilir.

Bu desen, sistemde bir kullanıcının oturum açması için kopyalanmış kodla sonuçlanabilecek diğer uygulamaların bir **bileşimi** olduğunda kullanılabilir. Birden fazla giriş noktası, kullanıcının sistemin farklı bölümlerine erişirken gereksiz bilgileri sağlamasını gerektiren farklı verilere ihtiyaç duyabilir. Bu daha büyük bir saldırı yüzeyiyle sonuçlanacağından, sistemi **daha az güvenli** hâle getirir ve daha kolay kontrol akışı sağlayarak sistemin **test edilmesini** kolaylaştırır.



Şekil 16-24: Tek Erişim Noktası Tasarım Deseni Yapısı

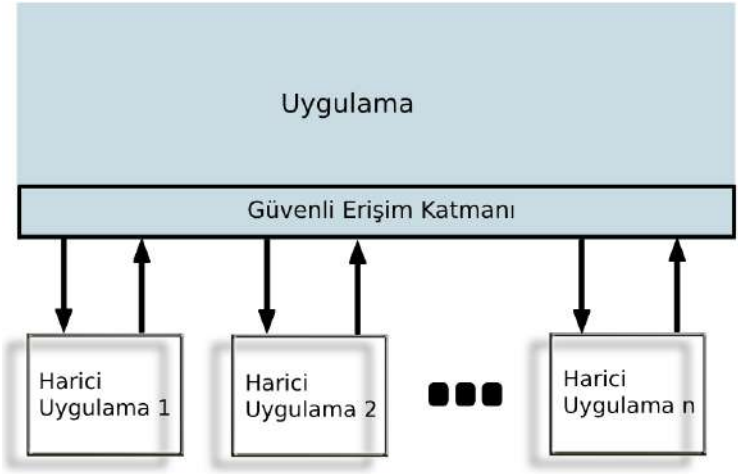
Şekil 16-24'te Tek Erişim Noktası Güvenli Tasarım Deseni gösterilmektedir. Şekilde *İstemciler, Tek Erişim Noktası, Sistem* ve *Uygulama* şeklinde dört katılımcı bulunmaktadır. Müşteri, **Sisteme erişmek isteyen kullanıcı veya süreçtir**. Bu, Tek Erişim Noktasından geçerek yapılır. Sistem, Müşterinin erişmek istediği uygulamadan oluşur.

p) Güvenli Oturum Tasarım Deseni

Güvenli oturum (*Secure Session*) tasarım deseni, bir uygulamada kullanılmak üzere, kullanıcının **kullanıcı adı** ve **rolleri** gibi genel olarak alakalı bilgileri **depolamak** için kullanılabilir. Oturum, genellikle web uygulamalarında bir kullanıcının istekler arasında bilgilerini depolamak için kullanılır. HTTP durumsuz olduğundan, Oturum kullanmak bir kullanıcının erişim kimlik bilgilerini yeniden girmek zorunda kalmadan uygulamayla **etkileşimde bulunmasını sağlar**. Güvenli Oturum, tüm bileşenlere sunulan ortak verileri depolamak için bir konum sağlar.

r) Güvenli Erişim Katmanı Tasarım Deseni

Güvenli erişim katmanı (*Secure Access*) tasarım deseni, hâricî sistemlerle iletişim kurmak için güvenli bir yöntem sağlar. Sistem entegrasyonu genellikle ilk sistemlerin **Uygulama Programlama Arayüzleri** (**API**) arasındaki sınırlarda **zayıflıklar** bırakır. Bu sınır noktalarının her iki tarafına da güvenlik kontrolleri uygulamak, entegre sistemini korumak ve test etmek için daha zor hâle getiren yinelenen kodlar verebilir. Bu desen, güvenli olmayan hâricî sistemlerde **arabirim yaparken** kullanılmalıdır.



Şekil 16-25: Güvenli Erişim Katmanı Tasarım Deseni Yapısı

Şekil 16-25'te gösterilen Güvenli Erişim Katmanı, farklı uygulamalar arasında iletişimi izole ederek sistemin bakımını ve testini kolaylaştırır. Güvenli Erişim Katmanı kullanılarak bir sistem **daha kolay bir şekilde farklı bir platforma taşınabilir**.

s) Güvenli Kanallar Tasarım Deseni

Güvenli kanallar (*Secure Channels*) tasarım deseni, özellikle ortak ağlar üzerinden iletişim kurarken istemciler ve sunucular arasındaki bağlantıların güvenli olmasını sağlar. Güvenli kanallar tasarım deseni, şifrelemenin ek masrafını azaltmak için **diğer verileri net bir şekilde iletirken hassas verileri şifreler**. Bu desen, istemci ve sunucunun **güvenli bağlantılar** kurmasını gerektirir.

Bu deseni kullanmanın faydaları, hassas olmayan verilerin değişimini etkilemeden güvenli iletişim uygulamak için mevcut teknolojinin kullanılması nedeniyle geliştirilmiş **güvenlik** ve **minimum geliştirme süresi**dir. Bu desenle ilgili sorunlar, şifreleme süresinden kaynaklanan düşük performans, müşteri iletişiminin **ölçeklenebilirliği** ve **artan maliyettir**.

WEB PROGRAMCILIĞI

17

Kitabımızın bu bölümünde **internet programcılığından** bahsediyor olacağız. İnternet programcılığının tarihine kısaca değinmek gerekirse aslında bana göre her şey **masaüstü programcılığı** ile başladı. Nesne yönelimli yazılımların yükselişe geçtiği 2000 yılından bu yana **web 2.0** ile beraber internet tabanlı programlar hayatımızın hemen her yerinde karşımıza çıkmaya başlamıştır. Nesneler ile masaüstü uygulamalarını **görsel açıdan yapmak kolaydır**. Ama söz konusu, bu görselliği web ortamına taşımaya gelince **Html**, **Css**, **Javascript** gibi pek çok yapıyla karşı karşıya geliriz. Web 2.0 yapısı ile birlikte hayatımıza giren etkileşimli web sitelerinin ilk örneklerini spor müsabakalarının sonuçlarının canlı olarak web sitelerine girilmesi süreçlerinde görmekteydik. Klasik web sitelerinde yönetim paneli, kullanıcı etkileşimi gibi özellikler olmadığı için yapılacak bir değişiklik için web sitelerinin kaynak kodlarına müdahale edilmesi gerekiyordu. Bu da hem **best practicise olarak karşılanmıyor** hem de **hatalara neden olabiliyordu**. Etkileşimsiz web sayfalarında aynı zamanda kullanıcı ile site yöneticisi veya diğer kullanıcılar arasında da bir etkileşim söz konusu değildi. Bugünün web sitelerine baktığımız zaman hemen hepsi bir şekilde kullanıcılar ile etkileşim hâlinde ve kullanıcının çeşitli bilgilerini işlemekteler. En güzel örnekleri sosyal medya sitelerine verebiliriz.



Şekil 17-1: İnternet Programcılığı

Yazılımları internet ortamına taşımak, tüm platformlardan erişilmesini sağlamak için mükemmel bir yöntemdir. Ancak günümüzün de konusu olan **güvenlik** unutulmamalıdır. Bir yazılımı kullanıcının bilgisayarına **setup** olarak yüklemek ile **internet ortamına taşımak** arasında çok ciddi güvenlik farklılıkları bulunmaktadır. Özellikle veri tabanı güvenliği ile ilgili olarak çeşitli aksiyonlar alınmaktadır. SQL Injection zafiyeti yaşamamak için çeşitli **ORM yapıları** ortaya çıkmış ve geliştiriciler tarafından hızla kabul görmüştür. Öte yandan sunucu taraflı güvenlik işlemlerini yapabilmek amacıyla web sunucu yazılımlarına (*IIS, Apache, Tomcat gibi*) güvenlik katmanları inşaa edilmiş ve SSL gibi sonradan sertifikasyon yüklemeleri ile güvenlik hususu sağlanmaya çalışılmaktadır. Elbette bir web sitesinin güvenliği yalnızca **sertifika** veya **veri tabanı işlemleri ile tamamen sağlanmış olmaz**. Önceki başlıkta yer verdiğimiz kodlama yapılırken uyulması gereken tasarım şablonları ve güvenli tasarım şablonları (secure design patterns) geliştirme esnasında bir standart benimsettiği için hem **yazılımın içinde boğulmayacağı kodlar** yazmasına neden olur hem de **güvenli bir desen sağlamış olur**. Güvenlik konusu ayrı bir disiplin olduğu için bu bölümde güvenliğin, internet kaynaklarından, kodlamadan, veri tabanı ve barındırma yöntemlerinden ve IPS/IDS gibi çeşitli yazılım veya cihazlarla alınan önlemlerle sağlanan çok geniş bir çalışma alanı olduğunu bilmemiz yetecektir.

Bir web sitesi geliştirirken öncelikle bir **tasarıma** ihtiyacımız vardır. Masaüstü programcılığında tasarım genellikle form üzerine konumlandırılan nesnelerle sağlanırken web tarafında iş biraz **grafikseldir**. Günümüzde her ne kadar insanlar hazır tasarımlara yöneliyor olsa da aslında bu işin **başlangıcı grafik tasarımdan** gelir.

Bizler bu bölüm okuyucuları olarak kendimizi **yazılım geliştirici** olarak tarif ettiğimiz için grafiğin temin edildiğini varsayacağız. Grafik tasarımı hem farklı bir disiplin hem de içerisinde kompozisyon, kullanıcı alışkanlıkları gibi süreçleri içerdiği için bir geliştiricinin aynı zamanda grafik tasarımcı olabileceğini söylemek benim hoşuma gitmiyor.

Photoshop gibi detaylı bir grafik tasarım programıyla yapılan sitelerin sayfaları, **tasarım aşamasından sonra** ne yazık ki **olduğu gibi kullanılamaz**. Bu statik resimlerin, **hareketli parçalar** hâline gelmesi gerekmektedir.

Örneğin Mouse ile üzerine gelince renk değiştiren bir buton, anasayfada bulunan bir resim geçiş slider nesnesi veya üzerine tıklanabilir bir buton oluşturmak için grafik tasarımcıdan gelen resimler üzerine **HTML** ve **CSS** giydirmesi işlemi yapılmalıdır. Hazır olarak template şeklinde temin edilirse bu iş zaten bizim için yapılmış olacaktır. Ancak elimizde bir tasarım resim formatında bulunuyorsa bunu HTML ve CSS giydirme dediğimiz yöntemle **dinamik hâle getirmek** gerecektir. Bu işlemi yapabilmek için başlıca yapılarımız **HTML**, **CSS**, **JavaScript** ve **çeşitli kütüphaneler**dir. Örneğin responsive yani tüm cihazlarda sorunsuz çalışabilen boyutlara sahip olmasını istiyorsak ki günümüzde cep telefonu kullanımının yaygın olduğunu düşündüğümüzde tasarımın **responsive** olması bir zorunluluktur. Böyle bir durumda **Bootstrap** kütüphanesinden faydalanırız. Bu kütüphane sadece farklı ekran çözünürlüklerine çözüm olmuyor aynı zamanda içerisinde çok güzel fontlar ve iconlar da barındırır. Herhangi bir web sayfasında **sağ tık** yaptıktan sonra **sayfa özelliklerini görüntüle** denirse üst kısımlardaki **<head> </head>** etiketleri arasında `bootstrap` kütüphanelerinin projede olduğunu belirten yollara (*path*) rastlarız.

Kabaca yer ettiğini düşündüğüm kavramların neler olduğuna tek tek değinelim. Ancak bu bölüm ayrı bir disiplin olması sebebiyle kitabımızda derinlemesine yer vermek hem konuyla ilgili profesyonel kitapları bulunan yazarlarımıza haksızlık etmemek amacıyla hem de kitabımızın konu bütünlüğünü dağıtmamak amacıyla **temel seviyede** yer vereceğim. Bu sayede bir **web kültürü** oluşmuş olacaktır.

HTML

Hyper Text Markup Language kelimelerinin baş harflerini birleştirerek telaffuz ettiğimiz (*heyçtiemel*) bir **programlama dili değil**, **işaretleme dilidir**. HTML ile web sayfalarımızın kullanıcıya tasarımlarda göründüğü gibi görünmesini sağlayan **işaretleme etiketleri** ile **kodlar yazarız**. Bu kodlar web **browser**'lar tarafından anlamlı yapılara dönüştürülür ve sonuç olarak bizlere web sayfalarının görüntülerini sunarlar.

HTML kodlarında her bir etiket **küçüktür** (<) işareti ve **büyüktür** işareti (>) arasına yazılır ve muhakkak **kapatılmak zorundadır**. Kapatma işlemi işlemi için **bölme işareti** (/) kullanılır.

Örneğin:

```
<html>
      ….…
 </html>
```

Programlama dillerinden aşina olduğumuz gibi **Türkçe karakter kullanılmaz** ve **büyük karakterlerde hoş karşılanmaz**. Ekrana **`anasayfa`**, **`hakkımızda`**, **`iletişim`** menüleri ile **`hoş geldiniz`** başlığını yazan **html kodları** aşağıdaki gibidir.

```
<html>
<head>
<title>ÖRNEK</title>
</head>
<body>
<h1>HOŞ GELDİNİZ</h1>
<ul>
<li><a href=”anasayfa.html”>anasayfa</a></li>
<li><a href=”hakkimizda.html”>hakkımızda</a></li>
<li><a href=”iletisim.html”>iletişim</a></li>
</ul>
</body>
</html>
```

Yukarıdaki kodları herhangi bir text editörde yazıp, **index.html** uzantısıyla kaydedip çalıştırdığınızda ekranda **menüler** ve **başlık** görüntülenecektir.

CSS

Yukarıdaki örneği çalıştırdıysanız eğer, ekranda tasarımsal herhangi bir şey olmadığını görmüşsünüzdür. Yani ne bir renk var ne bir yazı fontu tanımı ne de arka plan resmi veya menülerin üzerine gelince renk değişimi gibi şeyler. Bu tür işlemleri ham html dosyasının üzerine **CSS** ile yapmaktayız. CSS, **Cascading Style Sheets** kelimelerinin baş harflerinin kısaltmasıyla (***sieses***) olarak telaffuz ettiğimiz **stil şablonlardır**. Evet, Türkçe olarak **Stil Şablonları** şeklinde bilinmektedir.

CSS kodları, projelerde ayrı bir dosyada yazılabileceği gibi doğrudan html dosyaları içerisinde de yazılabilir. Örneğin bir linkin (** link adı **) yazı rengini değiştirmek için ayrı bir dosyada tüm *a etiketleri için şu rengi kullan* gibi bir tanım yapıp **HTML** tarafında **CSS özelliği ile** bu tanımı kazandırabilir veya doğrudan kod içerisinde bu işlemi gerçekleştirebiliriz. CSS işlemleri kullanıcının gözleriyle gördüğü **tüm etkileşimli tasarım hamleleridir** aslında.

```
<style>
.a {
Color: Red;
Font-family:Arial;
}
</style>
```

Yukarıdaki kodlar **ayrı bir CSS dosyası içerisinde yapılan tanıma örnektir**. Bu kodlar, herhangi bir text editöründe yazılıp **stil.css** uzantısı ile kaydedilir ve **index.html** içerisinde ilgili yere referans gösterilirse tüm <a> etiketleri bu tanımlamada yapılan özelliklerden **etkilenecektir**.

JAVASCRIPT

Web sitesinde *yazıları* ve *nesneleri* **HTML** ile oluşturabildiğimizi öğrendik, *tasarımsal dokunuşları* ve *iyileştirmeleri* de **CSS** ile kazandırabildiğimizi artık biliyoruz. Peki ama örneğin bir kutuya veya butona tıkladığımızda ekrana açılır bir mesaj kutusu çıkmasını sağlayan ya da anasayfalarda bulunan resim geçişi (*slider*) nesnelerindeki resimlerin **efektli** ve **bir kurala göre** (*örneğin: 1 saniye bir*) değişmesini sağlayan işlemleri *hangi platform ile yapacağız?* İşte, bu gibi işlemlerde karşımıza **JavaScript** çıkmaktadır.

Günümüz tarayıcılarının hepsi JavaScript'i destekler çünkü JavaScript tarayıcı üzerinde çalışmaktadır. Ancak yeni nesil teknolojilerde örneğin Node JS gibi kodlar tarayıcı, yani **client tarafında değil** sunucu tarafında **efektif** olarak çalışabilmektedir. JavaScript kodları da HTML ve CSS'de olduğu gibi **kendisine ait bir etiket alanında** çalışır. Bu alan `<script> </script>` şeklindedir. Çalışacağı işleme göre BODY etiketinde ya hemen önce tanımlanır ya da hemen sonra tanımlanır. Sayfaya yüklenme esnasında yapacağı işlere göre buna karar verilebilir. Javascript'i **en yaygın gördüğümüz yer** aslında bir web sayfasından çıktı almak istediğimiz zaman butona tıkladığımızda karşımıza gelen **yazdırma seçenekleri ekranının bize sunulması** işlemidir. Burada javascript `window.print()` gibi bir metot çalıştırmaktadır. Veya bir işlem yapmak üzere bir butona tıkladığımızda bize "**emin misin**" şeklinde uyarılar veren fonksiyon `confirm return('emin misiniz?');` şeklindedir.

Günümüzde farklı yeni nesil kütüphanelerle karşımıza hâlen gelmekte olan JavaScript, **eski ama güncel** bir dildir. Kullanabilmek için `<script>` etiketlerini HTML dosyasında **direkt gömülü olarak** çalıştırabileceğimiz gibi büyük projelerde karmaşıklık olmaması açısından hâricî bir dosya üzerinde yazılarak projeye **referans edilir**. JavaScript yapısı incelenirken **EcmaScript**, **TypeScript**, **Node.JS** ve **Angular.Js** yapıları da incelenmelidir.

JQUERY

JavaScript kodlarını **kısaltması** ve **az kodla çok iş yapması** gibi özellikleriyle bilinen Jquery, aslında bir **JavaScript yapısı kullanmaktadır**. Ancak gelişmiş özelliklerini kullanabileceği bir jquery kütüphanesine ihtiyaç duymaktadır. Tarayıcıların web sitelerini tarama üzerindeki davranışları da incelendiğinde özellikle Google'ın Ajax kullanması **jQuery kullanımını yaygınlaştıran** bir özelliktir. JQuery ile kullanıcılara çok **daha etkişelimli özellikler** sunabilirsiniz.

PHP

Hypertext Preprocessor kelimelerinin baş harflerini bir araya getirerek (*piheçpi*) olarak telaffuz ettiğimiz bu dil **açık kaynak kodlu** olması sebebiyle oldukça geniş bir kitle tarafından kullanılan ve geliştirilen **betik dil**dir. PHP, **C diline oldukça benzer** bir yapıdadır. PHP **web programlama dili**dir. Bir web sitesi geliştirileceği zaman tercih edilebilecek dillerden bir tanesidir. PHP çalışabilmesi için **Apache web sunucusuna ihtiyaç duymaktadır**. Wamp, Xamp veya Easy Php gibi yazılımlar ile ihtiyaç duyulan tüm paketler tek seferde kurulabilir. PHP **ücretsiz platform** sağlaması sebebiyle ve **kolay öğrenilebilirliği** ile çoğu geliştirici tarafından kabul görmüştür. Kütüphane kullanılmadan geliştirilen PHP uygulamaları için güvenlik bir sorun teşkil ediyor olsa da müdahale edilebilme oranı yüksek olduğu için günümüzde **hâlen efektif bir şekilde** kullanılabilmektedir.

AJAX

Ajax'ı anlatabilmek için çok klişe bir örnek vereceğim. Web sayfaları üzerinde **farklı farklı işlemleri gerçekleştiren bölümler bulunur**. Kullanıcı bir işlem yapacağında **tüm sayfaların yeniden yüklenmesi tercih edilen bir durum değildir**. Web sayfasının tamamının yüklenmesi yerine gerekli olan kısımlardaki veriyi **dinamik olarak almak** veya **geri göndermek** görevini üstlenir. Klişe örneğimize gelecek olursak; e-ticaret sitelerinde bulunan alışveriş sepetine yeni bir ürün eklediğimizde yalnızca sepet güncellenmektedir. Tüm sayfa güncellenmez.

Ajax: **Asynchronous JavaScript And XML** kelimelerinin baş harflerinden oluşur (*ayax, ajax*) olarak telaffuz edilir. Türkçe karşılığına eş zamansız **JavaScript** ve **XML** diyebiliriz. Ajax programlama dilleri ile sorunsuz anlaşır ve bu uyumluğu

ile kullanımı tercih sebebidir. Ajax'ı iyi kullanabilmek için **JavaScript bilgisinin yeterince iyi olması gerekmektedir**. Aynı şekilde Javascript bilgisi iyi olan birisi ecmascript, nodejs gibi yapıları da kolayca öğrenecektir. Ajax ile ilgili en yaygın örneklerden bir tanesi **kategori sistemleridir**. Örneğin ana ürünü seçince o ürüne ait alt ürünün gelmesi veya şehir seçince ilçelerin gelmesi gibi.

JSON

Veri değişim formatı olarak özellikle **servis tabanlı projelerde** sıklıkla kullandığımız bir yapı. JSON, Javascript Object Notation olarak bilinir ve bir notasyon olarak kabul edilir. JSON dataları okumak veya üzerinde işlem yapmak nispeten kolaydır. JSON dataları kullanmak veri alışverişi esnasında **boyut optimizasyonu** sağlamaktadır. Yapısal olarak C, C++ gibi dillere benzerlik gösterse de herhangi bir programlama diline **bağımlılığı yoktur**. JSON datalar **Number**, **Array**, **String**, **Object**, **Boolean** ve **Null** veri tiplerine sahip olabilirler.

XML

Veri iletişimini standart hâle getiren, platformlar arasındaki veri alış verişi sorunlarını ortadan kaldıran herkes için **ortak bir dil niteliğinde olan işaretleme dilidir**. **Extensible Markum Language** kelimelerinin baş harflerinden oluşan (*xemel, xamıl*) gibi telaffuz edilen genişletilmiş işaretleme dilidir. Oracle veri tabanı içerisindeki büyük verilerin XML olarak tutulabilmesinde, e-ticaret sistemlerinin karşılaştırma uygulamalarındaki alış verişlerde veya asp.net platformunda web.config dosyasında aslında **pek çok yerde XML görürürüz**.

ASP.NET

Bir başka web programlama dili ise Microsoft .Net platformu ile birlikte gelen **Asp.Net**'tir. Asp.Net, MS .Net desteğinden önce **klasik asp** olarak bilinmekteydi. MS .Net çatısı altında çok fazla geliştirildi ve en önemli özelliği web forms uygulamalarında masaüstü programcılığında kullanılan nesnelerin **web platformunda kullanılabilir yapısıydı**. Bu yapı ile masaüstü programcılığında nasıl bir form oluşturulup üzerine nesneler Toolbox üzerinden sürükle bırak ile bırakılabiliyorsa, asp.net web forms projelerinde de nesneler aynı şekilde **sürükle bırak** ile oluşturulabilmektedir.

Asp.Net, Visual Studio platformunda kullanıldığından **geliştirme arayüzü oldukça esnektir**. Hem tasarım görünümü hem de kod görünümü **aynı IDe üzerinden yönetebileceğimiz bir yapı** sunar. MS .Net Framework kütüphanelerini kullandığımız asp.net projelerine **çeşitli üçüncü parti bileşenler** de eklenebilir. HTML, CSS gibi işlemleri source code bölümünden yönetebildiğimiz teknolojide geliştirme için kullanacağımız programlama dili **C#.Net** veya **VB.Net** olabilir.

JSP

Java Server Page kelimelerinin kısaltmasıdır. Kısaca JAVA programlama dili kullanarak web projeleri geliştirebileceğimiz platformdur. Servlet teknolojisinin özelliklerini barındırdığı için uygulaması ve geliştirmesi **JAVA programlama dilini bilenler için oldukça kolay**dır.

ASP.NET MVC

MVC'nin bir **desen** (*pattern*) olduğuna kitabımızın önceki bölümlerinde değinmiştik. Asp.Net MVC, Asp.Net'e eklenen bir desendir. Microsoft tarafından eklenen bu desen, **Model View Controller** kelimelerinin baş harflerinden oluşmaktadır. Model katmanında kısaca veri tabanı ile ilgili işlemler gerçekleştrilirken, Controller katmanı; **istemciden gelen istekleri işler** ve **model ile view arasında sürekli bağlantı** hâlindedir. View ise kullanıcının browser'da gördüğü tasarım arayüz katmanıdır.

ASP.NET MVC CORE

Asp.Net ve MVC Microsoft çatısı altında olduğu için **ücretli** yazılımlardır. Ancak geliştiricilerin geliştirme ortamına lisans bedeli ödemek istememesi oldukça haklı bir sebep olduğu için **Microsoft açık kaynak kodlu bir geliştirme ortamını ortaya koydu**. .Net Core olarak bilinen bu platform sadece ücretsiz değil aynı zamanda **nesne gibi yüklerinden de kurtulduğu için performansla beraber gelmektedir**. PHP gibi dillerin performans gibi artılarını alarak, üstüne bir de Microsoft gücünü ekleyince ortaya çıkan ürün hiç şüphesiz ki günümüzün **en güncel** ve **en güçlü** web geliştirme platformlarından bir tanesi hâline gelmiştir. Container desteği de olan **.Net Core MVC** performanslı, **açık kodlu**, **Unit Test destekli**, **CSRF korumalı** ve **oldukça verimli** bir platformdur.

ADO.NET

ADO.NET, **ActiveX Data Objects** kelimelerinin kısaltmasıdır. Microsoft tarafından sunulduğu için **.Net** platformuyla gelmektedir. Veri tabanı ile program arasındaki bağlantıyı sağlayan aslında **köprü görevi** gören bir yapıdır. Örneğin Create, Read, Update, Delete gibi işlemleri ve SQL sorgularını bu yapı aracılığı ile gerçekleştirmekteyiz. ADO.NET ile veri sorgulama yapılırken aklımıza yalnızca Microsoft VTYS sistemleri, yani SQL Server veya Access gelmemeli **ORACLE** gibi veri tabanı sistemleri için de **ADO.NET** kullanabiliriz. SQL Server'da yazdığımız **SELECT * FROM…**'lu cümleleri programlama içerisinde yapabildiğimiz ve veri tabanına **bağlantı açıp kapatabildiğimiz yapıdır**. Microsoft Visual Studio platformuna örneğin C# dili ile geliştirme yaparken Ado.Net kullanılacaksa **Data.SqlClient** kütüphanesini eklememiz gerekmektedir. (*namespace*)

LINQ TO SQL

ADO.NET ile yapabildiğimiz veri erişim işlemleri **çok daha kolay hâle** getiren yapıdır. Linq ile SQL cümlecikleri değil **daha anlaşılır sorgular** yazabiliriz. Syntax olarak programlama diline yakındır. Veri erişim yöntemi yalnızca Linq To Sql Class değil, **Linq To XML** ve **Linq To Object** yöntemleri de vardır. Linq ve daha sonra gelişen ORM yaklaşımlarının her biri güvenlik temelli geliştirilmiştir. ADO.NET ile geliştirilen uygulamalarda veri tabanı açıklarından, SQL injection riskinden çok bahsedebiliriz ancak Linq ve sonraki teknolojilerde özellikle SQL injection zafiyetinden oldukça az bahsedebiliriz. Elbette kodlama tekniği de güvenlik için önemli bir unsurdur. Linq To Sql ile yazdığımız komutlar yine **ADO.NET** sağlayıcılarına iletilmektedir. Bu ifadeler veri tabanına iletilir ve sonuçlar **IQueryable** tipi ile **döner**.

Linq To Objects ile bellekteki nesnelerden oluşan koleksiyonlar sorgulanırken Linq To XML adın da anlayabileceğimiz gibi **XML verilerini sorgulamamızı sağlar**. Linq yapısı veri tabanı sorgularında **çok daha az sorgu ile çok daha fazla iş yapmamıza olanak sağlar**. Aynı zamanda Linq tek tırnak çift tırnak gibi programcının karıştırmasının yüksek olduğu sorgular içerisindeki özel ifadelerde de sorun yaşamaz çünkü sorgular **tırnaklar içerisine yazılmaz** doğrudan programlama syntax'ı ile yazılır. Tüm bunlar doğal olarak performansa olumlu katkılar sağlamaktadır.

ENTITY FRAMEWORK

ADO.NET altyapısını kullanan bir **ORM teknoloji**dir. Tasarım desenlerinden (*design pattern*) önceki bölümde bahsettiğimiz için söylemekte sakınca görmüyorum, **Entity Framework** Unit Of Work design pattern'ini yani **tasarım desenini** barındırmaktadır. İşlemler tıpkı Linq To Sql yaklaşımında olduğu gibi toplu bir şekilde yapılır. Yani ADO.NET'te olduğu gibi işlemler yapılırken o an veri tabanına yansıyan bir iletişim yoktur. Entity Framework ile birlikte gelen bir konsept daha veri tabanının oluşturulma şekli ve aşamasıdır. Örneğin veri tabanını diğer yaklaşımlarda olduğu gibi önceden SQL Server Management Studio ile oluşturmak yerine **Visual Studio** ile oluşturabilmekteyiz. Bu sisteme **Model First** denilmektedir. Diğer yaklaşımlarda ve tekniklerdeki gibi var olan yani önceden oluşturulmuş bir veri tabanı ile çalışmak istendiğinde ise **Database First** tekniği benimsenmektedir. Veri tabanı sorgu cümlelerinde artık SQL sorgu cümleleri değil de programlama dili ile yaptığımıza göre Code First yöntemi ile yani yeni veri tabanını **kod yazarak** da oluşturabilmekteyiz. Bu yönteme de **Code First** denilmektedir.

DAPPER

ADO.NET hızlıdır fakat sorgu sayısı arttığında geliştirici hızı olarak Linq ve Entity öne çıkmaktadır. Ancak çalışma performansını düşündüğümüzde Entity Framework yazdığımız sorguları veri tabanına gönderirken arka planda çeşitli dönüşümler yaptığı için **bir nebze yavaş kalabilmektedir**. Mikro düzeyde bir incelemeyle kıyaslandığında fark edilecek bir hız kaybından bahsediyoruz elbette. **Dapper**, performans denilince akla gelen **ORM yaklaşımlarından bir tanesidir**. Literatürde Mikro ORM olarak karşılık bulan Dapper, Nugget üzerinden **Visual Studio projelerine eklenebilmektedir**. Dapper, mapping yapmaz tek bir DLL'den oluşur bu yüzden oldukça **verimli** çalışır. Bir avantajı ise Entity Framework veya Linq To Sql kavramlarından aşina olduğunuz **kolay sorgu yazma mimarisine** sahiptir.

IIS



Microsoft tabanlı işletim sistemlerindeki sistemleri barındırmak için kurulan bir **web sunucu hizmeti**dir. **Internet Information Services** kelimelerinin baş harflerinden oluşmaktadır. **Bir site asp.net ile geliştirilmiş ise sunucusu IIS**'tir. IIS yönetim konsolundan alt alan adları eklenmesi, SSL sertifikası yüklenmesi, sertifika .csr kodu oluşturulması, sunucu ve uygulama güvenliğine dair işlemler, http isteklerini https olarak yönlendirme kuralları ve daha birçok kural, uzantılara veya belli türdeki isteklere (*GET, TRACE, POST*) **izin verme** veya **reddetme** gibi işlemlerin hepsi buradan gerçekleştirilir. Microsoft işletim sistemlerine özellikle server yani **sunuculara** kurulur.

APACHE



Açık kaynak kodlu bir web sunucusudur. IIS aksine *ücretsizdir* ve *asp.net teknolojisine sahip siteler barındırılmaz*. Bunun yerine PHP ile geliştirilen siteleri Apache sunucularda görürüz. **Linux** işletim sistemi üzerine kurulur. Hosting firmalarından hosting satın alırken de Linux sunucuların Windows sunuculardan **çok daha hesaplı** olduğunu görmekteyiz. Bunun nedeni Microsoft tarafındaki **lisanslama ücretleri**dir. Ancak günümüzde hybrid hosting paketleri satıldığı için bu fiyat farklı eskisi gibi görülmemektedir. IIS veya Apache ile çalışan hosting'lere sahipseniz C Panel veya Plesk panel gibi lisanslı panellerden kolayca yönetebilirsiniz. Tüm yapılandırma işlemini **.htaccess** dosyasından yapabiliriz.

TOMCAT



Apache tarafından geliştirilen bir **web sunucusu**dur. Tomcat; *Java Servlet ve Java Server Page veya WebSocket* gibi java tabanlı web uygulamalarını barındırmak için kullanılır. Amaç Joomle gibi siteleri barındırmak ise doğru adres **Apache** sunucusudur. Windows uygulamaları için *IIS, JSP* gibi **Java tabanlı** uygulamaları barındırmak için ise doğru adres **Tomcat** olacaktır.

Bazı Portlar

Bilişim hayatımızda karşımıza **en çok çıkacak olan portları** aşağıdaki tablodan inceleyebiliriz. Tabloda port numarasını, adını, hangi amaçla kullanıldığını incelebilirsiniz.

PORT Numarası	Port Adı	Açılımı	Ne İşe Yarar?
80	HTTP	HyperText Transfer Protocol	İnternet sitelerinin çalıştığı protokoldür.
443	HTTPS	HyperText Transfer Protocol Secure	http ile aynıdır ancak daha güvenlidir. TSL/SSL barındırır.
53	DNS	Domain Name System	IP-İsim Çözümleyici
67/68	DHCP	Dynamic Host Configuration	IP dağıtır.
25/465	SMTP	Single Mail Transfer Protocol	Mail server'da kullanılır.
110/995	POP3	Post Office Protocol	İstemci tarafındaki mail protokolüdür.
143/993	IMAP4	Internet Message Access Protocol	İstemci tarafındaki mail protokolüdür.
20/21	FTP	File Transfer Protocol	Dosya transfer protokolüdür.
22	SFTP	Secure File Transfer Protocol	FTP üzerinden çalışır. TLS/SSL barındırır. Güvenlidir.
161/162	SNMP	Simple Network Management Protocol	Ağda bulunan tüm aktif cihazlarla ilgili bilgi veren protokoldür.
3389	RDP	Remote Desktop Protocol	Windows uzak masaüstü bağlantı protokolüdür.
23	TELNET	-	İletişimin şifrelenmeden yapıldığı RDP gibi düşünebiliriz.

SSL Çeşitleri



Şekil 17-5: SSL

SSL teriminin açılımı, **Secure Sockets Layer**'dir. **Güvenli giriş katmanı** anlamında gelmektedir. SSL sertifikasına sahip siteler her ne kadar günümüzde fazla olsa da en çok **e-ticaret sitelerinde** görmekteyiz. SSL sertifikası olmayan bir e-ticaret sitesinden asla alış veriş yapmamalıyız. SSL sertifikasının olduğunu anlayabilmek için bakılacak ilk yer browser URL satırının hemen başındaki **https ibaresi** ve genellikle **yeşil bar**dır. Yeşil kilitli şekilde duran asma kilit, sitede **SSL sertifikasının yüklü olduğu anlamına** gelmektedir. Üzerine tıklayınca da sertifika hakkında bilgilere ulaşılır. Ne zaman biteceği, hangi sağlayıcıdan alındığı ve türü vb.

SSL ile istemci ve sunucu arasında **güvenli bir kanal** kurulur. Yapılan istekler tekrar döndüğünde açıkça gelmez. Örneğin Wireshark ile en temel ağ dinleme operasyonlarında http bir siteye yapılan bir istek sonucu dönen cevapta bilgi açıkça görülebilirken **HTTPS** ile yayın yapan bir sitede dönen cevap **şifreli dönecektir**. Sertifika satın alırken firma **.csr uzantılı bir kod istemektedir**. Bu kodu online oluşturuculardan bile temin edebilirsiniz ancak bu hiç **tavsiye edilmemektedir**. Genellikle web sunucu üzerinden oluşturulduktan sonra bu kod firmaya iletilir. Firma da satın alma süreci bittikten sonra SSL dosyalarını ilgili sistem yöneticisine mail üzerinden iletilir. Gelen dosyalar tamamen satın alırken belirtilen ihtiyaca göre değişir. Bu değişikliklerden dolayı SSL sertifikaları geldiği gibi kullanılmaz. Apache için farklı, IIS için farklı versiyonlara dönüştürülmesi gerekebilir. Örneğin Apache .key uzantılı dosyayı kullanıyor olabilir. IIS .pfx uzantılı veya .crt uzantılı kullanıyor olabilir. Özellikle wildcard

SSL dosyalarında bu dönüşümler yapılır.

SSL sertifikalarını türlerine göre inceleyecek olursak;

Domain türüne göre sınıflandırılan SSL sertifikaları;

Standart SSL → Yalnızca tek domainin olduğu sistemlerde kullanılır.

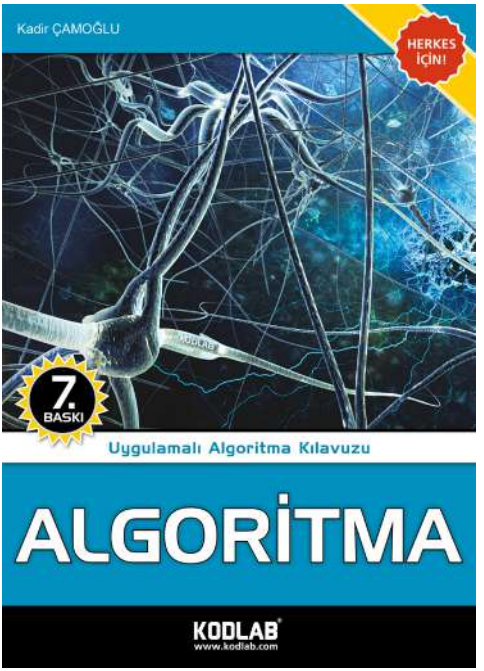
Multi Domain SSL → Birden çok domainin tek sertifikayla doğrulandığı sistemlerde kullanılır.

Wildcard SSL → Domainle birlikte alt alan adları yani subdomainlerin de doğrulandığı sistemlerde kullanılır.

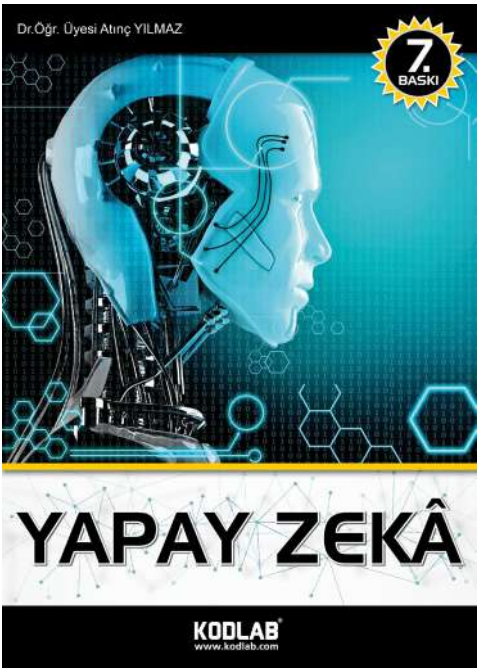
SSL sertifikalarını doğrulama türlerine göre inceleyecek olursak;

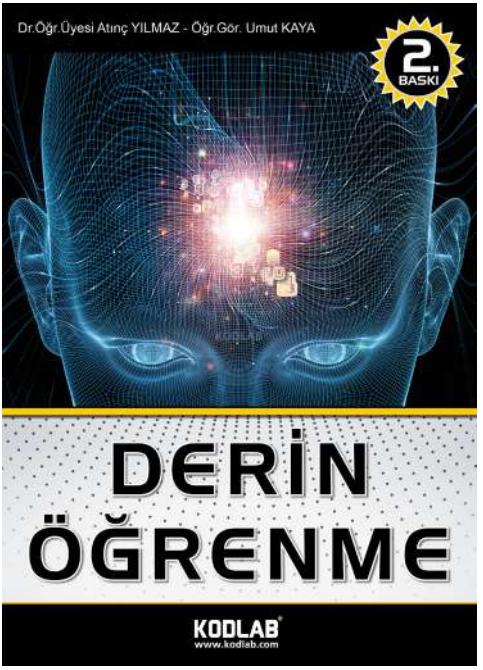
Domain Validation SSL → Yalnızca sunucu adını doğrulayan sistemlerde kullanılır.

Organizational Validation SSL → Sunucu ile birlikte sertifikayı satın alan kurum bilgilerini de doğrulayan sertifika türüdür.


Kadir ÇAMOĞLU
HERKES İÇİN!
7. BASKI
Uygulamalı Algoritma Kılavuzu
ALGORİTMA
KODLAB
www.kodlab.com


Dr. Bülent ÇOBANOĞLU
KODLAB.TV
OKU, İZLE, DİNLE, ÖĞREN!
C/C++
PROGRAMLAMA
İNTERAKTİF EĞİTİM VİDEOLARI KODLAB.TV 'DE!
KODLAB
www.kodlab.com


Dr.Öğr. Üyesi Atınç YILMAZ
7.
BASKI
YAPAY ZEKÂ
KODLAB
www.kodlab.com


Dr.Öğr.Üyesi Atınç YILMAZ - Öğr.Gör. Umut KAYA
2.
BASKI
DERİN
ÖĞRENME
KODLAB
www.kodlab.com