

## **Modül 1: Programlamaya giriş**

### **Hedefler**

- ➔ Program nedir?
- ➔ Programcı kimdir?
- ➔ Programlama Dilleri
- ➔ Programlama Dillerinin Tarihçesi
- ➔ Programın Derlenmesi

Bu modülde, bir programcının bilmesi gerek temel programlama kavramlarına giriş yapılacaktır. Bir programı oluşturan öğeler, çalışma süreci tamamlanana kadar geçtiği aşamalar ayrı ayrı işlenecektir. Bu kavramlar programcının ve programlama dillerinin tanımlanmasına yardımcı olacaktır.

Bu modülün sonunda:

- Bir programın çalışma prensibini açıklayabilecek,
- Programcı kavramını tanımlayabilecek,
- Değişik programlama dillerinin gelişimini açıklayabilecek,
- Derleme işlemini tanımlayabileceksiniz.

## Konu 1: Program nedir?

- Bilgisayarın, bir işi yapması için tasarlanan komutlar zinciri
- Program Türleri
  - Sistem Programları
  - Sürücüler (Driver)
  - Uygulamalar

Günümüzde bilgisayarların kullanım alanları büyük ölçüde artmıştır. Dolayısıyla işlerimizi daha hızlı ve düzenli bir şekilde yapmamız, bilgisayarları ne kadar iyi kullandığımıza bağlıdır. Bunun için ise, çeşitli amaçlara göre yazılan programları kullanırız.

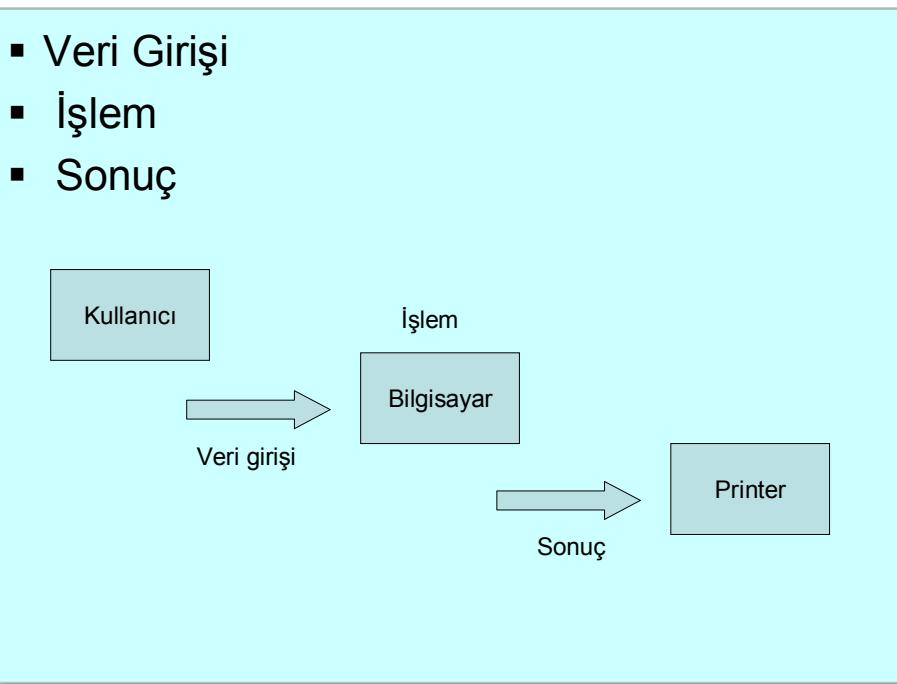
Program, bilgisayarın belli bir işi yapması için tasarlanan komutların tümüdür. Kullanım amaçları ve yerlerine göre birçok değişik program türü vardır:

- Sistem programları  
Her program, bir işletim sistemi üzerinde çalışır. İşletim sistemi, diğer programların çalışması için gerekli olan kaynakları ve ortamı sağlar.
- Sürücüler (Driver)  
İşletim sistemi ile donanım aygıtları arasında iletişim sağlayan programlardır. Klavye ile yazıların algılanması için, klavyenin sürücü programı kullanılır.
- Uygulamalar  
İşletim sistemi üzerinde çalışan, kullanıcıların ihtiyaç duyduğu işlevleri sağlayan programlardır.

Bir internet sitesini gezmek istediğimizde, Internet Explorer tarayıcısı kullanılabilir. Bu uygulama, işletim sistemine sitenin istenilen sayfadaki yazı ve resimleri almasını ister. İşletim sistemi, ağ kartıyla (Ethernet) sürücü programı sayesinde internet sitesinin sunucusuna isteği gönderir.

(Ref: MOC 2667 Introduction to Programming - Module 1 - sayfa 3)

## Programların çalışma modeli



Programların kullanılmasındaki amaç, girilen bilgilerin işlenip istenilen şekilde sonuçların üretilmesidir.

- Veri girişi  
Program, kullanıcıların veri girmesi ile başlar. Girilen veriler daha sonra işlenmek üzere hafızada saklanır.
- İşlem  
Veriler, programın yazılma şekline göre bir dizi işleminden geçirilir.
- Sonuç  
İşlenen veriler kullanıcıya aktarılır.

Programlar, belli kurallar çerçevesinde yazılır. Bu yazım kuralları sayesinde bilgisayar, programın işleyişini anlar ve gerekli sonuçları çıkartır. Yazılan programlar, belirtilen yazım kuralları kontrol edilerek derlenir. Bu derleme işlemi

sonunda, yazılan kaynak kodlar bilgisayarın anlayacağı tek dile çevrilir. Makine dili denilen bu dil, sadece 1 ve 0 sayılarından oluşmaktadır.

Örnek: ATM makinesinden para çekmek

1. Kullanıcı ATM makinesine kartını yerleştirir
2. Şifresini girer.
3. ATM cihazında çalışan uygulama kartta yazan bilgileri okur
4. Şifre kontrolü işlemi yapılır.
5. Şifre doğru girilmişse kullanıcı çekmek istediği miktarı girer.
6. Bankadaki hesap kontrol edilir.
7. Uygunsa kullanıcıya ödeme yapar.

## Konu 2: Programcı Kimdir?

- Belirli işlevlere sahip programlar geliştirir
- Kullanılan teknolojiyi, platformu iyi tanımıası gereklidir
  
- Programcı türleri
  - Mimar
  - Geliştirici
  - Test Mühendisi

Programcı, belirli işlevlere sahip programlar geliştirebilen uzmanlardır. Bir programcının, üzerinde çalıştığı platformu, kullandığı teknolojileri iyi tanımı ve bilgisayarın anlayacağı mantıksal dilde düşününebilmesi gerekmektedir. Programcılar çoğun genellikle aynı işi gerçekleştirse de, üstlendikleri görevlere göre üç gruba ayrılabilir:

- Mimar

Programların yazılması için gerekli teknolojileri belirleyen, gerekli durumlarda programın daha kolay yönetilmesi için küçük parçalara ayıran programcidir.

- Geliştirici  
Programı yazan kişidir.
- Test mühendisi  
Programın geliştirilmesi aşamasında, hatanın kaynaklarını bulan ve geliştiricilere raporlayan programcidir.

(Ref: MOC 2667 Introduction to Programming - Module 1 - sayfa 9)

## Konu 3: Programlama Dilleri

- Programcı ile bilgisayarın haberleşmesini sağlar
- Programlar 1 ve 0 sayılarından oluşan makine diline çevrildikten sonra çalıştırılır
- Programlama Dilinin özellikleri:
  - Sözdizimi (Syntax)
  - Gramer
  - Semantik
- 2500'den fazla programlama dili mevcuttur.

Dünyada konuşulan her dilin amacı iletişim sağlamaktır. Farklı kültürlerden insanların anlaşabilmesi için ortak konuştuğu bir dil gereklidir. Programlama dillerinin amacı da bilgisayar ile programcının haberleşmesidir. Programcı, bilgisayara hangi komutların çalıştırması gerektiğini bilgisayarın anlayacağı dilden konuşarak söyler.

Bilgisayarda, programlar makine diline çevrildikten sonra çalışır. 1 ve 0 sayılarından oluşan bu makine dili, en alt seviye dildir. Dolayısıyla programların bu dilde yazılması oldukça zordur. Programcılar konuşma diline daha yakın, kolay anlaşılabilecek diller kullanmaktadır. Bu dillere yüksek seviye programlama dilleri denir. Programlama dillerinin seviyeleri makine diline yakın olup olmaması ile ölçülür.

Bir programlama dili şu unsurlardan oluşur:

- **Söz dizimi (Syntax)**

Bir dil, kendine ait kelimeler ile konuşulur. Programlama dillerinin de benzer bir davranışı vardır. Programlama dillerindeki bu kelimeler, programlama dilinin anahtar kelimeleridir - komutlardır.

- **Gramer**

Programlama dillerini kullanmak için sadece kelimeleri bilmek yeterli değildir. Eğer anlamlı bir şekilde bir araya getiremiyorsa, bu kelimeler hiçbir anlam ifade etmez.

- **Semantik (anlamsal)**

Bir dili, kelimeleri doğru bir gramer kullanımı ile bir araya getirerek kullanabiliriz. Ancak konuşulan kelimelerin ne için kullanıldığı da önemlidir. Bir programlama dilinin özelliklerinin nasıl ve ne için kullanıldığı da, bu dilin semantiğidir.

Örneğin bir finans programı, Yeni Türk Lirası cinsinden bir miktarı dolara çevirecektir. Yapılacak işlem o andaki parite değerini merkez bankasından çektiğten sonra, girilen miktarı bu değerle çarpıp kullanıcıya gösterecektir. Kullanılan programlama dili ÇARP, GÖSTER, EŞİTLE komutları ile bu işlemi gerçekleştirecektir.

ÇARP EŞİTLE GÖSTER miktar parite sonuç

Bu şekilde yazılan program söz dizimi açısından doğrudur. Girilen veriler ve komutlar dışında, programlama dilinin anlamayacağı bir kelime kullanılmamıştır. Ancak komutlar yanlış sırada kullanılmıştır. ÇARP komutu hangi sayıları çarpması gerektiğini bilemeyecektir.

parite EŞİTLE sonuç ÇARP miktar

GÖSTER parite

Komutları ve değişkenleri, programlama dilinin gramerine göre doğru yerlerde kullanmamız gereklidir. Bu şekilde kullanılan komutlar doğru bir şekilde çalışır. Fakat GÖSTER komutunun ne için kullanıldığı yani semantiği de önemlidir. İstenilen, miktar ile pariteyi çarpmak, sonuca eşitlemek ve sonucu gösterecektir.

sonuç EŞİTLE miktar ÇARP parite

GÖSTER sonuç

(Ref: <http://www.cs.sfu.ca/~cameron/Teaching/383/syn-sem-prag-meta.html>)

Şu ana kadar 2500'den fazla programlama dili yazılmıştır. (Ref: [http://www.oreilly.com/pub/a/oreilly/news/languageposter\\_0504.html](http://www.oreilly.com/pub/a/oreilly/news/languageposter_0504.html))

Bunlardan bazıları Pascal, Basic, C, C++, Java, Javascript, Cobol, Perl, Python, Ada, Fortran, Visual Basic .NET, Microsoft Visual C# programlama dilleridir.

Yüksek seviye programlama dillerine Visual Basic .NET, Microsoft Visual C++ dillerini örnek verebiliriz. C ile işletim sistemi yazılabilirliğinden daha alt seviye bir dil olarak değerlendirilir.

### Programlama Dillerinin Tarihçesi

- Makine dili 10110110, 11011110
- Yordamların (Subroutine) ve Kütüphanelerin (Library) oluşması
- 1957 FORTRAN
- 1959 COBOL
- 1968 Pascal
- 1972 C
- Nesneye Yönelik Programlama Dilleri:  
C++, JAVA
- 2000 .NET  
Visual Basic .NET, Visual C#

Bilgisayarlar, icat edilmeleriyle birlikte belli bir iş yapmak için bir dizi komutlara ihtiyaç duymuşlardır. En başta çok basit işlemler yapan bu komutlar zamanla nesneye yönelme (object orientation) gibi ileri seviyede özellikler kazanmıştır.

İlk programlama dilleri, bilgisayarların üzerinde bazı araçların yerlerini değiştirerek veya yeni bileşenler eklenerek yapılmıştı. Programın işlemesi için bir devinime ihtiyaç vardı. Eskiden programlar fiziksel olarak yazılıyordu. Daha sonra fiziksel programlama yerini elektrik sinyaline bıraktı. Artık, kurulan elektronik devrelere düşük ya da yüksel voltajda akım gönderilerek bilgisayarın davranışını belirlenmeye başlandı. Yüksel voltaj 1, düşük voltaj 0 sayılarını ifade ediyordu. Böylelikle bugün de kullanılan makine dilinin ortaya çıkması için ilk adımlar atılmış oldu.

Ancak bu şekilde programlar yazmak, sistemi oluşturan elektronik devrelerin her program için baştan kurulmasını gerektiriyordu. Böylelikle programlar bazı

kavamlar çerçevesinde yazılmasına başlandı. Öncelikle bilgisayar donanımı her program için baştan kurulmamalı, bunun yerine basit bir donanımın üzerine yazılan komutlar kullanılmalıdır. Daha sonra, programlar tek bir komutlar zinciri yerine, küçük parçalar halinde yazılmalıdır. Bu parçaların programın içinde defalarca kullanılabilmesi yordam (subroutine) kavramını ortaya çıkarmıştır. Bu modelin kullanılması ise mantıksal karşılaşmaları, döngülerin kullanılmasını ve yazılan kodlar tekrar kullanıldığı için kütüphane (library) mantığını ortaya çıkarmıştır.

1957 yılında IBM, düşük seviye (makine diline yakın) bir programlama dili olan FORTRAN dilini ortaya çıkardı. FORTRAN ile beraber basit mantıksal karşılaşmalar, döngüler, (true-false) lojik ve (integer, double) sayısal değişkenler kullanılmaya başlandı.

1959 yılında, bu programlama dilinin özelliklerini alıp, giriş çıkış (Input – Output IO) gibi yeni işlevler sağlayan COBOL dili ortaya çıktı. Daha sonra 1968 yılında, COBOL ve FORTRAN dillerinin en iyi özelliklerini alarak Pascal ortaya çıktı. Ayrıca Pascal dili, hafızadaki adresler üzerinde işlem yapmaya olanak veren işaretçi (pointer) kavramını beraberinde getirdi.

1972 yılında C, Pascal dilindeki birçok hatayı gidererek ortaya çıktı. C dili ilk defa Unix işletim sistemini yazmak için kullanılmaya başlanmıştır. C, düşük seviye bir dil olması, kuvvetli giriş çıkış işlemleri sağlama gibi birçok özelliği ile işletim sistemleri yazılımasında tercih edilmiştir.

Bütün programlama dilleri birçok özelliğe sahip olmasına rağmen, modüler programlamanın birçok eksikliğini gidermek amacıyla, yeni bir programlama modeli olan nesneye yönelik programlama - OOP (object oriented programming) ortaya çıkarıldı. C dilinin ve OOP modelinin tüm özellikleriyle C++ dili oluşturuldu.

C++ dilini, Sun Microsystems tarafından çıkartılan Java takip etti. Java dilinin kullanım alanları, nesneye yönelik bir programlama dili olması ve beraberinde getirdiği çöp toplama GC (garbage collection) gibi performans artırmacı özellikleri ile büyük ölçüde genişledi.

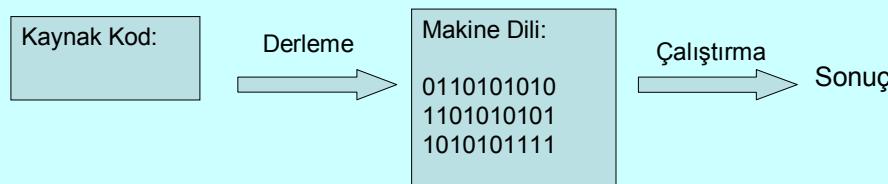
Microsoft, 2000 yılında .NET platformunu sunarak, otuzdan fazla programlama dilini aynı çatı altına topladı. VisualBasic.NET ve VisualC# .NET platformunu kullanan günümüzdeki en güçlü yüksek seviyeli programlama dilleri arasında yer almışlardır. .NET platformu hakkında daha detaylı bilgi için Modül 2'ye bakın.

(Ref:

[http://www.princeton.edu/~ferguson/adw/programming\\_languages.shtml](http://www.princeton.edu/~ferguson/adw/programming_languages.shtml)

## Konu 4: Programın Derlenmesi

- **Programlama dili derleyicisi:**
  - Gramer ve söz dizimi kontrolü
  - Kaynak kodların makine diline çevrilmesi
- **Makine diline çevrilen kodların çalıştırılması**



Programlar yazıldıktan sonra, çalışmaya uygun hale getirilene kadar bir dizi işleminden geçer. Bu işlemi gerçekleştiren, programlama dilinin derleyicisidir. (Compiler)

- Programlar, bir programlama dilinin gramer ve söz dizimi yapısına uygun bir şekilde yazıılır.
- Yazılan kodlar o dilin derleyicisi tarafından kontrol edilir.
- Kontrol işleminden sonra, bu kodlar bilgisayarın anlayacağı makine diline çevrilir. Ancak bir yazım ve ya mantık hatası varsa, programcıya gerekli hata mesajını vererek derleme işlemi iptal edilir.
- Makine diline çevrilen kodlar çalıştırılır.

## Modül Sonu Soruları & Alıştırmalar

### Özet

- ↳ Program nedir?
- ↳ Programcı kimdir?
- ↳ Programlama Dilleri
- ↳ Programlama Dillerinin Tarihçesi
- ↳ Programın Derlenmesi

1. Var olan bir metin dosyasını (.txt) görüntülemek için Notepad programını kullanabiliriz. İşletim sistemi, dosyayı kullanıcılarla göstermek için monitör ile iletişim kurar. Monitör işletim sisteminden gelen verilerle gerekli görüntüleme işlemlerini yapar.

Bu senaryodaki program çeşitlerini belirtin.

2. Bir arkadaşımıza e-posta yollamak istediğimizde, e-posta adresi, konu, mesaj bilgilerini gireriz. Daha sonra e-posta uygulaması mesajımızı verilen adrese yollar.

Programın çalışma modelinin aşamalarını belirtin.

3. C dilini kullanarak yazdığınız kodların bilgisayar tarafından çalıştırılabilir hale gelmesi için hangi aşamaların gerçekleşmesi gereklidir?

## **Modül 2: Microsoft .NET Platformu**

Microsoft .NET, uygulama geliştiricilerin yazılım geliştirme sürecinde altyapı işlemleri için harcadığı eforu en aza indirmek ve daha güvenli, güvenilir ve sağlıklı uygulamalar geliştirebilmelerini sağlamak için geliştirilmiş altyapıdır.

Bu modülü tamamladıktan sonra

- Microsof.NET platformu hakkında genel bilgi sahibi olacak,
- .NET Framework ve bileşenlerini açıklayabilecek,
- Microsoft .NET platformunun yazılım geliştiricilere sunduğu avantajları tanımlayabileceksiniz.

## **Konu 1: Yazılım Geliştirme Dünyası**

Microsoft 1975 yılında Bill Gates ve Paul Allen tarafından kurulduğunda vizyonu “Her eve, her masaya bir PC” idi. Donanım ve yazılım alanlarındaki gelişmelerin hızı ve birbirlerini sürekli tetiklemesinin sonucunda bilgisayar kullanıcı sayısı hızla arttı. Artan kullanıcı sayısı beraberinde yeni gereksinim ve talepleri ortaya çıkardı. Bu taleplerin doğal sonucu olarak da farklı platformlar ve farklı servis sağlayıcıları ortaya çıktı. İletişim, finansal hizmetler, ticaret, eğlence kullanıcılarının (özellikle internetin yaygınlaşmasıyla birlikte) en yoğun talep gösterdiği hizmetler halini aldı. Günümüze baktığımızda Microsoft'un çıkış noktasındaki hedefine büyük oranda ulaştığını görebiliyoruz. Ancak geldiğimiz noktada hızla artan bilgisayar ve internet kullanıcı sayısı, beraberinde güvenlik, iletişim, entegrasyon v.b. alanlarda çeşitli engellerin ortayamasına neden oldu.

Gelişmelere kendi açımızdan, yani yazılım geliştiriciler açısından baktığımızda işler çok daha zor ve zahmetli durumda. Kurumsal uygulamaların geliştirilmesinde performans, güvenlik, süreklilik gibi konularda belirli bir seviyeyi yakalamak için oldukça fazla efor sarfetmemiz gerekiyor. Örneğin elektronik cihazlarla soket iletişimini kuracak uygulamaları geliştirebilmek için iki alternatifimiz var. Birincisi 3. parti firmalar tarafından geliştirilmiş olan bileşenler satın almak ve uygulamamıza entegre etmek. Diğer alternatifimiz ise oldukça uzun sürecek bir kodlama ile benzer bir iletişim katmanını geliştirmek. Her ikiside firmaların birinci tercihi olmayacağındır. Sorunumuz sadece soket iletişimini noktasında değil elbette. Bölümün başında da belirttiğimiz gibi güvenlik, performans, yetkilendirme gibi pek çok konuda uygulama geliştiriciler oldukça zahmetli altyapı kodlarını geliştirmekle uğraşmak zorunda kalıyor. İşin kötü yanı geliştirilen bu altyapı kodları çoğu zaman istenilen verimliliği sunmaktan oldukça uzak kalıyor. Kabul etmemiz gereken şey, bu altyapı kodlarını geliştirecek bilgiye sahip olmadığımız, sahip olsak bile altyapı kodlarını yazacak zamana ve iş gücüne sahip olmadığımız, zaman ve iş gücü konusundaki ihtiyaçlarımızı karşılayabilesek bile bu kodların testi, güvenliği, güvenilirliği, performansı ve uygulamalara entegrasyonu konusunda hiç bir

zaman istenilen düzeye ulaşamayacağımızdır. Keşke ihtiyaç duyduğumuz tüm altyapı işlemleri için hazır, kullanımı kolay ve esnek bir platform olsaydı.

Hayalini kurduğum aslında şöyle bir sistem:

“Bir sanal mağazada cep telefonlarından sorumlu departmanda satış müdürü olarak çalışıyorsunuz. İş dışındasınız ve akıllı cihazınıza bir mesaj geliyor: “Henüz piyasaya yeni çıkmış olan telefonumuz inanılmaz satışlar yapıyor, telefon çok popüler ve stoklarınızda oldukça azalmış durumda.” Bu mesajın hemen ardından akıllı cihazınız üzerinden, şirketiniz için fiyat ve teslim zamanı açısından en uygun olan tedarikçi bulup ihtiyacınız kadar telefonu sipariş edebiliyorsunuz. Peki ya bu koşullar altında çalışmıyor olsaydınız? Şirketinizden sizi cep telefonunuzdan arayacaklar ve problemi ileteceklerdi. Sonra da siz şirketinize ancak donebildiğiniz zaman tedarikçilerle teker teker irtibata geçerek hangisinin şirketiniz için en yararlı olduğuna karar verecektiniz. Sipariş ve teslimat bilgileri üzerinde anlaştıktan sonra işleminizi tamamlamış olacaktınız. Yani sadece bir kaç dakikada yapabileceğiniz basit bir işlem için belki de bütün bir gününü kaybedecektiniz. Verimliliğiniz düşerken zamanınızı etkili şekilde kullanamayacaktınız. Oysa akıllı cihazınız üzerinden tüm bu işlemleri kısa bir şekilde çözebildiğinizden işe gitmenize bile gerek kalmadan zamanınızı en az şekilde kullanarak şirketiniz için en iyi olan seçimi yapabilirsiniz.”

Kesinlikle işler çok daha verimli ve kolay ilerlerdi.. Elbette bu kurulabilecek hayallerin sadece mobil platforma yönelik bölümünden bir kesit.

## Sorunun Temeli

Microsoft, vizyonu doğrultusunda attığı adımların yazılım geliştiricilere yansıyan sonuçlarını sürekli izliyordu ve yazılım geliştiricilerin sorunlarını şu başlıklar altında ele alıyordu.

- Uygulamaların, sistemlerin ve kurumlardaki birimlerin ve farklı kurumların arasındaki iletişim sorunu.
- Çalışanların ihtiyaç duydukları verilere, ihtiyaç duydukları an, kesintisiz, hatasız ve güvenli bir şekilde ve istedikleri platformdan erişebilmeleri.
- Uygulama geliştirme sürecinde, geliştiricilerin altyapı kodları ile uğraşması ve bunun sonucunda uygulama geliştirme ve test süresinin uzaması.
- Bir uygulamanın farklı platformlarda çalıştırılabilmesi için aynı işlemleri gerçekleştirecek kodların tekrar yazılması ihtiyacı.

Microsoft 1990 yılında, yaşanacak 10 yılda öngörerek bu ve benzeri sorunlara çözüm sunacak, uygulama geliştiricilerin ve son kullanıcıların işlerini kolaylaştıracak bir platform geliştirmeye başladı. Microsoft bu platforma öylesine inanıyorduğu kaynaklarının %80'inden daha fazlasını, yani kaderini bu platforma bağlamıştı. Çok geniş bir analiz ve geliştirme ekibinin çalışmalarının sonucunda ortaya çıkan ürün 2000 yılında dünyaya sunulduğuna insanların karşılaşlarında gördükleri yapı karşısında hissettiğini tanımlamak için kullanılabilecek en uygun kelime; "Hayranlık"ti.

Microsoft.NET Platformu her türlü yazılım geliştirme ihtiyacına yönelik hazır bir altyapı sunarak uygulama geliştiricilerin windows, web ve mobil platformlara yönelik uygulamaları, çok daha hızlı, kolay ve güçlü bir şekilde geliştirebilmelerine olanak tanıyordu. Uygulama geliştiriciler şifreleme, kimlik doğrulama, yetkilendirme, soket iletişim, her türlü veri kaynağına yönelik veritabanı işlemleri, xml ve web servisi teknolojilerine kadar burada saymadığımız (editörler bir modülün 100 sayfayı geçmesine pek sıcak bakmıyorlar) ve hatta milyonlarca sınıf ve fonksiyonları hazır şekilde karşılaşlarında gördüler. Bu güne kadar günler, haftalar ve hatta aylar harcayarak geliştirmeye çalışıkları bu yapıların hepsini karşılaşlarında kullanıma hazır bir şekilde görmekten de son derece memnunlardı.

## Modül 3: Microsoft Visual Studio Arayüzü

### Hedefler

- ↳ Visual Studio çalışma ortamı
- ↳ Start Page
- ↳ Menüler
- ↳ Solution Explorer Paneli
- ↳ Toolbox Paneli
- ↳ Properties Paneli
- ↳ Help Kullanımı

Bu modül, Microsoft Visual Studio ara yüzünü tanımayı sağlar ve etkili bir biçimde kullanmayı gösterir. Ev ve iş yerindeki çalışma ortamını düzenlemek daha verimli çalışmayı sağlar. Yazılım geliştirilirken de çalışılan ortamı tanımak ve kişiselleştirmek rahat çalışılması açısından önemlidir.

Bu modülü tamamladıktan sonra:

- Microsoft Visual Studio çalışma ortamını tanıယacak,
- Menülerin işlevlerini açıklayabilecek,
- Başlangıç sayfasının özelliklerini kullanabilecek,
- Solution Explorer, Toolbox, Properties panellerini tanıယacak,
- Microsoft Visual Studio Yardımı etkili bir şekilde kullanabileceksiniz.

## Konu 1: Visual Studio Çalışma Ortamı

- Visual Studio bir dosya editörüdür
- Çalışma Sayfaları
  - Sekmeler halinde gösterilir
- Araç Çubukları
  - Menü komutlarına görsel ara yüz
  - Özel araç çubukları tanımlanabilir
- Menüler
- Paneller
  - Sabitlenebilir, Kayan, Gizlenebilir pencereler

Visual Studio, çok gelişmiş özelliklere ve yardımcı araçlara sahip bir dosya editörüdür. .NET platformu üzerinde geliştirilen proje dosyaları dışında metin dosyaları, sql, rtf uzantılı dosyalar da düzenlenebilir. Visual Studio ortamını oluşturan ve kullanımını kolaylaştıran dört ana bileşen vardır:

- Çalışma Sayfaları (Tab Pages)

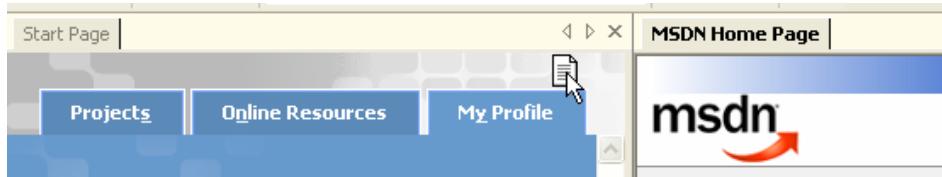
Visual Studio ortamında dosyalar, birer çalışma sayfası olarak açılır. Bu dosyalar sekemler halinde sıralanır. Sayfalar arasında **CTRL-TAB** kısa yolu ile geçiş yapılır.

Bu çalışma modelinde, sadece bir sayfa görünür ve üzerinde çalışma yapılır. Ancak Visual Studio bize, çalışma ortamını parçalara bölmeye imkânı verir.

Örnek:

- Visual Studio çalışma ortamını açın. Başlangıç sayfası karşınıza çıkar. (Eğer başlangıç sayfasını göremiyorsanız, **Help** menüsünden **Show Start Page** komutunu seçin)
- **View** menüsünden, **Web Browser** alt menüsüne işaret edin ve **Show Browser** komutunu seçin. Visual Studio açmak istediğimiz Internet tarayıcısi için yeni bir sayfa oluşturur.
- **CTRL** tuşuna basılı tutarak **TAB** tuşuna basın. Açıığınız Internet tarayıcısından başlangıç sayfasına döner.

- Başlangıç sayfasına sağ tıklayın ve çıkan menüden **New Vertical Tab Group** komutunu seçin. Visual Studio birden fazla sayfa üzerinde çalışma imkânını, sayfaları “sekme gruplarına” ayırarak sağlar.
- Başlangıç sayfasını, sayfa başlığına basılı tutarak, Internet tarayıcısının bulunduğu sekme grubuna taşıyın.



**İPUCU:** Visual Studio ortamını bir web tarayıcısı olarak kullanabilirsiniz.

- Araç Çubukları (Toolbars)

Visual Studio, menü komutlarını için görsel kısa yolları araç çubukları ile sunar. Benzer işlemler için kullanılan komutlar bir araç çubuğuunda gruplanır. Örneğin Standart araç çubuğu, yeni dosya oluşturmak, bir dosyayı açmak - kaydetmek gibi genel dosya işlemleri için kullanılır.

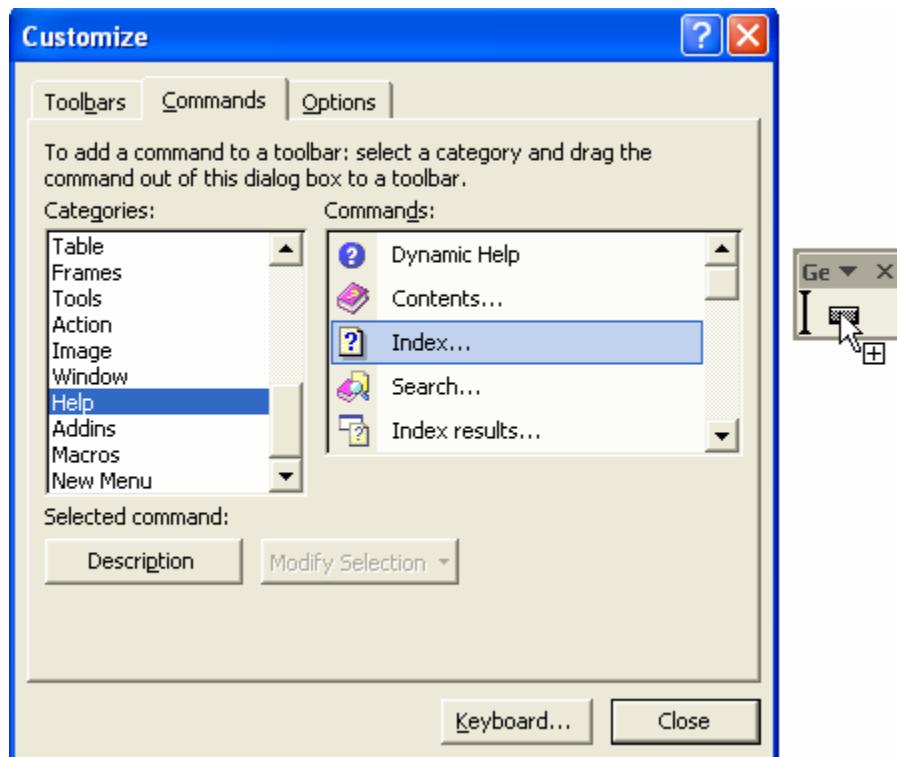
Araç çubukları, varsayılan olarak menülerin altında bulunur. Ancak çubukları taşınarak yerlerini değiştirebilir veya kayan duruma getirebilir. Ayrıca istenen çubuklar saklanılabilir veya gösterilebilir. Araç çubuklarını listesini görmek için View menüsünden Toolbars alt menüsüne işaret edin.

Visual Studio bize kendi araç çubuklarımızı oluşturma imkânı da verir. Farklı işlevlere sahip komutlar gruplanıp, kişisel araç çubuğu oluşturulabilir.

Örnek:

- Başlangıç sayfasının üstündeki bir araç çubuğuuna sağ tıklayın. Çıkan menü, var olan tüm araç çubuklarını listeler. İşaretti olan çubuklar eklenmiş çubuklardır. Bu listeden Web araç çubuğuunu seçin.
- Web araç çubuğu üzerine çift tıklayın. Bu işlem çubuğu **Floating** (kayan menü) duruma getirir. Tekrar çift tıklandığında, çubuk **Dockable** (sabit duruma) gelir.
- Araç çubuğuuna sağ tıklayın. Listenin en altındaki **Customize** ( özelleştir ) komutunu seçin.
- Toolbars sekmesinde **New** (yeni) komutuna tıklayın. Çıkan pencerede çubuğa “Genel İşlemlerim” yazın. **Ok** tuşuna basın. Visual Studio verilen isimde bir araç çubuğu oluşturur ve kayan durumda görüntüler.

- **Commands** (komutlar) sekmesinde, **Categories** (kategoriler) listesinden Help kategorisini seçin. Bu listenin yan tarafında bulunan **Commands** listesinden Index komutunu bulun. Bu komutu taşıyip, oluşturduğumuz “Genel İşlemlerim” araç çubuğuuna bırakın.



Bu şekilde şu komutları da ekleyin.

Categories	Commands
Tools	Options
File	Exit
View	Show web Browser
Window	Close All Documents

- Araç çubuğunu, çalışma ortamının altına taşıyarak sabitleyin.
- Araç çubuğu sağı tıklayın ve listeden “Genel İşlemlerim” çubuğunu seçerek çalışma ortamından kaldırın.
- Menüler

Birçok çalışma ortamının yaptığı gibi Visual Studio da, benzer öğeler üzerinde işlevleri olan komutları menüler halinde grupper. Araç çubuklarından farklı sabit

olmaları ve özelleştirmeye açık olmamalarıdır. Menüler bu modülde detaylı olarak ele alınacaktır.

- Paneller

Paneller, Visual Studio içindeki pencerelerdir. Çalışma ortamında birçok panel bulunmasıyla beraber, Solution Explorer, Toolbox, Object Browser, Properties, Watch, Output, Search Result, Task List gibi sıkça kullandığımız paneller vardır.

**İPUCU:** Görmek istenilen paneller View menüsünden seçilebilir.

Paneller, Visual Studio ortamı içerisinde istenilen yere taşınabilir veya sabitlenebilir. Panellerin birkaç genel özelliği vardır:

- **Auto Hide** (Otomatik Gizle):

Panelin, fare üzerindeyken gözükmesi ve fare çekildikten sonra gizlenmesidir.

- **Dockable** (Sabitlenebilir):

Panelin, Visual Studio ortamı içerisinde bir yerde sabitlenebilme özelliğidir.

- **Floating** (Kayan):

Kayan paneller herhangi bir yere sabitlenemez. Ancak her sayfanın üstünde durur ve böylece sürekli görünür.

Panellerin bu özellikleri **Window** menüsünden erişilebilir.

Örnek:

- **View** menüsünden **Other Windows** alt menüsünü işaret edin ve **Favorites** panelini seçin. Panelin başlığında, biri **Auto Hide** diğeri **Close** olan iki düğme görülür.
- **Auto Hide** düğmesine basarak paneli gizleyin.
- Paneli tekrar seçin, **Window** menüsünden **Auto Hide** özelliğini seçin. Daha sonra aynı menüden **Floating** özelliğini seçin. Panelin taşınabildiği ancak sabitlenemediği görülür.
- Panel seçili iken, **Window** menüsünden **Dockable** özelliğini seçin. Bu sefer panelin, taşındığı zaman çalışma ortamının herhangi bir yerine sabitlenebildiği görülür.
- Panel seçili iken, **Window** menüsünden **Hide** komutunu seçin. Paneli tekrar açmak için bu etapları tekrarlayın.

## Konu 2: Start Page

- Visual Studio ortamının başlangıç sayfasıdır
- Projects
  - Oluşturulan Visual Studio projeleri listesi
- Online Resources
  - Internet üzerindeki kaynaklar
  - Kod örnekleri, güncellemeler, makaleler
- My Profile
  - Çalışma şekline göre özel ayarlar

Visual Studio Çalışma ortamını açtığımız zaman karşımıza ilk gelen başlangıç sayfasıdır. Bu sayfa üç bölümden oluşur.

- **Projects**

O ana kadar çalışığınız projeleri gösterir. Bu menüden son projelerinizi açabilirsiniz. Son projelerde gözükmemeyen bir proje (**Open Project**) veya yeni bir proje (**New Project**) açabilirsiniz.

- **Online Resources**

Bu bölümde örnek uygulamalar (**Find Samples**) ipuçları bulabilir, en yeni teknolojileri, güncellemeleri veya en son eklenen haberleri takip edebilir, MSDN kütüphanelerinde kod örnekleri, makaleler araştırabilirsiniz.

- **My Profile**

Bu bölümde çalışma şeklinize göre bir profil seçebilirsiniz. Profiller; kullanılan kısa yollara, panellerin yerlerine ve görünümlerine, Visual Studio yardımını kullanırken yapılan filtrelemeye göre değişir.

Örneğin, profili Visual C# Developer olarak ayarlısak Dynamic Help paneli, sayfaların sağ tarafında civili olarak durur. Yardım panelinde bir arama yapmak istediğimizde ise, sonuçlar Visual C# filtresine göre çıkar. Ayrıca Solution Explorer paneli **CTRL-ALT-R** kısa yolu ile açılır.



Verify that the following settings are personalized for you:

**Profile:**

Visual Basic Developer	<input type="button" value="▼"/>
Keyboard Scheme:	Visual Basic 6
Window Layout:	Visual Basic 6
Help Filter:	Visual Basic

**Show Help:**  Internal Help  External Help

**At Startup:** Show Start Page

Görünüm, kısa yollar ve yardım filtresi birbirinden bağımsız olarak da ayarlanabilir. Bu durumda seçilen profil, **custom** (özel) olarak gözükecektir.

**At Startup** seçeneklerinden, Visual Studio açılırken hangi pencerenin gözükeceğini belirleyebilirsiniz. Örneğin, başlangıçta en son çalıştığınız projenin açılmasını istiyorsanız, “Load last loaded solution” seçeneğini tercih etmelisiniz.

**İPUCU:** Giriş sayfasını kapattıktan sonra, Help menüsünden Show Start Page seçeneğine tıklayarak açabilirsiniz.

## Konu 3: Menüler

- Birçok uygulamada kullanılan benzer menü görünümü
- File, Edit
  - Dosya, metin düzeni işlemleri
- View, Window
  - Paneller, çalışma sayfaları görüntümleri
- Project, Build, Debug
  - Proje, derleme ve hata ayıklama işlemleri
- Tools, Help
  - Yardımcı araçlar, yardım seçenekleri

Visual Studio menüleri birçok uygulamanın menülerine benzer niteliktedir. Menü isimlerinde, belirli bir harfinin altı çizilmiştir. Belirtilen harfler **ALT** tuşu ile birlikte basıldığında, o menülere kısa yolla ulaşılır. Menü komutlarının bazlarında ise, sadece o komuta özel bir kısa yol tanımlıdır. Bu kısa yollar **CTRL** veya **SHIFT** gibi birkaç tuş kombinasyonu ile gerçekleşir.

- **File** (Dosya)

Tüm dosya işlemleri bu menü altındadır. "Standart" araç çubuğu ile bu menüdeki bazı komutlara ulaşılır. **File** menüsündeki komutlar ile:

- Yeni bir proje, bir dosya veya boş bir solution (çözüm) oluşturmak,
- Oluşturulmuş bir projeyi veya var olan bir dosyayı açmak,
- Web üzerinde paylaştırılmış dosya veya projeler açmak,
- Açık olan dosya veya projeleri kapatmak,
- **Recent Files** (en son kullanılan dosya veya projeler) açmak,
- Dosyaları kaydetmek, yazdırma mümkün.

- **Edit** (Düzenle)

Tüm yazı düzenleme işlemleri için, bu menüdeki komutlar kullanılır. "Text Editor" araç çubuğu da bu menünün komutlarına kısa yoldur. **Edit** menüsündeki komutlar ile:

- **Copy, Cut, Paste, Delete, Select All** gibi temel işlemleri
- **Find And Replace, Go, Bookmark** gibi navigation işlemleri

- **Outlining** ile metinleri graplama işlemleri
- Satırları yorum satırı yapma, yorum satırlarını kaldırma, büyük küçük harf çevrimi gibi ileri seviye işlemler gerçekleştirilir.
- **View** (Görünüm)  
Visual Studio çalışma ortamındaki tüm paneller bu menü komutlarıyla gösterilir. Ayrıca **Navigate Backward** ve **Navigate Forward** komutlarıyla en son çalışılan satıra geri dönülür.
- **Project** (Proje)  
Proje dosya eklemek, çıkarmak, proje özelliklerini göstermek için bu menü kullanılır.
- **Build** (Derleme)  
Projelerin çalışmak üzere derlenmesi için gereken komutlar, bu menü altındadır.
- **Debug** (Hata Ayıklama)  
Projede hata ayıklarken gereken komutlar **Debug** menüsü altındadır. Projenin debug durumunda başlatmak, **BreakPoints** (hata ayıklarken durulması gerek satırları ayarlamak) gibi işlemler yapılır.
- **Tools** (Araçlar)  
Visual Studio ile beraber yüklenen yardımcı araçların listelendiği menüdür. Araç çubuklarını özelleştirmek için kullanılan **Customize** seçeneği gibi **Options** seçeneği de en sık kullanılan özelliklerden biridir.

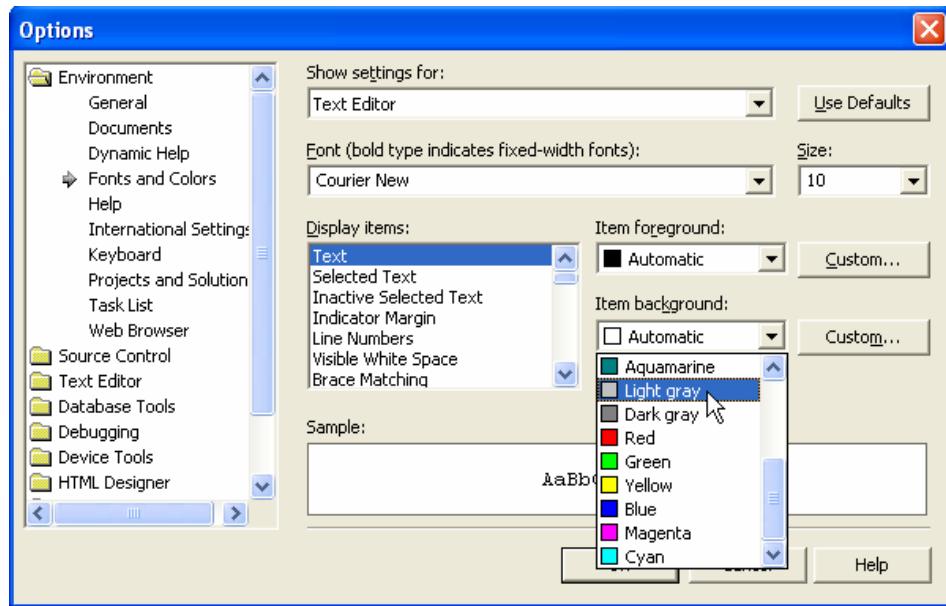
Visual Studio çalışma ortamının tüm ayarları **Options** menüsünden yapılır. **Environment** ve **Text Editor** en sık kullanılan seçeneklerdir.

**Environment** (Ortam): Sayfa düzeni ve görünüm ayarları, yazı tipi (font) ve renk ayarları, komutlar için kısa yol ayarları, Internet tarayıcısı ayarları, yardım ve dinamik yardım ayarları yapılır.

**Text Editor** (Metin Düzenleyicisi): Farklı programlama dillerine özgü yazı düzeni yapılır.

Örnek:

- **Tools** menüsünden **Options** komutunu seçin.
- Sol panelde bulunan **Environment** menüsünden **Fonts and Colors** (Yazı düzeni ve Renkler) sekmesine gelin.
- Sağ panelde bulunan **Display items** (Öğeleri Listele) menüsünden Text alanının seçin ve **Item background** (Öge arka planı) özelliğini **Light Grey** (Açık Gri) olarak belirleyin. Tüm sayfaların arka plan rengi açık gri olacaktır.



- Sol panelde **Environment** menüsünden **web Browser** sekmesine gelin. Home Page (ana sayfa) özelliğinin altındaki **Use Default** seçeneğini kaldırın ve metin kutusuna [www.bilgeadam.com](http://www.bilgeadam.com) yazın.
- Sol panelde **Text Editor** menüsünden **C#** alt menüsünü seçin. Burada Visual C# diline özel metin düzenleme seçenekleri bulunur. Sağ panelde, **Display** sekmesinin altında **Line Numbers** (Satır Numaraları) seçeneğini işaretleyin. Bu seçenek, Visual C# projelerinde çalışırken satır numaralarını gösterir.
- **Window** (Pencere)  
Sayfaların ve panellerin görünümleri ve özelliklerini değiştirmek için kullanılan komutlar bu menü altında bulunur. Tüm açık çalışma sayfaları bu menü altında görüldüğü gibi, istenen sayfa seçilerek ön plana getirilir. Ayrıca, **Close All Documents** (Tüm Sayfaları Kapat) komutu ile açık olan bütün sayfalar kapatılır. **Auto Hide All** (Tümünü Otomatik Gizle) komutu ile, sabit hale getirilmiş tüm paneller gizlenir.
- **Help** (Yardım)  
Visual Studio çalışma ortamında çok sık kullanılan yardım panellerinin görünümü bu menü ile sağlanır. Bu menü ile ayrıca, kullanılan Visual Studio çalışma ortamının sürümü hakkında bilgi alınır, son güncellemeler kontrol edilir, teknik destek için gereken e-posta adreslerine veya telefonlara ulaşılır.

Yardım kullanımı bu modülde detaylı olarak ele alınacaktır.

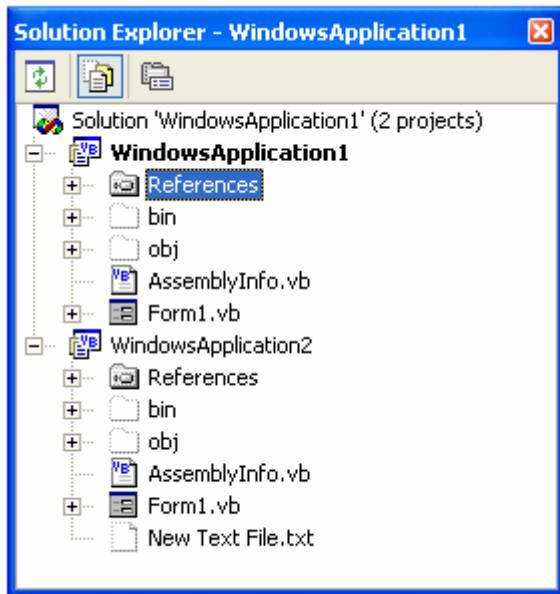
## Konu 4: Solution Explorer Paneli

- Visual Studio projeleri, bir “Solution” altında toplar
- Solution içinde bulunan tüm dosyalar, klasörler görüntülenir
- Panelye ait araç çubuğu basit işlemler gerçekleştirir
  - Refresh, Show All Files, Properties
- Visual Basic profilinde, CTRL-ALT-L ile ulaşılır

Visual Studio çalışma ortamında projeler bir solution (çözüm) altında açılır. Bir solution içine farklı dilde ve tipte projeler dâhil edilebilir. Visual Studio ile bir solution açıldığında, Solution Explorer paneli ile solution içinde bulunan tüm projeleri, ilgili dosya ve klasörleri görüntüler. Panelde gözüken yazı tipinde gözüken proje, solution içindeki başlangıç projesidir.

Bu panelden, öğeler üzerinde silme, kopyalama, taşıma, ismini değiştirme işlemleri yapılabilir. Ayrıca panelin üst kısmında, seçilen öğe üzerinde basit işlemler gerçekleştirmek için bir araç çubuğu bulunur.

- **Refresh** (Yenile)  
Proje dosyaları üzerindeki değişikliklerin gözükmesini sağlar.
- **Show All Files** (Bütün Dosyaları Göster)  
Seçilen projenin bulunduğu klasördeki tüm dosyaları ve alt klasörleri gösterir. Panelde gözüken beyaz öğeler proje içine dâhil edilmemiş öğelerdir. Projede kapsamında kullanılmak istenen öğeler (örneğin arka plan resmi), üzerine sağ tıklanıp **Include In Project** komutu ile projeye dâhil edilmelidir.
- **Properties** (Özellikler)  
Paneldeki tüm öğelerin özellikleri, Properties komutu ile görülebilir. Bu komut seçildiğinde, öğenin özellikleri Properties paneli ile görüntülenir. (Properties paneli bu modülde detaylı olarak ele alınacaktır.)



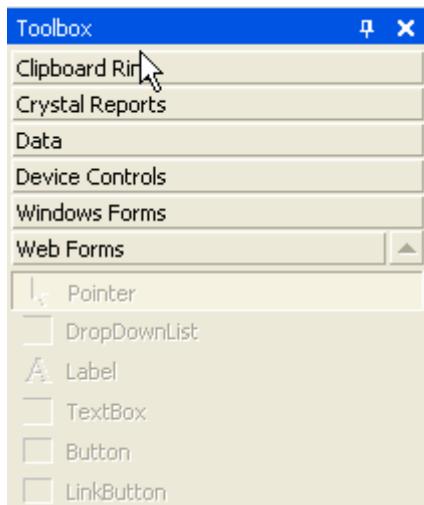
Solution Explorer paneli, **View** menüsünden görülebildiği gibi, varsayılan klavye seçeneklerinde **CTRL-ALT-L** kısa yolu ile de görülebilir.

(Ref: MSDN, Solution Explorer)

## Konu 5: Toolbox Paneli

- Projelerde kullanılan çeşitli bileşenler listelenir
- Nesneler, sekmeler halinde gruplanır
  - Windows Forms, Web Forms, ClipBoard Ring
- Visual Basic profilinde, CTRL-ALT-X ile ulaşılır

Toolbox (Araç Kutusu) paneli, projelerde kullanılan çeşitli bileşenlerin listelendiği paneldir. Buradaki öğeler, sekmeler içinde gruplanmıştır. Her sekme, ortak platformlarda çalışan veya benzer işlevleri olan nesnelere sahiptir. Örneğin, **Data** sekmesinde veri tabanı işlemlerinde kullanılan bileşenler vardır. **Windows Forms** bileşenleri Windows platformunda çalışan projelerde, **Web Forms** bileşenleri ise Web tabanlı projelerde kullanılan nesnelerdir. **Clipboard Ring** sekmesinde ise kopyalanan metinler bulunur. Nesnenin silik gözükmesi, o anda çalışılan sayfada kullanılamayacağı anlamına gelir.



Toolbox panelinde nesneler, en sık kullanıldan en az kullanıla göre sıralanmaktadır. Örneğin, **Windows Forms** sekmesinde en üstte **Label**, **Link Label**, **Button**, **TextBox** nesneleri bulunur. Nesneler, yerleri ve sıraları taşınarak değiştirilebilir, ayrıca başka bir sekmeye de taşınılabilir. Varsayılan sıralama dışında, alfabetik olarak da sıralama yapılabilir.

Visual Studio çalışma ortamın, Toolbox panelindeki nesnelere yeni isim verme, nesneleri silme veya panele yeni sekmeler ve nesneler ekleme imkânlarını da sağlar.

Örnek:

- **View** menüsünden **Toolbox** panelini seçin
- Panelde herhangi bir yere sağ tıklayın ve **Show All Tab** (Bütün Sekmeleri Göster) komutunu seçin
- **Windows Forms** sekmesinde **TextBox** nesnene sağ tıklayın. Çıkan menüden **Rename Item** (Ad Değiştir) komutunu seçin ve "Metin Kutusu" yazın.
- "Metin Kutusu" nesnesini taşıyarak sekmenin en üstüne getirin.

- Paneye sağ tıklayın ve **Sort Items Alphabetically** (Nesneleri Alfabetik olarak Sırala) komutunu seçin. Metin Kutusu nesnesinin, alfabetik sırada yerini aldığı görülür.
- Paneye sağ tıklayın ve **Add Tab** (Sekme Ekle) komutunu seçin. Sekmeye "Medya" ismini verin.
- Sekmeye sağ tıklayın ve **Add/Remove Items** (Nesne Ekle/Kaldır) komutunu seçin. **Customize Toolbox** diyalog kutusu çıkar. Burada Toolbox paneline eklenebilecek tüm bileşenler listelenir. **Com Components** sekmesine gelin ve listeden Windows Media Player nesnesini işaretleyin. **Ok** tuşuna basarak, diyalog kutusunu kapatın. Windows Media Player nesnesinin, oluşturulan Medya sekmesine eklendiği görülecektir.

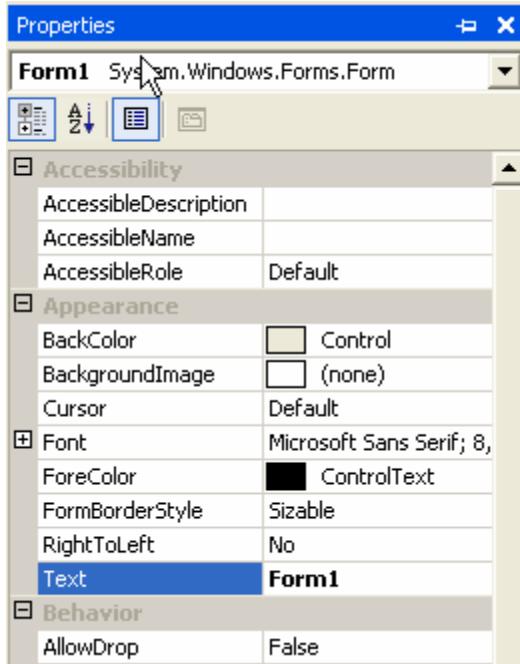
Toolbox paneli varsayılan klavye seçeneklerinde **CTRL-ALT-X** kısa yolu ile ulaşılır.

## Konu 6: Properties Paneli

- Visual Studio ortamındaki nesnelerin özelliklerini listeler
- Özellik adı – Değeri
- Özellikler kategorilere göre gruplanmıştır, alfabetik olarak da sıralanabilir
- 
- F4 ile her yerden ulaşılır

Properties (Özellikler) paneli, seçilen bir nesnenin özelliklerini görüntüler. Paneldeki görünüm, Özellik adı - değeri şeklidir. Silik olarak gözüken özellikler salt okunurdur ve değiştirilemez. Panelin üzerindeki açılır liste, çalışma sayfasındaki nesneleri listeler. Buradan istenilen nesne seçilerek özellikleri görüntülenir.

Paneldeki özellikler kategorilere göre gruplanmıştır, ancak alfabetik olarak da dizilir. Panelin üzerinde bulunan araç kutusundan **Categorized** (Kategorileştirilmiş) veya **Alphabetic** (Alfabetic) seçeneklerle özelliklerin görünümleri değiştirilir.



Panelin en altında bulunan bölümde, her özelliğin açıklaması bulunur.

İpucu: Bir nesnenin üzerindeyken **F4** tuşuna basınca, Properties paneli görüntülenir.

## Konu 7: Help Kullanımı

- En sık kullanılan kaynaktır
- MSDN (Microsoft Developer Network) kütüphaneleri
- Dynamic Help
  - İçeriği, seçilen nesnelere göre değişir
  - F1 ile dinamik yardım
- Search
  - Zengin arama seçenekleri
- Index
  - Alfabetik konu dizini
- Contents
  - MSDN kütüphanelerinin hiyerarşik görünümü

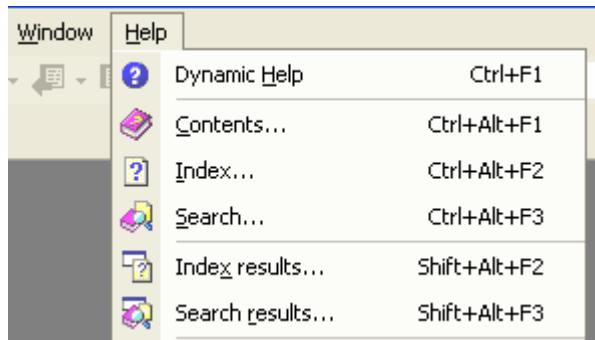
Yazılım geliştirirken en çok kullanacağımız kaynaklar yardım dosyaları olacaktır. Bir programlama dilinin çok çeşitli özellikleri, kullanım farklılıklarını olabilir. İyi bir programcı bütün bu özellikleri ezbere bilen değil, bu özellikleri en kısa sürede bulan, öğrenen ve kullanandır. Yardım dosyalarının kullanımını bilmek, programcılığın temelini oluşturan önemli unsurlardan biridir.

**DİKKAT:** Visual Studio Yardımını kullanmak için, MSDN (Microsoft Developer Network) yardım kütüphanelerinin yüklenmiş olması gerekiyor.

Visual Studio Yardımı programcıya, gelişmiş özelliklere sahip paneller ve yardım dosyaları ile geniş bir kullanım kolaylığı sağlar.

Visual Studio Yardım dosyalarının yapısı başlık, içerik, ilişkili konular (**See Also**) bölümlerinden oluşmuştur. Ayrıca her yardım dosyasının altında bulunan **Send Comments** bağlantısı ile konu hakkında yorum gönderilebilir.

Visual Studio Yardım panelleri **Dynamic Help, Search, Index ve Contents** olarak dörde ayrılır. Bu paneller, Help menüsünden ulaşılabilir.



## Dynamic Help (Dinamik Yardım)

**Dynamic Help** paneli, içeriği otomatik olarak değişen bir araçtır. Kod yazarken, panellerde veya sayfalarda nesneler seçildiğinde, kullanıcının başka bir işlem yapmasına gerek kalmadan, o nesne hakkında yardım konularını listeler. **F1** tuşuna bastığımız zaman ise seçilen nesneye ait, **Dynamic Help** panelindeki ilk yardım konusu yeni bir sayfada görüntülenir.

Paneldeki yardım konuları **Help**, **Samples** ve **Getting Started** olarak üç bölüme ayrılmıştır. Help bölümünde, seçilen nesne, bu nesneyle ilişkili olan kavramların listelendiği bölümdür. **Samples**, konuya ilgili kod örnekleri bulunan yardım dosyalarını gösterir. **Getting Started** çalışılan sayfalara göre değişen, temel işlemleri içeren başlangıç yazılarını gösterir.

## Search (Arama)

MSDN kütüphanelerinde arama yapılan paneldir. Look for metin kutusuna, aranacak anahtar kelimeler girilir. Filtreleme (**Filtered by**) ile arama sonuçları belli konulara göre sınırlanır ve istenmeyen seçeneklerin gösterilmesi engellenir.

Search panelinde, **Search in titles only**, **Match related words**, **Search in previous results**, **Highlight search hits** arama seçenekleri bulunur:

**Search in titles only:** Sadece konu başlıklarında arama yapar, içerik kısmına bakmaz

**Match related words:** Kelimeleri yazıldığı gibi arar, benzer yazılı kelimeyi aramaz.

**Search in previous results:** İlk aramadan sonra aktif olan bu seçenek ile kelimeler, bir önceki aramada bulunan sonuçlar arasına bakılır.

**Highlight search hits:** Bulunan yardım sayfalarında, aranan kelimelerin seçili olmasını sağlar.

Bulunan sonuçlar **Search Results** (Arama Sonuçları) panelinde gösterilir. Bu panelde

- **Title**, konunun başlığını
- **Location**, MSDN kütüphanelerinde hangi başlık altında bulunduğu
- **Rank**, konunun, aranılan kelimeye olan yakınlık derecesini ifade eder.

## Index (Dizin)

Yardım dosyalarındaki bütün konuları alfabetik sırada dizer. Filtreleme işlevi, arama panelinde olduğu gibidir. Bu panelin özelliği, aranacak kelime yazılırken, bu kelime ile başlayan tüm konuların alfabetik sırada gösterilmesidir. Bu şekilde, aranan konulara çok hızlı bir şekilde ulaşılabilir.

Eğer bir konu ile ilgili birden fazla yardım dosyası varsa, **Index Results** (Dizin Sonuçları) panelinde bu seçenekler gösterilir.

## Contents (İçerik)

Contents panelinde, tüm MSDN içeriği konulara göre hiyerarşik yapıda, kategorilere ayrılmış olarak gösterilir. Bu panelde de aynı şekilde filtreleme yapılarak istenmeyen içerikler çıkartılabilir.

Bir yardım dosyası açıkken, **Help** menüsünden **Sync Contents** (İçerik Senkronizasyonu) komutu seçerek o yardım dosyasının Contents panelindeki yeri bulunabilir.

## LAB 3.1: Help Kullanımı

Bu lab tamamlandıktan sonra:

- **Dynamic Help** kullanımını öğrenecek,
- **Search** paneli ile arama yapabilecek,
- **Contents** paneli ile MSDN kütüphanelerinin hiyerarşik yapısını öğrenecek,
- **Index** paneli ile içeriğe hızlı bir şekilde ulaşabilecek,
- Yardım dosyalarını yorumlayabileceksiniz.

Bu labı tamamlamak için, MSDN yardım kütüphaneleri yüklenmiş olmalıdır.

## Dynamic Help

1. **Help** menüsünden **Show Start Page** komutuna tıklayın.
2. **Help** menüsünden **Dynamic Help** komutunu seçerek **Dynamic Help** panelini açın. Panelde gösterilen ilk konunun ismi nedir?

3. **CTRL-ALT-X** tuşlarına basıp Toolbox panelini açın. **Dynamic Help** menüsünde ne değişti?
4. Toolbox panelinde, **windows Forms** tabında **Button** nesnesini seçin. **Dynamic Help** panelindeki ilk konunun ismi ne olarak değişti? **Button** seçiliyken **F1** tuşuna basın. Açılan sayfanın ismi nedir?

## Contents

1. Help menüsünden **Sync Contents** komutuna tıklayın. **Button Members** konulu yardım dosyası hangi konuların altında bulunuyor?
2. **Contents** panelinin ilk başlığı olan Visual Studio .NET altında, **Getttings Assitance** altında, "Using Help in Visual Studio .NET" altında, "Tips for Using the Help Keyword Index" konulu yardımcı açın. **File** menüsünden **Print** komutunu seçin ve sayfayı yazdırın.

**DİKKAT:** Sayfayı yazdirmak için bilgisayarınıza bağlı bir yazıcı bulunması gerekiyor.

**İPUCU:** Yardım dosyalarını yazdirmak, özellikle uzun metinlerde, kolay çalışma imkânı sağlar.

3. **Contents** panelini kapatın.

## Search

1. **Help** menüsünden **Search** komutunu seçin. **Look for** metin kutusuna Visual Studio .NET yazın. **Search in titles only**, **Match related words** seçeneklerini işaretleyin. **Search** düğmesine basın.

Kaç tane konu bulundu? En üst dereceli konu nedir?

2. **Search in previous results** seçeneğini işaretleyin. MSDN kelimesini aratın. Kaç konu bulundu?
3. **Search in previous results** seçeneğini kaldırın. MSDN kelimesini tekrar arattığınız zaman kaç konu bulunur? **Search in titles only** seçeneğini kaldırınca kaç konu bulunur?
4. **Search Results** ve **Search** panellerini kapatın.

## Index

1. **Help** menüsünden **Index** komutuna tıklayın. **Look for** metin kutusuna "file types" yazın. İlk çıkan konu nedir?
2. Filtre olarak Visual C# seçin. İlk hangi konu gösteriliyor?
3. "File Types" konusu üzerine tıklayın. Açılan sayfada Solution Files (.sln and .suo) adlı bölümü inceleyin.
4. **Project Files** başlığında, Visual Basic and Visual C# alt başlığı altında, "File Types and File Extentions in Visual Basic and Visual C#" konusuna tıklayın.

5. Açılan yardım dosyasını inceledikten sonra, sayfanın **See Also** başlığı altında “What's New in Projects” konusuna sağ tıklayın. Açılan menüden “Open Link in New Window” komutunu seçin. **Window** menüsünden “New Vertical Tab Group” komutunu seçin.  
Bir önceki yardım dosyasıyla arasındaki benzerlikleri inceleyin.
6. **Window** menüsünden “Close All Documents” seçeneği ile bütün sayfaları kapatın ve Visual Studio ortamından çıkışın.

## Modül Sonu Soruları & Alıştırmalar

### Özet

- ➔ Visual Studio çalışma ortamı
- ➔ Start Page
- ➔ Menüler
- ➔ Solution Explorer Paneli
- ➔ Toolbox Paneli
- ➔ Properties Paneli
- ➔ Help Kullanımı

1. Visual C# profili için, **Object Browser** paneline hangi kısa yolla ulaşılır?
2. Visual Studio ortamında tüm sabitlenmiş panelleri gizlemek için hangi menü komutu kullanılır?
3. Properties panelindeki özellikler alfabetik olarak nasıl sıralanır?

## **Modül 4: Visual C# .NET İle Windows Tabanlı Programlama**

### **Hedefler**

- ↳ Windows Tabanlı Uygulamalar
- ↳ Özellikler, Metotlar, Olaylar
- ↳ Windows kontrolleri
- ↳ Değişken, Sabit Tanımları
- ↳ Veri Tipleri
- ↳ Operatörler

Windows tabanlı uygulamalar, Windows işletim sistemi üzerinde çalışan uygulamalardır. Windows uygulamaları Windows formları ve kontrollerinden oluşur. Visual Studio bu formların ve üzerindeki kontrollerin tasarımını, kodların yazılımını büyük ölçüde kolaylaştırarak uygulama geliştirme sürecini daha hızlı ve kolay hale getirir.

Bu modülü tamamladıktan sonra:

- Windows tabanlı programlamada kullanılan kontrolleri tanıyacak,
- Kontrollerin özellik, metot ve olay kavramlarını öğrenecek,
- Visual C# .NET dilinde değişken, sabit tanımlamayı öğrenecek,
- Veri tiplerini tanıyacak,
- Operatörleri kullanabileceksiniz.

## Konu 1: İlk Uygulama (Hello World, The Time Is..)

### Windows tabanlı ilk uygulama

```
private void Button1_Click(object sender, System.EventArgs e)
{
    MsgBox("Hello World! The time is " + DateTime.Now);
}
```

Visual C#.NET ile yazacağımız Windows uygulaması ekrana, “Hello World!” yazısını ve o anki zamanı gösteren bir bilgi mesajını çıkartır.

- Visual Studio çalışma ortamını açın.
- **File** menüsünden, **New** alt menüsüne işaret edin ve **Project** komutunu seçin. “New Project” diyalog kutusu, yazılmacıği dile, çalışacağı ortama göre değişen projeleri tiplerini listeler.
- Proje tiplerinden Visual C#Project ve Windows Application tipinin seçili olduğunu kontrol edin.
- **Name** özelliğine **HelloWorld** yazın ve **Ok** tuşuna basın. Açılan Windows projesinde başlangıç olarak bir adet Windows Form tasarım görünümünde açılır.
- Toolbox panelinden **Button** kontrolünü formun üzerine sürükleyip bırakın. Properties panelini açarak **Button** kontrolünün **Text** özelliğine “Hello World!” yazın.
- Eklenen **Button** kontrolüne çift tıklayarak kod sayfasına geçin. **Button** kontrolüne basıldığındá çalıştırılacak kodu yazın:  
**MessageBox.Show("Hello World! The time is " + DateTime.Now);**

**NOT:** Yazdığınız kodun ne anlamına geldiğini belirtmek için yorum satırları kullanmak, kodları okumayı kolaylaştırır. Yorum satırları // ile başlayarak yazılmalıdır.

- **MessageBox.Show** metodunun yazıldığı kodun üstüne, yapılmak istenileni belirten bir yorum satırı yazın.  
`// MessageBox.Show metodu ile kullanıcıya Merhaba diyoruz.`  
`// Now özelliği ile o andaki saat ve gün`  
`// değerlerini de kullanıcıya gösteriyoruz.`
- **F5** tuşuna basarak projeyi çalıştırın.

**İPUCU:** Çalışma sayfaların isimlerinin yanında yıldız işaretinin gözükmesi, o sayfada değişiklik yapıldığını ancak daha kaydedilmediğini belirtir. Proje dosyalarınızı **CTRL-S** tuşlarına basarak sıkça kaydedin.

## Konu 2: Özellikler, Metodlar Ve Olaylar

### ▪ Özellikler

- Görünüm, yerleşim, davranışlara özgüdür
- Properties paneli
- Text, Name, Size, BackColor

### ▪ Metotlar

- Yapılan işlemler
- Parametre ile, Parametresiz çağrırlılar
- Focus, Select, Hide, Show

### ▪ Olaylar

- Başlarına gelen işlemlerdir
- Click, MouseDown, Enter

.NET Kontrolleri üç temel kavramdan oluşur.

## Özellikler

Özellikler, kontrollerin görünümü, yerleşimi veya davranışlarına özel niteliklerdir. Örneğin bir **Button** kontrolünün **Text** özelliği, üzerinde yazan yazıya erişmemizi sağlar.

Kontrollerin özellikleri, tasarım alanında **Properties** panelinden ulaşılabilcegi gibi, kod tarafında da okunup değiştirilebilir.

Kontrollerin birçok özelliği hem okunabilir hem de değiştirilebilir. Ancak bazı özellikler salt okunur (**ReadOnly**) ve salt yazılır (**WriteOnly**) olabilir. Bu tip özellikler **Properties** panelinde gözükmeler.

Kontrollerin birçok ortak özellikleri vardır.

- **Text** (Yazı)

Kontrollerin **Text** özelliği, üzerinde görüntülenen yazıdır. Bu özellik çalışma anında sıkça okunup değiştirilerek, kullanıcıyla iletişim sağlanır.

**TextBox** kontrolüne girilen bir değerin okunup **Label** kontrolüne yazılması için, kontrollerin **Text** özellikleri kullanılır.

```
private void button1_Click(object sender,
System.EventArgs e)
{
    label1.Text = textBox1.Text;
}
```



**Name** (İsim)

**Name** özelliği kontrollere ulaşmak için kullanılan özelliktir. Birçok kontrolün **Text** özelliği aynı olabilir. Ancak her biri ayrı birer nesne oldukları için, Name özellikleri benzersiz olması gereklidir.

```
textBox2.Text = textBox1.Text;
```

İki **TextBox** kontrolünün yazıları aynı, fakat isimleri farklıdır.

- **Size** (Büyüklük)

Kontrollerin büyülüklük özelliği. **Height** (yükseklik) ve **Width** (genişlik) özelliklerinden oluşur. Genellikle tasarım anında belirlenen bu özellik, çalışma anında da değiştirilebilir.

```
label1.Height = 10;
label1.Width = 20;
```

- **BackColor** (Arka plan rengi)

Kontrollerin arka plan renginin ayarlandığı özelliktir. Bu özelliğin değeri, **Color** (renk) nesnesinde tanımlı değerler ile belirlenir.

- **ForeColor** (Önalan rengi)

Kontrollerin üzerindeki yazıların rengini belirler.

```
private void button1_Click(object sender,
System.EventArgs e)
{
    button1.BackColor = Color.Black;
    button1.ForeColor = Color.White;
}
```



- **visible** (Görünür)

Kontrollerin ekranda görünüp görünmediklerini belirleyen özelliktir. **True** ve **False** olmak üzere iki değer alabilir. **Boolean** veri tiplerinden bu modülde bahsedilecektir.

```
private void button1_Click(object sender,
System.EventArgs e)
{
    // Label kontrolünü gizle
    label1.Visible = false;

    // Label kontrolünü göster
    label1.Visible = true;
}
```

## Metotlar

Metotlar kontrollerin yaptığı işlemlerdir. Metotlar parametreyle veya parametresiz çağrılabılır. Parametreyle çağrılmak, metodun girilen değere göre işlem yapacağını belirtir. Örneğin **Focus** (Odaklan) metodu, parametre beklemeden çalışır ve kontrolün seçilmesini sağlar.

```
private void button1_Click(object sender,
System.EventArgs e)
{
    // İşlem yapıldıktan sonra
    // TextBox kontrolüne odaklan
    textBox1.Focus();
}
```

Kontrollerin bazı ortak metotları vardır.

- **Select** (Seç)

**Select** metodu **Focus** ile aynıdır ama **TextBox** kontrolünün **Select** metodunun diğerlerinden bir farkı daha vardır. **TextBox** içindeki yazıyı, verilen parametreler göre belli bir kısmını ya da hepsini seçer.

```
private void button1_Click(object sender,
System.EventArgs e)
{
    textBox1.Text = "Yazılım Uzmanı";
    textBox1.Focus();

    // Sekizinci karakterden sonra,
```

```
// beş karakter seç
textBox1.Select(8, 5);
}
```



- **BringToFront** (Öne Getir)  
Üst üste duran kontroller arasından en öne getirir.
- **SendToBack** (Arkaya Gönder)  
Üst üste duran kontrollerin en arkasına gönderir.
- **Hide** (Sakla)  
Kontrolün gözükmemesini engeller.
- **Show** (Göster)  
Kontrolün gözükmemesini sağlar.

## Olaylar:

Olaylar kontrollerin başına gelen işlemlerdir. Olayların metotlardan farkı, bu işlemler kontrollerin elinde olmadan gerçekleşmesidir. Örneğin bir **Button** kontrolüne tıklanması, o kontrolün isteği dışında yapılmıştır. Bu olayın tetiklenmesinde kontrolün bir rolü yoktur. Bu olaylar gerçekleştiği zaman yapılması gereken işlemler, ilgili olayın yordamına yazılır. **Button1** isimli kontrolün üzerine tıklandığı zaman gerçekleştirmek istenen eylemler **Button1\_Click** yordamına yazılır.

Visual Studio, olayların yordam isimlerini **Kontrolİsmi\_Olayİsmi** olarak biçiminde yazar.

Kontroller ile çalışırken benzer olaylar kullanılır.

- **Click** (Tıklandığında)  
Kontrol üzerine tıklandığı zaman tetiklenen olaydır. Windows tabanlı programlamada en sık kullanılan olaylardan biridir.
- **MouseDown** (Mouse tuşu basıldığında)  
Fare, kontrolün üzerindeyken herhangi bir tuşuna basıldığı zaman gerçekleşen olaydır. Bu olay, **Click** olayından önce çalışır.
- **MouseUp** (Mouse tuşu bırakıldığında)  
Fare, kontrolün üzerindeyken basılan tuş bırakıldığı zaman çalışır.
- **Enter** (Girildiğinde)  
Kontrol seçildiği veya üzerine odaklanıldığı zaman gerçekleşen olaydır.
- **Leave** (Çıktıdığında)

Başka bir kontrol seçilmek üzere çıkışında, bu kontrolün **Leave** olayı tetiklenir.

- **VisibleChanged** (Görünürlüğü değiştiğinde)

Kontrolün görünüp görünmediğini belirten **Visible** özelliği değiştiği zaman tetiklenir.

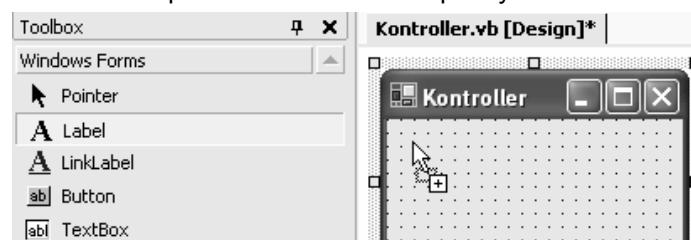
**İPUCU:** Olayların çalışma sıralarını test etmek için tüm olay yordamlarına, mesaj kutusu çikaran (**MessageBox.Show**) kod yazın. Daha sonra projeyi çalıştırıp kontroller üzerinde yapılan değişikliklere göre olayların çalışma sıralarına bakın.

## Konu 3: Visual C# .NET'e Kontrollerin Eklentimesi

### Visual Studio'ya Kontrol Eklentimesi

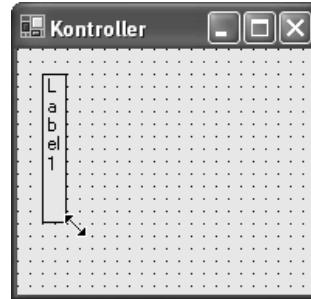
- Toolbox panelinden kontrollerin eklenmesi
- Form
- Button
- TextBox
- Label
- ComboBox
- ListBox
- Timer

Windows tabanlı uygulamalar geliştirirken sıkça kullanacağımız bir grup kontrol vardır. Form kontrolü hariç diğer bütün kontroller **Toolbox** panelinden seçilir. Bu kontroller sürükleenip Form üzerine istenilen pozisyon'a bırakılır.



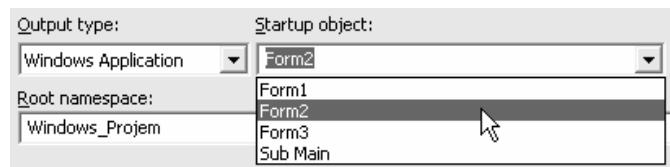
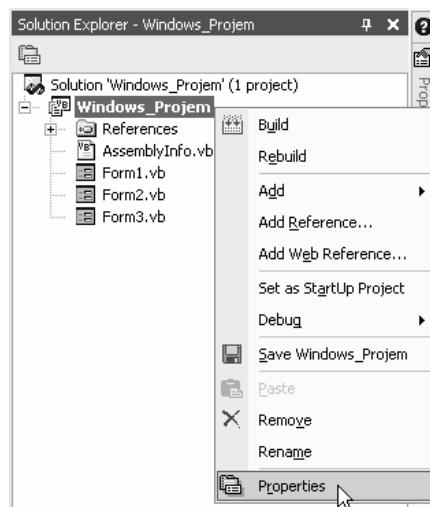
Kontroller, **ToolBox** panelinde üzerine çift tıklayarak da eklenebilir.

Kontrollerin tasarım alanında büyütükleri, yerleri **Size** ve **Location** özellikleri ile değiştirilebileceği gibi, fare ile de istenilen şekilde ayarlanabilir.



## Form

Windows uygulamaları, Windows kontrollerinin tutulduğu pencereler olan formlardan oluşur. Bir Windows projesi açıldığı zaman Form kontrolü otomatik olarak eklenir. İkinci bir form eklemek için **Project** menüsünden **Add Windows Form** komutunu seçilir. Proje çalıştığı zaman başlangıç formu görüntülenir. Başlangıç formu projenin özelliklerinden değiştirilir.



Visual Studio ortamında formlar, tasarım sayfası ve kod sayfası olmak üzere iki farklı sayfada görüntülenir. Tasarım sayfası, formun ve üzerindeki kontrollerin görünümlerini kolay bir şekilde değiştirmeyi sağlar. Visual Studio bu sayfada yapılan değişiklikleri kod sayfasında eş zamanlı olarak günceller. Örneğin bir Button kontrolünün genişliğini fare ile değiştirdiğimiz zaman, kod sayfasında bu

kontrolün **Width** özelliği yapılan değişikliğe göre güncellenecektir. Aynı değişiklikler **Properties** panelinde de görülebilir.

Formların, diğer kontrollerin özelliklerinden farklı bazı özellikleri vardır.

- **ControlBox** (Denetim Kutusu)

Form üzerindeki simge durumunda küçültme, ekranı kaplama ve formu kapama kutularının görünümünü ve erişebilirliğini kontrol eder.

**NOT:** Formun **ControlBox** özelliği **False** iken uygulama, Debug menüsünden Stop Debugging komutu seçilerek kapatılabilir.

- **StartPosition** (Başlangıç Pozisyonu)

Form açıldığı zaman nerede gözükeceğini belirler. **CenterScreen** seçeneği formu ekranın ortasında gösterir.

Formlar açıldığı zaman **Load** olayı gerçekleşir. Eğer form, başlangıç formu olarak seçilmişse, proje başladığı zaman çalıştırılmak istenen kodlar bu olayın yordamına yazılır.

```
private void Form1_Load(object sender,
System.EventArgs e)
{
    label1.Text = "Proje başlatıldı. Kayıt zamanı: "
    + DateTime.Now;
}
```

## Button

Bir Windows düğmesini temsil eder. **Button** kontrolüne basıldığında **Click** olayı tetiklenir. Bu olay gerçekleştiği zaman yapılacak işlemler, **ButtonIsmi\_Click** yordamında yazılır.

```
private void btnRenkDegistir_Click(object sender,
System.EventArgs e)
{
    btnRenkDegistir.ForeColor = Color.Gray;
}
```

## TextBox

Bir Windows metin kutusunu temsil eder. Kullanıcıların değer girerek program ile haberleşmesini sağlamak amacıyla kullanılır. **TextBox** kontrolündeki yazı değiştiği zaman **TextChanged** olayı gerçekleşir.

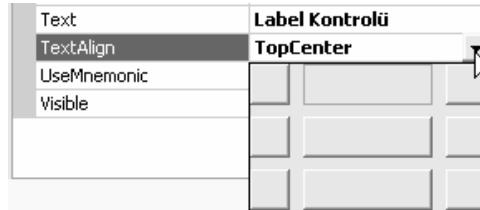
```
private void textBox1_TextChanged(object sender,
System.EventArgs e)
{
    // TextBox içindeki yazı değiştiği zaman
    // aşağıdaki kod çalışır.
    MessageBox.Show("Yazı değiştirildi: " +
textBox1.Text);
}
```

## Label

Bir Windows etiketini temsil eder. Kullanıcıya, form üzerinde bir yazıyı göstermek amaçlı kullanılır. Bu yazının görünümü, **Label** kontrolünün bazı özellikleri ile değiştirilir.

- **TextAlign** (Yazı Hızalama)

Yazının **Label** kontrolü üzerinde nerede duracağını belirler.



### Font (Yazı Tipi)

Font özelliği birçok alt özellik taşır. Bunlardan bazıları en sık kullanılan özelliklerdir.

- **Name**

Yazı tipinin ismini belirler. Varsayılan **Microsoft Sans Serif** seçilidir.

- **Size**

Karakterlerin boyutunu belirler. Varsayılan büyülüklük **8,5** değerini alır.

- **Bold** (Kalın)

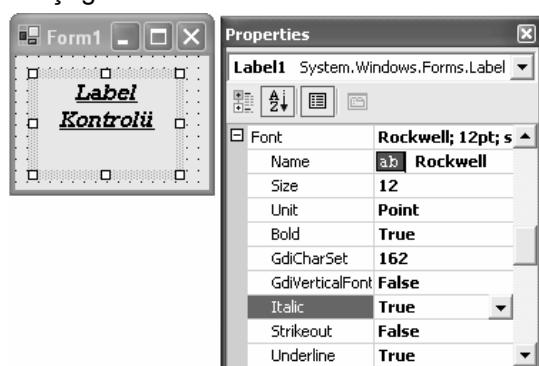
Yazının kalın tipte olmasını belirler.

- **Italic** (Yatay)

Yazının italik tipte olmasını belirler.

- **UnderLine** (Altı Çizgili)

Yazının altı çizgili olmasını belirler.



## ComboBox

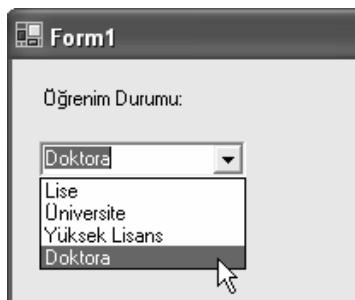
Bir Windows açılan kutusunu temsil eder. **ComboBox** kontrolü, kullanıcıların bazı değerleri açılan bir listeden seçmesini sağlar. Listeye tasarım anında veya çalışma anında öğe eklenebilir. Listeye öğe eklemek için kontrolün **Items** özelliğinden faydalanılır.

Tasarım anında öğe eklemek için **Properties** panelinden **Items** özelliği seçilir. **String Collection Editor** penceresinde, her ögenin değeri tek bir satırda yazılır.



Çalışma anında öğe eklemek için kod sayfasında, kontrolün **Items** özelliğinin **Add** metodu kullanılır.

```
private void Form1_Load(object sender,
System.EventArgs e)
{
    comboBox1.Items.Add("Lise");
    comboBox1.Items.Add("Universite");
    comboBox1.Items.Add("Yüksek Lisans");
    comboBox1.Items.Add("Doktora");
}
```



## ListBox

Bir Windows liste kutusunu temsil eder. Kontroldeki öğeler sabit bir liste olarak görüntülenir. **ListBox** kontrolüne öğe ekleme işlemi, **ComboBox** kontrolündeki işlemlere ile aynıdır. **ComboBox** kontrolünden farkı, birden fazla öğe seçilebilir olmasıdır.

```
private void btnBossiniflar_Click(object sender,
System.EventArgs e)
{
    listBox1.Items.Add("YU6501");
    listBox1.Items.Add("YM6221");
    listBox1.Items.Add("YM6102");
    listBox1.Items.Add("YU6412");
}
```



## Timer

Bir Windows sayacını temsil eder. Sayaç çalışmaya başladığı zaman, belirli zaman aralıklarında **Tick** olayı gerçekleşir. **Timer** kontrolünün **Interval** değeri, **Tick** olayının kaç milisaniyede bir gerçekleşeceğini belirler. Örneğin **Interval** değeri 2000 olan bir sayaç, **Tick** olayında yazılan kodları iki saniyede bir çalıştıracaktır.

Sayacı başlatmak için kontrolün **Start** metodu, durdurmak için ise **Stop** metodu kullanılır. **Enabled** özelliği, sayacın aktif olup olmadığını belirler.

```
private void btnBasla_Click(object sender,
System.EventArgs e)
{
    // Sayaç 5 saniyede bir çalışacak
    timer1.Interval = 5000;
    timer1.Start();
}

private void timer1_Tick(object sender,
System.EventArgs e)
{
    MessageBox.Show("Sayaç çalışıyor...");
}

private void btnDur_Click(object sender,
System.EventArgs e)
{
    timer1.Stop();
}
```

## LAB 4.1: Kronometre Uygulaması

Bu labı tamamladıktan sonra:

- Form ve üzerindeki kontrollerin görünüm özelliklerini öğrenecek,
- **ComboBox**, **ListBox** kontrollerine öğe ekleyebilecek,
- **TextBox** kontrolünden değer okuyabilecek,
- **Timer** kontrolünün çalışma şeklini öğreneceksiniz.

## Form üzerine kontrollerin eklenmesi, biçimlendirin yapılması

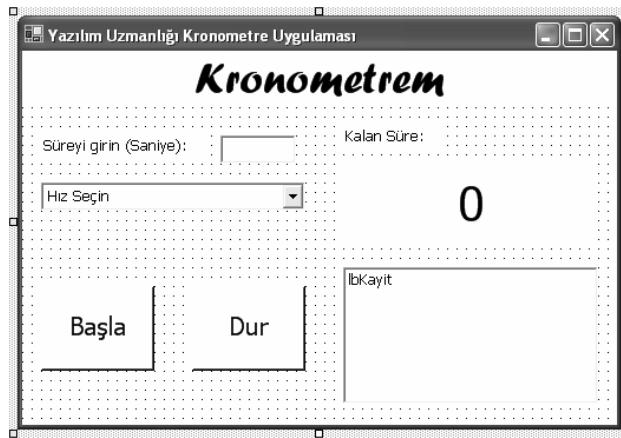
1. "Kronometre" isminde yeni bir Windows projesi açın.
2. **Properties** panelinden, **Form1** nesnesinin **BackColor** özelliğini "Menu" olarak seçin. **Font** özelliğini, yanındaki + tuşuna basarak genişletin. **Font** özelliğinin alt özellikleri listelenir.
  - **Name** özelliğini Tahoma,
  - **Text** özelliğini "Yazılım Uzmanlığı Kronometre Uygulaması",
  - **Size** özelliğini 10 olarak ayarlayın.
- Form görünüm özellikleri, eklenecek kontrollerin (değiştirilmeyenleri sürece) görünümlerini de etkiler.
3. **Toolbox** panelinden Form üzerine bir **Label** ekleyin. Özelliklerini atayın:
  - **Text**: Kronometrem
  - **Font – Name**: Forte, **Font – Size**: 28
  - **Dock**: Top
  - **TextAlign**: BottomCenter
4. Bir **Label** kontrolü ekleyin. Özelliklerini atayın:
  - **Text**: 0
  - **Font – Size**: 30
  - **TextAlign**: MiddleCenter
  - **Name**: lblSure
5. Forma bir **Timer** kontrolü ekleyin. **Name** özelliğini tmrKronometre olarak değiştirin.

**İPUCU:** Kod tarafında kullanacağınız kontrollerin isimlerini değiştirmek, daha sonra ulaşmak için zaman kazandıracaktır.

6. Bir **ComboBox** ekleyin. **Text** özelliğini "Hız Seçin" olarak, **Name** özelliğini de cmbInterval olarak değiştirin. **Items Collection** içine sırayla 1000, 2000, 3000, 4000 değerlerini girin.

Bu kontrol, çalışma anında **Timer** kontrolünün **Interval** özelliğini değiştirmeyi, dolayısıyla kronometrenin hızını ayarlamayı sağlayacak.

7. Biri "Dur", diğeri "Başla" **Text** özelliklerine sahip iki **Button** ekleyin. Kontrollerin **Name** özelliklerini sırayla btnDur ve btnBasla olarak değiştirin.
8. Bir **ListBox** kontrolü ekleyin ve **Name** özelliğini lbKayit olarak değiştirin. Bu kontrol kronometrenin başlama ve durma zamanlarını kaydetmeyi sağlayacak.
9. Bir **TextBox** kontrolü ekleyin. **Name** özelliğini txtSure olarak değiştirin ve **Text** özelliğinde yazan yazıyı silin.
10. Eklenen kontrolleri, resim (Resim numarası) de görünen şekilde düzenleyin.



## Kodların yazılması

1. Formun üzerine sağ tıklayın ve **View Code** komutunu seçin.
2. Açılan kod sayfasında, KalanSure isimli bir değişken tanımlayın.

```
public int KalanSure;
```

3. Formun tasarım görünümüne dönün ve Başla isimli **Button** kontrolüne çift tıklayın. btnBasla\_Click yordamı içine **Timer** kontrolünü ayarlayıp başlatan, **ListBox** kontrolüne kayıtları giren, kalan süreyi **Label** kontrolünde görüntüleyen kodları yazın.

```
private void btnBasla_Click( System.Object sender,
System.EventArgs e ) {
    // Başlangıç zamanı "KalanSure" değişkenine
    // atanır.
    KalanSure = System.Convert.ToInt32( txtSure.Text );
    // Kalan süre kullanıcıya gösterilir.
    lblSure.Text = System.Convert.ToString(
    KalanSure );
    // ListBox kontrolüne kayıt girilir.
    lbKayit.Items.Add( "Kronometre başlad: " +
    DateTime.Now.TimeOfDay.ToString() );
    // ComboBox kontrolünden seçilen değer,
    // Timer kontrolünün çalışma hızını
    // belirler.
    tmrKronometre.Interval = System.Convert.ToInt32(
    cmbInterval.Text );
    // Timer kontrolünü çalıştırır.
    tmrKronometre.Start();
}
```

4. Dur isimli **Button** kontrolüne çift tıklayın. **btnDur\_Click** yordamı içine **Timer** kontrolünü durduracak ve **ListBox** kontrolüne kayıtları ekleyecek kodları yazın.

```
private void btnDur_Click( System.Object sender,
System.EventArgs e ) {
    // Timer kontrolünü durdurur.
    tmrKronometre.Stop();

    // ListBox kontrolüne kayıt girilir.
    lbKayit.Items.Add( "Kronometre durduruldu: " +
DateAndTime.Now.TimeOfDay.ToString() );
}
```

5. Tasarım görünümünde **tmrKronometre** isimli **Timer** kontrolüne çift tıklayın. **tmrKronometre\_Tick** yordamı içine kalan süreyi azaltacak ve süre sıfırlandığında kronometreyi durduracak kodları yazın.

```
private void tmrKronometre_Tick( System.Object
sender, System.EventArgs e ) {
    // Her saniye geçtiğinde sure değeri 1
azalacaktır.
    KalanSure = KalanSure - 1;

    // KalanSure değeri kullanıcıya gösterilir
    lblSure.Text = System.Convert.ToString(
KalanSure );

    // KalanSure değeri sıfıra ulaşmışsa kronometre
durdurulur.
    if ( KalanSure == 0 ) {
        tmrKronometre.Stop();
        lbKayit.Items.Add( "Süre Doldu: " +
DateAndTime.Now.TimeOfDay.ToString() );

        MessageBox.Show( "Süre doldu" );
    }
}
```

6. Projeyi başlatın, metin kutusuna 5 değerini girin. Hız Seçin açılan kutusundan 1000 değerini seçin ve Başla düğmesine basın.
- Süre başladıkten ve bittiğinden sonra **ListBox** kontrolündeki değişiklikler nelerdir?
  - Hız 3000 olarak seçildiğinde başlama ve bitiş zamanları arasındaki süre ne kadardır?

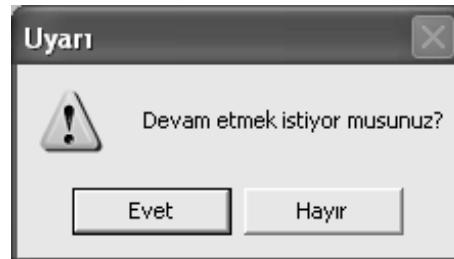
## Konu 4: **MessageBox**

**MessageBox**, kullanıcıya bilgi göstermek için açılan mesaj kutusudur. Bu mesaj kutusu dört öğeden oluşur.

- **Text** (Yazı): Mesaj kutusunda verilmek istenen bilgiyi tutan yazıdır

- **Caption** (Başlık): Mesaj kutusunun başlığıdır
- **Buttons** (Düğmeler): Mesaj kutusunda hangi düğmelerin gösterileceğini belirler.
- **Icon** (Simge): Mesaj kutusunda gösterilecek olan simgeyi ve açıldığı zaman çıkartılacak sesi belirler.

```
MessageBox.Show("Devam etmek istiyor musunuz?", "Uyarı",  
MessageBoxButtons.YesNo, MessageBoxIcon.Warning);
```



Mesaj kutusu, kapanırken hangi düğmenin basıldığını **DialogResult** nesnesi ile programciya bildirir.

```
if(MessageBox.Show("Değişiklikler kaydedilsin mi?", "Kayıt",  
MessageBoxButtons.YesNoCancel) == DialogResult.Cancel)  
{  
    // İptal tuşuna basıldığı zaman  
    // buraya girilir.  
}
```

## Konu 5: Değişken – Sabit Nedir, Değişkenlerin ve Sabitlerin Tanımlanması

### Değişkenlerin - Sabitlerin Tanımlanması

- Dim anahtar kelimesi ile tanımlanır

```
int sayı;  
string kelime;
```

- Option Explicit Off, tanımlanmamış değişkenlerin kullanımına izin verir
- Değişkenlere, kapsam alanı dışından erişilemez
- Sabitler tanımlandıktan sonra değiştirilemez

```
const int buffer = 255;
```

### Değişken Nedir, Nasıl Tanımlanır

Program yazarken, bazı verilerin daha sonra kullanılmak üzere bir yerde tutulması gerekebilir. Örneğin bir hesaplama yapılırken, önceden hesaplanmış verileri kullanmak istenirse, bu verileri tekrar hesaplamak yerine hafızada tutmak performansı artıracaktır. Veya veritabanından alınan bir kullanıcı isminin hafızada tutulması, bu değer her istendiğinde veritabanına bağlanıp alınmasına tercih edilmelidir. Verilerin bu şekilde hafızada tutulması değişkenler ile sağlanır.

Değişkenler farklı türde verileri tuttuğu için, farklı tiplere sahip olabilirler. Bir negatif veya pozitif sayıyı tutan değişken ile yazı tutan bir değişken farklı tiplere sahiptirler.

```
int sayı;  
string kelime;
```

Tanımlanan değişkenlerin tipleri değişken isminden önce belirtilir.

Değişken isimlerini belirlerken bazı noktalara dikkat etmek gereklidir.

- Boşluk, nokta, soru işaretçi, noktalı virgül, çift tırnak, tek tırnak, aritmetik operatörler, karşılaştırma ve atama operatörleri, parantezler kullanılamaz.
- Sayı ile başlayamaz.

- Visual C#.NET dilinde tanımlı anahtar kelimeler kullanılamaz.

**İPUCU:** Değişken isimlerinde Türkçe karakter kullanılırsa, farklı dil seçenekli işletim sistemlerinde çalışma anında hata üretecektir.

Hatalı bazı değişken tanımları:

```
int int;
short (sayi);
int 333sayisi;
string "kelime";
string <isim>;
```

Aynı tipteki değişkenler tek bir satır içinde tanımlanabilir.

```
int sayil, sayi2;
```

Değişkenlere değer atamak = operatörü ile yapılır. Eşitliğin sağ tarafındaki değer, sol tarafta bulunan değişkene atanır. Dolayısıyla sağ taraftaki ifadenin değeri değişmez.

```
sayi1 = 10;
sayi2 = sayi1;
```

Değişkenler tanımlandıkları sırada başlangıç değeri alabilirler.

```
string isim = "Enis Günesen";
```

Değişkenler program içinde, tuttukları verilere ulaşmak için kullanılır. Ancak değişkenlere ulaşmak, tanımlandıkları yerde veya alt bloklarda mümkündür. Bu kavrama değişkenlerin kapsam alanı (**Scope**) denir.

Kapsam alanı dışındaki bir yerden değişkene ulaşılamaz.

```
namespace NameSpace1
{
    class Class1
    {
        int SınıfDegiskeni;

        void Sub1()
        {
            int YordamDegiskeni;

            while(true)
            {
                int DonguDegiskeni;
            }
        }

        void Sub2()
        {
            int YordamDegiskeni2;
        }
    }
}
```

Tablo 0-i, kod bloklarından hangi değişkenlere ulaşılındığını gösterir.

	Class1	Sub1	Sub2	Loop
--	--------	------	------	------

<b>SinifDegiskeni</b>	Evet	Evet	Evet	Evet
<b>YordamDegiskeni</b>		Evet		Evet
<b>YordamDegiskeni2</b>			Evet	Evet
<b>DonguDegiskeni</b>				Evet

Tablo 0-i

Uygulamanın çalışması değişkenlerin kapsam alanlarındayken, bu değişkenler bellekte tutulur. Dolayısıyla değişkenlerin tanımlandıkları yer, kullanılacağı amaca göre seçilmelidir. Örneğin bir değişken birden fazla yordamda kullanılacaksa, bir üst düzeyde (Class düzeyinde) tanımlanmaları gereklidir. Ancak sadece bir yordam içinde kullanılan değişkenler class düzeyinde tanımlanırsa, fazladan bellekte yer tutar ve performans düşer. Class seviyesindeki değişkenler, aynı class içindeki fonksiyonlar ile değiştirebilir ve class örneğinin yaşam süresinde ilgili özelliklerine erişim sağlanabilir.

## Sabit Nedir, Nasıl Tanımlanır

Sabit, sürekli aynı değeri tutan değişkendir. Uygulamanın çalışması boyunca değişimyen bir değer kullanılıyorsa sabit kullanılması, kodun kolay okunmasını sağlayacaktır.

Sabitler tanımlandıktan sonra değiştirilemeyeceği için, tanımlandıkları anda değerlerinin verilmesi gereklidir.

```
const int x = 1;
```

Sabitlerin kapsam alanları değişkenler ile aynıdır.

## Veri Tipleri

- Boolean
- Byte
- Char
- Date
- Decimal
- Double
- Int16 - Short
- Int32 - Integer
- Int64 – Long
- Single
- String

```
String yazi = "Veri tipleri örnekleri";

boolean bool = True;
char karakter = "A";
date tarih = #4/23/2005#;

decimal numerik = -123456789;
double cift = -1.234E-120;
single tek = 3.32E+100;
byte bayt = 255;

short kisaSayi = -32000;
int tamSayi = 2000000000;
long uzunSayi = -123456789123456789;
```

Veri tipi, değişkenlerin tuttuğu değerlerin türünü ve bellekte tutulacak boyutunu tanımlar. Değişkenleri veri tipleri ile tanımlarken verinin boyutuna göre bir veri tipi seçilmelidir.

Visual C#.NET veri tipleri Tablo 1'de listelenmiştir.

Veri Tipi	Boyut	Değer
bool	2 Bayt	true – false
byte	1 Bayt	0 – 255
char	2 Bayt	Tek bir Unicode karakteri tutar
decimal	16 Bayt	Maksimum 29 haneli sayı tutar. +/- 79,228,162,514,264,337,593,543,950,335 arasında değer alır
double	8 Bayt	Negatif sayı aralığı: -1.79769E+308 ile -4.94065E-324 Pozitif sayı aralığı: 4.94065E-324 ile 1.79769E+308
Int32	4 Bayt	-2,147,483,648 – 2,147,483,647
Int16	2 Bayt	32,768 – 32,767
Int64	8 Bayt	-9,223,372,036,854,775,808 – 9,223,372,036,854,775,807.
float	4 Bayt	Negatif sayı aralığı: -3.4028235E+38 ile -1.401298E-45 Pozitif sayı aralığı:

		1.401298E-45 ile 3.4028235E+38
string		Maksimum 2,147,483,647 Unicode karakter tutar

Tablo 1

**double** ve **float** veri tiplerinin aralığında belirtilen “E + sayı” ifadesi,  $10^{\wedge}$  sayı ile çarpılacağını belirtir. Örneğin 12 E-3 ifadesi,  $12 * 0.001$  anlamına gelir.

-1.7E-5 = -0.000017

-1.7E+10 = -17000000000.0

0.7432E+2 = 74.32

7432E-3 = 7.432

**NOT:** **Int16**, **Int32**, **Int64** .NET veri tipleridir. Visual C# dilindeki karşılıkları **short**, **int**, **long** veri tipleridir.

```
string yazi = "Veri tipleri örnekleri";

bool b = true;
char karakter = "A";

decimal numerik = -123456789;
double cift = -1.234E-120;
float tek = 3.32E+100;
byte bayt = 255;

short kisasayi = -32000;
int tamsayi = 2000000000;
long uzunSayi = -123456789123456789;
```

Büyük veri tiplerinden küçük veri tiplerine dönüşüm sırasında, değer kayıpları meydana gelebilir. Örneğin **float** tipinden **short** tipine yapılacak bir dönüşümde virgülden sonraki sayılar kaybedilecektir.

```
float virgullu = 1.12;
short kisasayi = (float) virgullu;
// kisasayi değişkenin son değeri 1 olur
```

## struct

### ▪ Kullanıcı tanımlı veri tipi

```
struct Nokta
{
    int x;
    int y;
    void Degistir(int yeniX , int yeniY)
    {
        x = yeniX;
        y = yeniY;
    }
}
struct Ucgen
{
    Nokta n1;
    Nokta n2;
    Nokta n3;
}
```

**struct** veri tipleri, programcıların kendilerinin tanımladığı veri tipleridir. **struct**, birkaç veri tipinin bir araya getirilip oluşturulduğu bileşik bir tiptir. **struct** veri tiplerinde yordam tanımları da yapılabilir.

```
struct Nokta
{
    int x;
    int y;

    void Degistir(int yeniX , int yeniY)
    {
        x = yeniX;
        y = yeniY;
    }
}

struct Ucgen
{
    Nokta n1;
    Nokta n2;
    Nokta n3;
}
```

## Dizilerle Çalışmak

### Diziler

- Aynı tipte veriyi bir arada tutar
  - Birden fazla boyutlu olabilir
- ```
string [] isimler;
string [] isimler = new string[10];
string [] isimler;
// ...
isimler = new string[4];
```
- Length, Rank
  - GetLength, Clear, Reverse, IndexOf

Dizi değişkenleri, aynı tipte birçok veriyi bir arada tutmayı sağlar. Benzer işlemlerde kullanılan değişkenler bir dizi altında listelenebilir. Örneğin kullanıcidan alınan isimler **String** tipinde bir dizi içinde toplanabilir.

```
string [] isimler;
```

Dizilerin kaç eleman içereceği, dizi tanımlanırken ya da daha sonra belirtilebilir:

```
string [] isimler = new string[10];

string [] isimler;
// ...
isimler = new string[4];
```

Dizilerin indisleri sıfırdan başlar. Örnekteki isimler dizisinin 4 tane **String** tipinden elemanı vardır.

Dizilerin elemanlarına ulaşmak için, istenilen elemanın indisini verilmesi gereklidir.

```
isimler[0] = "Ali";
isimler[1] = "Ahmet";
isimler[2] = "Mehmet";
isimler[3] = "Ayşe";
```

```
MessageBox.Show(isimler[3]);
```

Dizilere tek tek değer atanabildiği gibi, tanımlarken de başlangıç değerleri atanabilir.

```
string [] isimler = {"Ali","Ahmet","Mehmet","Ayşe"};
```

Diziler tek boyutlu olduğu gibi, birkaç boyutlu diziler de tanımlanabilir.

```
// İlk boyuttunda 5, ikinci boyuttunda 6 int değeri olan
// 2 boyutlu dizi
int [,] matris = new int[5,6];
```

Burada dizinin ilk boyuttunda 5 tane eleman vardır. İlk boyuttaki her eleman için ikinci boyutta 6 eleman bulunur. Dolayısıyla toplam 30 elemanlı bir dizidir. Bu dizide bir boyut daha olsaydı, o boyutun her elemanı için diğer boyutlardaki 30 eleman bulunacaktı.

Çok boyutlu dizilerin eleman sayıları boyutlarındaki eleman sayılarını çarparak hesaplanabilir.

```
int [,,,] dizi = new int[boyut1,boyut2,boyut3,... ,boyutn];
// Eleman sayısı:
// boyut1 * boyut2 * ... * boyutN
```

Çok boyutlu dizilere başlangıç değerleri, dizinin boyutu dikkate alınarak verilmelidir. Boyutlardaki elemanlar kümeye parantezleri ile grüplanmalıdır.

```
// İlk boyuttunda 2, ikinci boyuttunda 4 eleman olan
// 2 boyutlu dizi
int [,] matris = {{1, 2, 3, 4}, {5, 6, 7, 8}};
```

Çok boyutlu dizilerin elemanlarına ulaşmak için, her boyut için indis göstermek gerekir.

```
matris[0, 0] = 1;
```

### **Bazı dizi özellikleri ve metodları**

Diziler, .NET Framework içinde tanımlı **Array** sınıfı temsil eder. Tüm diziler **Array** sınıfında tanımlı özellikleri ve metodları kullanırlar.

- **Length**

Dizinin bütün boyutlarındaki toplam eleman sayısını veren özelliktir.

```
ComboBox [] ComboBoxDizisi = new ComboBox[20];
MessageBox.Show(ComboBoxDizisi.Length.ToString());
// Sonuç = 20
```

```
int [] dizi = new int[1, 4, 4, 5, 6];
MessageBox.Show(dizi.Length.ToString());
// Sonuç = 2 * 5 * 5 * 6 * 7 = 2100
```

- **Rank**

Dizinin boyut sayısını veren özelliktir.

```
MessageBox.Show(dizi.Rank.ToString());
// Sonuç = 5
```

- **GetLength**

İndisi verilen boyutun kaç elemanlı olduğunu gösterir. Burada indisin sıfırdan başladığına dikkat edilmelidir.

```
int [,,,] dizi = new int[10, 40, 50, 80, 90];
MessageBox.Show(dizi.GetLength(4).ToString());
// Sonuç = 90
```

**Clear, Reverse, Indexof** metodları **Array** sınıfında **Shared** (paylaştırılmış) olarak tanımlı metotlardır. İşlemin yapıldığı dizi parametre olarak verilmelidir.

- **Clear**

Parametre olarak verilen dizinin, belirtilen indis aralığındaki tüm değerleri temizler. Temizleme işleminde atanmış değer, dizi elemanlarının tiplerine göre değişir. Örneğin **int** tipinde tanımlı bir dizinin elemanları temizlenirse **0** değerini alacaktır. Buna karşın **String** tipindeki elemanlar "" (boş yazı) değerini alır.

```
int []dizi= {12, 13, 14, 15};
// 1. indisten başlayarak, 3 elemanı temizle
Array.Clear(dizi, 1, 3);
MessageBox.Show(dizi[2].ToString());
//Sonuç = 0

// Dizinin tüm elemanlarını temizler
Array.Clear(dizi, 0, dizi.Length);
```

- **Reverse**

Parametre olarak verilen dizinin eleman sırasını tersine çevirir. Dizinin tüm elemanlarının veya belirli indis aralığındaki elemanlarının sırası tersine çevrilebilir.

```
string [] harfler = {"A", "B", "C"};
Array.Reverse(harfler);
MessageBox.Show(harfler[2]);
// Sonuç = A

string [] harfler = {"A", "B", "C"};
Array.Reverse(harfler, 0, 1);
MessageBox.Show(harfler[2]);
// Sonuç = C
```

- **Indexof**

İlk parametrede verilen dizide, ikinci parametrede verilen değeri arar.

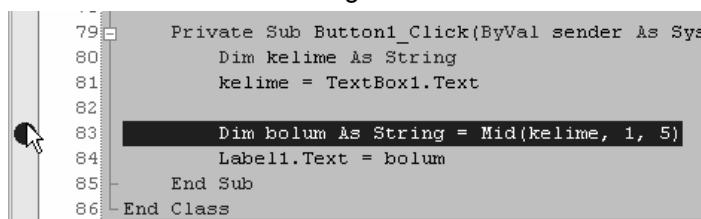
Aranan değer dizide bulunursa indis, bulunamazsa -1 döndürür.

```
float [] notlar = {78.1, 99.9, 100, 12.2};
float maxNot = 100;
MessageBox.Show(Array.IndexOf(notlar, maxNot).ToString());
// Aranan maxNot değerinin indis = 2
```

## Debug

- ### Debug
- BreakPoint ile çalışma durdurulur
  - Değişkenlerin durumları izlenir
    - Autos, Locals, Watch panelleri
  - Kodlar arasında ilerlenerek her etapta değişkenler izlenir
    - Step Into, Step Over, Step Out, Continue

Visual Studio Debug aracı, çalışma anında kodlar arasında satır satır ilerleyerek hataları bulmayı sağlar. İncelemeye başlamak istenen kod satırı üzerinde bir **BreakPoint** (durma noktası) konarak, hata ayıklayıcının bu satır çalıştırılmadan önce orada durması sağlanır.



```
79 Private Sub Button1_Click(ByVal sender As System.EventArgs)
80     Dim kelime As String
81     kelime = TextBox1.Text
82
83     Dim bolum As String = Mid(kelime, 1, 5)
84     Label1.Text = bolum
85 End Sub
86 End Class
```

A screenshot of the Visual Studio code editor. A mouse cursor is pointing at the line number 83. The line number 83 is highlighted in red, indicating it is a BreakPoint. The code itself is in black text on a white background.

Uygulama çalıştırıldığında, **BreakPoint** konulan kod satırına kadar durmaz. Belirtilen satıra sıra gelindiğinde, kod sayfasında, o an üzerinde bulunan satır ok ile gösterilir. Visual Studio ile hata ayıklarken, tanımlanan değişkenlerin o andaki değerler incelenerek mantıksal hatalar bulunabilir.

The screenshot shows a code editor window with the following VB.NET code:

```

79  Private Sub Button1_Click(ByVal sender As Syst
80  Button1.Click
81      Dim kelime As String
82      kelime = TextBox1.Text
83      kelime = "BilgeAdam"
84
85      Dim bolum As String = Mid(kelime, 1, 5)
86      Label1.Text = bolum
87  End Sub
End Class

```

Below the code editor is the 'Autos' panel, which displays variable values:

| Name          | Value       | Type   |
|---------------|-------------|--------|
| Label1.Text   | ""          | String |
| TextBox1.Text | "BilgeAdam" | String |
| bolum         | Nothing     | String |
| kelime        | "BilgeAdam" | String |

Hata ayıklama sırasında bazı Visual Studio panelleri, değişkenlerin, kontrollerin ve nesnelerin değerlerini listelemek için kullanılabilir. Bu paneller **Debug** menüsünde Windows alt menüsünden gösterilir.

- **Autos** Paneli

Çalışmakta olan satırda ifade ile bir önceki ifadede bulunan değişken ve kontrollerin değerlerini listeler.

- **Locals** Paneli

İçinde bulunan kapsam alanındaki tüm değişkenlerin değerlerini listeler.

- **Watch** Paneli

Değeri incelenmek istenen değişken veya kontroller bu panele elle yazılmalıdır.

Kodlar arasında ilerlemek ve hata ayıklamaya devam etmek için dört yol vardır. Bu komutlar **Debug** menüsünden veya Debug araç çubuğundan ulaşılabilir.

1. **Step Into**

Kod satırında bir yordam çalıştırılacaksa, bu yordamın içine girer. Bu yordam farklı bir yerde ise, ilgili sayfa açılır ve hata ayıklamaya devam edilir.

2. **Step Over**

Herhangi bir yordam içine girmeden, içindeki kapsam alanında çalışmaya devam eder.

3. **Step Out**

Bulunan yordamdan çıkararak hata ayıklamaya devam eder.

4. **Continue**

Birden fazla durma noktası yerleştirilmişse, bir sonraki noktaya kadar çalışmaya devam eder.

Hata ayıklama, çalıştırılacak hiçbir satır kalmadığında durur ve uygulama normal çalışmasına devam eder. Durma noktaları kaldırılarak ya da pasif hale getirilerek uygulamanın durması engellenebilir.

Bütün durma noktalarını kaldırmak için **Debug** menüsünden **Clear All Breakpoints** komutu, pasif hale getirmek için **Disable All Breakpoints** komutu verilmelidir. Durma noktalarını aktif hale getirmek için tekrar aynı komut seçilmelidir.

## Aliştırma

Bu uygulamada veri tiplerinin kullanım yerlerine, diziler ile çalışma örneklerine bakılacaktır.

### struct veri tipi

1. Sınıf isminde bir Windows projesi açın.
2. Açılan form üzerine sağ tıklayarak **View Code** komutunu seçin. Kod sayfasında class düzeyinde bir **struct** tanımlayın.

```
public struct Ogrenci {
    public string Isim;
    public string Soyad;
    public char Sube;
    public float OrtalamaNotu;
    public bool DevamEdiyor;
}
```

3. Ogrenci tipindeki değerleri tutmak için, class düzeyinde iki elemanlı bir dizi tanımlayın

```
public Ogrenci[] ogrenciler = new Ogrenci[2];
```

4. Formun **Load** olayına, uygulama açılırken yeni öğrenci ekleme kodlarını yazın.

```
Ogrenci ogrenci1 = new Ogrenci();
```

```
ogrenci1.Isim = "Ali";
ogrenci1.Soyad = "Veli";
ogrenci1.Sube = "C";
ogrenci1.OrtalamaNotu = 67.1;
ogrenci1.DevamEdiyor = true;
```

```
Ogrenci ogrenci2 = new Ogrenci();
```

```
ogrenci2.Isim = "Ahmet";
ogrenci2.Soyad = "Veli";
ogrenci2.Sube = "C";
ogrenci2.OrtalamaNotu = 72.9;
ogrenci2.DevamEdiyor = true;
```

```
ogrenciler[ 0 ] = ogrenci1;
ogrenciler[ 1 ] = ogrenci2;
```

5. Forma btnOgrenciEkle isminde bir **Button** kontrolü yerleştirin. Bu kontrolün **Click** olayına, diziye yeni bir öğrenci kaydı ekleyen kodu ekleyin.

```
// ogrenciler dizinde boş yer kalmadığı için
// diziyi, eski değerleri kaybetmeden tekrar
// boyutlandırmak gereklidir.
Ogrenci[] gecici = new Ogrenci[ 3 ];
System.Array.Copy( ogrenciler, gecici,2 );
ogrenciler = gecici;
```

```
Ogrenci ogrenci = new Ogrenci();
```

```
ogrenci.Isim = "Veli";
ogrenci.Soyad = "veli";
ogrenci.Sube = char.Parse( "D" );
```

```

ogrenci.OrtalamaNotu = System.Convert.ToSingle( 92.1 );
ogrenci.DevamEdiyor = false;

ogrenciler[ 2 ] = ogrenci;

```

ogrenciler dizisine başka bir yordamdan nasıl erişildi?

Formun **Load** olayında ogrenci isimli bir değişken tanımlandığı halde, **Button** kontrolünün **Click** olayında aynı isimde bir değişken nasıl tanımlanabiliyor?

### Dizi işlemleri

1. Forma bntOzellikleriGoruntule isminde bir **Button** kontrolü ekleyin ve **Click** olayında, diziden indisini verilen öğrenciyi alan kodları yazın.

```

int indis = Int32.Parse(textBox1.Text);
Ogrenci secilenOgrenci = new Ogrenci();
secilenOgrenci = (Ogrenci)ogrenciler[indis];

string bilgiler = null;

bilgiler += secilenOgrenci.Isim + " " + secilenOgrenci.Soyad;
bilgiler += "\n";
bilgiler += "Notu: " + secilenOgrenci.OrtalamaNotu + "\n";
bilgiler += "Şubesı: " + secilenOgrenci.Sube + "\n";
bilgiler += "Devam ediyor mu: " + secilenOgrenci.DevamEdiyor;

MessageBox.Show(bilgiler, MsgBoxStyle.Information, "Öğrenci Bilgileri");

```

**İPUCU:** "\n" ifadesi, **String** değişkenlerinde yeni satırı geçilmesini sağlar.

### Aritmetik işlemler

1. Forma btnOrtalamaHesapla isminde bir **Button** kontrolü ekleyin ve **Click** olayında sınıfın ortalamasını hesaplayan kodu yazın.

```

double not1 = ogrenciler[ 0 ].OrtalamaNotu;
double not2 = ogrenciler[ 1 ].OrtalamaNotu;
double not3 = ogrenciler[ 2 ].OrtalamaNotu;

int ortalama = ( ( int )( ( not1 + not2 + not3 ) / 3 ) );
MessageBox.Show( ortalama.ToString() );

```

2. Not3 değişkeninin tanımlandığı yere **BreakPoint** koyn ve projeyi çalıştırın.
3. Form açıldığında **btnOrtalamaHesapla** düğmesine basın. Uygulamanın çalışması durma noktası konulan yerde duracaktır.
4. **Debug** menüsünden **Windows** alt menüsünden **Autos** komutunu seçin. **Autos** panelinde not1 ve not2 değişkenlerinin değerlerini inceleyin.
5. **Debug** menüsünden **Windows** alt menüsünden **Watch** komutunu seçin. **Watch** panelinde **Name** sütununa "ogrenciler" yazın. ogrenciler dizisini + düğmesine basarak genişletin ve dizinin elemanlarının değerlerini inceleyin.

6. **Debug** menüsünden **Step Into** komutunu seçin. Bu işlemi **Debug** araç çubuğu ile ya da **F11** tuşuna basarak yapabilirsiniz.
7. Gösterilen hata mesajını inceleyin. **Continue** düğmesine basarak uygulamayı sonlandırın.
8. Uygulamayı tekrar çalıştırın ve önce **btnOGrenciEkle** düğmesine daha sonra **btnOrtaLamaHesapla** düğmesine basın.

## Konu 8: Operatörler

### Operatörler

- Aritmetik Operatörler
  - Çarpma \*, Bölme /, Toplama +, Çıkarma -
  - Üs alma ^, Mod alma (Mod)
- Karşılaştırma Operatörleri
  - Küçük <, Küçük Eşit =<, Büyük >
  - Büyük Eşit >=, Eşit =, Eşit Değil <>
- String Operatörleri
  - &, Split, ToCharArray, Insert, Remove

Visual C# .NET dilinde çalışırken, değişkenler üzerinde birçok işlem yapılır. Hesaplamlarda aritmetik işlemler, kontrollerde karşılaştırma işlemleri veya mantıksal işlemler yapılır. Bu işlemler için Visual C# .NET dilinde tanımlı operatörler kullanılır.

### Aritmetiksel Operatörler

Bu operatörler aritmetik işlemlerinde, sayılarla veya sayı tutan ifadelerle kullanılır.

- Çarpma

```
int sayı = 100;  
sayı = 200 * 2;
```

- Bölme

```
double bölüm;  
bölüm = sayı / 23;
```

- Çıkarma

```
int sonuc = bolum - 100;  
• Toplama  
int toplam;  
toplam += sonuc;  
// Bu ifade, "toplasm = toplam + sonuc" ile aynı anlama gelir
```

**İPUCU:** Aritmetik operatörleri, eşittir ifadesi ile beraber kullanılırsa, işlem değişkenin kendisi ile yapılır.

```
• Mod alma  
int kalan = toplam % 42;  
// Sonuç, toplam değişkenindeki değerin 42 ile  
// bölümünden kalan sayıdır.
```

## Karşılaştırma Operatörleri

Bu operatörler veri tiplerini birbirleriyle karşılaştırmak için kullanılır. Bu operatörler ile yapılan işlemlerin sonucunda **true** ya da **false** değeri döner. Karşılaştırma operatörleri yalnızca sayı tipleri üzerinde yapılmaz.

- Küçük

```
double sayı = 1.5;  
float sayı2 = 1.3;  
  
sayı2 < sayı  
// Sonuç: True
```

- Küçük Eşit

```
sayı2 <= sayı  
// Sonuç: True
```

- Büyüк

```
sayı2 > sayı  
// Sonuç: False
```

- Büyüк Eşit

```
sayı2 >= sayı  
// Sonuç: False
```

- Eşit

```
sayı2 == sayı  
// Sonuç: False
```

- Eşit Değil

```
sayı2 != sayı  
// Sonuç: True
```

## String Operatörleri

String tipleri üzerinde gerçekleştirilen işlemler için tanımlı operatörlerdir.

- String tipindeki değişkenleri birbirine bağlama `+` operatörü ile gerçekleştirir.

```
string isim, soyad;
string IsimSoyad = isim + " " + soyad;
    • Split
```

Belirtilen ayraca göre yazıyı böler, çıkan sonuç String dizisinde tutulur.

Ayraç karakterleri sonuç dizisinde yer almaz.

```
string Kelime = "Kelime1:Kelime2:Kelime3";
string [] parcalar;
parcalar = Kelime.Split(':');
// parcalar dizisinin üç elemanı olur:
// Kelime1
// Kelime2
// Kelime3

string [] parcalar2;
parcalar2 = Kelime.Split('m');
// parcalar2 dizisinin dört elemanı olur:
// Keli
// e1:Keli
// e2:Keli
// e3
```

- **ToCharArray**

String değerinin belli bir bölümündeki karakterleri ya da tüm karakterlerini, `char` dizisi olarak döndürür.

```
char [] harfler = "Kelime".ToCharArray();

// Dizinin 1. elemanından başlayarak 4 karakter oku
char [] harfler = "Kelime".ToCharArray(1,4);
```

- **Insert**

String tipinde bir değişkenin değerine, ilk parametrede belirtilen yerden başlayarak ikinci parametredeki değeri ekler. Ancak bu değişkenin değeriyle oynamaz. Yeni oluşturulan String ifadesini döndürür.

```
string sayilar = "0123456789";
string yeniSayilar;
yeniSayilar = sayilar.Insert(5, " --- Rakamlar --- ");
MessageBox.Show(yeniSayilar);
// Sonuç: 01234--- Rakamlar ---56789
```

- **Remove**

İlk parametrede verilen değerden başlayarak, ikinci parametredeki değer kadar karakter, değişkenden çıkarılır.

```
yeniSayilar = yeniSayilar.Remove(4, yeniSayilar.Length - 4);
MessageBox.Show(yeniSayilar);
// Sonuç: 0123
```

## Modül Sonu Soruları & Alıştırmalar

### Özet

- ↳ Windows Tabanlı Uygulamalar
- ↳ Özellikler, Metotlar, Olaylar
- ↳ Windows kontrolleri
- ↳ Değişken, Sabit Tanımları
- ↳ Veri Tipleri
- ↳ Operatörler

1. Arabanın fren yapması ve arabaya çarpması, .NET nesnelerinin hangi kavramlarına girer?
2. 10 saniyede bir, ListBox kontrolüne, kullanıcidan alınan değerleri ekleyen kodları yazın.
3. Değişkenler ile sabitlerin farkı nedir?
4.  $5 < 6 = -1$  ifadesi hangi Boolean değerini döndürür, neden? Option Strict On seçildiğinde çıkan hata mesajını inceleyin.

## Modül 5: Algoritma ve Dump Coding

### Hedefler

- ➔ Algoritma kurmak
- ➔ Dump Coding çözümlemesi
- ➔ Akış diyagramları

Programlamanın temelinde, çalışma akışını, izlenecek yolları belirleyen algoritmalar vardır. Bir iş yapılmaya başlanmadan önce nasıl planlanırsa, kodlamaya geçilmeden önce de bir çalışma planı belirlenmelidir. Programlar, bu planda yazılan kodları belli bir sıra ile okur ve işler. Dolayısıyla algoritma yapısını çok iyi kurmak gereklidir. Kurulan algoritmalar akış diyagramları ile görsel zenginlik kazanırlar.

Dump Coding yöntemi algoritmaları çözmenin uzun fakat etkili bir yoludur. Bu yöntem, adımları tek tek inceleyerek algoritma akışını çözer.

Bu modül tamamlandıktan sonra:

- Algoritma kurmayı öğrenecek,
- Dump Coding ile algoritmaları çözümleyecek,
- Akış diyagramları ile algoritmaları görsel olarak ifade edebileceksiniz.

## Konu 1: Algoritma Nedir?

### Algoritma

- İşin yapılma sırasının belirlenmesidir.
- İş, en küçük etaplara ayrılır.
- Olası tüm hataların tespit edilmesi, gerekli kontrollerin yapılması gereklidir.
- Algoritmanın yönü belirlenmelidir
  - Veri girişi
  - Kararlar
  - İşlemler

Algoritma, bir işin hangi etaplardan geçilerek yapılacağını gösteren çalışma planıdır. Algoritma bir programlama dili değildir. Programlama dillerine yol gösteren bir yöntem dizisidir. Her dilde algoritma yazılıp uygulanabilir. Örneğin bir cep telefonunun el kitapçığında yazan, rehber kaydı girmek için izlenecek yollar, o işin algoritmasıdır.

Algoritma yazarken, programın çalışması için kullanılan kaynakların, yapılması gereken kontrollerin veya işlemlerin açıkça ifade edilmesi gereklidir. Ayrıca iyi bir algoritma, tüm ihtimalleri kontrol edip istenmeyen durumlarda ne yapılması gerektiğini belirtmesi gereklidir.

Örneğin, bir e-ticaret uygulamasında ürün satış algoritması çıkarılır. Satın alınacak ürün seçildikten sonra, kullanıcidan adet miktarı bilgisi alınır. Uygulama yazılırken, bu değerin Int16 veri tipinde olacağına karar verildiği düşünülürse; kullanıcının girdiği adet miktarı bu değişkene atanmadan önce kontrol edilmelidir. Eğer Int16 veri tipinin tutamayacağı bir değer girilmişse, çalışma anında uygulamanın beklenmedik şekilde durduğu ya da istenmeyen sonuçların üretildiği gözlemlenir. Ayrıca sistemin verdiği hata, kullanıcının anlamayacağı bir mesaj içereceği için, uygulamanın imajını da kötü yönde etkiler.

**Veri girişi:** Çalışma zamanında çoğu zaman, işleyişin tamamlanması için dışarıdan bir bilgi girilmesi gereklidir. Algoritmanın çalışması için ihtiyaç duyduğu

veriler, işlemi başlatan kişiden veya belirtilen bir kaynaktan alınabilir. Bu bilgiler sağlanmadan işlem devam etmez.

**Kararlar:** Karar ve kontrol yapıları algoritmanın akışını yönlendiren en önemli kavamlardır. Girilen veya işlem sonucunda elde edilen veriler, işlemin amacına göre kontrol edilir ve sonuca göre algoritma akışı istenilen yere yönlendirilir.

**İşlemler:** Algoritmanın akışı boyunca veriler üzerinde değişiklikler, yeni değer atamaları gibi işlemlere ihtiyaç duyulur. Algoritmalar kurulurken, yapılan işlemlerin yalnız halde, tek tek yazılması okunabilirliği arttırmır.

Algoritmalar adım sırası ile çalışır ve karar yapıları sonucunda farklı bir yere yönlendirilmemiği müddetçe, bir sonraki adım ile işlemeye devam eder.

Örnek: Telefon kulübesinden telefon açmak için örnek bir algoritma

1. Telefon kulübesine git
2. Telefon kartı al
3. Telefon sırasında kaç kişi olduğuna bak
4. Kişi sayısı sıfırdan fazlaysa 3 e dön
5. Kapı kapalısa kapıyı aç
6. İçeri gir, kapıyı kapat
7. Telefon kartını telefona yerleştir
8. Ahizeyi kaldır
9. Numarayı çevir
10. Konuşmanın bitip bitmediğine bak
11. Konuşma bittiye kartı al, bitmediye 10 a dön
12. Bir daha konuşma yapılacaksa 7 e dön
13. Kapıyı aç, dışarı çı

Bu algoritmanın işlemesi için, her ihtimal gözden geçirilerek, algoritma akışı gerekli yerlere yönlendirilir. Örneğin kapının kapalı olması durumunda kapıyı açmak için gerekli komutlar verilmelidir. Bu algoritmanın ihtiyaç duyduğu veriler, ya kullanıcı tarafından verilir ya da işlem başlamadan önce belirlidir. Sıradaki kişi sayısı, telefon kartı gibi veriler kullanıcı tarafından sağlanmış; çevrilecek numara, algoritma başlamadan önce belirlenmiştir.

## Konu 2: Dump Coding Nedir?

### Dump Coding

- Karışık algoritmaların çözümlenmesi
- Değişkenlerin değerleri yazılıarak işleyiş takip edilir.

Dump Coding, aptal kodlama anlamına gelir. Bu yöntem birçok karışık algoritmayı çözümlememizi sağlar.

Dump coding yöntemi, algoritmanın her adımında, değişkenlerin tek tek değerlerini yazıp işleyışı takip etmektir.

Örnek: İki sayının OBEB ini (ortak bölenlerin en büyüğünü) alan algoritmalarından bir tanesi Euclid tarafından geliştirilmiştir.

1. İki sayı gir. Büyük A, küçük B
2. A sayısı B sayısına böl. Tam bölündüyorsa, OBEB B sayısıdır. Çıkış
3. A sayısının değerini, Kalan sayının değeri yap
4. A ile B sayılarını yer değiştir. İkinci etaba dön

Bu algoritmanın çalışma mantığı, Dump Coding yöntemi ile adım adım incelenir.

1. İki sayı girilir.  $A = 12$  ve  $B = 8$
2. A sayısı, B sayısına tam bölünmüyor. Algoritma diğer etaptan devam eder.
3. Kalan sayı = 4. Dolayısıyla  $A = 4$  olur.
4. A sayısı ile B sayısı yer değiştirilir.  $A = 8$  ve  $B = 4$  olur. İkinci etaba dönülür.

5. A sayısı B sayısına tam bölünüyor. OBEB = 4

## Konu 3: Akış Diyagramlarında Kullanılan Semboller

### Akış Diyagramı

- Başla – Bitir
- Veri Girişi
- Karar Verme
- Veri Tabanı
- Ekran
- Printer
- Fonksiyon
- Devam

Madde madde yazılan algoritmaların okunması kolaydır ancak işleyişin bütününe görmek çoğu zaman mümkün değildir. Akış diyagramları, algoritmaları görsel biçimde göstermeyi, dolayısıyla daha anlaşılır hale getirmeyi sağlar. Algoritmada yapılacak işlemlerin çeşitlerine göre çeşitli semboller kullanılır.

- Başla – Bitir

Algoritmanın hangi aşamadan başlayacağını ve ne zaman biteceğini gösteren semboldür. Bir algoritmayı temsil eden akış diyagramında, bir tane Başla ve bir tane Bitir sembolü olmalıdır.

Başla

Bitir

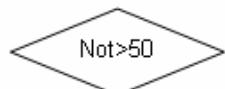
- Veri Girişi

Kullanıcıdan ve başka bir kaynaktan alınan verilerin isimlerini tutar.



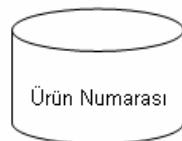
- Karar Verme

Karar yapısını belirten semboldür. Üstünde koşul ifadesi belirtilir.



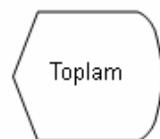
- Veri tabanı

Veri tabanında okuma veya yazma işlemi yapıldığını gösterir.



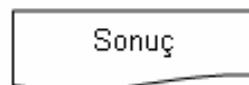
- Ekran

Üzerinde yazılan yazının bilgi olarak ekranda gözükeceğini belirtir.



- Printer

Üzerinde yazılan yazının yazıcıdan çıkarılacağını belirtir.



- İşlem

Bir işlem yapılacağını belirten semboldür. Her işlem için ayrı bir fonksiyon sembolü kullanılması, akış diyagramını daha anlaşılır kılar.

$$\text{Sayaç} = \text{Sayaç} + 1$$

- Fonksiyon

İşlem simbolüne yazılacak büyülükte işlemler, alt işlem olarak bu simbolle belirtilir.

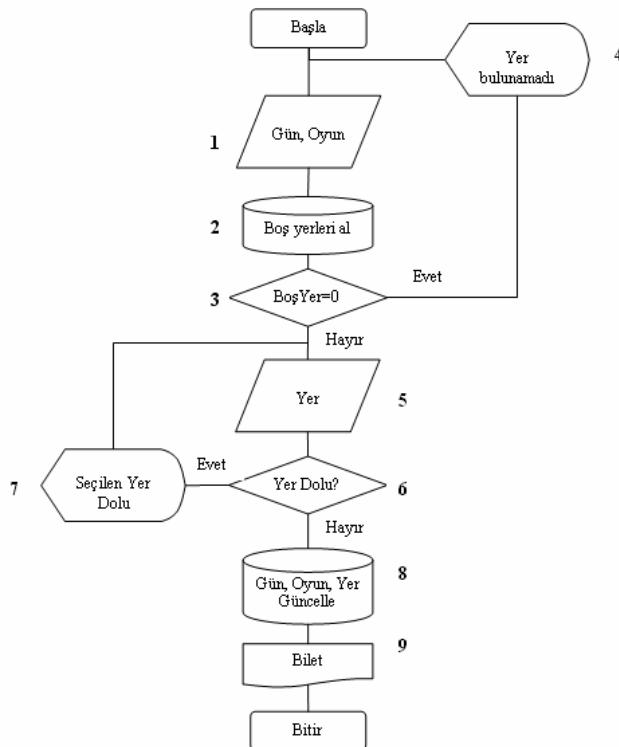


## Konu 4: Algoritma Uygulamaları

### Bilet Satma

Bir tiyatro uygulamasının sürekli gerçekleştireceği temel işlem bilet satmaktadır. Bu işlemi gerçekleştirmek için gerekli kodlar yazılmadan önce, algoritma kurulmalıdır.

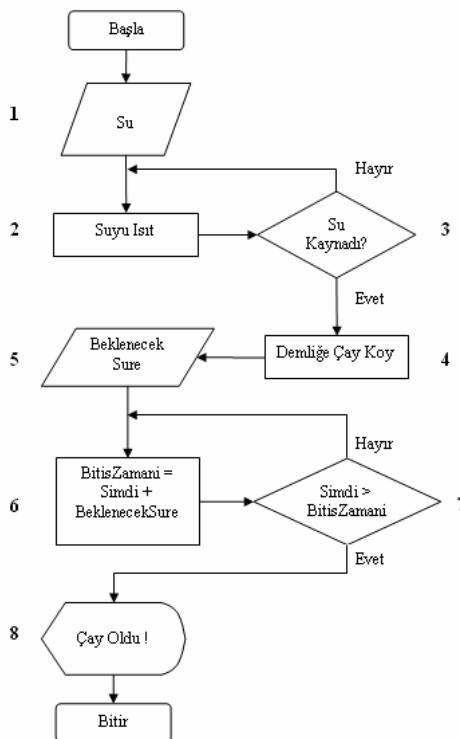
1. Kullanıcının istediği oyun, gün ve yer bilgileri alınır.
2. Veritabanı sorgulanarak, belirtilen günde oynayan oyunun boş yerleri çıkartılır.
3. Boş yer sayısı sıfırsa, o günde belirtilen oyun oynamıyor ya da oyundaki bütün yerler satılmıştır.
4. Her iki durumda da bilet kesilemediği için ekrana hata mesajı gösterilir. Gün ve oyun bilgilerini baştan almak için ilk etaba dönülür.
5. Kullanıcıdan oturmak istediği yer bilgisi alınır.
6. İstediği yerin dolu olup olmadığı kontrol edilir.
7. Yer dolu ise ekrana hata mesajı gösterilir ve yer bilgisi tekrar alınmak üzere 5. etaba dönülür.
8. Yer boşsa, veritabanında oyunun yer kayıtları güncellenir.
9. İstenilen gün, oyun ve yer bilgilerini içeren bilet yazıcıdan çıkartılır.



## Çay Demleme

Bu örnekte, bir çay demleme işleminde yapılması gereken işlemleri, kontrol edilmesi gereken olayları içeren algoritma kurulur.

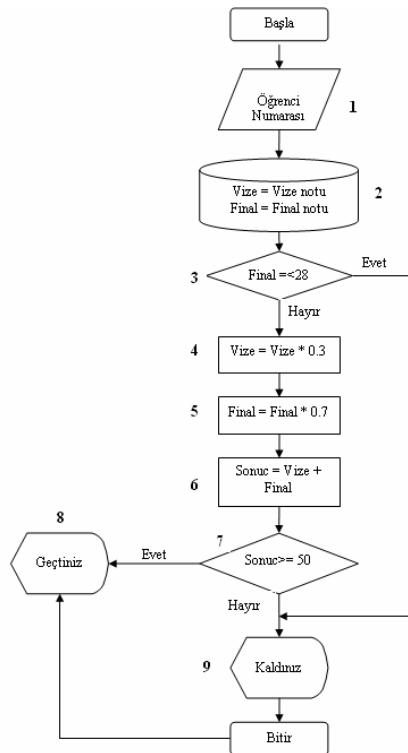
1. Kullanıcıdan su vermesi beklenir.
2. Suyu ısıtma işlemi yapılır.
3. Suyun kaynayıp kaynamadığı kontrol edilir. Kaynamamışsa 2. etaba dönülür.
4. Çay daha önceden hazır olduğu için, kullanıcıdan beklenmez. Demliğe çay koyma işlemi yapılır.
5. Kullanıcıdan, demleme işleminin ne kadar süreceği bilgisi alınır.
6. Kullanıcıdan alınan demleme süresi ile şimdiki zaman (çayın demlenmeye başladığı zaman) toplanır. Çıkan değer, BitisZamani isimli değişkene atılır. Bu değişken demleme işleminin ne zaman biteceği bilgisini tutar.
7. Şimdiki zaman, bitiş zamanından küçükse çayın demlenmesi için ayrılan süre daha dolmamış demektir. Bu süre dolana kadar 7. etap tekrarlanır.
8. Çayın demlendiğini, kullanıcıya ekran üzerinde bildiren bir mesaj çıkartılır.



## Üniversite Eğitim Notunu Hesaplama

Üniversitede bir dersin başarı notu, genelde bir vize ve bir final notu hesaplanır. Vize notunun katsayısı finalden daha düşüktür. Sonuçta çıkan not 50 ve üstüyse öğrenci geçer, 50 altıysa kalır. Bu örnek, vizenin %30 ve finalin %70 ağırlıklı olduğu başarı notunun hesaplanması akış diyagramı ile gösterir.

1. Notu hesaplanacak öğrencinin numarası kullanıcıdan alınır.
2. Veritabanından öğrencinin vize ve final notları çekilir.
3. Eğer final notu 28 veya daha düşükse öğrenci kalır ve 9. etaba gidilir. Bu durumda vize notu 100 olsa da, sonuç olarak toplanan not 50 altında olur. Dolayısıyla öğrencinin kalması kesinleşir. Böyle bir kontrol yapılması, gereksiz işlemlerin yapılmasını engeller.
4. Vize değişkenine, veritabanından alınan vize notunun %30 u atanır.
5. Final değişkenine, veritabanından alınan final notunun %70 i atanır.
6. Sonuc değişkenine, vize ve final değerlerinin toplamı atanır.
7. Sonuc değerinin 50'den büyük olup olmadığı kontrol edilir.
8. Sonuc 50'den büyükse ekrana "Geçtiniz" yazan bir mesaj çıkartılır. Algoritmadan çıkarılır.
9. Sonuc 50'den küçükse ekrana "Kaldınız" yazan bir mesaj çıkartılır.



## Modül Sonu Soruları & Alıştırmalar

### Özet

- ↳ Algoritma kurmak
- ↳ Dump Coding çözümlemesi
- ↳ Akış diyagramları

1. Algoritma kurulurken esas alınacak noktalar nelerdir?

2. Dump Coding ile Algoritmanın farkı nedir?
3. Bir ürünün bilgilerinin, veri tabanından çekilerek kullanıcıya görüntüleme işleminin algoritmasını kurun
4. Bu algoritmayı akış diyagramı ile gösterin.

## Modül 6: Karar Yapıları ve Döngüler

### Konu 1: Karar Yapıları

#### Hedefler

- ↳ If Then Elself ile akış kontrolü
- ↳ Koşul Operatörleri
- ↳ Select Case
- ↳ Karar yapılarının kullanım yerleri

Uygulamalar çalıştırılırken, yazılan kodların çalışma sırası, satırların teker teker işlenmesi ile gerçekleşir. Ancak çoğu zaman, bazı kodların sadece belli durumlarda çalışması istenir. Örneğin uygulama açılırken kullanıcı adı ve parola sorulması, kullanıcıların seviyelerine göre erişim izinleri tanımlanması gibi durumlarda kontrol işlemleri yapılmalıdır. Bu kontroller de karar yapıları ile gerçekleştirilebilir.

Algoritmaların akışını kontrol etmekte en büyük rol, karar yapılarındandır. Visual C# .NET dilinde farklı şekillerde kullanılan ancak benzer görevlere sahip karar yapıları tanımlıdır.

Bu bölüm tamamlandıktan sonra

- **if else if** karar yapıları ile akış kontrolü yapabilecek,
- Kontrollerde kullanılan koşul operatörlerinin tanıယacak,
- **switch** karar yapısını kullanabilecek,
- Hangi karar yapısının nerede kullanılacağını öğreneceksiniz.

**if****If**

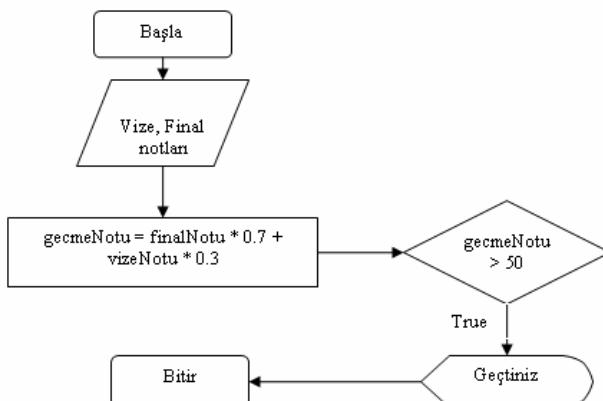
- Koşul ifadesi True ise If bloğuna girilir.
- Verilen koşul sağlandığı zaman yapılan işlemleri tutar.

```
if (gecmeNotu > 50) {
    MsgBox("Geçtiniz tebrikler...");
}
```

**if** karar yapısı, bir koşul sağlandığı zaman yapılacak işlemleri kapsar. Kontrol edilecek koşul ifadesinin sonucu **true** değerini alırsa, kümeye parantezleri arasındaki kodlar çalıştırılır.

```
if(Koşul)
{
}
```

Örnek: Vize ve final notunu kullanıcıdan aldıktan sonra, geçme notunu hesaplayan ve notun 50'den büyük olması durumunda ekrana “geçtiniz” mesajını çıkartan algoritma.



```
double gecmeNotu;
short finalNotu = short.Parse(textBox1.Text);
short vizeNotu = short.Parse(textBox2.Text);

gecmeNotu = finalNotu * 0.7 + vizeNotu * 0.3;

if(gecmeNotu > 50)
{
    MessageBox.Show("Geçtiniz tebrikler...");
```

## Koşul Operatörleri

### Koşul Operatörleri

- And
- Or
- Not
- AndAlso
- OrElse

Veri tipleri ve değişkenler üzerinde kontrol yapılırken birden fazla koşula ihtiyaç duyulabilir. Bu durumda, koşulları birbirleriyle karşılaştıracak operatörler kullanılır. Bu kontrollerden dönen değerler **Boolean** tipinde olduğu için, koşul operatörleri de bu değerler üzerinde işlem yaparlar.

#### & (And)

Bu ifade, verilen koşulların kesişimini alır. Eğer tüm koşulların değeri **true** ise sonuç da **true** olur. En az bir tane **false** değeri olan koşul varsa, sonuç **false** olur.

| Koşul 1      | Koşul 2      | Koşul 1 <b>&amp;&amp;</b> Koşul 2 |
|--------------|--------------|-----------------------------------|
| <b>true</b>  | <b>true</b>  | <b>true</b>                       |
| <b>true</b>  | <b>false</b> | <b>false</b>                      |
| <b>false</b> | <b>true</b>  | <b>false</b>                      |

|              |              |              |
|--------------|--------------|--------------|
| <b>false</b> | <b>false</b> | <b>false</b> |
|--------------|--------------|--------------|

**| (Or)**

Bu ifade, verilen koşulların birleşimini alır. Eğer tüm koşulların değeri **false** ise sonuç **false** olur. En az bir tane **true** değeri varsa sonuç **true** olur.

| Koşul 1      | Koşul 2      | Koşul 1    Koşul 2 |
|--------------|--------------|--------------------|
| <b>true</b>  | <b>true</b>  | <b>true</b>        |
| <b>true</b>  | <b>false</b> | <b>true</b>        |
| <b>false</b> | <b>true</b>  | <b>true</b>        |
| <b>false</b> | <b>false</b> | <b>false</b>       |

**&& (AndAlso)**

Koşullardan biri **False** ise, diğerleri kontrol edilmeden **False** değeri döndürülür. Bu tip bir kullanım, birçok koşulun kontrol edilmesi gerektiğinde performansı arttırmır.

```
string [] dizi;
// Diziye eleman ekleme işlemleri
// ...

if (dizi.Length > 0 && dizi[1].EndsWith("."))

{
    label1.Text = "Cümle sonundaki kelime: " + dizi[1];
}
```

Bu örnekte, dizinin ilk elemanı üzerinde bir kontrol yapılmak isteniyor. Ancak diziye eleman eklenmemişse, ilk elemana ulaşırken hata üretilicektir. Dolayısıyla dizinin uzunluğunu da kontrol etmek gerekir. Kontrol **And** ifadesi ile yapılsaydı, dizi elemanın noktaya bitip bitmediği ve dizinin uzunluğu kontrol edilecekti. Bu durumda iç içe **if** ifadeleri ile uzun bir kod yazılacaktı. Pek çok kıyaslama gerekecek ve performans düşecekti. Ancak burada, dizi uzunluğu koşulu sağlanmazsa, diğer koşula geçilmeden **if** kontrolünden çıkarılır.

**|| (OrElse)**

Koşullardan biri **True** ise, diğerleri kontrol edilmeden **True** değeri döndürülür.

```
string Rol;
// Veritabanından, kullanıcının rolü alınır.
// ...

// Sadece Administrator, Moderator ve Power User rolündeki
// kullanıcılar dosya silme işlemi yapabilirler.
if (Rol == "Administrator" || Rol == "Moderator"
    || Rol == "Power User")
{
    // Dosya silme işlemleri
}
```

Dosya silme işlemi için, kullanıcının rolü veritabanından alındıktan sonra, kontrol işlemi yapılır. Eğer bir kullanıcın rolü Administrator, Moderator veya Power User rolünden biriyse diğer kontrollerin yapılması gerekmez. Bu ömekte Rol değişkeni Administrator değerine eşitse, diğer iki koşul kontrol edilmeden **true** ifadesi döner.

### **! (Değil)**

Bir koşulun değerini tersine çevirir. Koşul **false** ise **true**, **true** ise **false** olur.

| Koşul        | ! Koşul      |
|--------------|--------------|
| <b>true</b>  | <b>false</b> |
| <b>false</b> | <b>true</b>  |

## **if else**

### If Then Else

- If koşulunda sağlanmayan tüm durumlar için Else ifadesi kullanılır.

### ElseIf

- Koşulların sağlanmadığı durumlarda, yeni kontrollerin yapılması için kullanılır.

### Select Case

- Elseif işlevini görür ancak yazılması okunması daha kolaydır.

**else** ifadesi, **if** yapısındaki koşulun sağlanmadığı bütün durumlarda devreye girer.

```
if(Koşul)
{
    //Diğer kodlar
}
else
{
    //Diğer kodlar
```

```
}
```

Örnek: Her 100 milisaniyede bir, formun renginin siyahken beyaz olması, beyazken ise siyah olması için, formun renginin kontrolü yapılması gerekiyor.

```
public bool Beyaz = true;

private void timer1_Tick(object sender,
System.EventArgs e)
{
    if(Beyaz)
    {
        this.BackColor = Color.Black;
        Beyaz = false;
    }
    else
    {
        this.BackColor = Color.White;
        Beyaz = true;
    }
}
```

**if** kontrolünde formun beyaz olup olmadığı **bool** tipindeki bir değişkende tutuluyor. Koşulda beyaz adlı değişken eğer **true** ise, formun arka planı siyah yapılır. Bu koşulun sağlanmadığı durumda, yani beyaz değişkeninin **false** olduğu durumda, **else** içindeki kodlar çalışacaktır ve formun arka planı beyaz yapılacaktır. Her kontrolden sonra beyaz değişkenin değiştirilmesinin nedeni, formun bir siyah bir beyaz olması istediği içindir.

## else if

**if** deyimindeki koşul sağlanmadıysa **else** deyimindeki kodlar çalışıyordu. Ancak bazı durumlarda **else** içinde de kontrol yapmak gerebilir.

```
if(Koşul1)
{
}
else if(Koşul2)
{
}
else if(Koşul3)
{
}
```

Örnek: Günün saatine göre karşılama mesajı çıkartmak için, saat değişkeni birçok kez kontrol edilmesi gereklidir. Sadece bir **if** kontrolü yapılsaydı, sadece iki karşılama mesajı çıkartılabilirdi.

```
string karsilamaMesaji = "BilgeAdama hoşgeldiniz!";
int saat = DateTime.Now.Hour;
```

```

if ((9 <= saat && saat < 12))
{
    karsilamaMesaji = karsilamaMesaji.Insert(0,
"Günaydın,");
}
else if (12 <= saat && saat < 16)
{
    karsilamaMesaji = karsilamaMesaji.Insert(0, "İyi
günler,");
}
else if (16 <= saat && saat < 18)
{
    karsilamaMesaji = karsilamaMesaji.Insert(0, "İyi
akşamlar,");
}

// Formun başlığı karşılama mesajını
// gösterecek şekilde ayarlanır
this.Text = karsilamaMesaji;

```

## switch

**switch** deyimi **else if** ile benzer işlevi görür, ancak okunması daha kolaydır. **switch** ile seçilen bir değerin kontrol edilmesi **Case** ifadelerinde yapılır.

```

string karsilamaMesaji = " BilgeAdama hoşgeldiniz!";
int saat = DateTime.Now.Hour;

switch (saat)
{
    case 9:
    case 10:
    case 11:
        karsilamaMesaji = karsilamaMesaji.Insert(0,
"Günaydın,");
        break;

    case 12:
    case 13:
    case 14:
    case 15:
        karsilamaMesaji = karsilamaMesaji.Insert(0,
"İyi günler,");
        break;

    case 16:
    case 17:
        karsilamaMesaji = karsilamaMesaji.Insert(0,
"İyi akşamlar,");
        break;

    default:
        karsilamaMesaji = karsilamaMesaji.Insert(0,
"Merhaba,");
        break;
}

```

Buradaki **switch** kullanımı, saat değerine göre işlem gerçekleştirilmesidir. Sayı, **case** ifadelerinde verilen değerler eşitse ilgili kodlar çalıştırılır. **default** ise, diğer koşulların sağlanmadığı tüm durumlarda devreye girer.

**break** ifadesi, kontrolün durması gerektiğini belirtir. Sayının belli değerler aralığında kontrolü yapılması için boş **case** ifadeleri kullanılır.

## Hangi Karar Cümlesi Nerede Kullanılır?

### Karar Yapılarının Kullanım Yerleri

- Select Case ifadesinin yazılışı ve okunuşu daha kolaydır.
- And, AndAlso operatörleri If yapısı ile kullanılabilir.

**if** ve **switch** karar yapıları benzer işlevler görseler de kullanım yerlerine ve birbirlerine göre değişik avantajları vardır. **if else if** karar yapılarında, kontrol edilen değişkenlerin ya da değerlerin her seferinde tekrar yazılması gereklidir. Bu durumda **switch** karar yapısı, kodların yazılışını ve okunuşunu kolaylaştırması açısından tercih edilmelidir. Ancak **switch** ile belli değerler aralığında kontrol yapılması, **case** ifadelerinin ardı ardına yazılması gereği için zordur.

Bir grup **RadioButton** kontrolü içinden sadece bir tanesi seçilebildiği için, seçilen kontrolü bulmak için **else if** yapısının kullanımı yeterli olacaktır.

```
if (RadioButton1.Checked)
{
}
else if (RadioButton2.Checked)
{
}
else if (RadioButton3.Checked)
{
}
```

Ancak bu kontroller, **CheckBox** kontrolünün kullanım yapısına uymaz. Formlarda birden fazla **CheckBox** kontrolü seçilebildiği için, seçilen kontrolleri bulmak için sadece **if** blokları kullanılmalıdır.

```
if (checkbox1.Checked)
{
}

if (checkbox2.Checked)
{
}

if (checkbox3.Checked)
{
}
```

## Uygulama

Bu uygulama kullanıcıya, stok durumuna veya tarihe göre değişen görüntüleme seçenekleri sunarak, ürün katalogu tanıtılmır. Ürünler kategorilere göre ayrılmış bulunmaktadır ve **ComboBox** kontrolleri ile filtrelerden biri seçilmemiği takdirde işlem gerçekleşmez.

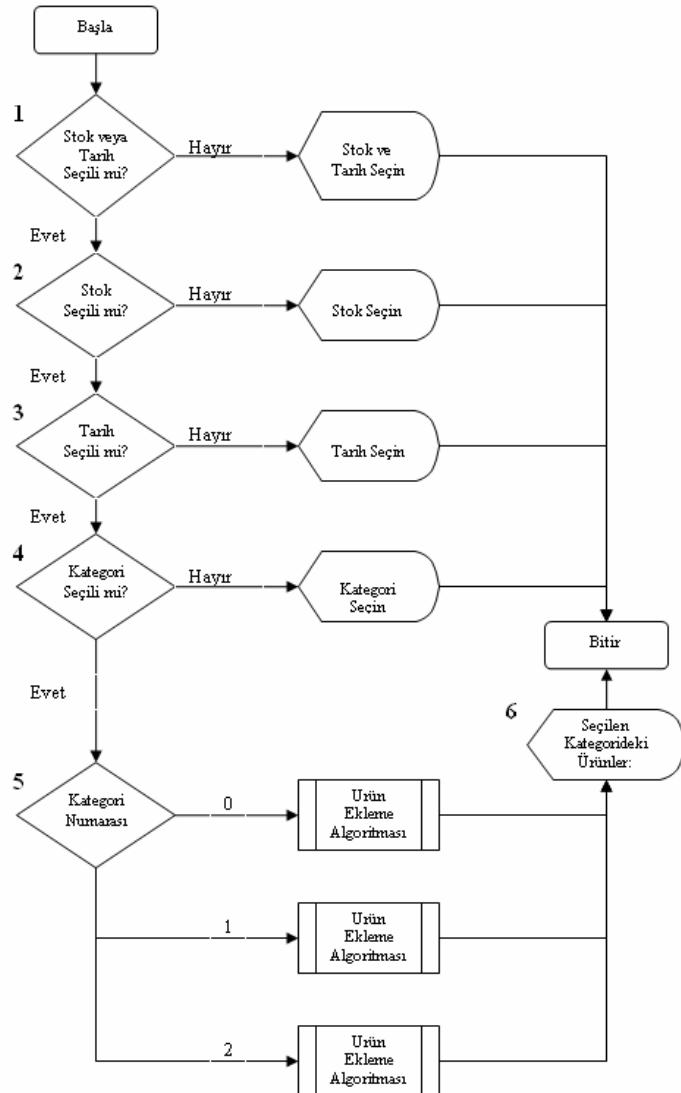
Stok durumu filtresi ile sadece stokta bulunan ya da stokta kalmamış satılmakta olan ürünler listelenebilir. Tarihe göre filtreleme ile yeni çıkan ürünler ya da tüm ürünler gözlenebilir.

Uygulamada, akış diyagramından koda geçiş aşaması rahat bir şekilde görülecektir.

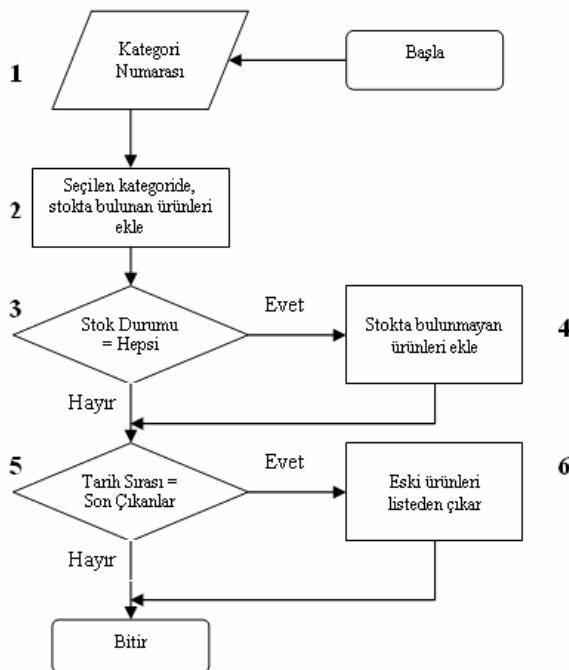
## Algoritmanın incelenmesi

Uygulamanın algoritması başlangıç ve ürün ekleme algoritması olarak ikiye ayrılmıştır.

Başlangıç algoritmasında ürün kategorisinin, stok durumu ve tarih filtrelerin seçiliş seçilmediği kontrolü yapılır. Eğer herhangi biri seçilmediği zaman kullanıcıya ilgili mesaj gösterilir ve algoritmadan çıkarılır. Tüm kontroller yapıldıktan sonra, ilgili kategorideki ürünlerin listeye eklenmesi için diğer algoritma devreye girer.



Ürün ekleme algoritması, ilk algoritmada seçilen kriterlere göre, kullanıcıya gösterilecek ürün listesini doldurur. Bu algoritma başlangıç olarak kategori numarasını alır. Bu kategorideki stokta bulunan ürünleri listeye ekler. Stok durumu filtresinde "Hepsi" değeri seçiliyse, stokta o an bulunmayan ürünler de listeye eklenir. Tarih filtresinde "Son çıkanlar" değeri seçiliyse, eski ürünler listeden çıkarılır.



## Forma kontrollerin eklenmesi

1. **UrunYelpazesi** isminde bir Visual C# Windows Projesi açın.
2. Forma biri **1bUrunler** diğeri **1bKategoriler** isminde iki **ListBox** kontrolü ekleyin. **1bUrunler** liste kutusu tüm filtreler uygulandıktan sonra çıkan ürünleri listeler. **1bKategoriler** liste kutusuna kategori isimleri ekleyin:
  - Video
  - Kitap
  - Yazılım
3. Forma biri **cmbTarihsirasi** diğeri **cmbStokDurumu** isminde iki **ComboBox** kontrolü ekleyin. **cmbTarihsirasi** son ürünler; **cmbStokDurumu** stoktaki ürünleri gösteren filtre olarak kullanılacaktır. **cmbTarihsirasi** elemanlarına "Son Çıkanlar" ve "Tüm Ürünler" değerlerini, **cmbStokDurumu** elemanlarına "Sadece Stoktakiler" ve "Hepsi" değerlerini ekleyin.
4. **1b1Mesaj** isminde bir **Label** kontrolü ekleyin. **Dock** özelliğini **Bottom** yapın. Bu kontrol filtrelerin seçilmediği durumda hata mesajlarını gösterecektir.
5. **1b1Secilenurunler** isminde bir **Label** kontrolü ekleyin ve **1bUrunler** liste kutusunun üzerine yerleştirin. Bu kontrol, seçilen ürünlerin hangi kategoride olduğunu gösterecektir.
6. Forma **btnListele** isminde bir **Button** kontrolü ekleyin.

## Kodların yazılması

Bu uygulamada kodların tamamı **btnListele** düğmesinin **Click** olayına yazılacaktır. Kodlar arasındaki numaralar akış diyagramında işlenen durumlara referans gönderir. Algoritma 1, başlangıç algoritmasındaki numaraları; Algoritma 2, ürün ekleme algoritmasındaki numaraları ifade eder.

1. **btnListele** düğmesine çift tıklayın ve **Click** olayına gelin. Düğmeye her basıldığından liste kutusuna ardi ardına öğeler eklenmemesi ve hata mesajlarının temizlenmesi için gerekli kodları yazın.

```
lblMesaj.Text = "";
lburunler.Items.Clear();
```

2. Kategori listesinden, stok ve tarih filtreleri için açılan kutulardan öğelerin seçili olup olmadığı kontrolü yapılır. Eğer seçilmemiş bir değer varsa, ilgili hata mesajı **lblMesaj** etiketinde görüntülenir.

```
// Algoritma 1 - 1
if ( cmbStokDurumu.SelectedIndex == -1 &&
cmbTarihSırası.SelectedIndex == -1 ) {
    lblMesaj.Text = "Stok Durumu ve Tarih Sırası
seçiniz.";
    // Algoritma 1 - 2
}
else if ( cmbStokDurumu.SelectedIndex == -1 ) {
    lblMesaj.Text = "Stok Durumunu seçiniz.";
    // Algoritma 1 - 3
}
else if ( cmbTarihSırası.SelectedIndex == -1 ) {
    lblMesaj.Text = "Tarih Sırasını seçiniz.";
    // Algoritma 1 - 4
}
else if ( lbKategoriler.SelectedIndex == -1 ) {
    lblMesaj.Text = "Kategori seçiniz.";
}
else {
    // Algoritma 1 - 5
}
```

3. **if else if** deyimlerinde tüm kontroller yapıldıktan sonra **else** ifadesine geçilir. Algoritmanın akışı bundan sonra ürün ekleme işlemiyle devam edecektir.

```
switch ( lbKategoriler.SelectedIndex ) {
    case 0:
        // Sadece stokta bulunan ürünler eklenir.
        // Algoritma 2 - 2
        lburunler.Items.Add( "MSDN Tv Visual C# 5" );
        lburunler.Items.Add( "MSDN Tv Visual C# 4" );

        // Stokta bulunan veya bulunmayan ürünlerin Hepsini
        // seçiliyse, kalan ürünler de listeye eklenir.
        // Algoritma 2 - 3
        if ( cmbStokDurumu.SelectedIndex == 1 ) {
            // Algoritma 2 - 4
            lburunler.Items.Add( "MSDN Tv Visual C#" );
            lburunler.Items.Add( "MSDN Tv Visual C# 2" );
            lburunler.Items.Add( "MSDN Tv Visual C# 3" );
        }
    }
```

```
// Eski ürünlerin gösterilmesi istenmiyorsa
// listeden çıkartılır.
// Algoritma 2 - 5
if ( cmbTarihSırası.SelectedIndex == 0 ) {
    // Algoritma 2 - 6
    lstUruler.Items.Remove( "MSDN TV Visual C#" );
    lstUruler.Items.Remove( "MSDN TV Visual C# 2" );
}
break;
```

4. Diğer iki kategori için liste ekleme işlemleri aynıdır.

```
case 1:
    lstUruler.Items.Add( "Yazılım Uzmanlığı 1" );
    lstUruler.Items.Add( "Yazılım Uzmanlığı 2" );
    lstUruler.Items.Add( "Yazılım Mühendisliği Orta Düzey" );
);
    lstUruler.Items.Add( "Yazılım Mühendisliği İleri Düzey" );
);

if ( cmbStokDurumu.SelectedIndex == 1 )
{
    lstUruler.Items.Add( "Yazılım Mühendisliği Başlangıç
Düzeyi" );
    lstUruler.Items.Add( "Access Giriş" );
}

if ( cmbTarihSırası.SelectedIndex == 0 )
{
    lstUruler.Items.Remove( "Yazılım Uzmanlığı 1" );
}
break;

case 2:
    lstUruler.Items.Add( "Visual Studio 6.0" );
    lstUruler.Items.Add( "visual C# .NET Standard 2003" );
    lstUruler.Items.Add( "visual C# C# Standard 2003" );

if ( cmbStokDurumu.SelectedIndex == 1 )
{
    lstUruler.Items.Add( "visual studio .NET 2005" );
}

if ( cmbTarihSırası.SelectedIndex == 0 )
{
    lstUruler.Items.Remove( "Visual Studio 6.0" );
}
break;

}
```

5. **switch** ifadesinde tüm eklemeler yapıldıktan sonra ikinci algoritma biter. İlk algoritmanın son aşaması olan, kullanıcıya hangi kategoride ürün seçildiğini gösteren mesaj yazılır ve **if** karar yapısı sonlanır.

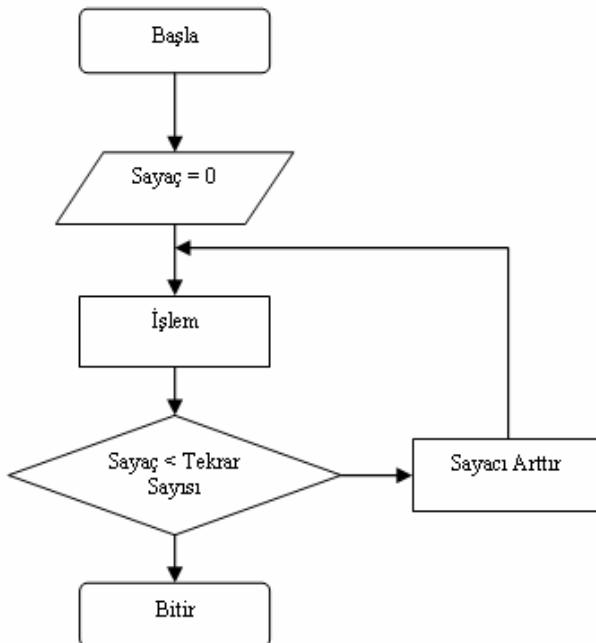
```
// Algoritma 1 - 6
    lblSecilenUruler.Text = lblKategoriler.Text + "
Kategorisindeki Ürünler";
}
```

## Konu 2: Döngüler

### Hedefler

- ➔ For Next Döngüsü
- ➔ While, Until Döngüleri
- ➔ Do Loop Döngüsü
- ➔ İç içe döngüler
- ➔ Döngülerin kullanım yerleri

Algoritmalarla bazı işlemlerin tekrar çalışması için, onları her seferinde yazmak gereklidir. Ancak bu çözüm, çok fazla tekrar için hem yazmayı hem de okumayı zorlaştırır. Örneğin yüz elemanlı bir diziye rasgele sayı atanması için işlemin yüz defa yapılması gereklidir. Döngüler ile işlem sadece bir defa yazılır ve tekrar sayısına göre bu işlemi geri dönülür.



Bu bölüm tamamlandıktan sonra

- For ve While döngüsünü tanıyacak,
- İç içe döngüler kullanabilecek,
- Hangi döngünün nerede kullanıldığını öğreneceksiniz.

## For

### For Next

- Sayaç, belirtilen aralıkta olana kadar işlem yapılır.
- Step ifadesi, sayacın artacağı ya da azalacağı miktarı belirler.
- Next ifadesi, sayacı otomatik artırır ya da azaltır.

```
int fahr;
int derece;
for (int derece = 0; derece <= 100; derece += 10)
{
    fahr = derece * 1.8 + 32;
    Label1.Text += fahr + " Fahrenheit= ";
    Label1.Text += derece + " Celcius" + "\r\n";
}
```

**For** döngüsü bir işlemin belirli sayıda yapılması için kullanılır.

```
for (int i = 0; i < 10; i++)
{
    MessageBox.Show("Merhaba");
}
```

Bu döngüde 3 parametre vardır.

- İlk parametre sayacın başlangıç değerini belirler. Örnekte, sayaç değişkeni tanımlanıp 0 değeri atanmıştır.
- İkinci parametre bir koşul ifadesidir. Bu koşul sağlandığı sürece döngü devam eder. Örnekteki döngü, i değeri 10dan küçük olduğu sürece devam edecektir.
- Üçüncü parametre, her döngüden sonra yapılması gereken işlemi belirtir. Örnekte, her işleminden sonra i değeri bir artırılır.

Döngülerde kullanılan sayıçalar sadece belli bir sayıda işlem yapmayı sağlamaz. Sayaçların artma veya azalma adımları belirli olduğu için, kod içerisinde çoğu zaman bu avantajdan yararlanılır.

```
ListBox1.Items.Add("Karakter - ASCII kod karşılığı");
for (int i = 50; i <= 255; i = i + 2)
{
    listBox1.Items.Add(Microsoft.VisualBasic.Strings.Chr(i)
& " - " & i)
}
```

Örneğin dizi işlemlerinde, dizinin her elemanına ulaşmak için sayıç kullanabilir. Sayacın artma hızı bir olduğu için **dizi[sayac]** ifadesi, sırayla dizinin elemanlarına ulaşmayı sağlar.

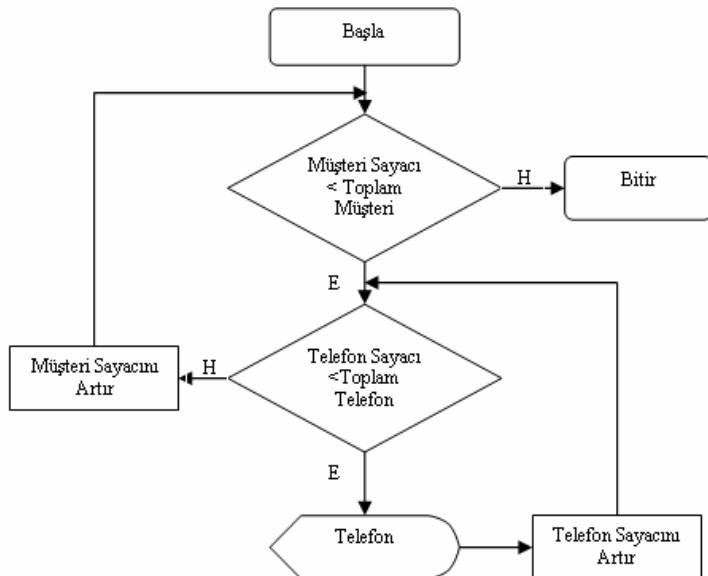
```
int [] dizi = new int[10];
Random r = new Random();

for (int i = 0; i < dizi.Length ; i++)
{
    dizi[i] = r.Next(100);
}
```

**DİKKAT:** Değişken tanımlamaları **For** döngüsünün içinde de yapılabilir. Bu durumda, değişkenin kapsam alanı bu döngüyle sınırlı kalır.

## For döngülerinin iç içe kullanımı

Çoğu zaman **For** döngülerindeki her etap için başka bir döngünün kurulması gereklidir. Örneğin bir müşterinin birden fazla telefon numarası bir dizi içinde tutuluyorsa, bütün müşterilerin telefonlarını listelemek için iki döngü kullanılması gereklidir. İlk döngü tek tek müşterileri almak için, alt döngü ise her müşterinin telefonlarını almak için kullanılmalıdır.



```

public struct Musteri {
    public string Isim;
    public string Soyad;
    public string[] Telefonlari;
}

public Musteri[] Musteriler;
// Musteriler dizisi dolduruyor
// ...

private void Button1_Click( System.Object sender,
System.EventArgs e )
{
    int i, j;

    for ( i=0; i<=Musteriler.Length - 1; i++ ) {
        // İlk m̄teri seiliyor
        Musteri m = Musteriler[ i ] ;

        Label1.Text += m.Isim + " " + m.Soyad;
        Label1.Text += " müşterisinin telefonları:"
+ "\n";

        for ( j=0; j<=m.Telefonlari.Length - 1; j++ )
            Label1.Text += m.Telefonlari[ j ] ;
        Label1.Text += "\n";
    }
}
}

```

Birden fazla boyutlu dizilerde işlem yaparken de **For** döngüsü iç içe kullanılabilir. Örneğin iki boyutlu bir tabloda, ilk boyut için bir **for** döngüsü, diğer boyut için de başka bir **for** döngüsü kullanılarak dizinin tüm elemanlarına ulaşılabilir.

```

Random r = new Random();
string[,] tablo = new string[ 5, 5 ];

byte i = 0;
for ( i=1; i<=4; i++ )
{
    tablo[ 0, i ] = "Yazar " + i;
    tablo[ i, 0 ] = "Kitap " + i;
    tablo[ ( ( int )( r.Next(3) ) + 1, ( ( int )( r.Next(3) ) + 1 ] = "X";
}

```

**tablo** isminde **String** değerleri tutan bir dizi oluşturulur ve dizinin ilk satırına yazar isimleri, ilk sütununa da kitap isimleri konur. **For** döngüsünün sayacı birden başladığı için dizinin 0,0 koordinatlı ilk elemanına değer atanmaz.

|         | Yazar 1 | Yazar 2 | Yazar 3 | Yazar 4 |
|---------|---------|---------|---------|---------|
| Kitap 1 |         |         |         |         |
| Kitap 2 |         |         |         |         |
| Kitap 3 |         |         |         |         |
| Kitap 4 |         |         |         |         |

Daha sonra tablonun diğer elemanlarına rasgele X değerleri atanır. Bu değer hangi yazarın hangi kitabı yazdığını gösterecektir.

**r.Next(3) + 1** ifadesi 1 – 4 arasında rasgele sayı üretir. Bu sayı tablo dizisine indis olarak verildiğinde ise, kalan hücrelerde X değeri elde edilir.

|         | Yazar 1 | Yazar 2 | Yazar 3 | Yazar 4 |
|---------|---------|---------|---------|---------|
| Kitap 1 | X       |         |         |         |
| Kitap 2 | X       |         |         |         |
| Kitap 3 |         |         | X       |         |
| Kitap 4 |         |         |         | X       |

Tablonun tüm elemanlarını listelemek için iç içe iki **For** döngüsü kullanılmalıdır.

```
for ( int j=0; j<=tablo.GetLength( 0 ) - 1; j++ )
{
    for ( int h=0; h<=tablo.GetLength( 1 ) - 1; h++ ) {
        Label1.Text += System.Convert.ToString( tablo[ j, h ] );
    }
    Label1.Text += "\n";
}
```

Yazarlar ve Kitaplar tablosu hazırlandıktan sonra, hangi yazarın hangi kitabı yazdığını bulmak için yine tablo elemanları içinde dolaşip X değerini aramak gereklidir. İlk **For** döngüsü ile Kitaplar satırında, ikinci **For** döngüsü ile Yazarlar sütunlarında gezilir.

```
for (int j=0; j<=tablo.GetLength( 0 ) - 1; j++ )
{
    for (int h=0; h<=tablo.GetLength( 1 ) - 1; h++ )
    {
        // Tablonun her elemanının değeri
        // X değeri ile karşılaştırılır.
        if ( tablo[ j, h ] == "X" ) {
            Label2.Text = tablo[ 0, h ] + ", ";
            Label2.Text += tablo[ j, 0 ];
            Label2.Text += " kitabı yazıyor";

            break;
        }
    }
    Label1.Text += "\n";
}
```

Tablonun her elemanı kontrol edilir ve X değeri bulunduğu zaman yazar ismi ve kitap ismi ekrana yazdırılır. Yazar isimleri, dizinin ikinci boyutunun ilk sırasında tutulduğu için **tablo[0,h]** kodu ile ulaşılır. Kitap isimleri ise, dizinin ilk boyutunun ilk sırasında tutulduğu için **tablo[j,0]** kodu ulaşılır. Buradaki h ve j değişkenleri o anda kontrol edilen elemanın tablodaki indisleridir.

**break** ifadesi, o anda bulunan **For** döngüsünden çıkmayı sağlar. Bu örnekte ikinci **For** döngüsü yazarlar sütunu üzerinde döndüğü için, bu döngüden çıktıduğunda, ilk **For** döngüsüne tekrar geçilir. Bu sefer yeni bir kitap için

yazarlar kontrol edilir. Sonuçta bir kitabı birden fazla yazar yazmasına rağmen, görüntülenecek olan sadece ilk yazardır.

## While

- Verilen koşul gerçekleştiği sürece işlem yapılır.
- Sayacı değiştirmek için kod yazılması gereklidir.

```
int toplam = 0;
short sayac = InputBox("Bir sayı girin");
while (sayac > 0) {
    toplam += sayac;
    sayac -= 1;
}
```

**while** döngüsü bir koşul gerçekleştiği sürece çalışan döngüdür.

```
while(Koşul)
{
}
```

Birden ona kadar olan sayıların toplamını hesaplamak için, bir ve on arasındaki sayılar tek tek yazılp toplanabilir. İyi bir yöntem olmasa da sonucu verir. Ancak kullanıcının girdiği bir sayıya kadar toplam almak için bir döngü gereklidir.

```
int toplam = 0;
int sayac = int.Parse(txtSayiGiris.Text);
while (sayac > 0)
{
    toplam += sayac;
    sayac -= 1;

}
```

Burada kullanıcının girdiği sayıdan itibaren sıfıra kadar giden bir döngü kurulur. Döngü sayacın sıfırdan büyük olduğu her durum için çalışacaktır. Sayaç sıfırlandığında ise döngüden çıkarılır.

## Sonsuz Döngüler

**while** döngüsü sayaç ile kullanılırken, sayacın değiştirilmesine dikkat edilmesi gereklidir. Eğer sayaç değiştirilmemezse, **while** ifadesindeki koşul hep **true** değeri alacağı için sonsuz döngüye girilir.

Sadece sayacın kontrol edilmmediği durumlarda değil, koşulların yazılmasındaki mantık hataları da sonsuz döngüye sebebiyet verir.

```
int i = 0;

while (i < 10 | i > 5)
{
    label1.Text = "Sonsuz döngüye girildi";
    i += 1;
}
```

**For** döngüsünde sayaç, artırma ifadesindeki değerden fazla bir sayıda azaltılırsa yine sonsuz döngüye girilir. Bu döngünün çalışması, **int** veri tipinin alabileceği minimum değere ulaşınca hata ile sonlanır.

```
for(int i = 0; i <= 9; i += 3)
{
    MessageBox.Show("Sonsuz döngü");
    i -= 4;
}
```

## Uygulama

Bu uygulamada, bir satranç tahtası üzerindeki bir filin hareket alanı hesaplanır. Satranç tahtası rasgele taşlarla doldurulur. Verilen bir koordinatta bulunan fil, çapraz hareketlerine göre nereye ilerleyebileceği bulunur. Eğer filin önünde bir taş varsa, bu taşın bulunduğu yere ve daha gerisine ilerleyemeyecektir. Filin dört bir yanına çapraz olarak hareket edebileceği göz önünde bulundurulması gereklidir.

### Tahtanın doldurulması

1. Satranc isminde bir Windows projesi açın.
2. Form üzerine **TbHareketAlani** isminde bir **ListBox**, **btnGoster** isminde bir **Button** ekleyin.
3. **btnGoster** düğmesinin **Click** olayına 8 x 8 boyutunda bir dizi tanımlayıp dolduran kodları yazın. Bu dizi **bool** tipinde değerler taşıır. Verilen indisteki elemanın değeri **true** olması, o koordinatta bir taşın bulunduğu belirtir.

```
tbHareketAlani.Items.Clear();
int a,b;

Random r = new Random();
bool[,] tahta = new bool[ 8, 8 ];

for ( a=0; a<=7; a++ ) {
    for ( b=0; b<=7; b++ )
    {
        int sonuc = (r.Next() % 2);
```

```

        if (sonuc == 1)
        {
            tahta[ a, b ] = true;
        }
        else
        {
            tahta[ a, b ] = true;
        }
    }
}

```

### Hareket Alanı

Tahta üzerindeki bir fil, dört çapraz yöne doğru ilerleyebilir. Dizide çapraz olarak ilerlemek x ve y koordinatlarının eşit oranda artması ve azalması demektir. Dizide ilerlerken x ve y koordinatlarının sıfırdan küçük ve dizinin boyutundan büyük olmamasına dikkat edilmelidir. Dört farklı yöne göre, koordinatlar artacak ya da azalacaktır.

1. Fili tahta üzerine yerleştirmek için kullanıcidan koordinatları alın.

```
byte x = byte.Parse( txtFilinXkoordinati.Text );
byte y = byte.Parse( txtFilinYkoordinati.Text );
```

2. 0, 0 yönüne doğru olan yoldaki taşların kontrolünü yapın. Filin x ve y koordinatlarını birer düşürerek, koordinatlarda taş var mı yok mu kontrol edilir. Eğer taş yoksa bu kareye ilerlenebildiğini, **1bHareketAlani** liste kutusuna koordinat eklenecek gösterilir. Yol üzerinde bir taş varsa, daha fazla ilerlenemeyeceği için **while** döngüsünden çıkarılır.

```

int i = 1;
while ( x - i >= 0 & y - i >= 0 )
{
    if ( !( tahta[ x - i, y - i ] ) ) {
        1bHareketAlani.Items.Add( (x -
i).ToString() + " - " + (y - i).ToString() );
        i += 1;
    }
    else {
        break;
    }
}

```

3. 7, 0 yönüne doğru ilerlenir ve olası hareketler liste kutusuna eklenir.

```

i = 1;
while ( x + i < tahta.GetLength( 0 ) & y - i >=
0 ) {
    if ( !( tahta[ x + i, y - i ] ) ) {
        1bHareketAlani.Items.Add( (x +
i).ToString() + " - " + (y - i).ToString() );
        i += 1;
    }
    else {
        break;
    }
}

```

4. 7, 7 yönüne doğru kontrol yapılır.

```

        i = 1;
        while ( !(x + i) >= tahta.GetLength( 0 ) || y +
i >= tahta.GetLength( 1 ) ) {
            if ( !(tahta[ x + i, y + i ] ) ) {
                lbHareketAlani.Items.Add( (x +
i).ToString() + " - " + (y + i).ToString());
                i += 1;
            }
            else {
                break;
            }
        }
    }
}

```

5. 0, 7 yönüne doğru taşlar kontrol edilir.

```

        i = 1;
        while ( x - i >= 0 & y + i < tahta.GetLength( 1
) ) {
            if ( !(tahta[ x - i, y + i ] ) ) {
                lbHareketAlani.Items.Add( (x -
i).ToString() + " - " + (y + i).ToString());
                i += 1;
            }
            else {
                break;
            }
        }
    }
}

```

### Debug

Kodlar yazıldıktan sonra indislerin doğru kullanıldığını, döngülerde mantıksal hataları yapılmadığını kontrol etmenin en kolay yolu hata ayıklayıcı ile çalışmaktadır. Bu örnekte birçok döngü kurulmuştur, filin dört hareket yönündeki engeller kontrol edilir. Tüm hareketlerin işleyişine Debug ile kolayca bakılır.

1. **TextBox** ile filin koordinatlarının alındığı yere **BreakPoint** koyn ve projeyi çalıştırın.
2. x ve y koordinatlarına 0 değerini girin. Filin, tahtanın sol üst köşesinde olduğu varsayıılır.

Bu durumda fil hangi yönde ilerleyebilir? **Step Into** komutu ile kodlar arasında ilerleyin ve hangi döngü içine girdiğini bulun.

3. **Locals** panelinde i değişkeninin değerini izleyin. Aynı panelde tahta dizisinin, o anda kontrol edilen değerine bakın. x ve y değerlerinin bir fazlasını alarak, diğer değerin **true** ya da **false** olduğunu kontrol edin.

## Konu 3: Hata Yakalama

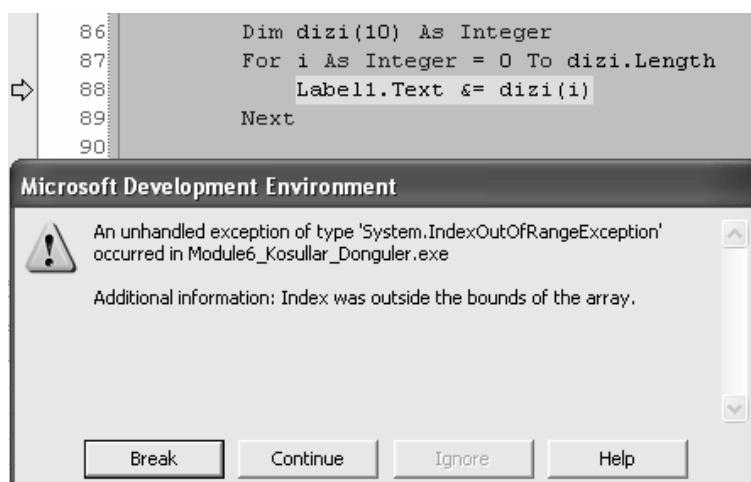
Bir uygulama geliştiricisi program yazarken çok çeşitli hatalarla karşılaşabilir. Visual C# .NET ortaya çıkan hata durumlarında, çok detaylı hata mesajlarını uygulama geliştiricisine gönderir. Bu hata mesajları, hataların nerede, nasıl yapıldığını çok detaylı bir şekilde gösterir. Hataların en ince ayrıntısına kadar işlenmesi, uygulama geliştirmede büyük kolaylık sağlar.

Visual C# .NET hata mesajları, çalışma zamanı (Run Time) ve tasarım zamanı (Design Time) hataları olarak ayrılabilir.

Tasarım zamanı hataları, kodların yazılması sırasında derleyici tarafından bulunan ve **Task List** panelinde gösterilen hatalardır. **Task List** panelinde hatanın açıklaması, hatanın projenin hangi dosyasında ve dosyanın kaçinci satırında bulunduğu gösterir. Tasarım zamanı hataları sözdizimi yanlış kullanıldığında meydana gelir.

Çalışma zamanı hataları, uygulama çalışırken yapılması imkânsız bir işlemin gerçekleştirilmesi sırasında meydana gelir. Örneğin **TextBox** ile bir sayının alınması sırasında, kullanıcı **String** tipinde bir değer girerse çalışma zamanında bir hata oluşur.

```
int []dizi = new int[10];
for (int i = 0; i <= dizi.Length; i++)
{
    Label1.Text &= dizi(i)
}
```



Buradaki hata mesajı, dizinin büyüklüğünün dışında bir indis verildiğini belirtir.

Visual C# .NET dilinde uygulama geliştirirken oluşabilecek tüm hatalar .NET FrameWork çatısı altında **Exception** sınıfları halinde tanımlanır. Örneğin dizinin büyüklüğünden farklı bir indis verildiğinde **IndexOutOfRangeException** hatası ortaya çıkar. Tüm hatalar gibi bu hata da **Exception** taban sınıfından türemiştir.

## Try Catch Finally

# Try Catch Finally

- Çalışma zamanında çıkan hataların işlenmesini sağlar.
- Try, hata doğurabilecek kodları tutar.
- Catch, hata yakalandıktan sonra çalışacak kodları tutar.
- Finally, her iki durumdan sonra çalışacak kodları tutar.

```
int dosya = FreeFile();
try {
    string kayit = "Kayıt Zamanı: " + Now;
    kayit += "\r\n" + "Uygulama kayıtları...";
    FileOpen(dosya, "C:\\Log.txt", OpenMode.Binary, OpenAccess.Write);
    FilePut(dosya, kayit);
} catch (Exception ex) {
    MsgBox(ex.Message);
} finally {
    FileClose(dosya);
}
```

Çalışma zamanında ortaya çıkan hatalar uygulamanın beklenmedik bir şekilde sonlanması neden olur. Uygulamanın devam etmesi için bu hataların yakalanıp işlenmesi gereklidir. **Try** **Catch** **Finally** blokları içinde, çalışma zamanı hataları meydana geldiği durumlarda çalışması istenen kodlar yazılır. **Try** bloğu içine, çalışırken hata üretebilecek kodlar yazılırken, **Catch** bloğu içine, hata oluştuğunda yapılması gereken işlemler yazılır.

```
int sayı, sonuc;

try
{
    Random r = new Random();
    sayı = r.Next(3);
    sonuc = 100 / sayı;
    MessageBox.Show("Bölme işlemi başarılı, sonuç: " +
+ sonuc.ToString());

}
catch (Exception ex)
{
    MessageBox.Show("Bölme işlemi başarısız. Hata
Mesajı: " + ex.Message);

}
```

Bu örnekte üretilen rasgele bir sayı ile bölme işlemi yapılıyor. Sayı sıfır değerini aldığımda bölüm işlemi hata verecektir. Dolayısıyla bu işlem **Try** bloğu içine yazılmalıdır. **Catch** bloğunda, işlemin başarısız olduğunu belirten bir mesaj

yazılır. **Exception** nesnesinin **Message** özelliği, hatanın olduğu zaman üretilen mesajı tutar. **Exception** nesnesinin özellikleri **Catch** içinde kullanılmayacaksızın tanımlanmasına gerek yoktur.

```
try
{
}
Catch
{
    label1.Text = "Exception kullanılmıyor.";
}
```

**Finally** bloğunda, **Try Catch** içinde yapılan tüm işlemlerden sonra çalıştırılacak kodlar yazılır. **Finally** bloğunda yazılan kodlar hata meydana gelse de çalışmaya devam eder.

```
try
{
    // Dosya aç
    // Dosya işlemleri
}
catch (Exception ex)
{
    // Dosya açılırken veya işlem yapılırken
    // hata meydana geldi.
}
finally
{
    // Dosya kapat
}
```

**Finally** bloğunda, genellikle, kullanılan kaynaklar serbest bırakılır. Örnekte, bir dosya açılıyor. Dosya açma veya dosyaya veri yazma işlemlerinde bir hata meydana geldiğinde, **Catch** ifadesinde bu hata yakalanıp ilgili mesaj kullanıcıya gösterilir. **Finally** bloğu her halükarda çalışacağı için dosya kapama işlemi burada yapılır.

## Lab 1: Şifreleme Algoritması

Bu uygulamada, verilen bir yazı şifrelerek bir dizi sayıya çevrilir. Bu sayılar, yazıda geçen karakterlerin Ascii kodlarının karışmış bir halidir. Şifreyi çözmek için, şifrelemede izlenen yolların tersi uygulanır.

**İPUCU:** Şifreleme algoritmalarında yazılı şifrelerken izlenen yolların geri dönüşü olmalıdır. Örneğin rasgele sayılar kullanılarak şifrelenen bir yazılı, tekrar rasgele sayılar kullanarak çözülemez.

### Şifreleme:

Verilen yazının şifrelenmesi üç etaptan oluşur:

- Yazının karakterleri Ascii kodlarına çevrilir.
- Kodlar, gerekiyorsa başlarına 0 konarak, 4 haneli yapılır ve sıralı halde bir **String** değişkeninde tutulur.

- Sıralı şekilde yazılan kodlar, bir baştan bir sondan karakter alınarak tekrar düzenlenir.

Örnek: "acf" kelimesinin şifrelenmesi:

- a c f karakterleri Ascii kodlarına çevrilir.

a = 97

c = 99

f = 102

- Kodlar başlarına sıfır konarak 4 haneli yapılır.

0097

0099

0102

Sıralı halde bir **String** değişkenine yazılır.

009700990102

- Sayıının ortasına kadar, önce baştan daha sonra sondan rakam alınarak tekrar yazılır. Siyah olarak gösterilen rakamlar, dizinin sonundan alınmıştır.

0

**0 2**

0 2 0

0 2 0 0

0 2 0 0 9

0 2 0 0 9 1

0 2 0 0 9 1 7

0 2 0 0 9 1 7 0

0 2 0 0 9 1 7 0 0

0 2 0 0 9 1 7 0 0 9

0 2 0 0 9 1 7 0 0 9 0

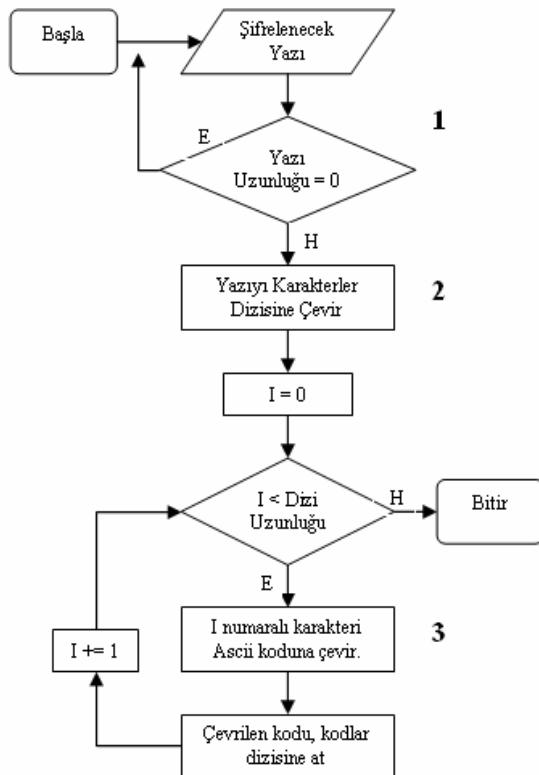
0 2 0 0 9 1 7 0 0 9 0 9 = Şifre

### Projenin açılması

- Visual Studio ortamında, Sifreleme isminde bir Windows projesi açın.
- Açılan forma **lblSifre** isminde bir **Label** kontrolü, **btnSifrele** isminde bir **Button** kontrolü ekleyin. Bu kontroller kullanıcidan alınan yazının şifrelenip görüntülenmesini sağlayacaktır.
- Açılan forma **lblDesifre** isminde bir **Label** kontrolü ve **btnSifreyiCoz** isminde bir **Button** kontrolü ekleyin. Bu kontroller şifrelenmiş yazının **lblSifre** kontrolünden alınarak, şifrelenip görüntülenmesini sağlayacaktır.

**DİKKAT:** Sifreleme algoritmasının tüm kodları **btnSifrele** kontrolünün **Click** olayına yazılacaktır.

### Ascii Kodlarına Çevirme



- Şifrelenecek yazının girilmesi için gerekli kodu yazın. Bir yazı girilene kadar kullanıcıdan yazı istemek için **do while** döngüsünü kullanın.

```

string yazi = null;

// Algoritma 1 - 1
do {
    yazi = txtSifrelenecekYazi.Text;
} while ( ! ( yazi.Length > 0 ) );

```

- Girilen yazının karakterlerini bir dize toplamak için **String** değişkeninin **ToCharArray()** metodunu kullanın.

```

// Algoritma 1 - 2
char[] karakterler = yazi.ToCharArray();

```

- Karakterlerin Ascii kodu karşılığını tutmak için **kodlar** isminde bir dizi yaratın. Karakterler dizisindeki tüm elemanlar üzerinde işlem yapmak için bir döngü kurun. Karakterler dizisindeki her karakteri **Asc** hazır fonksiyonu ile Ascii koduna çevirin.

```

int uzunluk = karakterler.Length - 1 ;
string[] kodlar = new string[ uzunluk + 1 ];

int i;
// kelimedeki her karakterin ASCII kodu alınır
for ( i=0; i<=uzunluk; i++ )
{

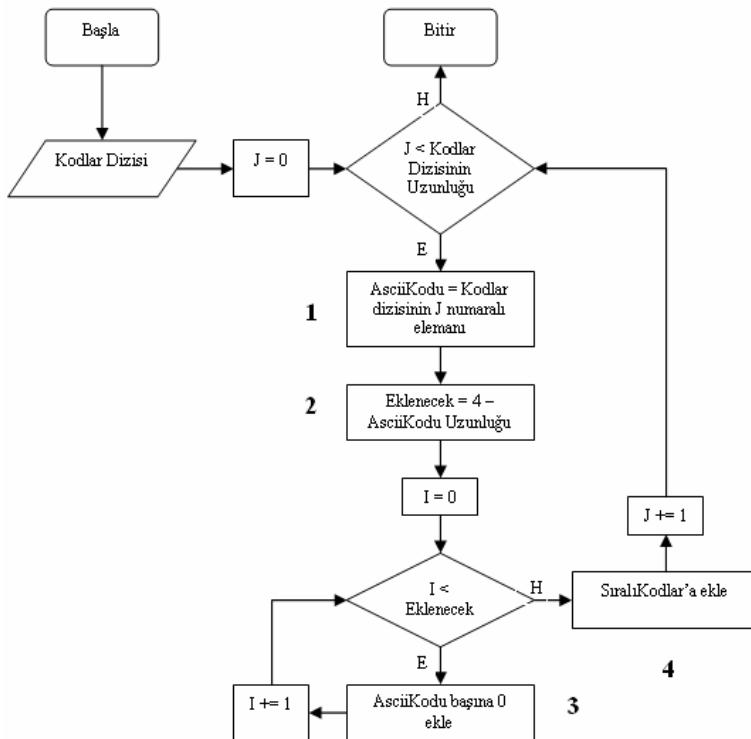
```

```
// Algoritma 1 - 3
kodlar[ i ] = ( Microsoft.VisualBasic.Strings.Asc(
karakterler[ i ] ) ).ToString();
}
```

4. Bu algoritma sonunda elde edilen **kodlar** dizisi, şifrelenecek olan yazının her karakterinin Ascii kodunu tutar. Bu dizi diğer algoritmanın giriş değeri olarak kullanılacaktır.

### Sıralı Kodlara Çevirme

Ascii karakter kodları 0 – 255 arasında değer alır. Dolayısıyla her kod maksimum üç haneli olacaktır. Şifre oluşturulurken yapılan son düzenlemede kolaylık sağlamak için, bu kodlar 4 haneli yapılır. Daha sonra bu kodlar diziden çekilerek **Sıralıkodlar** adlı bir **String** değişkenine yazılır.



1. Dizideki kodları sıralı bir şekilde tutmak için **Sıralıkodlar** adlı bir değişken tanımlayın. İlk algoritmadan alınan Ascii kodlarını tutan **kodlar** dizisi üzerinde bir döngü kurun.

```
string Sıralıkodlar = null;
short j = 0;
while ( j <= kodlar.Length - 1 ) {
    j += 1;
}
```

**DİKKAT:** 2 – 4 etaplarında yapılacak tüm kodlar **While** döngüsünün içine yazılacaktır.

Bu döngüde kullanılacak Ascii kodunu bir değişkene atan kodu yazın.

```
// Alogritma 2 - 1
string AsciiKodu = kodlar[ j ];
```

2. **AsciiKodu** değişkeninde tutulan kodun 4 haneli hale getirilmesi için kaç tane sıfır eklenmesi gerektiğini bulun. Eklenecek sıfırların sayısı, 4 – **AsciiKodu** değişkeninin uzunluğu kadardır. Örneğin 192 kodlu bir değişkene eklenmesi gerekken sıfır sayısı  $4 - 3 = 1$  tanedir.

```
byte eklenecek = System.Convert.ToByte( 4 - AsciiKodu.Length );
```

3. Eklenecek sayı kadar çalışacak bir döngü içinde, sıfır ekleme işlemini yapın.

```
for ( i=0; i<eklenecek - 1; i++ ) {
    // Alogritma 2 - 3
    AsciiKodu = AsciiKodu.Insert( 0, "0" );
}
```

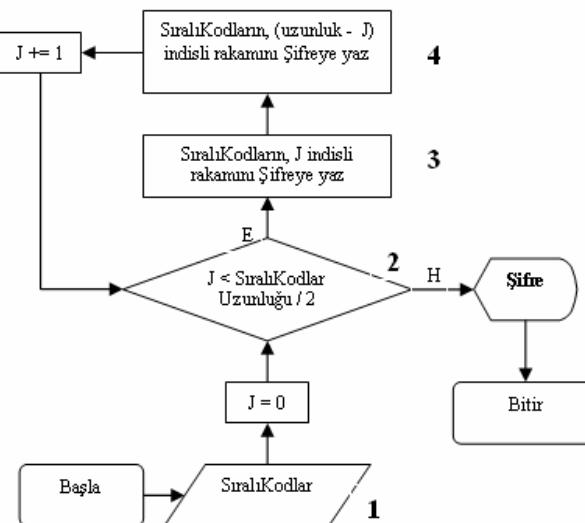
4. Düzenlenmiş **AsciiKodu**, **SıralıKodlar** değişkenine yazın ve sayacı bir artırarak diğer Ascii koduna geçin.

```
// Alogritma 2 - 4
Siralikodlar += AsciiKodu;
j += 1;
```

Algoritma sonunda ortaya çıkan değer, karakterlerin 4 haneli Ascii kodlarını tutan bir **String** değişkenidir. Bu değişken diğer algoritmada, tekrar düzenlenmek üzere kullanılacaktır.

### Şifrenin oluşturulması

Bir önceki algoritmada elde edilen **SıralıKodlar** değişkeni halen istenilen şifreli yazı değildir. Çünkü 4 haneli kodlar sıralı bir şekilde durur ve kolayca çözülebilir. Şifrenin ilk bakışta anlaşılmasını daha da zorlaştırmak için, sıralanmış kodlar biraz daha karıştırılır.



- Döngüde kullanılacak J sayacını sıfırlayın ve şifrenin tutulacağı değişkeni tanımlayın

```
// Algoritma 3 - 1
j = 0;
string Sifre = null;
```

- Sıralı Kodlar** değişkeni üzerinde yapılacak işlem sayısı, bir baştan bir sondan ilerlediği için, değişkenin uzunluğunun yarısı kadardır. Sayacın bu uzunluğa kadar tanımlı olan bir döngü oluşturun.

```
// Algoritma 3 - 2
while ( j < Siralikodlar.Length / 2 ) {

    j += 1;
}
```

**DİKKAT:** 3 – 4 etaplarında yazılacak tüm kodlar **Do While** döngüsünün içine yazılacaktır.

- Şifreye, **Sıralı Kodların** j indisli karakterini ekleyin.

```
//Algoritma 3 - 3
Sifre &= Mid(Siralikodlar, j + 1, 1)
```

- Şifreye, **Sıralı Kodların** uzunluk – j indisli karakterini ekleyin.

```
// Algoritma 3 - 3
Sifre += Strings.Mid( Siralikodlar, j + 1, 1 );
```

- Sonuç olarak çıkan şifre, girilen yazının Ascii kodlarının karışık düzende tutulması ile oluşturulur.

```
// Algoritma 3 - 4
Sifre += Siralikodlar.Substring(Siralikodlar.Length - j - 1,
1 );
```

## Şifreyi Çözmek

Şifreleme algoritması kullanılarak oluşturulan şifrenin çözülmesi, izlenen yolların tersi uygulanarak gerçekleştirilir. Deşifre algoritması iki etaptan oluşur.

- Bir baştan bir sondan karakter alınarak şifrelenen Ascii kodları, sıralı kodlar haline dönüştürülür.
- 4 haneli olarak duran sıralı kodlar, karakterlere çevrilir. Karakterler ardı ardına konarak deşifre işlemi gerçekleştirilir.

Örnek: "acı" kelimesinin şifrelenmiş hali 020091700909 şeklidir. Bu kelime şifrelenirken, karakterleri 4 haneli Ascii kodların çevrilmiş ve bu kodların rakamlarının sırası değiştirilmiştir. Bu şifrenin önce 4 haneli sıralı kodlar haline getirilmesi için, şifrelenen yöntemin tersi işlenir. Sırayla okunan rakamlar önce başa daha sonra sona yazılır.

Şifre: 0 2 0 0 9 1 7 0 0 9 0 9

Sıralı kodlara çevrim:

| Sıranın ilk yarısı | Sıranın son yarısı |
|--------------------|--------------------|
| 0                  | 2                  |
| 0 0                | 0 2                |
| 0 0 9              | 1 0 2              |
| 0 0 9 7            | 0 1 0 2            |
| 0 0 9 7 0          | 9 0 1 0 2          |
| 0 0 9 7 0 0        | 9 9 0 1 0 2        |

Sonuç olarak elde edilen sıralı Ascii kodları, sıranın ilk yarısı ve son yarısının birleşimi olur: 0097 0099 0102

**DİKKAT:** Sıranın ilk yarısı oluşturulurken, rakamlar sona eklenir. Ancak sıranın son yarısı oluşturulurken rakamlar başa eklenir.

Bu 4 haneli kodlar **String** değerinden **Integer** değerine çevrilir ve bu değerlerin karşılığı olan karakterler yazılır.

0097 → 97 → a

0099 → 99 → c

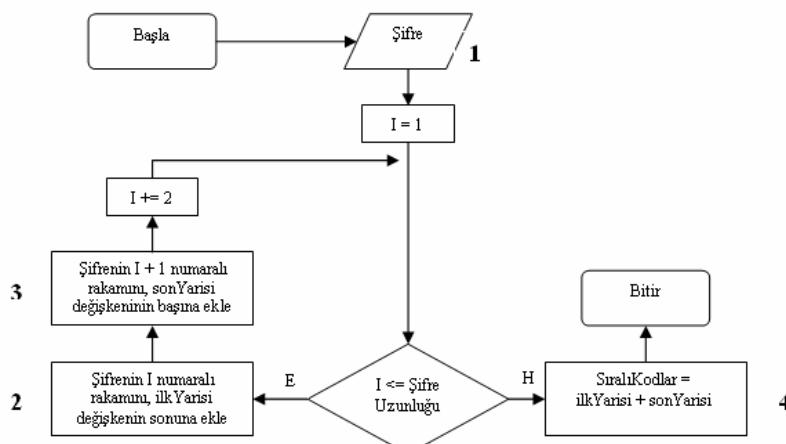
0102 → 102 → f

Elde edilen karakterler birleştirildiğinde şifre çözülmüş olur: "acf"

**DİKKAT:** Deşifre algoritmasının tüm kodları **btnSifreyiCoz** kontrolünün **Click** olayına yazılacaktır.

### Şifreyi Sıralı Kodlara Dönüşümme

Bu algoritma verilen şifreyi sıralı Ascii kodlarına dönüştürür.



1. Şifreyi **lblSifre** etiketinden alın ve sıralı kodların oluşturulması için gereken değişkenleri tanımlayın.

```

// Algoritma 1 - 1
string Sifre = lblSifre.Text;
string Siralikodlar = null;
short i = 0;
  
```

```

// Başa ve sona rakam ekleneceği için
// değişkenlere başlangıç değerleri verilir
    string ilkYarisi = "";
    string sonYarisi = "";

2. Şifrenin tüm elemanları üzerinde bir döngü kurarak, sıralı kodların ilk
ve son yarısını oluşturun. Kodların ilk yarısı, şifrenin tek haneli
rakamları ile; kodların son yarısı, şifrenin çift haneli rakamları ile
oluşturulur. Dolayısıyla döngünün sayacı ikişer ikişer artmalıdır.
Şifrenin i indisli rakamını sıranın ilk yarısına, yanındaki rakamı (i + 1
indisli rakamı) sıranın son yarısına ekleyen kodları yazın.

for ( i=1; i<=System.Convert.ToInt16( Sifre.Length ); i+=2 )
    // Algoritma 1 - 2
    // Sıranın ilk yarısının sonuna rakam eklenir.
    ilkYarisi += Strings.Mid( Sifre, i, 1 );

    // Algoritma 1 - 3
    // Sıranın son yarısının başına rakam eklenir.
sonYarisi = sonYarisi.Insert( 0, Sifre.Substring(i , 1 ) );
}

```

3. Sıralı kodların ilk yarısı ve son yarısı birleştirilir. Elde edilen değer, 4 haneli Ascii kodlarının sırayla tutulduğu bir **String** değeridir.

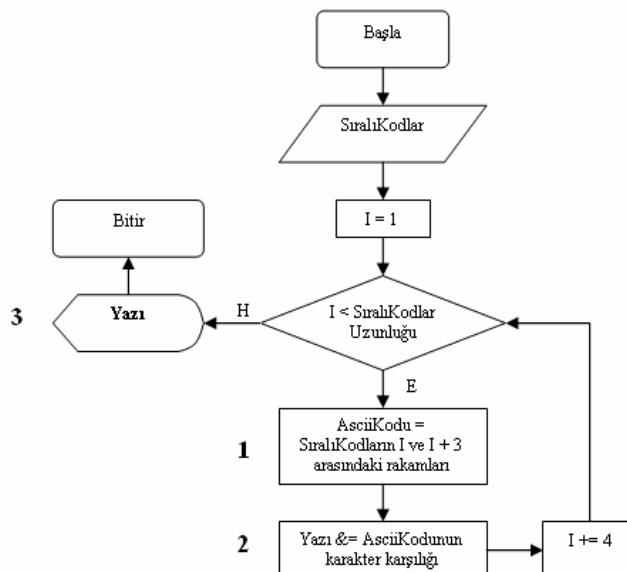
```

// Algoritma 1 - 4
Siralikodlar = ilkYarisi + sonYarisi;

```

### Sıralı Kodların Okunması

İlk algoritmada elde edilen sıralı Ascii kodları, bu algoritmada okunarak karakterlere çevrilir ve şifre çözülmüş olur.



1. Şifre çözüldüğü zamanki değerinin tutulacağı değişkeni tanımlayın ve sıralı kodlar üzerinde bir döngü kurun. Sıralı kodların 4 haneli kodlardan

oluştuğu için, döngüde bir seferde 4 rakam alınacaktır. Bunun için döngünün sayacı 4 artırılmalıdır.

```
i = 0;
string yazi = "";
while ( i < SiraliKodlar.Length ) {

    i += 4;
}
```

**DİKKAT:** 2 – 3 etaplarında yazılacak tüm kodlar **Do While** döngüsünün içine yazılacaktır.

2. Döngü her seferinde bir Ascii kodu alır. Bu değeri tutan bir değişken tanımlanır ve sıralı kodlardan 4 haneli rakam bu değişkene atanır.

```
int AsciiKodu;
```

```
// Algoritma 2 - 1
AsciiKodu = int.Parse( SiraliKodlar.Substring(i , 4 ) ) ;
```

3. Alınan Ascii kodunun karakter karşılığı bulunur ve **yazi** değişkenine eklenir.

```
// Algoritma 2 - 2
yazi += Microsoft.VisualBasic.Strings.Chr( AsciiKodu );
```

4. Döngü sonunda elde edilen değer **lblDesifre** etiketine yazılır.

```
// Algoritma 2 - 3
lblDesifre.Text = yazi;
```

## Lab 2: Sıralama Algoritması

Bu algoritma, bir dizinin elemanlarını küçükten büyüğe sıralar.

### Dizinin Doldurulması

1. Sıralama isimli bir Windows projesi açın
2. Form üzerine biri **1bSirasız**, diğeri **1bsiralı** isimli iki **ListBox** ekleyin.

Bu kontroller dizinin sırasız ve sıralı halini listeler.

3. **btnListele** ve **btnSırala** isimli iki **Button** ekleyin.

4. Formun kod tarafına geçin ve bir dizi tanımlayın. Bu dizi bir çok yordamın içinde kullanılacağı için global olarak tanımlanır.

```
public string[] dizi = new string[ 5 ];
```

5. **btnListele** düğmesinin **Click** olayına, diziyi karışık bir şekilde isimlerle dolduran kodları yazın:

```
dizi[ 0 ] = "Enis";
dizi[ 1 ] = "Engin";
dizi[ 2 ] = "Tamer";
dizi[ 3 ] = "Kadir";
dizi[ 4 ] = "Fulya";

int i;
for ( i=0; i<=dizi.Length - 1; i++ ) {
    ListBox1.Items.Add( dizi[ i ] );
}
```

## Dizinin Sıralanması

Sıralama algoritması, dizi üzerinde bir döngü kurar ve sırayla dizinin bir elemanı seçilir. Bu eleman için bir başka döngü kurulur ve seçilen elemanın indisine kadar olan tüm elemanlarla bir karşılaştırma yapılır. Küçük olan sıranın başına konmak için büyük olan ile yer değiştirilir.

### Örnek

1. Dizinin 2. elemanı seçilir: "Engin"

Dizinin 2. indisine kadar olan elemanlarla karşılaştırılır. "Engin" değeri alfabetik sırada "Enis" değerinden küçük olduğu için, bu iki değer yer değiştirilir.

Sıra, Engin Enis Tamer Kadir Fulya olur.

2. Dizinin 3. elemanı seçilir: "Tamer"

Dizinin 3. indisine kadar olan elemanlarla karşılaştırılır. "Tamer" değeri, "Enis" ve "Engin" değerlerinden büyük olduğu için sıralama değişmez.

3. Dizinin 4. elemanı seçilir: "Kadir"

Dizinin 4. indisine kadar olan elemanlarla karşılaştırılır. "Kadir" < "Tamer" olduğu için bu iki değer yer değiştirilir.

Sıra, Engin Enis Kadir Tamer Fulya olur.

"Kadir" değeri, "Enis" ve "Engin" değerlerinden büyük olduğu için sıralama değişmez.

4. Dizinin 5. elemanı seçilir: "Fulya"

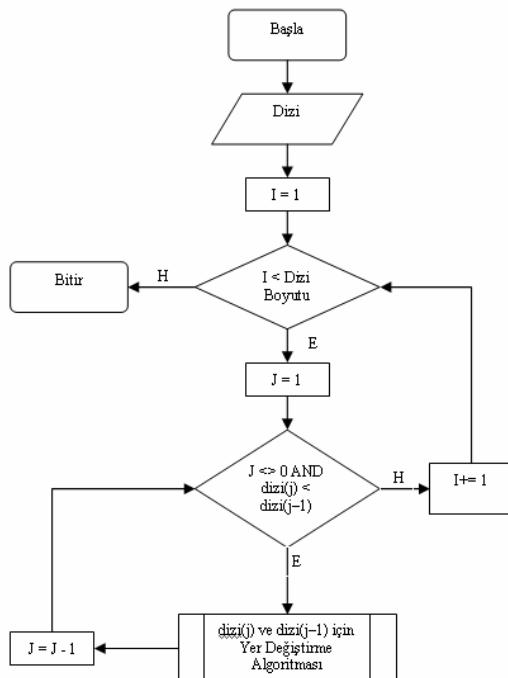
Dizinin 5. indisine kadar olan elemanlarla karşılaştırılır. "Fulya" < "Tamer" olduğu için bu iki değer yer değiştirilir.

Sıra, Engin Enis Kadir Fulya Tamer olur.

"Fulya" < "Kadir" olduğu için bu iki değer yer değiştirilir.

Sıra, Engin Enis Fulya Kadir Tamer olur.

Dizideki tüm değerler kontrol edildiği için algoritmadan çıkarılır.



1. **btnsira** düğmesinin **Click** olayına, dizi üzerinde bir döngü tanımlayın.

Bu döngü dizinin (1 indisli) ikinci elemanından başlayarak dizi sonuna kadar devam edecektir. Daha sonra bu döngü içine başka bir döngü daha yazın. Bu döngü, ilk döngünün sayacından başlar ve sıfır olana kadar devam eder. İkinci döngünün amacı, ilk döngüde seçilen elemani, dizinin başına kadar olan elemanlarla karşılaştırmaktır.

```

int i;

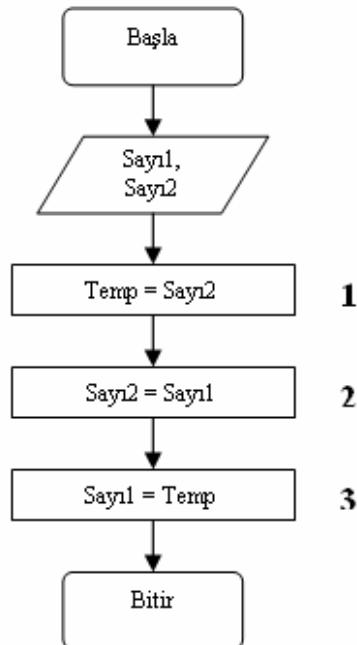
for ( i=1; i<=dizi.Length - 1; i++ ) {
    int j = i;
    while ( j != 0 && String.Compare(dizi[ j ], dizi[ j - 1
]) == -1 ) {
        // Yer değiştirme Algoritması

        j -= 1;
    }
}
  
```

**while** döngüsü, **j** değeri sıfır olana kadar ve dizinin kontrol edilen değeri bir önceki değerden küçük olana kadar devam eder. Burada, dizi elemanlarının kontrolünün sadece bir defa (bir önceki eleman ile) yapıldığı düşünülebilir. Ancak küçük eleman yer değiştirildiğinde **j** değeri bir düşürülür. Döngü tekrar çalıştığı zaman, aynı eleman bu sefer dizinin kalan elemanlarıyla karşılaştırılır.

**AndAlso** operatörü, **j** değerinin sıfır olma durumunda diğer kontrolün yapılmaması için kullanılır. Diğer kontrolde **dizi(j - 1)** ifadesi, negatif indisli değere ulaşılmak istediği için hata mesajı verir.

2. Yer değiştirme algoritması, bir değişkenin değerinin geçici bir yerde tutulması ile gerçekleştirilir.



Sıralama algoritmasında dizinin **j** ve **j - 1** indisli değerleri yer değiştirilir. **while** döngüsü içinde “Yer değiştirme Algoritması” yazan yorum satırını kaldırın ve yerine algoritma kodlarını yazın.

```
// Yer değiştirme
string temp = dizi[ j - 1 ];
dizi[ j - 1 ] = dizi[ j ];
dizi[ j ] = temp ;
```

3. **İbsirali** liste kutusunda dizinin yeni sırasını görüntüleyen kodları yazın.

```
for (int t=0; t<=dizi.Length - 1; t++ ) {
    ListBox2.Items.Add( dizi[ t ] );
}
```

## Lab 3: Arama Algoritması

Arama algoritmaları, sıralı bir liste üzerinde bir değerin aranmasıdır. Karışık sırada olan bir listede yapılan arama, ancak listenin başından sonuna kadar tüm elemanlarının kontrol edilmesi ile gerçekleştirir. Bu yöntem büyük dizilerde performansı düşürür. Belirli bir sırada olan dizilerde ise daha hızlı arama yöntemleri kullanılmalıdır. Bu labda ikili arama yöntemi (Binary Search) incelenecektir.

**DİKKAT:** İkili arama yöntemi sadece sıralı bir dizi üzerinde uygulanabilir. Ya da elimizdeki dizi öncelikle sıralanır.

**NOT:** İkili arama yönteminde büyük küçük kıyaslaması yapıldığından dizinin sıralı olması gereklidir.

## Dizinin sıralanması

Arama algoritması sıralı bir dizi üzerinde çalışacağı için, dizi oluşturulduktan sonra sıralanması gereklidir.

1. İkiliArama isminde bir Windows projesi açın.
2. Forma **btnAra** isimli bir **Button** ve **TbDizi** adlı bir **ListBox** ekleyin.
3. Kod sayfasına geçin ve global bir dizi tanımlayın.

```
public int[] dizi = new int[ 11 ];
```

4. Formun **Load** olayına diziyi rasgele sayılar ile doldurmak için gereken kodları yazın.

```
int i;
Random r = new Random();

for (i=0; i<=10; i++ )
{
    dizi[ i ] = r.Next(1000);
}
```

5. Diziyi sıralayın ve değerlerini **TbDizi** adlı listeye ekleyin.

```
Array.Sort( dizi );
for ( i=0; i<=10; i++ ) {
    ListBox1.Items.Add( dizi[ i ] );
}
```

## Arama algoritması

İkili arama algoritması, dizi üzerinde aranacak değeri önce sıranın ortasındaki değerle karşılaştırır. Dizi küçükten büyüğe sıralı olduğu için, eğer aranan değer ortadaki değerden küçükse arama, dizinin ilk yarısında devam eder. Dizinin diğer yarısı aranan değerden büyük değerler içeriği için, aramaya dahil edilmez.

### Örnek

Küçükten büyüğe sıralı dizi üzerinde 9 değeri aranacaktır.

Dizi: 1 2 4 7 9 10 12 18

1. Son, baş ve orta değişkenleri tanımlanır: Son değeri dizinin son elemanın indisini, baş değeri dizinin ilk elemanın indisini, orta değeri ise son + baş / 2 değerini alır. Orta değeri virgülü bir değer alırsa tam sayıya çevrilir.
2. Başlangıç olarak baş -1, son dizi uzunluğu değerini alır.

Baş = -1

Son = 8

Orta =  $(8 - 1) / 2 = 3$

3. Dizinin orta indisli değeri alınır. Dizi(3) = 7

4. Aranan 9 değeri, yediden büyük olduğu için, dizini son yarısında aranır. Baş değişkenine orta değeri verilirse, dizinin başlangıç indisinden değiştirilir, böylece aramalar dizinin son yarısında gerçekleşmiş olur.

Dizi: **9 10 12 18**

Baş = **3**

Son = **8**

$$\text{Orta} = (8 + 3) / 2 = \mathbf{5}$$

5. Dizinin orta indisli değeri alınır. Dizi(5) = 10

6. 9 değeri, ondan küçük olduğu için, kalan dizinin ilk yarısında aranır.

Dizi: **9 10**

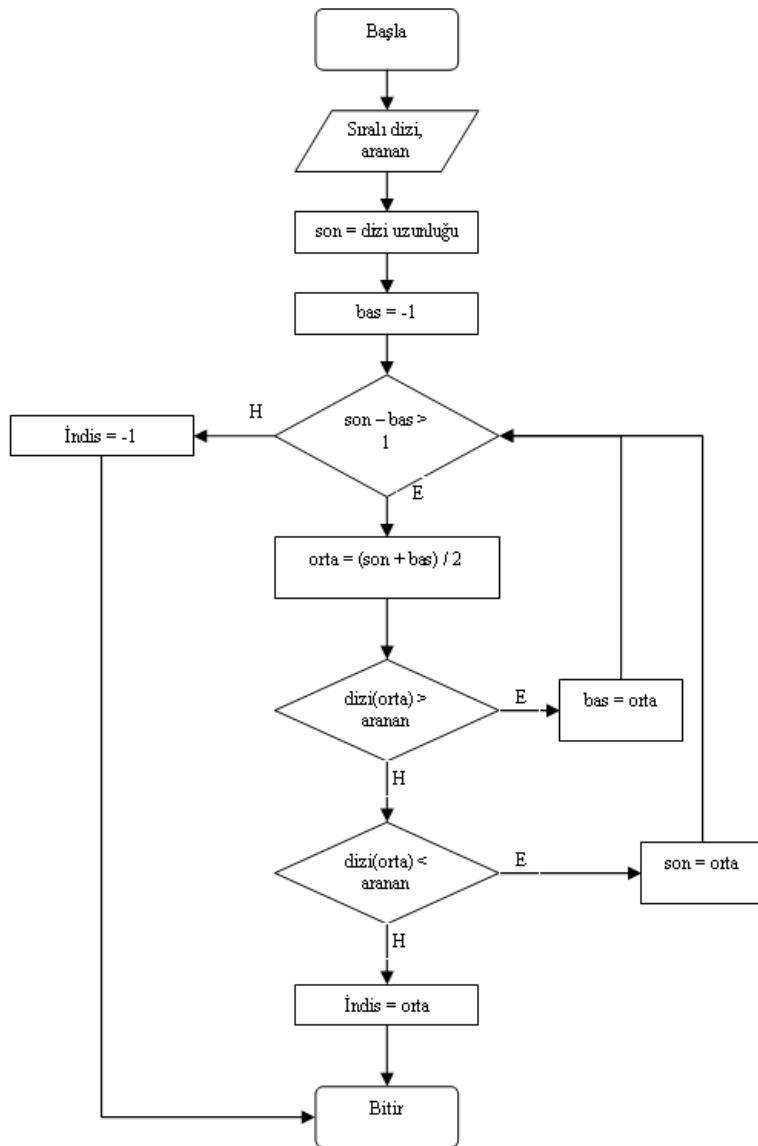
Baş = **3**

Son = **5**

$$\text{Orta} = (5 + 3) / 2 = \mathbf{4}$$

7. Dizinin ortasındaki değer alınır: 9

8. Böylece 9 değerinin indis orta değeri olur.



## Kodlar

Arama algoritmasının kodları **btnAra** düğmesinin **Click** olayına yazılacaktır.

- Algoritma için gerekli **bas**, **son** ve **orta** değişkenlerini tanımlayın ve başlangıç değerlerini verin. Aranan değerin indisini tutmak için de bir değişken tanımlayın.

```

int son = dizi.Length;
int bas = -1;
int orta;
int indis;
  
```

- Kullanıcıdan aranacak değeri girmesini isteyin.

```
int hedef = int.Parse(textBox1.Text);
```

3. Dizide aranacak değer kalmadığı zaman çıkan bir döngü kurun. **Son** ve **bas** değerleri arasındaki fark bire düştüğünde, dizide aranacak değer kalmamıştır.

```
while ( son - bas > 1 ) {  
    }  
}
```

4. **while** döngüsü **içine**, dizinin **orta** indisli değerini alan ve bu değeri aranan değerle karşılaştırın kodları yazın.

```
orta = ( son + bas ) / 2;  
if ( dizi[ orta ] > hedef ) {  
    son = orta;  
}  
else if ( dizi[ orta ] < hedef ) {  
    bas = orta;  
}  
else {  
    indis = orta;  
    MessageBox.Show( "İndis: " +  
indis.ToString());  
    return;  
}
```

Eğer dizinin ortasındaki değer aranan değerse, indis bulunmuş demektir. **orta** değişkeni kontrolün yapıldığı değerin indisini tuttuğu için, sonuç **orta** değeri olur ve yordamdan çıkarılır.

5. Eğer istenen değer bulunamadan döngüden çıkışılırsa, indis -1 değerini alır. **End while** ifadesinden **sonra**, aranan değerin bulunamadığını belirten kodu yazın.

```
indis = -1;  
MessageBox.Show( "İndis: " + indis + " Aranan değer  
bulunamadı");
```

## Modül Sonu Soruları & Alıştırmalar

### Özet

- ↳ Algoritma kurmak
- ↳ Dump Coding çözümlemesi
- ↳ Akış diyagramları

1. **if** ifadesi hangi veri tipini kontrol eder?
2. **if - switch** karar yapılarının farkları nelerdir?
3. **true | false & false**  
**(true | false) & false**  
ifadeleri hangi değerleri döndürür?
4. 4 boyutlu bir dizinin tüm elemanlarını doldurmak için, dizi üzerinde kaç tane döngü kurulmalıdır?
5. Uygulamaları derledikten sonra hatalar hangi yollarla görülebilir?
6. **Finally** bloğundaki kodlar ne zaman çalışır?
7. Sıralı arama yönteminde (Linear Search) dizinin elemanlarının sıralı olması gerekli midir?
8. Sıralı arama yöntemin bir dizi üzerinde uygulayın.

## Modül 7: Fonksiyonlar ve Yordamlar

### Hedefler

- ↳ Sub – Function kullanımı
- ↳ .NET Tarih, String, Matematik fonksiyonları
- ↳ Online Offline yardımının etkin kullanımı

Uygulama geliştirirken, bir işlemin birçok yerde kullanıldığı zamanlar olur. Bu gibi durumlarda bir kere yazılan kodlar, farklı yerlerde tekrar yazılır. Uygulama üzerinde bir değişiklik yapılmak istenirse, tekrar yazılan kodların tek tek bulunup değiştirilmesi gereklidir. Böylece hem uygulamanın yazımı zorlaşır hem de değişik yapmak giderek imkânsız hale gelir. Bu problemler, birçok yerde yapılması istenen işlemlerin fonksiyonlar ve yordamlar içinde yazılması ile çözülür. Sadece fonksiyon ve yordamların isimleri kullanılarak, istenen yerlerde kodlar çalıştırılır.

Yapılan işlemin sonucunda oluşan değer isteniyorsa fonksiyonlar kullanılır. Örneğin veritabanına yeni bir kullanıcı ekledikten sonra kullanıcının ID numarası isteniyorsa fonksiyon kullanılmalıdır. Eğer yapılan işlemlerin sonunda bir değer döndürülmüyorsa yordamlar kullanılır. Örneğin bir **ComboBox** kontrolüne öğe ekleme işlemi yordam içine yazılabilir.

.NET çatısındaki nesnelerin birçok fonksiyon ve yordamları vardır. Tüm fonksiyon ve yordamların kaç parametre aldığı, geriye dönüş değerinin ne olduğu, hangi nesneye ait oldukları ezberlenemez. Dolayısıyla Visual Studio yardımının kullanılması kaçınılmazdır.

Bu modül tamamlandıktan sonra:

- Yordam ve fonksiyon kullanarak kodlarınızın yönetilebilirliğini ve esnekliğini artıracak,
- Fonksiyon ve yordamların farklarını ayırt edebilecek,

- .NET çatısındaki tarih ve zaman, matematik, String fonksiyonlarını tanıyacak,
- Offline ve Online yardımcı etkin bir şekilde kullanabileceksiniz.

## Konu 1: Void (Yordam)

### Void

- Dönüş değeri olmayan kod bloklarıdır.
- Birçok yerde kullanılacak kodlar, yordamlar ile gruplanmalıdır.

```
void Temizle()
{
    Label1.Text = "";
    ListBox1.Items.Clear();
}
```

Yordamları dönüş değeri olmayan kod bloklarıdır. Bu yordamlar **void** ifadesi ile belirtilir.

```
void Yordamİsmi()
{
}
```

Uygulama içinde birçok yerde çalışacak kodlar yordam içinde yazılır. Bu kodlar, içine yazıldıkları yordamın ismi ile çağrılarak, istenilen yerde tekrar çalıştırılabilir. Örneğin bir uygulama başlarken form üzerindeki kontrollerin temizlenmesi gerekiyorsa, bu kodları bir daha yazmamak için yordam kullanılabilir.

```
void Temizle()
{
    Label1.Text = "";
    ListBox1.Items.Clear();
}
```

Yordamı tanımlarken parantezler içine, alabileceği parametreler yazılır. Eğer yordam parametre almıyorsa parantezlerin içi boş bırakılır.

```
void YazılımUrunleriEkle()
{
    ComboBox1.Items.Add("Yazılım Uzmanlığı");
    ComboBox1.Items.Add("Yazılım Mühendisliği");
    Label1.Text = "Yazılım paketleri eklendi...";
}
```

Yordamları tanımladıktan sonra başka bir yordam veya fonksiyon içinde kullanılır. Yordamı kullanmak için, gerekli yere isminin yazılması yeterlidir.

```
Void DersleriListele()
{
    switch( ComboBox1.SelectedIndex )
    {
        case 0:
            Temizle();

            ListBox1.Items.Add("Access - İlişkisel
Veritabanları");
            ListBox1.Items.Add("Programlamaya Giriş ve
Algoritma");
            ListBox1.Items.Add(".NET Framework");
            ListBox1.Items.Add("VB.NET ile Windows Tabanlı
Programlama");
            ListBox1.Items.Add("ASP.NET ile Web Tabanlı
Programlama");

            Label1.Text = "Yazılım Uzmanlığı dersleri
yüklandı." ;
            break;

        case 1:
            Temizle();

            ListBox1.Items.Add("SQL Server Veritabanı
Yönetimi");
            ListBox1.Items.Add("Visual Studio .NET ile
Uygulama Geliştirme");
            ListBox1.Items.Add("ADO.NET ile Veri Yönetimi ve
XML");
            ListBox1.Items.Add("XML Web Services, .NET
Remoting ve COM+");
            ListBox1.Items.Add("Proje Yönetimi");

            Label1.Text = "Yazılım Mühendisliği dersleri
yüklandı.";

            break;

        default:
            Temizle();
            Label1.Text = "Yazılım paketi seçiniz.";
            break;
    }
}
```

Burada **ComboBox** kontrolünden seçilen değerin kontrolün indis üzerinden yapılması, **YazılımUrunleriEkle** yordamında eklenen elemanların sırası değiştirirse problem yaratır. Liste kutusuna eklenen dersler yanlış paketlerde gözükmür. Ancak **ComboBox** kontrolünün seçili metni üzerinden kontrol yapılırsa

da, eklenen isimler değiştiği zaman bir problem ortaya çıkar. Bu durumda iki yordamın birbirine bağımlılığı görülür. Bu örnekte, bir yordamda değişiklik yapıldığı zaman diğer yordamın çalışma şekli de kontrol edilmelidir.

**Label** ve **ListBox** kontrollerini temizleyen kodlar sadece iki satır olduğu için **Temizle** yordamında yazılmayabilirdi. Ancak bu kodlar **DersleriListele** yordamında üç defa kullanıldığı için her değişiklikte, kodun yazıldığı üç yer bulunup gerekli düzeltmeler yapılacaktır. Örneğin temizle işlemi, liste kutusunda "Dersler" metni gözükecek şekilde değiştirebilir. Bu durumda, değişikliği sadece **Temizle** yordamında yapmak yeterli olur.

```
void Temizle()
{
    Label1.Text = "";
    ListBox1.Items.Clear();
    ListBox1.Items.Add("Dersler: ");
}
```

## Parametre Kullanımı

### Parametre Kullanımı

- Parametreler ile yordamların davranışları değiştirilir.
- Params, aynı tipten sınırsız parametre girilmesini sağlar.

Yordamların bazı değerlere göre farklı işlem yapması istenebilir. İşlemin bağlı olduğu bu değerlere parametre veya argüman denir. Yordamlar parametre alacak şekilde tanımlanıp, çağrıldıkları sırada istedikleri parametreleri verilerek kullanılır.

```
void YordamIsmini(VeriTipi Parametre1, VeriTipi Parametre2, ...)
{
```

```
}
```

Örneğin uygulamanın birçok yerinde kullanıcıya bilgi vermek amaçlı mesaj kutuları kullanılır. Eğer bu mesajlar bir yordam içine yazılırsa, daha sonra mesajları bir **Label1** üzerinde gösterilecek şekilde değiştirmek kolay olacaktır. Yordamın göstereceği mesajlar parametre olarak verilmesi gereklidir.

```
Void MesajGoster(string mesaj)
{
    Label1.Text = mesaj;
}

Void Yordam1()
{
    //...
    MesajGoster("1. Yordam içinden çağırılır.");
}

Void Yordam2()
{
    //...
    MesajGoster("2. Yordam içinden çağırılır.");
}

Void Yordam3()
{
    // ...
    MesajGoster("3. Yordam içinden çağırılır.");
}
```

Yordamları çağrıırken tüm parametrelerin belirtilen veri tipte verilmesi gereklidir. Yordamları tanımlarken parametreleri isimleri ve veri tipleri belirtilmelidir. Ayrıca parametreler değer tipi ya da referans tipi olarak geçileceği belirtilmelidir.

Yordamın normal akışından çıkmak istenirse **Return** ifadesi kullanılabilir.

```
Void MusteriBilgisi(int MusteriId)
{
    if( ! MusteriId > 0 )
        Return

    // MusteriId değerine göre
    // müşteri bilgileri veritabanından çekilir.
}
```

Diziler parametre olarak kullanıldıklarında büyülükleri verilmez. Fakat parantezler kullanılarak, verilen parametrenin dizi olduğu belirtilmelidir.

```
void MatrisTopla(int [,] matris1 , int [,] matris2)
{
    int x = matris1.GetLength(0);
    int y = matris1.GetLength(1);

    if (x != matris2.GetLength(0) || y != matris2.GetLength(1))
    {
        MessageBox.Show("Matris boyutlarının büyülükleri birbirile aynı olmalıdır.");
    }
}
```

```

        return;
    }

    int[,] sonuc = new int[x - 1, y - 1];

    for(int i = 0; i < x; i++)
    {
        for (int j = 0 ; j < y; j++)
        {
            sonuc[i, j] = matris1[i, j] + matris2[i, j];
        }
    }
}

```

Diziler yordamlara parametre olarak geçirilirken, sadece isimleri verilir.

```

int [,]m1 = {{1, 3, 5}, {7, 9, 11}};
int [,]m2 = {{0, 2, 4}, {6, 8, 10}};

MatrisTopla(m1, m2)

```

## Params

Yordamları ve fonksiyonları çağrıırken parametrelerin mutlaka girilmeleri gereklidir. Ancak bazı durumlarda yordamlara ve fonksiyonlara girilecek parametrelerin sayısı tasarım zamanında belli olmaz. **params** anahtar kelimesi ile yordamlara, aynı veri tipinde parametre dizisi verilebilir. **params** ile verilen dizi yordamın son parametresi olarak tanımlanmalıdır.

```

public void YasOrtalaması( string sınıf, params
byte[] Yaşlar ) {
    int toplam = 0;
    double ortalama = 0.0;

    int i;
    for ( i=0; i<=Yaşlar.Length - 1; i++ ) {
        toplam += Yaşlar[ i ];
    }

    // Parametre verilmezse i = 0 olur
    if ( i > 0 ) {
        ortalama = toplam / i;
    }
    MessageBox.Show( sınıf + " sınıfının yaş
ortalaması: " + ortalama );
}

```

Yaş ortalamasını hesaplayan bu yordamın ilk parametresi verilmek zorundadır. **params** ile tanımlı olan dizi, yordam çağrırlarken girilen tüm parametreleri tutar. Fakat girilen bu parametrelerin veri tipleri aynı olmak zorundadır. Bu örnekte girilecek yaşlar **Byte** tipinde olacaktır.

```

        private void Button1_Click2( System.Object sender,
System.EventArgs e ) {
    // İlk parametre verildikten sonra,
    // istenen sayıda parametre verilebilir
    YasOrtalaması("YU6112", 45, 14, 25, 28);

    // Yaşlar parametre olarak verilmeyebilir
    YasOrtalaması("YU6112");
}

```

Parametrelerin sınırlı olmaması, dizilere eleman ekleme işlemini kolaylaştırır. Örneğin bir diziye birçok eleman eklemek için, bu elemanların bir dizi içinde parametre verilmesi gereklidir.

```

public string[] Raf;
public void KitapEkle( string[] Kitap ) {
// Raf dizisine, kitaplar dizisinin
// elemanları eklenir.
}

```

Bu yordamın kullanımı için, eklenecek değerlerin önce bir diziye aktarılması gereklidir. Yordamın yazılması kolay ancak kullanımı zordur. Bu yordamı kullanacak programcının işi **params** ile kolaylaştırılır.

```

public void Mesaj( string msg ) {
    Label1.Text += msg + "\n";
}

public void KitapEkle( string Kitap, byte Genisletme
) {
    if ( Genisletme == 0 ) {
        Mesaj( "Dizi boyutu genişletilemedi..." );
        return ;
    }

    // Genişletme faktörü kullanıcıya bırakıldığı için
    // dizide boş alanlar olabilir. İlk boş alan bulunup
    // veri buraya aktarılır.
    int i;
    while ( i < Raf.Length ) {
        if ( Raf( i ) == ( ( System.String[] )( "" ) )
    ) {
        Raf( i ) = ( ( System.String[] )( Kitap
) );
        return;
    }
    i += 1;
}

// Dizide boş yer yoksa yeniden boyutlandırılır.
string[] Temp = new string[ i + Genisletme ];
System.Array.Copy( Raf, Temp, Raf.Length );
Raf = Temp;
Raf[ i + 1 ] = Kitap;
}

```

Önce, diziye bir tek eleman ekleyen yordam yazılmıştır. Dizinin tüm alanları doluyaşa, genişletme parametresinde verilen değer kadar tekrar boyutlandırılmıştır.

Genişletme değişkeni **Byte** veri tipinde tanımlı olduğu için negatif değer alamaz. Dolayısıyla dizinin boyutunun küçültülmesi engellenmiş olur. Bu yordam tek başına kullanılabilir olduğu gibi, diziye birçok eleman ekleyecek yordama yardımcı niteliğindedir.

```
public void Ekle( params string[] Kitaplar ) {
    for ( i=0; i<=Kitaplar.Length - 1; i++ ) {
        // Genişletme faktörü 5 ile tek tek kitap eklenir.
        KitapEkle( Kitaplar[ i ], 5 );
    }
    Mesaj( Kitaplar.Length + " kitap rafa eklendi." );
}
```

Bu yordam ise sınırsız parametre alarak, dizi işlemlerinde programcıya kolaylık sağlar.

```
private void Button1_Click( System.Object sender,
    System.EventArgs e ) {
    Ekle( "Kitap1", "Kitap2", "Kitap3" );
}
```

## **Void Main**

### Void Main

- Başlangıç yordamıdır.
- Application sınıfı kullanılarak, istenen formlar yüklenir.

```
public static void Main()
{
    Application.Run( new Form1() );
}
```

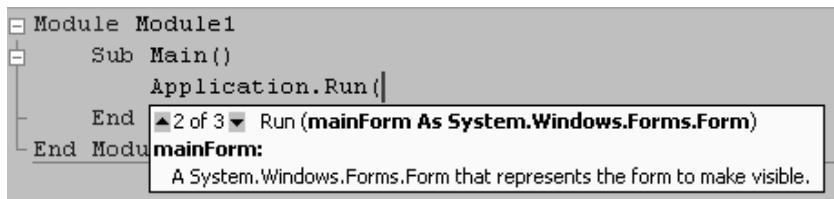
Yeni bir yordam tanımlarken **Main** yordamı hariç istenilen isim verilebilir. **Main** yordamı bütün uygulamaların giriş noktasıdır. Windows uygulamalarında formlar yüklenmeden önce o form içinde tanımlı **Main** yordamı çalıştırılır. Bu **Main** yordamında **Application** sınıfı başlangıç formunu **Run** metodu ile

yükler. **Application** sınıfı, .NET Framework çatısında, uygulamaları başlatmak, yönetmek ve sonlandırmak için kullanılır.

Projenin özelliklerinden başlangıç nesnesi **Sub Main** olarak ayarlanırsa, uygulama çalıştığı zaman tüm projede **Main** yordamı arar. Windows uygulamaları geliştirirken **Main** yordamı yazılırsa başlangıç formunun da bu yordam içinde belirtilmesi gereklidir. Bu yordam bir modülün içinde tanımlanabilir.

```
public static void Main()
{
    Application.Run( new Form1() );
}
```

**Application** sınıfının **Run** metodu, parametre olarak başlangıç formu ister. Uygulama başladığı zaman hangi formun çalışması isteniyorsa, bu formdan oluşturulup parametre olarak verilir. **New** anahtar kelimesi, sınıfları oluşturmak için kullanılır.



Başlangıç formu olarak seçilen bir Windows formunda **Main** yordamı tanımlanırsa, bu yordam **static** olarak tanımlanmalıdır. **static** metodlar uygulama genelinde paylaştırılan sabit metodlardır.

```
public static void Main()
{
    MessageBox.Show("Başlangıç formları kod ile
yüklenmelidir.");
}
```

Başlangıç formu olarak ayarlanmış bir formun içine bu **Main** yordamı tanımlanırsa, formu yüklemek için herhangi bir kod yazılmadığı için uygulama sadece mesaj kutusunu gösterecektir.

## Konu 2: Fonksiyonlar

### Function

- İşlem yapıldıktan sonra değer döndürülür.
- Return ifadesinden sonraki kodlar çalıştırılmaz.

```
bool KontrolOk()
{
    if (TextBox1.Text.Length > 0 &
    ComboBox1.SelectedIndex > -1)
    {
        return false;
    }
    return true;
}
```

Fonksiyonlar bir işlem yaptıktan sonra geriye değer döndürürler. Örneğin bir çarpma fonksiyonunun dönüş değerini, parametre olarak verilen iki sayının çarpımı olacaktır. Fonksiyonların tanımları değişkenler gibidir.

```
DönüşVeriTipi Fonksiyon(VeriTipi Param1, ...)  
{  
}  
}
```

Fonksiyonların geriye dönüş değerleri **Return** ifadesi ile yapılır.

```
bool KontrolOk()
{
    if (TextBox1.Text.Length > 0 & ComboBox1.SelectedIndex
> -1)
    {
        return false;
    }

    return true;
}

private void Button1_Click( System.Object sender,
System.EventArgs e ) {

    if (! KontrolOk())
    {
        MessageBox.Show("Seçiminizi yaptıktan sonra devam
edebilirsiniz.");
        return;
    }
}
```

```

        }

    // Kontrol tamamlandıktan sonra yapılacak işlemler
}

```

Bu fonksiyonun çalışması **Return** ifadesinden sonra yazılan değerin döndürülmesiyle sonlanır. Burada dikkat edilmesi gereken nokta, fonksiyon değer döndürdükten sonra sonlandığı için **Return** ifadesinden sonra gelen hiçbir kod çalıştırılmaz. Eğer dönüş değerini belirledikten sonra başka bir işlemin yapılması isteniyorsa, fonksiyon ismi kullanılır. Fonksiyonun ismi bir değişken gibi gözüke de, temsil ettiği değer fonksiyonun dönüş değeridir.

```

float GunlukKur(string Cinsi)
{
    switch(Cinsi)
    {
        Case "d":
        Case "D":
            Return 1.43;

        Case "e":
        Case "E":
            Return 1.81;

        Case "s": "
        Case "S":
            Return 2.91;
    }
}

public double KurHesapla( float Miktar, string Cinsi )
{
    return Miktar * GunlukKur( Cinsi );
}

// Bu satırdan sonra yazılan kodlar işlenmez.
private void Button1_Click1( System.Object sender,
System.EventArgs e ) {
    Label1.Text      =      System.Convert.ToString(
KurHesapla( -1000, "d" ) );
}

```

Örnek: Sınıf geçme notunun hesaplanması, geriye bir sonuç döndürüleceği için fonksiyon ile yazılması gereklidir. Parametre olarak final ve vize notları alınır ve bu değerlerle hesaplanan geçme notu sonuç olarak döndürülür. Vize notlarının girilmesi zorunlu değildir, dolayısıyla bu değerler **params** olarak verilebilir.

```

public int NotHesapla( int Final,     float
VizeKatSayisi, params int[] vizeler ) {
    int vizeToplam = 0;
    double vizeOrtalama = 0.0;

    int i;

```

```

        for ( i=0; i<=vizeLength - 1; i++ ) {
            vizeToplam += vize( i );
        }

        if ( i > 0 ) {
            vizeOrtalama = vizeToplam / i;
        }

        float finalKatSayisi = 1 - VizeKatSayisi;

        return finalKatSayisi * Final + VizeKatSayisi *
vizeOrtalama;
    }
}

```

Fonksiyonun ilk parametresi final notudur. Final notu bir tane olacağı için girilmesi zorunludur. Daha sonra vize notlarının ortalaması hesaplanarak final notu ile toplanır. Parametre olarak verilen vize katsayısı, vize notlarının ortalamadaki ağırlıklarını belirler.

```

private void Button1_Click( System.Object sender,
System.EventArgs e ) {
    int gecmeNotu;
    gecmeNotu = NotHesapla( 70,
System.Convert.ToSingle( 0.6 ), 90, 80, 86, 75, 90 );
    MsgBox( gecmeNotu );
}

```

## Fonksiyonlar ve Yordamların Aşırı Yüklenmesi

### Function – Sub OverLoad

- Aynı isimde birden fazla metod yapılmasıdır.
- Parametreleri farklı olmalıdır.

```

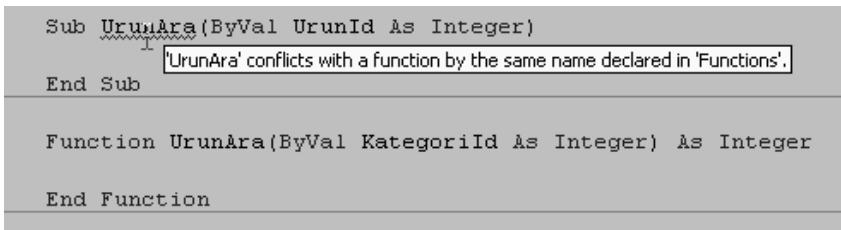
public void UrunAra( int UrunId ) {
    // Ürün numarasına göre arama yapılır.
}
public int UrunAra( string UrunIsimi )
{
    // Ürün ismine göre arama yapılır.
    // Bulunan ürünün numarası döndürülür.
}

```

Fonksiyon ve yordamları kullanırken, aynı isimde birden fazla tanımlanabildikleri görülür. Buna Aşırı Yüklenme (**OverLoad**) denir. Bir yordamın ve fonksiyonun aşırı yüklenmesi kullanımını kolaylaştırır. Aynı isimde farklı seçenekler sunması metodların kullanımı arttırmıştır.

```
public void UrunAra( int UrunId ) {  
    // Ürün numarasına göre arama yapılır.  
}  
  
public int UrunAra( string UrunIsmi ) {  
    // Ürün ismine göre arama yapılır.  
    // Bulunan ürünün numarası döndürülür.  
}  
  
public int UrunAra( string UrunIsmi, DateTime  
UretimTarihi ) {  
    // Ürün ismine ve üretim tarihine göre arama yapılır.  
    // Bulunan ürünün numarası döndürülür.  
}  
  
public int UrunAra( DateTime UretimTarihi ) {  
    // Üretim tarihine göre arama yapılır.  
    // Bulunan ürünün numarası döndürülür.  
}
```

Metotların aynı isimde olmasının ayrimı parametrelerin veri tipi ve sayısına göre yapılır. Metotların isimleri, parametre sayısı ve parametrelerin veri tipleri metotların imzalarını (**Method Signatures**) oluşturur. Örneğin ürün numarasına göre arama yapan yordamın imzası **UrunAra(int)** şeklindedir. **int** parametre alan bir **UrunAra** isminden başka bir yordam veya fonksiyon tanımlanamaz. Fonksiyonların dönüş tipleri ile imzaları tanımlanmaz.



```
Sub UrunAra(ByVal UrunId As Integer)  
    'UrunAra' conflicts with a function by the same name declared in 'Functions'.  
End Sub  
  
Function UrunAra(ByVal KategoriId As Integer) As Integer  
End Function
```

Metotları aşırı yüklerken dikkat edilmesi gereken bazı noktalar vardır.

- İmzaları aynı olan metodlar tanımlanamaz
- Fonksiyonlar yordamlarla, yordamlar da fonksiyonlarla aşırı yüklenebilir.
- Fonksiyonlar dönüş tiplerine göre aşırı yüklenemez.

## Konu 3: String Fonksiyonları

### String Fonksiyonları

- CompareTo
- Concat
- CopyTo
- EndsWith & StartsWith
- ToUpper & ToLower
- Join
- SubString
- Trim, TrimEnd, TrimStart

**String** fonksiyonları, kullanıldığı **String** değeri üzerinde verilen parametrelere göre değişen işlemler yaparlar. Sonuç olarak geriye döndürdükleri değerler, fonksiyonun işleyiş amacına göre değişir.

- **CompareTo**

Bu fonksiyon, işlemin yapılacakı değeri parametre olarak verilen değerle karşılaştırır. İki değer bir birine eşitse 0, parametredeki değer alfabetik olarak önde ise 1, değilse -1 değeri döndürülür.

```
string yazi1 = "BilgeAdam";
string yazi2 = TextBox1.Text;

switch (yazi1.CompareTo(yazi2)) {
    case 0:
        MessageBox.Show("Yazilar birbirine
eşit");
        break;
    case 1:
        MessageBox.Show(yazi1 + ", " + yazi2 +
" kelimesinden sonra geliyor");
        break;
    case -1:
        MessageBox.Show(yazi1 + ", " + yazi2 +
" kelimesinden önce geliyor");
        break;
}
```

- **Concat**

**String** değerlerini birleştirmek için kullanılır. Parametre tipi **params** olduğu için, sınırsız **String** değişkeni birleştirilebilir.

```

string kurum = "BilgeAdam";

Label1.Text = String.Concat("Kurum: ", kurum, "Şubeler: ",
"\n", "Fatih", "Bakırköy", "Kadıköy", "Beşiktaş", "Town
Center");

```

- **CopyTo**

BU fonksiyon ile bir **String** değişkenin belli bir kısmı, bir karakter dizisine kopyalanır. Ayrıca kopyalanacak dizinin hangi indisten itibaren başlanacağı da belirtilir.

```

string yazi = "BilgeAdam";
char[] Karakterler = new char[ 11 ];

// Yazının 5. karakterinden itibaren alınan 4 karakter,
// karakterler dizisinin 3. indisinden başlanarak
// dizeye kopyalanır.
yazi.CopyTo( 5, Karakterler, 3, 4 );

// Karakterler dizisinin son hali:
// _ _ _ A d a m _ _ _ -

```

Burada dikkat edilmesi gereken nokta, karakterlerin kopyalanacağı dizinin büyüğünün yeterli olup olmadığıdır. Dizinin kopyalanmaya başlanacak indis ile kopyalanacak karakterlerin uzunluğunun toplamı, dizi büyüğünden küçük olmalıdır

- **EndsWith & StartsWith**

Bu fonksiyonlar, **String** değişkeninin, parametrede verilen **String** değeriyle başladığını ya da bittiğini gösterir. Geriye dönüş değer **Boolean** tipindedir.

```

bool degisken.EndsWith(string deger)

bool degisken.StartsWith(string deger)

string HtmlTag = "<table>";
if ( HtmlTag.StartsWith( "<" ) &
HtmlTag.EndsWith( ">" ) )
{
    MessageBox.Show( "Yazım doğru" );
}

```

- **ToUpper & ToLower**

**ToUpper**, **String** değişkenin içindeki küçük karakterleri büyüğe; **ToLower**, büyük karakterleri küçüğe çevirir.

```

string yazi = "bilgeADAM";

MessageBox.Show(yazi.ToUpper());
// Sonuç: BİLGEADAM
MessageBox.Show (yazi.ToLower());
// Sonuç: bilgeadam

```

- **Join**

Bir **String** dizisindeki elemanları, parametre olarak verilen ayraç karakteri ile birleştirerek tek bir **String** değişkeni döndürür.

```
string [] yazi = {"İsim", "Soyad", "Adres", "Email",
"Telefon"};
MessageBox.Show(string.Join(";", yazi));
// Sonuç: İsim;Soyad;Adres;Email;Telefon
```

- **SubString**

Verilen bir String değerinin, bir bölümünü String olarak döndüren fonksiyondur. İstenen karakterlerin hangi indisten başlayacağı parametre olarak geçirilir. Bu durumda, başlangıç karakterinden sona kadar okunur. Ancak fonksiyonun, kaç karakter okunacağını belirten bir parametre kabul eden aşırı yüklemesi de vardır.

```
string yazi = "BilgeAdam";
MessageBox.Show (yazi.Substring(5));
// Sonuç : Adam

MessageBox.Show (yazi.Substring(5, 2));
// Sonuç : Ad
```

- **Trim, TrimEnd, TrimStart**

**Trim** fonksiyonu, parametre olarak verilen bir karakteri, String değişkeninin başından ve sonundan kaldırır.

**TrimEnd** fonksiyonu parametrede verilen karakteri String değişkeninin sadece sonundan, **TrimStart** ise sadece başından kaldırır.

```
string yazi = "-----Merhaba-----";

MessageBox.Show (yazi.Trim("-"));
// Sonuç: Merhaba

MessageBox.Show (yazi.TrimEnd("-"));
// Sonuç: -----Merhaba

MessageBox.Show (yazi.TrimStart("-"));
// Sonuç: Merhaba-----
```

## Konu 4: Matematiksel Fonksiyonlar

### Matematiksel Fonksiyonlar

- Abs
- Ceiling & Floor
- Cos, Sin, Tan
- Exp
- Log
- Max & Min
- Pow
- Sqrt

Uygulamalarda çoğu zaman matematiksel hesaplamalara ihtiyaç duyulur. Bu hesaplamaları kolaylaştıran hazır matematik fonksiyonları vardır. Bu fonksiyonlar .NET Framework çatısında **System.Math** uzay alanının içinde tanımlanmıştır.

- **Abs**

Verilen bir sayının mutlak değerini döndürür. Dönen değer her durumda pozitif olacaktır.

```
Math.Abs(-123)  
// Sonuç: 123
```

- **Ceiling & Floor**

**Ceiling** fonksiyonu, **Double** veri tipinde verilen bir sayıdan büyük, en küçük tamsayıyı verir. **Floor** fonksiyonu verilen sayıdan küçük, en büyük tam sayıyı verir.

```
Math.Ceiling(-12.231231)  
// Sonuç: -12
```

```
Math.Ceiling(12.231231)  
// Sonuç: 13
```

```
Math.Floor(-12.231231)  
// Sonuç: -13
```

```
Math.Floor(12.231231)  
// Sonuç: 12
```

- **Cos, Sin, Tan**

Bu fonksiyonlar temel trigonometrik işlemleri gerçekleştirir. **Cos** fonksiyonu verilen derecenin kosinüsünü, **Sin** sayının sinüsünü ve **Tan** sayının tanjantını hesaplar. Parametre olarak verilen derece radyan (360 derece) değeri olarak kabul edilir.

```
double Derece = 90;
Math.Cos(Math.PI * Derece / 180);
Math.Sin(Math.PI * Derece / 180);
Math.Tan(Math.PI * Derece / 180);
```

- **Exp**

Bu fonksiyon, e sabitinin değerini (yaklaşık 2,718281 değerini), parametrede verilen sayı ile üssünü alır.

```
Math.Exp(4)
// Sonuç yaklaşık: 54,59815
```

- **Log**

Logaritmik hesaplamlalar için kullanılan bir fonksiyondur. Taban parametresi verilmemezse sayının e tabanında logaritmasını alır.

```
Math.Log(1000, 10)
// Sonuç: 3
```

```
Math.Log(Math.E)
// Sonuç: 1
```

- **Max & Min**

**Max** fonksiyonu verilen iki sayıyı karşılaştırarak büyük olanı, **Min** fonksiyonu ise sayılardan küçük olanı döndürür.

```
Math.Max(100, 200)
// Sonuç: 200
Math.Min(100, 200)
// Sonuç: 100
```

- **Pow**

İlk parametrede verilen bir sayının, ikinci parametredeki değer kadar üssünü alır.

```
Math.Pow(10, 3)
// Sonuç: 1000
```

- **Sqrt**

Verilen sayının karekökünü hesaplar.

```
Math.Sqrt(441)
// Sonuç: 21
```

## Konu 5: Tarih ve Zaman Fonksiyonları

### Tarih ve Zaman Fonksiyonları

- DateAdd
- DateDiff
- CompareTo
- DaysInMonth
- IsLeapYear
- Parse
- ToLongDateString & ToLongTimeString
- ToShortDateString & ToShortTimeString

Tarih ve zaman fonksiyonları **Date** veri tipi üzerinde hesaplamalar yapan fonksiyonlardır. Bu fonksiyonlar **System.DateTime** uzay alanında tanımlıdır.

- **CompareTo**

**String** ifadelerinde olduğu gibi, tarih ve zaman değerleri üzerinde de karşılaştırma yapılabilir. **CompareTo** fonksiyonu, işlem yapılan tarih ile parametre olarak verilen tarihi karşılaştırır. Parametredeki tarih küçükse 1, büyükse -1 veya eşitse 0 döndürür.

```
DateTime d = #03/23/2002#;
MessageBox.Show (d.CompareTo(Now).ToString());
```

- **DaysInMonth**

İlk parametrede verilen yılın, ikinci parametrede verilen ayında kaç gün olduğunu döndürür.

```
DateTime.DaysInMonth(2002, 2)
// Sonuç: 28
```

- **IsLeapYear**

Verilen bir yılın artık yıl olup olmadığını hesaplar. Dönüş değeri **True** ya da **False** tipindedir.

```
DateTime.IsLeapYear(1200)
// Sonuç: True
```

- **Parse**

Parametrede verilen **String** bir ifadeden **Date** veri tipine çevrim işlemini yapar. **String** ifadesinde verilen ifadenin doğru bir tarih ve zaman tipinde olması gereklidir.

```
DateTime.Parse("23.04.2005 20:20:00")
DateTime.Parse("22 July 2005 02:00 PM")
DateTime.Parse("18 Haziran 1980")
```

İngilizceden farklı bir dilde girilen ay isimlerinin tarih tipine çevrilmesi için, uygulamanın kültürü o dilde ayarlanması gereklidir.

```
// Uygulama kültürü Fransızca yapılır.
Application.CurrentCulture = New Globalization.CultureInfo("fr-FR");
// temps değişkeninin değeri 23/05/2005 olacaktır.
DateTime temps = DateTime.Parse("23 Mai 2005");
// Bu kod hata verecektir.
DateTime zaman = DateTime.Parse("23 Mayıs 2005");
```

- **ToLongDateString & ToLongTimeString**

Verilen tarihi uzun tarih ve zaman formatında gösteren fonksiyonlardır.

```
DateTime d = #1/29/2005 12:59:22 PM#;
d.ToString("yyyy/MM/dd HH:mm:ss");
// Sonuç: 29 Ocak 2005 cumartesi

d.ToString("HH:mm:ss");
// Sonuç: 12:59:22
```

- **ToShortDateString & ToShortTimeString**

Verilen tarihi kısa tarih ve zaman formatında gösteren fonksiyonlardır.

```
DateTime d = #1/29/2005 12:59:22 PM#;
d.ToString("dd/MM/yyyy");
// Sonuç: 29 Ocak 2005

d.ToString("mm:ss");
// Sonuç: 12:59
```

## Konu 6: Offline ve Online Yardımın Etkin Kullanımı

Visual C#.NET dilinde uygulama geliştirirken .NET Framework içinde tanımlı bir çok nesnenin fonksiyon ve yordamları kullanılır. Ancak her yordam ve fonksiyonun aldığı parametreleri ve ne işe yaradıklarının ezbere bilinmesi mümkün değildir.

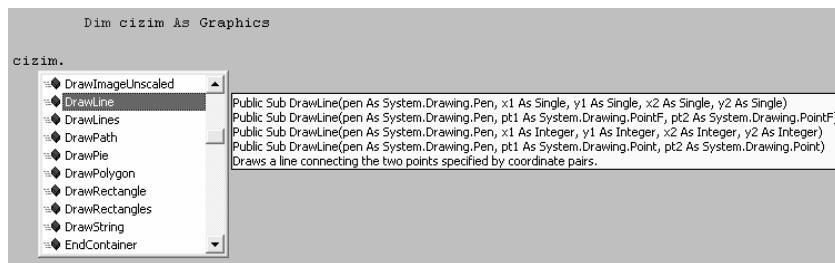
Modül 3 Help Kullanımı bölümünde MSDN offline yardımının kullanılmasından ve öneminden bahsedilmiştir. MSDN kütüphanelerinin Visual Studio içine kurulmaması durumunda online yardım araçları kullanılabilir. Visual Studio, başlangıç sayfasının **online Resources** sekmesinde birçok arama kolaylığı sunar.

### Offline Yardım

#### Offline Yardım

- IntelliSense
- Index, Search, Contens, Dynamic Help
- Uygulama: String.Format fonksiyonunun araştırılması

Uygulama geliştirirken, kodların yazılmasında **IntelliSense** aracından büyük ölçüde faydalananır. **IntelliSense**, bir kodun yazılması sırasında açıldığı zaman, yazılan kodlarla başlayan tüm metot, özellik ve nesneleri programcıya sunar. O anda üzerinde bulunan ögenin açıklaması, aldığı parametreler gibi bilgileri de gösterir.



Visual Studio içinde MSDN kütüphanelerinde istenen konuların aranması için **Index**, **Search**, **Contents** ve **Dynamic Help** panelleri kullanılır. Sonuç bulunduğu zaman yeni bir çalışma sayfasında gösterilir. Bu sayfada aranan kavram ile ilgili detaylı bilgiler ve örnekler mevcuttur.

Örnek: **String** veri tiplerinin yazdırılmasını değişik formatlarda yazdırılması **String.Format** fonksiyonu ile kullanılır.

1. Visual Studio ortamında bir proje açın ve kod sayfasında **String.Format** yazın. Fonksiyonu yazdıktan sonra parantezi açın ve **IntelliSense** aracının çıkardığı menüyü inceleyin.

Fonksiyon kaç parametre alabiliyor?

Aşağı ve yukarı oklarla menü içinde ilerleyerek fonksiyonun aşırı yüklenmiş durumlarını inceleyin.

Fonksiyonun kaç tane aşırı yüklemesi yazılmış?

2. Format yazısının üstüne geldikten sonra **F1** tuşuna basın ve dinamik yardımın açıldığı sayfaya bakın. Bu sayfa fonksiyonun tüm aşırı yüklemelerini gösterir.
3. Parametre olarak **String** ve **params Object** alan fonksiyona tıklayın. Çıkan sayfa fonksiyonun detaylarını listeler.

- İlk olarak fonksiyonun söz dizimi verilmiştir. Burada parametre isimleri ve tipleri üzerinde bağlantılar görünür. Bu bağlantılar ile ilgili yardım dosyası açılır.
- **Parameters** bölümünde bu fonksiyonun aldığı parametrelerin tipleri ve kullanım amaçlarını gösterilir.
- **Return value** fonksiyonun dönüş değerinin hangi tipte olduğu ve nasıloluştugu gösterilir.
- **Exceptions** bölümünde bu fonksiyon kullanılırken meydana gelebilecek hatalar listelenir.
- **Remarks** bölümü, fonksiyonun kullanım yerleri, parametrelerin nasıl kullanılacağı, parametreler kullanılırken dikkat edilmesi gereken yerler, bağlantılı konular gibi fonksiyon hakkında detaylı bilgi verir.
- **Example** bölümünde, fonksiyonun kullanımına örnekler verilir.
- **Requirements** bölümünde, fonksiyonun çalışabilmesi için gereken araçlar ve platformlar listelenir.

- **See Also** bölümü, fonksiyon ile ilişkili kavamlara bağlantılar sunar.

**Remark** bölümündeki tanımlamalardan ve **Examples** bölümündeki örneklerde fonksiyonun nasıl kullanıldığını inceleyin.

4. Kod sayfanıza geçin ve **String.Format** fonksiyona bir örnek yazın.

```
int ocak = 1000;  
int subat = 1100;
```

```
MessageBox.Show(String.Format("Ocak ayı maaşı {0:c} -- Şubat  
ayı maaşı: {1:c}", ocak, subat));
```

5. Formatlama işlemleri hakkında daha fazla bilgi almak için, fonksiyonun yardım sayfasına gelin ve **Remarks** bölümünde **Formatting Types** bağlantısına tıklayın. Ya da **Index** panelinden **Formatting Types** yazın ve yardım sayfasını açın.

Çıkan sayfada her veri tipi için kullanılan formatlama seçenekleri vardır.

**Numeric Format Strings** bağlantısına tıkladıktan sonra açılan sayfada **NumberFormatInfo** bağlantısına tıklayın.

**Format Character** tablosunda değişik formatlama seçeneklerini inceleyin ve kodunuzda deneyin.

6. **web** araç çubuğundan geri tuşuna basarak veya **Alt – Sol ok** kısa yolu ile **String.Format Method** başlıklı ilk açığınız sayfaya dönün.
7. Parametre olarak **IFormatProvider**, **String** ve **params Object** alan fonksiyon tanımına tıklayın. Fonksiyonun kullanımını inceledikten sonra, bu kullanıma bir örnek yazın.

```
MessageBox.Show (String.Format(New  
Globalization.CultureInfo("it-IT"), "Bugün: {0:ddd MM  
yyyy}", DateTime.Now));
```

8. Bu örnekte uygulamanın kültür ayarları değiştirilmeden, tarihin istenen kültür ayarı ile gösterilmiştir. Kültür ayarlarının tanımlanmasını incelemek için **Index** yardım panelinde **CultureInfo** yazın ve **about CultureInfo Class** indeksini seçin. Çıkan **Index Result** penceresinde **CultureInfo Class** indeksini seçin.
9. Bu sayfada çıkan kültür isimlerini örneğinizde kullanarak değişik sonuçları inceleyin.

**NOT:** Türkçe dil ailesi için **Globalization.CultureInfo("TR-tr")** kullanılır

## Online Yardım

### Online Yardım

- Online MSDN Kütüphaneleri
- Start Page Online Resources
- Uygulama: Undo yordamının araştırılması

MSDN kütüphanelerinde offline olarak yardım almak hızlı ve etkili bir yöntemdir. Ancak bu yardım dosyalarının güncellenmesi için MSDN sürümünün yenilenmesi gereklidir. Online yardım MSDN kütüphanelerinin internet ortamında yayınlanmasıdır. Yeni örnekler, makaleler ve düzeltmelerle güncellenen bu yardım dosyalarına <http://msdn.microsoft.com> adresinden ulaşılabilcegi gibi, Visual Studio ortamından da bu dosyalar içinde arama yapılabilir.

Örnek: Windows uygulamasında kullanılan bir metin kutusunda “Geri Al” (**Undo**) işlemi yapılmak isteniyor fakat fazladan kod yazılmak istenmiyor. Bunun için .NET Framework çatısında hazır bir metodun olup olmadığı kontrol edilmesi gereklidir. Online yardım ile gerekli arama yapıldıktan sonra çıkan sonuçlar yorumlanır.

1. Başlangıç sayfasını (Start Page) açın ve Online Resources sekmesine gelin.
2. Sol paneldeki menüden Search Online menüsüne gelin ve Search For altındaki metin kutusuna “TextBox Undo” yazın. Sonuçların MSDN Online içinde hangi duruma göre filtrelenebildiğini gösteren bağlantılar çıkar.  
Sonuçlar
  - Tüm MSDN içinde,
  - MSDN kod ve karşılık yûklemelerde,
  - MSDN teknik makalelerinde,
  - Microsoft bilgi veri kaynağında,

- Microsoft.com genelinde filtrelenebilir.
3. Search results for All of MSDN bağlantısına tıklayın ve çıkan sonuçları inceleyin. Aranan kaynak .NET Framework içinde kullanılabilmesi istediği için TextBoxBase.Undo Method (.NET Framework) yardım konusuna tıklayın.
  4. MSDN Online kütüphanelerinin sayfa düzeni, içeriği offline yardım ile aynıdır. **TextBoxBase** taban sınıfının **Undo** metodunu inceleyen bu yardım sayfasında, metot tanımlaması, **Remarks**, **Examples**, **Requirements** ve **See Also** bölümleri görülür. **Examples** bölümünde **visual C#** kodlarının altında **Undo** metodunun kullanımını inceleyin.
  5. **Undo** yapıldıktan sonra silinen kelimelerin bir listede tutulması ve listeye ekleme işleminin kolay bir şekilde yapılması isteniyor. Bunun için Sol panelde bulunan menülerin üstündeki Search For metin kutusuna “ArrayList” yazın ve çıkan sonuçlarda ilk bağlantıya tıklayın.
  6. **ArrayList** sınıfının **Count**, **Item** özelliklerini ve **Add** metodunu inceleyin. Ve uygulamanızı tamamlamak için bu özellikleri kodunuzda kullanın.

```
ArrayList silinenler = New ArrayList;
```

```
void GeriAl()
{
    // Metin kutusunda geri alınacak bir veri varsa
    if (TextBox1.CanUndo)
    {
        // Eski değerler listeye eklenir.
        silinenler.Add(TextBox1.Text);
        TextBox1.Undo();
        GeriAlinankelimeler();
    }
}

// Listeleme işlemini yapan yordam
void GeriAlinanKelimeler()
{
    ListBox1.Items.Clear();
    for (int i = 0; i < silinenler.Count; i++)
    {
        // i indisli Item, liste kutusuna eklenir.
        ListBox1.Items.Add(silinenler.Item(i));
    }
}
```

## Lab 1: Kelime Oyunu

Bu uygulamadaki oyun, girilen bir kelimenin son harfleriyle başlayan başka bir kelimenin girilmesidir. Oyunun seviyesi, girilecek kelimenin kontrol edilecek harf sayısıdır. Örneğin ikinci seviyede, ilk girilen kelime “Masa” ise, bir sonraki kelime “sa” ile başlamalıdır. Üçüncü seviyede bu kelime “asa” ile başlamalıdır. Kullanıcı, oyuna ilk seviyeden başlar ve beş kelime bildiği zaman bir sonraki seviyeye geçer. Toplam alınan puan, bilinen kelime sayısının seviye kadar kuvveti alınarak hesaplanır.

## Projenin Açılması

1. **KelimeOyunu** isminde bir Window projesi açın ve forma listedeki kontrolleri ekleyin.

- **btnBasla** ve **btnGiris** isminde iki **Button**
- **txtKelime** isminde bir **TextBox**
- **lblMesaj** isminde bir **Label**
- **tmrSure** isminde bir **Timer**

2. Projenizin kod sayfasına geçin ve uygulama boyunca kullanılacak global değişkenleri tanımlayın.

```
// Kontrol edilecek kelime
public string kelime;

// oyunun seviyesi
public byte oyunSeviyesi = 1;

// Timer kontrolünde kullanılacak süre
public int kalanSure = 5;

// Bilinen kelime sayısı
public int tekrar = 0;
```

3. Uygulamaya giriş **Sub Main** yordamından yapılır. Bu yordamda kullanıcidan, formun başlığında görüntülenecek bir kullanıcı adı istenir. Eğer kullanıcı adı boş girilirse form yüklenmeden uygulamadan çıkarılır.

```
// Uygulamanın giriş noktası
public static void Main() {
    string KullaniciAdi = null;
    KullaniciAdi =
Microsoft.VisualBasic.Interaction.InputBox( "Kullanıcı Adı
girin:", "", "", -1, -1 );

    if ( KullaniciAdi == "" ) {
        return;
    }

    Form1 oyun = new Form1();
    oyun.Text = KullaniciAdi + " yaryor";
    oyun.ShowDialog();
}
```

## Yardımcı Yordam ve Fonksiyonlar

Uygulamanın tamamında kullanılacak kodlar yordam ve fonksiyonlar halinde yazılarak hem yönetilmesi hem de kullanılabilirliği artırılır. Uygulamada kullanılacak yordam ve fonksiyonlar tabloda listelenmiştir.

| İsim               | Parametreler        | İşlev                                                           |
|--------------------|---------------------|-----------------------------------------------------------------|
| <b>Temizle</b>     |                     | Zamanı sıfırlar ve <b>TextBox</b> kontrolüne <b>Focus</b> verir |
| <b>OyunuBaslat</b> |                     | Başlangıç kelimesi alınarak <b>Timer</b> başlatılır.            |
| <b>OyunuBitir</b>  | <b>String</b> neden | Süreyi durdurur, puanı                                          |

|                           |                                              |                                                                            |
|---------------------------|----------------------------------------------|----------------------------------------------------------------------------|
|                           |                                              | ve bitiş nedeni kullanıcıya gösterir.                                      |
| <b>Bilgi</b>              | <b>String</b> mesaj                          | <b>Label1</b> kontrolünde mesaj görüntülenir.                              |
| <b>SonrakiKelimeBilgi</b> |                                              | Girilecek kelimenin hangi harflerle başlayacağını gösterir.                |
| <b>SeviyeAtla</b>         | <b>Byte</b> seviye                           | Oyunun seviyesini artırır.                                                 |
| <b>Kontrol</b>            | <b>String</b> kelime1, <b>String</b> kelime2 | İkinci kelimenin, ilk kelimenin harfleriyle başladığının kontrolü yapılır. |
| <b>PuanHesapTa</b>        | <b>Byte</b> seviye, <b>Short</b> tekrar      | Tekrar değerinin, seviye kadar üssü alınır.                                |

1. Yordamları ve fonksiyonları yazın

- Temizle yordamı

```
public void Temizle() {
    kalansure = 5;
    TextBox1.Text = "";
    TextBox1.Focus();
}
```

- OyunuBaslat yordamı

```
public void OyunuBaslat() {
    Temizle();
    kelime = TextBox1.Text;

    tmrSure.Start();

    SonrakikelimeBilgi();
}
```

- OyunuBitir yordamı

```
public void OyunuBitir( string neden ) {
    tmrSure.Stop();
    Bilgi( neden );

    Temizle();

    int puan;
    puan = PuanHesapla( oyunseviyesi,
    System.Convert.ToInt16( tekrar ) );
    MessageBox.Show( "Puanınız: " + puan );
}
```

- Bilgi yordamı

```
public void Bilgi( string kelime ) {
    Label1.Text = kelime;
}
```

- SonrakiKelimeBilgi yordamı

```

        public void SonrakiKelimeBilgi() {
            string mesaj = null;
            mesaj += Microsoft.VisualBasic.Strings.Right(
                kelime, OyunSeviyesi );
            mesaj += " ile başlayan bir kelime girin";

            Bilgi( mesaj );
        }
    • SeviyeAtla yordamı

    public void SeviyeAtla( byte seviye ) {
        OyunSeviyesi = seviye;

        OyunuBitir( seviye + ". seviyeye geçildi" );
        SonrakiKelimeBilgi();
    }
    • Kontrol fonksiyonu

    public bool Kontrol( string kelime1, string kelime2
    ) {
        // ikinci kelimenin başında oyun seviyesi kadar
        // karakter alınır.
        string bas = kelime2.Substring( 0, OyunSeviyesi
    );

        // ikinci kelime, ilk kelimenin sonu ile başlıyorsa
        // doğru girilmiştir. True değeri döner.
        return kelime1.EndsWith( bas );
    }

    • PuanHesapla yordamı

    public int PuanHesapla( byte seviye, short tekrar ) {
        return Math.Pow( tekrar, seviye );
    }
}

```

## Olayların yazılması

1. **tmrSure** kontrolünün **Tick** olayına kalan süreyi kontrol eden kodları yazın

```

private void tmrSure_Tick( System.Object sender,
System.EventArgs e ) {
    if ( kalansure <= 0 ) {
        OyunuBitir( "Süreniz doldu" );
    }
    else {
        kalansure -= 1;
    }
}

```

2. **btnBasla** düğmesinin **Click** olayına, oyunu başlatan yordamı yazın

```

private void btnBasla_Click( System.Object sender,
System.EventArgs e ) {
    OyunuBaslat();
}

```

3. **btnGiris** düğmesinin **Click** olayına, girilen kelimeyi alıp kontrolleri yapan kodu yazın. Burada dikkat edilmesi gereken nokta, tekrar sayısının seviye ile doğru orantılı olmasıdır.

```
private void btnGiris_Click( System.Object sender,
System.EventArgs e ) {
    string girilen = TextBox1.Text;

    if ( !( Kontrol( kelime, girilen ) ) ) {
        string neden = null;
        neden = "Girilen kelime, ilk kelimenin son
";
        neden += oyunSeviyesi + " harfi ile
başlamıyor";

        OyunuBitir( neden );
    }
    else if ( tekrar > 5 * oyunSeviyesi ) {
        SeviyeAtla( System.Convert.ToByte(
oyunSeviyesi + 1 ) );
    }
    else {
        tekrar += 1;
        kelime = girilen;
        SonrakikelimeBilgi();
        Temizle();
    }
}
```

## Modül Sonu Soruları & Alıştırmalar

### Özet

- ➔ Sub – Function kullanımı
- ➔ .NET Tarih, String, Matematik fonksiyonları
- ➔ Online Offline yardımın etkin kullanımı

1. Yordam ile fonksiyon arasındaki fark nedir?
2. **Main** yordamı formların ve modüllerin içinde nasıl tanımlanır. Kendi **Main** yordamınızı yazın.
3. Yordam ve fonksiyonlar uygulamalarda kod tekrarını nasıl önler?
4. Yordam ve fonksiyonların sınırsız parametre olmasını sağlayan **params** neden sonda tanımlanır?
5. Farklı kültürlerde tarih, zaman, metin değerlerini göstermek için gerekli olan sınıflar ve fonksiyonlar nelerdir?
6. Yordam ya da fonksiyon içerisinde yordam ya da fonksiyonlar çağrılabılır mı? Uygulamasını yazın.
7. Bir yordam ya da fonksiyon kendisini çağrıabilir mi? (Recursive) Uygulamasını yazın.

## Modül 8: Veri Tipleri Üzerine İleri Bakış

### Hedefler

- ↳ Değer Veri Tipleri
- ↳ Referans Veri Tipleri
- ↳ Organizasyon yapısı
- ↳ ByVal – ByRef

.NET içinde tanımlanabilen veri tipleri temel (primitive) veri tipleri ya da kullanıcının tanımladığı veri tipleridir. Temel veri tipleri .NET içinde tanımlanmış ve bazı önemli özellikleri olan tiplerdir. Örneğin 32 bitlik bir sayıyı temsil eden **Int32** değer tipi temel biriptir. Bu temel tipin üzerinde aritmetik işlemler yapılabilir. **struct** olarak tanımlanan kullanıcı veri tipleri üzerinde aritmetik işlemler yapılamaz.

Temel ve kullanıcı tanımlı veri tipleri, değer tipi ve referans tipi olarak ikiye ayrılır. **struct** bir **değer** tipi, **Class** ise bir **referans** tipidir. Değer tipleri belleğin stack bölgesinde, referans tipleri heap bölgesinde depolanır. Değer tiplerinin oluşturulması ve silinmesi, sadece değerleri üzerinde işlem yapıldığı için kolaydır. Değer tipinin ömrü bittiği zaman stack yapısından hemen kaldırılır. Referans tiplerinin oluşturulması, yok edilmesi ekstra bir performans gerektirir. Ancak iki veri tipinin de birbirlerine göre avantajları vardır.

Bu modül tamamlandıktan sonra:

- Temel ve kullanıcı tanımlı değer tiplerini tanıyacak,
- Temel ve kullanıcı tanımlı referans tiplerini tanıyacak,
- Veri tiplerinin belleği kullanımı öğrenecek,
- **ref** kavramını öğrenecek,
- Referans ve değer tiplerinin nerede kullanılacağını öğreneceksiniz.

## Konu 1: Değer Tipleri

### Değer Tipleri

- Built-In Değer Tipleri
  - .NET içinde tanımlı veri tipleridir.

```
// Visual C# tanımı değer tipi
short sayı = 10;
// .NET Framework tanımı değer tipi
Int16 sayı2 = 10;
```

- User-Defined Değer Tipleri

- Structure yapısı ile oluşturulan kullanıcı tanımlı veri tipleridir.

```
public struct Ucgen {
    public int kenar1;
    public int kenar2;
    public int kenar3;

    public Ucgen( int kenar_1, int kenar_2, int kenar_3 ) {
        this.kenar1 = kenar_1;
        this.kenar2 = kenar_2;
        this.kenar3 = kenar_3;
    }
}
```

Değer tipindeki değişkenlerin tuttukları değerler bellekte stack yapısında bulunur. Bir değer tipindeki değişkenin, başka bir değişkene atanması, değerin olduğu gibi kopyalanması ile gerçekleşir. Dolayısıyla ne zaman bir atama işlemi yapılsa, değer tipinin bir kopyası bellekte oluşturulur. Bu durum çok karmaşık değerler ve büyük veri blokları için performansı düşürür. Ancak değer tipleri, tanımlı olduğu yerden çıktılarında bellekten hemen silinir.

### Built-In Değer Tipleri

Built-In değer tipleri olarak bahsedilecek temel tipler, .NET içinde tanımlı olan veri tipleridir. Bu değer tipleri sayıları, ondalık sayıları, **bool** değerlerini, tarih zaman değerlerini, karakterleri temsil eden yapılardır. Bu tipler, tüm .NET dilleri tarafından kullanılabilir şekilde tanımlanır. Ancak Visual C# dilinde bu değer tiplerine belirli isimler ile ulaşılır.

| Visual C#      | .NET Framework        | Değer                               |
|----------------|-----------------------|-------------------------------------|
| <b>bool</b>    | <b>System.Boolean</b> | <b>True / False</b>                 |
| <b>Byte</b>    | <b>System.Byte</b>    | 8 bit uzunlığında sayı              |
| <b>Char</b>    | <b>System.Char</b>    | 16 bit uzunlığında Unicode karakter |
| <b>Decimal</b> | <b>System.Decimal</b> | 128 bit uzunlığında sayı            |

|               |                      |                                        |
|---------------|----------------------|----------------------------------------|
| <b>Double</b> | <b>System.Double</b> | 64 bit uzunluğunda<br>kayan tipte sayı |
| <b>int</b>    | <b>System.Int32</b>  | 32 bit uzunluğunda sayı                |
| <b>Long</b>   | <b>System.Int64</b>  | 64 bit uzunluğunda sayı                |
| <b>Short</b>  | <b>System.Int16</b>  | 16 bit uzunluğunda sayı                |
| <b>float</b>  | <b>System.Single</b> | 32 bit uzunluğunda<br>kayan tipte sayı |

```
// Visual C# tanımı değer tipi
short sayı = 10;

// .NET Framework tanımı değer tipi
Int16 sayı2 = 10;
```

## Kullanıcı Tanımlı Değer Tipleri

Uygulamalarda çoğu zaman Built-in değer tiplerinin sağlamadığı özel veri tiplerine ihtiyaç duyulur. Örneğin bir üçgen tipi, kenarları temsil eden üç tane sayı tutan bir değer tipi olarak oluşturulabilir.

Kullanıcı tanımlı değer tipleri Visual C# .NET dilinde **struct** ile oluşturulur.

```
public struct Ucgen {
    public int kenar1;
    public int kenar2;
    public int kenar3;

    public Ucgen( int kenar_1, int kenar_2, int
kenar_3 ) {
        this.kenar1 = kenar_1;
        this.kenar2 = kenar_2;
        this.kenar3 = kenar_3;
    }
}
```

- **struct** tiplerinde en az bir veri tipi tanımlı olması gereklidir.
- **struct** tiplerinde boş parametreli constructor tanımlanamaz. Değer tipleri tanımlandıklarında bu constructor ile oluşturulur. Ancak bir veya daha fazla parametre alan constructor metotları kullanılabilir.
- **struct** veri tipleri **Class** yapısına benzer, ancak değer tipi oldukları için oluşturulması yok edilmesi daha kolaydır.

## Konu 2: Referans Tipleri

### Referans Tipleri

- Built-In Referans Tipleri
  - Object, Built-In referans tipidir.
  - Array, dizilerin Built-In referans tipinde olmasını sağlar.
- User-Defined Referans Tipleri
  - Class yapısı ile oluşturulan kullanıcı tanımlı referans tipleridir.

```
// Kullanıcı tanımlı referans tipi
public class Class1 {
    public int Deger;
}
```

Referans tipindeki değerlere erişimler, bu değerlerin bellekte oluşturulduğu yerin adresi ile sağlanır. Bu değerler bellekteki heap bölgesinde oluşturulur. Referans tipindeki değişkenlerin, başka değişkenlere atama işlemleri bellekteki adreslerin kopyalanması ile gerçekleşir. Dolayısıyla aynı adresdeki veriyi gösterir. Bu iki değişkenden herhangi biri değiştiğinde, diğerinin de değişmiş olacaktır.

Sınıf ve dizi yapıları referans tipleridir. Dizilerin tuttukları değerlerin sayısı çoğu zaman önceden belli olmakla birlikte boyutları ve uzunlukları değişebilir. Dolayısıyla dizi değişkenlerinin ismi, elemanlarının bellekte tutuldukları ilk yerin adresini temsil eder. Ancak dizilerin tuttukları değerler referans tipinde olmayabilir.

### Built-in Referans Tipleri

.NET içinde tanımlı olan class, array yapıları **Object** sınıfında türemiştir. **Object** sınıfı .NET içinde tanımlı Built-in referans tipidir. Değişkenler tanımlandıkları sırada tipleri belirtilmemezse **Object** tipinde oluşturulur.

Kullanıcı tarafından oluşturulan diziler, bir **Array** sınıfından türetilir. Bu sınıf, diziler üzerinde işlemleri kolaylaşdıracak birçok özellik ve metod tanımlar. Örneğin **Length** özelliği dizinin toplam eleman sayısını verir, **Sort** metodu ise dizideki elemanların sıralanması işlemini yapar. Dolayısıyla **Array** sınıfı, diziler için Built-in referans tipi sağlar.

```
// Parametre olarak verilen dizinin başlangıç adresidir.  
// Dolayısıyla, bu adres bölgesinde tanımlı  
// değerlere ulaşılabilir.  
  
public void Goruntule( int[] dizi ) {  
    for ( i=0; i<=dizi.Length - 1; i++ ) {  
        Label1.Text += dizi[ i ];  
    }  
}
```

## Kullanıcı Tanımlı Referans Tipleri

.NET içinde tanımlı sınıflar kullanılabildiği gibi, birçok nesneye yönelik programlama dilinde kullanıcılar da kendi sınıflarını oluşturabilirler. .NET Framework içerisinde, kullanıcıların oluşturduğu bu sınıflar **Object** sınıfından türer. Dolayısıyla bu sınıflar kullanıcı tanımlı referans tipleridir.

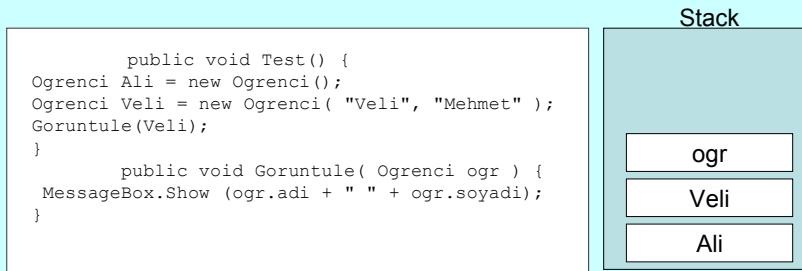
```
// Kullanıcı tanımlı referans tipi  
public class Class1 {  
    public int Deger;  
}  
  
public void Test1() {  
    Class1 sinif = new Class1();  
    sinif.Deger = 10;  
  
    Class1 sinif2 = null;  
  
    // Sinif değişkeninin tuttuğu adres bilgisi  
    // diğer değişkene aktarılır. Dolayısıyla Sinif2  
    // değişkeni de bellekte aynı yeri temsil eder.  
    sinif2 = sinif;  
  
    // Sinif değişkenin tuttuğu adres bölgesindeki  
    // değer değiştirilir.  
    sinif.Deger = 15;  
  
    // Sinif2 değişkeni de aynı adresi gösterdiği için  
    // sonuç 15 olur.  
    MessageBox.Show(sinif2.Deger.ToString());  
End Sub
```

## Konu 3: Organizasyon Yapısını İnceleme

### struct Organizasyon Yapısı Ve Belleğin İncelenmesi

#### Structure Organizasyon Yapısı

- Tanımlandıkları anda Stack bölümünde oluşturulur.
- Parametre geçilen tipler kopyalanır.



**struct** veri tipi, değer tipi olduğu için, tanımlandıkları anda bellekte **stack** bölümünde oluşturulur. Bellekte ayrılan yer **struct** içinde tanımlı olan Built-in veri tiplerinin toplam boyutu kadardır.

Visual C# .NET dilinde **struct** veri tipleri New anahtar kelimesiyle de oluşturulabilir. Ancak bu constructor metotları parametre alacak şekilde tanımlanmalıdır. Varsayılan parametresiz constructor metotları CLR tarafından işlenir. Dikkat edilmesi gereken bir durum da, New ile oluşturulan değişkenler yine bir değer tipidir ve **stack** alanında tutulur.

**NOT:** Classlardan nesne oluştururken New anahtar kelimesi kullanılır ancak bu nesneler heap alanında tutulur.

```

public struct Ogrenci {
    public string adı;
    public string soyadı;

    public Ogrenci( string isim, string soyisim ) {
        this.adı = isim;
        this.soyadı = soyisim;
    }
}

```

```

    }

    public void Test2() {
        // 1 - Öğrenci değeri tanımlandığı sırada
        // stack alanında yer ayrılır
        Ogrenci Ali = new Ogrenci();

        // 2 - New ile tanımlanan değişkenler de stack
        // alanında oluşturulur.
        // Farkı, bu değişkenin başlangıç değeri almasıdır.
        Ogrenci Veli = new Ogrenci( "Veli", "Mehmet" );

        // 3 - Parametre olarak sipariş nesnesinin
        // adresi verilir
        Goruntule(Veli);
    End Sub

    public void Goruntule( Ogrenci ogr ) {
        MessageBox.Show (ogr.adi + " " + ogr.soyadi);
    }
}

```

### 1. Ali değişkeni tanımlanırken Stack yapısı

|                                   |
|-----------------------------------|
|                                   |
|                                   |
| Değişken Ali.soyadi<br>Değer = "" |
| Değişken Ali.adi<br>Değer = ""    |
| <b>Ali Ogrenci</b>                |

### 2. Veli değişkeni tanımlanırken Stack yapısı

|                                          |
|------------------------------------------|
|                                          |
|                                          |
| Değişken Veli.soyadi<br>Değer = "Mehmet" |
| Değişken Veli.adi<br>Değer = "Veli"      |
| <b>Veli Ogrenci</b>                      |
|                                          |
| Değişken Ali.soyadi<br>Değer = ""        |
| Değişken Ali.adi<br>Değer = ""           |
| <b>Ali Ogrenci</b>                       |

Bu değişkenler oluşturulduktan sonra, bir yordama parametre olarak verildiklerinde, tüm değerleri kopyalanır. Değer tipindeki değişkenler atama işlemlerinde, oldukları gibi kopyalanır.

### 3. Görüntüle yordamı çağrıldığı zaman Stack yapısı

|                                          |
|------------------------------------------|
| Degisken ogr.soyadi<br>Değer = "Mehmet"  |
| Değişken ogr.adi<br>Değer = "Veli"       |
| <b>ogr Ogrenci</b>                       |
| Değişken Veli.soyadi<br>Değer = "Mehmet" |
| Değişken Veli.adi<br>Değer = "Veli"      |
| <b>Veli Ogrenci</b>                      |
| Değişken Ali.soyadi<br>Değer = ""        |
| Değişken Ali.adi<br>Değer = ""           |
| <b>Ali Ogrenci</b>                       |

Test isimli yordamdan çıkışınca, bu değişkenler oluşturuldukları sırayla stack yapısından kaldırılır.

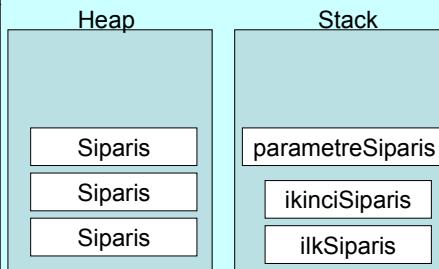
## Class Organizasyon Yapısı Ve Belleğin İncelenmesi

### Class Organizasyon Yapısı

- Tanımlandıkları anda Stack bölümünde oluşturulur.
- Parametre geçilen tipler kopyalanır.

```
public void Test() {
    Siparis ilkSiparis = null;
    ilkSiparis = new Siparis(
        DateTime.Now, "Enis Günesen", "Visual
        Studio.NET 2003" );
    ilkSiparis = new Siparis(
        DateTime.Now.AddYears( -1 ), "Enis
        Günesen", "Visual Studio.NET 2002" );

    Siparis ikinciSiparis = new Siparis();
    ilkSiparis = ikinciSiparis;
    ilkSiparis.Urun = "BilgeAdam Yazılım
    Uzmanlığı";
    Goruntule( ikinciSiparis );
}
public void Goruntule( Siparis
parametreSiparis ) {
    MessageBox.Show (parametreSiparis.Urun);
}
```



Classlardan (Sınıf) nesneler oluşturuldukları zaman bu nesnelerin değerleri **heap** bölgesinde tutulur. Ancak bu nesneleri gösteren bir adres tutucusu oluşturulur ve bu adresin değeri de **stack** alanında depolanır.

```
public class Siparis {
    public DateTime Tarih;
    public string AliciIsmi;
    public string Urun;

    public Siparis() {
    }

    public Siparis( DateTime Tarih, string Isim,
string Urun ) {
        this.Tarih = Tarih;
        this.AliciIsmi = Isim;
        this.Urun = Urun;
    }
}

public void Test() {
    // 1 - Sipariş referansı oluşturulur
    Siparis ilkSiparis = null;
    // 2 - Yeni bir nesne oluşturulup,adresi
```

```

        // bu referansa aktarılır
        ilksiparis = new Siparis( DateTime.Now, "Enis
Günesen", "visual Studio.NET 2003" );

        // 3 - Yeni bir nesne daha oluşturulur ve adresi
        // değişkene aktarılır
        ilkSiparis = new Siparis(
DateTime.Now.AddYears( -1 ), "Enis Günesen", "Visual
Studio.NET 2002" );

        // 4 - Yeni bir referans ve nesne oluşturulur
        Siparis ikincisiparis = new Siparis();

        // 5 - İki değişkenin aynı bellek alanını göstermesi
        // sağlanır
        ilkSiparis = ikincisiparis;

        // 6 - Bu alandaki Ürün ismi değiştirilir
        ilkSiparis.Urun = "BilgeAdam Yazılım Uzmanlığı";

        // 7 - Parametre olarak sipariş nesnesinin
        // adresi verilir
        Goruntule( ikincisiparis );
End Sub

public void Goruntule( Siparis parametresiparis ) {
    MessageBox.Show (parametresiparis.Urun);
}

```

### **1. ilkSiparis tanımlanırken Stack Alanı**

|                                                         |
|---------------------------------------------------------|
| Değer = 0x00000000<br>(Bellekte boş bir alanı gösterir) |
| ilkSiparis                                              |

### **Heap Alanı**

| Nesne | Adres Bilgisi |
|-------|---------------|
|       |               |

### **2. ilkSiparis oluşturulurken Stack Alanı**

|                    |
|--------------------|
| Değer = 0x00000012 |
| ilkSiparis         |

### **Heap Alanı**

| Nesne                                                                                             | Adres Bilgisi |
|---------------------------------------------------------------------------------------------------|---------------|
| <b>Siparis</b><br>Tarih = 10.05.2005<br>Alicilsmi = Enis Günesen<br>Urun = Visual Studio.NET 2003 | 0x000000012   |

### 3. ilkSiparis referansına başka bir nesne verilirken Stack Alanı

|                                                         |
|---------------------------------------------------------|
| Değer = 0x00000056<br>(Gösterdiği adres değeri değişir) |
| <b>ilkSiparis</b>                                       |

### Heap Alanı

| Nesne                                                                                             | Adres Bilgisi                                                      |
|---------------------------------------------------------------------------------------------------|--------------------------------------------------------------------|
| <b>Siparis</b><br>Tarih = 10.05.2004<br>Alicilsmi = Enis Günesen<br>Urun = Visual Studio.NET 2002 | 0x000000056                                                        |
| <b>Siparis</b><br>Tarih = 10.05.2005<br>Alicilsmi = Enis Günesen<br>Urun = Visual Studio.NET 2003 | 0x000000012<br>(Bu adres alanına artık hiçbir referans ulaşamıyor) |

### 4. ikinciSiparis oluşturulurken Stack alanı

|                      |
|----------------------|
| Değer = 0x00000088   |
| <b>ikinciSiparis</b> |
| Değer = 0x00000056   |
| <b>ilkSiparis</b>    |

### Heap Alanı

| Nesne                     | Adres Bilgisi |
|---------------------------|---------------|
| <b>Siparis</b><br>Tarih = | 0x000000088   |

|                               |            |
|-------------------------------|------------|
| Alicilsmi =                   |            |
| Urun =                        |            |
| <b>Siparis</b>                | 0x00000056 |
| Tarih = 10.05.2004            |            |
| Alicilsmi = Enis Günesen      |            |
| Urun = Visual Studio.NET 2002 |            |
| <b>Siparis</b>                | 0x00000012 |
| Tarih = 10.05.2005            |            |
| Alicilsmi = Enis Günesen      |            |
| Urun = Visual Studio.NET 2003 |            |

#### 5. ikinciSiparis in adres bilgisi ilkSiparis e atanırken Stack Alanı

|                                                                                 |
|---------------------------------------------------------------------------------|
|                                                                                 |
| Değer = 0x00000088                                                              |
| <b>ikinciSiparis</b>                                                            |
| Değer = 0x00000088<br>(Gösterdiği adres ikinci sipariş referansı ile aynı olur) |
| <b>ilkSiparis</b>                                                               |

#### Heap Alanı

| Nesne                         | Adres Bilgisi                                            |
|-------------------------------|----------------------------------------------------------|
| <b>Siparis</b>                | 0x00000088                                               |
| Tarih =                       |                                                          |
| Alicilsmi =                   |                                                          |
| Urun =                        |                                                          |
| <b>Siparis</b>                | 0x00000056<br>(Bu nesneyi gösteren referans kalmamıştır) |
| Tarih = 10.05.2004            |                                                          |
| Alicilsmi = Enis Günesen      |                                                          |
| Urun = Visual Studio.NET 2002 |                                                          |
| <b>Siparis</b>                | 0x00000012                                               |
| Tarih = 10.05.2005            |                                                          |
| Alicilsmi = Enis Günesen      |                                                          |
| Urun = Visual Studio.NET 2003 |                                                          |

#### 6. ilkSiparis in gösterdiği nesnenin Ürün ismi değiştirilirken Stack Alanı

|                      |
|----------------------|
|                      |
| Değer = 0x00000088   |
| <b>ikinciSiparis</b> |
|                      |
| Değer = 0x00000088   |
| <b>ilkSiparis</b>    |

**Heap Alanı**

| Nesne                                                                                             | Adres Bilgisi                                                                             |
|---------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| <b>Siparis</b><br>Tarih =<br>Alicilsmi =<br>Urun = BilgeAdam Yazılım Uzmanlığı                    | 0x000000088                                                                               |
| <b>Siparis</b><br>Tarih = 10.05.2004<br>Alicilsmi = Enis Günesen<br>Urun = Visual Studio.NET 2002 | 0x000000056<br>(Referanslarını kaybetmiş nesneleri,<br>Garbage Collector bellekten siler) |
| <b>Siparis</b><br>Tarih = 10.05.2005<br>Alicilsmi = Enis Günesen<br>Urun = Visual Studio.NET 2003 | 0x000000012<br>(Referanslarını kaybetmiş nesneleri,<br>Garbage Collector bellekten siler) |

**7. ikinciSiparis in gösterdiği değer Goruntule yordamı ile gösterilirken  
Stack alanı**

|                         |
|-------------------------|
|                         |
| Değer = 0x00000088      |
| <b>parametreSiparis</b> |
|                         |
| Değer = 0x00000088      |
| <b>ikinciSiparis</b>    |
|                         |
| Değer = 0x00000088      |
| <b>ilkSiparis</b>       |

**Heap Alanı**

| Nesne                                                                          | Adres Bilgisi |
|--------------------------------------------------------------------------------|---------------|
| <b>Siparis</b><br>Tarih =<br>Alicilsmi =<br>Urun = BilgeAdam Yazılım Uzmanlığı | 0x00000088    |

Sonuç olarak gösterilen değer, heap alanında 0x00000088 adres numaralı nesnenin ürün ismi olur. Parametre olarak verilen nesneler heap alanında tekrar oluşturulmazlar. Referans olarak geçen değişkenler aslında değer tipleridir. Ancak nesnenin tümü değil sadece adres değerinin kopyası oluşturulur.

## ByVal ve ByRef İncelemesi

### ByVal – ByRef

- ByVal
  - Parametreye, değişkenin değeri geçer.

```
public void ElemanDegistir( int[] dizi, int index, int
yeniDeger ) {
    dizi[ index ] = yeniDeger;
}
```

- ByRef
  - Parametreye, değişkenin adresi (referansı) geçer.

```
public void Ekle( ref string Kelime, string eklenenek ) {
    Kelime = Kelime.Insert( 0, eklenenek );
}
```

Fonksiyon ve yordamlara parametre verilirken varsayılan olarak, değişkenlerin değerleri verilir. Parametre olarak verilen değişkenler üzerinde değişiklik yapılması için bu parametrelerin bulunduğu adres bilgilerine ihtiyaç vardır. Referans tipindeki değerler parametre olarak geçildiklerinde, referansları verilir. Ancak değer tipleri parametre olarak verildiklerinde bu değerler kopyalanır ve asıl değişkenin tuttuğu değere ulaşılamaz. Bu karışıklıkları çözmek için, yordamlarda parametreler **ref** olarak belirtilir.

Normal parametre olarak verilecek değişkenin değeri ile işlem yapılacağını belirtir. Dolayısıyla bu parametrenin değeri değiştirilemez.

```
// Değişecek olan kelime normal verilmiştir
public void Ekle( string Kelime, string eklenenek )
{
    Kelime = Kelime.Insert( 0, eklenenek );
}

private void Button1_Click1( System.Object sender,
System.EventArgs e ) {
    string mesaj = "Hello";
    Ekle( mesaj, " World" );
    MessageBox.Show (mesaj);
}
```

**mesaj** değişkenin değeri, yordama değer olarak verilmiştir. Dolayısıyla yordamın üzerinde çalıştığı değer, mesajın bir kopyasıdır. Bellek alanında fiziksel olarak farklı yerlerde dururular. Yani değişiklik yapılan değer, sadece bir kopyadır. Yordam sonlandığında kopya olarak oluşturulan değer silinecek ve asıl değer değişmemiş olarak kalacaktır. Bu durumda parametre olarak mesaj değişkeninin adresi verilmelidir. Dolayısıyla yordamdaki parametrenin **ref** olarak tanımlanması gereklidir.

```
public void Ekle2( ref string Kelime, string eklenenek ) {
    Kelime = Kelime.Insert( 0, eklenenek );
}
```

Parametre olarak referans tipinde bir değişken verilirse bir fark olmaz. Referans tipleri, nesnelerin bulunduğu heap alanlarının adresini tutar. Dolayısıyla normal tanımlanan parametreye referans tipinin değeri (adres bilgisi) kopyalanır. Bu kopya üzerinden aynı adres alanında değişiklik yapılır.

```
// Parametre ByVal ile tanımlı olsa dahi, değiştirir
// dizinin belirtilen indexteki değeri,
// adres olarak erişilir.

public void ElemanDegistir( int[] dizi, int index,
int yeniDeger ) {
    dizi[ index ] = yeniDeger;
}

private void Button1_Click( System.Object sender,
System.EventArgs e ) {
    int[] sayilar = new int[ 3 ];
    sayilar[ 0 ] = 111;
    sayilar[ 1 ] = 222;
    sayilar[ 2 ] = 333;

    ElemanDegistir( sayilar, 1, 1000000 );
    MessageBox.Show( sayilar[ 1 ] );
// Sonuç = 1000000
```

{}

## Modül Sonu Soruları & Alıştırmalar

### Özet

- ➔ Değer Veri Tipleri
- ➔ Referans Veri Tipleri
- ➔ Organizasyon yapısı
- ➔ ByVal – ByRef

1. Değer değişkenleri ve referans değişkenleri arasındaki farkı açıklayınız. Her iki değişken tipinin de yer aldığı parametreleri içeren bir yordam yazın. Değişkenlerin verilerinin değişip değişmediğinin gözlemleyin.
2. struct yapısını açıklayınız. Kompleks bir veri tipi structure yapısında kodlayın. Yazılan veri tipinin uzunluğunu hesaplayın.

## Modül 9: Windows Programlama

### Hedefler

- ➔ Listeleme Kontrolleri
  - ➔ ListBox, TreeView, ComboBox
- ➔ Resim Kontrolleri
  - ➔ PictureBox, ImageList
- ➔ Düzenleme Kontrolleri
  - ➔ TabControl, Panel, HScrollBar, VScrollBar
- ➔ Zaman ve Tarih Kontrolleri
  - ➔ DateTimePicker, MonthCalendar
- ➔ Dinamik Kontroller
  - ➔ Çalışma anında eklenen kontroller

Visual C#.NET ile Windows Tabanlı Programlama modülünde, Windows Formlarına ve kontrollerine giriş yapılmıştı. .NET çatısında, Windows uygulamalarının görünüm ve kullanım zenginliğini artırmak için birçok kontrol vardır. Visual Studio ile varsayılan olarak gelen kontrollerin dışında birçok kontrol de Windows uygulamalarına eklenebilir.

Bu modül tamamlandıktan sonra:

- **ListBox, TreeView, ComboBox** gibi listeleme kontrollerini tanıyacak,
- **PictureBox, ImageList** gibi resim kontrollerini tanıyacak,
- **TabControl, Panel, HScrollBar, VScrollBar** gibi düzenleme kontrollerini tanıyacak,
- **DateTimePicker, MonthCalendar** gibi zaman ve tarih kontrollerini tanıyacak
- Çalışma anında forma yeni kontroller oluşturup ekleyebileceksiniz.

# Konu 1: Formlar ve Windows Forms Kontrolleri

## Form Nesnesi

### Formlar

- Kullanıcı ile iletişimini sağlar
- Show ve ShowDialog ile birden fazla form açılır.
- Başlangıç formu projenin özelliklerinden ayarlanır.

Windows uygulamaları, kullanıcı ile iletişimini Form nesneleri ile sağlar. Formlar, görünüm özellikleri, pencere stili değiştirilerek ve üzerine kontroller eklenerek özelleştirilir. Ayrıca birden çok form nesnesi kullanılarak, uygulamalar zenginleştirilir.

### Birden Fazla Form Oluşturmak

Windows uygulamaları birden fazla form nesnesinden olduğu için, projelere form eklemek her zaman gereklidir. Bir Windows projesine yeni bir form eklemek için:

1. Solution Explorer panelinden projeye sağ tıklayarak ya da **Project** menüsünden **Add Windows Form** komutunu seçilir.
2. Çıkan menüden Windows Form öğesinin seçili olduğuna kontrol edilir ve bir isim verilerek form eklenir.

Başlangıç formlarının ayarlanmasıının yanı sıra, uygulamada bir formdan başka bir formun açılması ve ayarlanması sık karşılaşılan bir durumdur. Form nesneleri, **System.Windows.Forms** namespace içinde bulunan Form

sınıfindan türemiş sınıflardır. Dolayısıyla yeni bir Form oluşturmak için, istenen Form sınıfından bir nesne oluşturulması yeterlidir.

```
frmYeni yeniForm = New frmYeni;
```

Yeni oluşturulan formların gösterilmesi, formun **Show** ve **ShowDialog** metodları ile yapılır. **ShowDialog** metodu, form gösterildikten sonra, kapanana kadar diğer formlara erişimi engeller. **ShowDialog** metodundan sonra yazılan kodlar, form kapandıktan sonra çalıştırılır.

```
frmYeni yeniForm = New frmYeni;
yeniForm.ShowDialog();
```

```
// Bu kodlar yeniForm kapandıktan sonra çalıştırılır
MessageBox.Show("Form kapandı...");
```

**ShowDialog** ile gösterilen formlar, hangi durum ile kapandıklarını belirten bir **DialogResult** sonucu döndürürler. Bu kullanım **MessageBox.Show** hazır fonksiyonu ile aynıdır.

```
frmSatis frm = New frmSatis;
if (frm.ShowDialog == DialogResult.Yes)
{
    // Verileri kaydet
}
```

Formun hangi diyalog sonucu ile döneceğini, üzerindeki **Button** kontrollerinin **DialogResult** özelliği ile belirlenir. Eğer düğmenin bu özelliği **Yes** olarak ayarlanmışsa, Form bu düğmeye basılıp kapandığı zaman, **DialogResult.Yes** değerini döndürür.

Örneğin bir Windows uygulamasının, kullanıcının girdiği verilere göre değişik formların açılması için **Main** yordamından faydalıdır.. Bu yordamda, kullanıcının istediği form dinamik olarak yüklenir.

```
public void Main1() {
    string grup = null, parola = null;

    grup = Interaction.InputBox("Kullanıcı grubu:",
    "", "", -1, -1);
    parola = Interaction.InputBox(grup + " grubuna
giriş için parola girin:", "", "", -1, -1);

    // Grupların parolası kontrol edilir
    // ve ilgili grubun formu açılır.
    // Eğer parola veya grup ismi yanlış girilirse
    // hata formu yüklenir.

    switch (grup.ToUpper()) {
        case "SATIŞ":
            if (parola.ToUpper() != "SATIS_PAROLA")
) {
                HataFormuYukle("Satış departmanı
parolası yanlış!");
}
else {
```

```

        frmSatis satisDepartmani = new
        frmSatis();
        satisDepartmani.ShowDialog();
    }

    break;
case "YÖNETİM":
    if ( parola.ToUpper() !=
"YONETIM_PAROLA" ) {
        HataFormuYukle( "Yönetim departmanı
parolası yanlış!" );
    }
    else {
        frmYonetim yonetimDepartmani = new
        frmYonetim();
        yonetimDepartmani.ShowDialog();
    }

    break;
default:
    HataFormuYukle( grup + " isminde bir
grup bulunamadı" );
}

break;
}

// Hata formu, verilen parametredeki mesajı
// gösterecek şekilde ayarlanır ve yüklenir.
public void HataFormuYukle( string mesaj ) {
    frmHata hataFormu = new frmHata();
    hataFormu.lblHataMesaji.Text = mesaj;
    hataFormu.ShowDialog();
}

```

Aynı Windows projesi içinde açılan formlar açılmadan önce kontrollerinin özellikleri değiştirilebilir. Örneğin hata formu gösterilmeden önce, üzerindeki **Label** kontrolünün **Text** özelliği ilgili hata mesajını gösterecek şekilde ayarlanabilir.

## Form Özellikleri:

| Özellik             | Değer Tipi    | Açıklama                                                                              |
|---------------------|---------------|---------------------------------------------------------------------------------------|
| <b>AcceptButton</b> | <b>Button</b> | Form üzerinde <b>Enter</b> tuşuna basıldığı zaman “tıklanacak” <b>Button</b> kontrolü |
| <b>CancelButton</b> | <b>Button</b> | Form üzerinde <b>Esc</b> tuşuna basıldığı zaman “tıklanacak” <b>Button</b> kontrolü   |
| <b>Opacity</b>      | <b>Double</b> | Formun şeffaflık oranı (0 -1 arası)                                                   |

|                        |                          |                                                                                    |
|------------------------|--------------------------|------------------------------------------------------------------------------------|
| <b>MaximizeBox</b>     | <b>Boolean</b>           | Ekranı Kapla düğmesinin görünürügü                                                 |
| <b>MinimizeBox</b>     | <b>Boolean</b>           | Simge Durumunda Küçült düğmesinin görünürüüğü                                      |
| <b>ControlBox</b>      | <b>Boolean</b>           | <b>Close</b> , <b>Maximize</b> ve <b>Minimize</b> düğmelerinin tümünün görünürüüğü |
| <b>StartPosition</b>   | <b>FormStartPosition</b> | Form açıldığı zaman, ekran üzerindeki konumu                                       |
| <b>TopMost</b>         | <b>Boolean</b>           | Formun tüm pencerelerin üzerinde gözükmesi                                         |
| <b>FormBorderStyle</b> | <b>FormBorderStyle</b>   | Formun kenar stili                                                                 |
| <b>MaximumSize</b>     | <b>Size</b>              | Formun alabileceği maksimum büyülüklük                                             |
| <b>MinimumSize</b>     | <b>Size</b>              | Formun alabileceği minimum büyülüklük                                              |

### Form Olayları:

| Olay           | Açıklama                                              |
|----------------|-------------------------------------------------------|
| <b>Click</b>   | Form üzerine tıklandığı zaman gerçekleşir             |
| <b>Closing</b> | Form kapanmadan hemen önce gerçekleşir                |
| <b>Closed</b>  | Form kapandıktan sonra gerçekleşir                    |
| <b>Load</b>    | Form yüklenirken gerçekleşir                          |
| <b>KeyDown</b> | Form üzerindeyken bir tuşun basılması ile gerçekleşir |
| <b>KeyUp</b>   | Basilan tuşun kaldırılması ile gerçekleşir            |

### Form Metotları:

| Metot       | Açıklama             |
|-------------|----------------------|
| <b>Hide</b> | Formu <b>Visible</b> |

|                   |                                                                                           |
|-------------------|-------------------------------------------------------------------------------------------|
|                   | özellikini<br>yaparak, gizler<br><b>False</b>                                             |
| <b>Close</b>      | Formu kapatır. Eğer form başlangıç formuysa uygulama sonlanır                             |
| <b>Show</b>       | Formu gösterir. <b>Hide</b> ile gizlenmişse, <b>Visible</b> özelliği <b>True</b> yapılır. |
| <b>ShowDialog</b> | Formu diyalog kutusu olarak gösterir.                                                     |

Örnek: Bir Windows formunun kapanmasını yönetmek için, o formun **Closing** olayına ve **Close** metoduna ihtiyaç vardır. Kapanmasını yavaşlatmak için bir **Timer** kontrolü kullanılır ve formun şeffaflığı yavaşça azaltılır.

```
private void Form1_Load( System.Object sender,
System.EventArgs e ) {
    this.Text = "Hogeldiniz... " + DateTime.Now;
}

private void Form1_Closing( object sender,
System.ComponentModel.CancelEventArgs e ) {
    // Kapanma olayı gerçekleşmeden önce iptal edilir
    e.Cancel = true;
    Timer1.Start();
}

private void Form1_KeyDown( object sender,
System.Windows.Forms.KeyEventArgs e ) {
    // Shift-Ctrl-F3 tuşları basıldığında uygulama kapanır
    if ( e.Shift & e.Control & e.KeyCode == Keys.F3 )
        this.Close();
}

private void Timer1_Tick( System.Object sender,
System.EventArgs e ) {
    // Formun görünmez hale gelince uygulama kapanır
    if ( this.Opacity == 0 ) {
        Application.Exit();
    }
    else {
        this.Opacity -= 0.1;
    }
}
```

## Label

### Label

- Kullanıcıya bilgi veren etikettir.



**Label** kontrolü Form üzerinde kullanıcıya bilgi vermek amaçlı kullanılan etikettir.

### Label Özellikleri

| Özellik            | Değer Tipi              | Açıklama                                                                                                                                                      |
|--------------------|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>TextAlign</b>   | <b>ContentAlignment</b> | Yazının, etiket üzerindeki pozisyonu belirler.                                                                                                                |
| <b>BorderStyle</b> | <b>BorderStyle</b>      | Kontrolün kenar stilidir.<br><b>FixedSingle</b> değeri, kontrolün kenar çizgilerini gösterir.<br><b>Fixed3D</b> değeri, kenarların üç boyutlu olmasını sağlar |
| <b>Image</b>       | <b>Drawing.Image</b>    | Etiket üzerinde görüntülenmek istenen resmi tutar                                                                                                             |
| <b>ImageAlign</b>  | <b>ContentAlignment</b> | Etiket üzerindeki resmin nerede duracağını                                                                                                                    |

|             |             | belirler                                                                                            |
|-------------|-------------|-----------------------------------------------------------------------------------------------------|
| RightToLeft | RightToLeft | Etiket üzerindeki yazının yönünü belirler. Eğer Yes değerini alırsa, yazılar sağdan sola gösterilir |

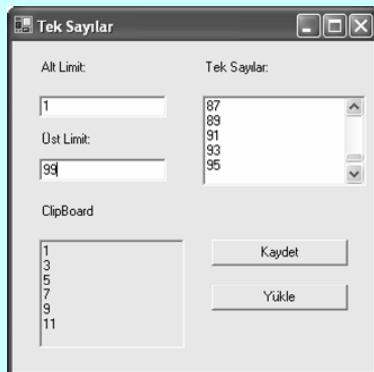
```
Label1.BorderStyle = BorderStyle.FixedSingle;  
  
// visual Studio klasörü altındaki simgeler kullanılabilir  
Label1.Image = Image.FromFile("C:\Program Files\_  
Microsoft Visual Studio .NET  
2003\Common7\Graphics\icons\Flags\FLGTURK.ICO");  
  
Label1.ImageAlign = ContentAlignment.MiddleRight;  
Label1.RightToLeft = RightToLeft.Yes;  
Label1.Text = "Türkçe";
```



**NOT:** Resmin bulunduğu yer kontrolün sağ tarafında bulunacak şekilde ayarlanmasına rağmen sol tarafta gözükmür. Bu durum, RightToLeft özelliğinin Yes olarak atanmasından kaynaklanır.

## TextBox

- ### TextBox
- Kullanıcıdan bilgi almak için kullanılır.



Metin kutuları, kullanıcıdan bilgi almak için kullanılır.

## TextBox Özellikleri

| Özellik             | Değer Tipi        | Açıklama                                                                                                                                                                                          |
|---------------------|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MultiLine</b>    | <b>Boolean</b>    | Metin kutusuna birden fazla satırda değer girilebilmesini sağlar. <b>False</b> durumunda ise, metin kutusunun yüksekliği değiştirilemez                                                           |
| <b>ScrollBars</b>   | <b>ScrollBars</b> | Metin kutusunda kaydırma çubuklarının görünmesi. Varsayılan olarak kaydırma çubuğu görüntülenmez, ancak <b>Horizontal</b> , <b>Vertical</b> kaydırma çubukları ya da ikisi birden gösterilebilir. |
| <b>PasswordChar</b> | <b>Char</b>       | Metin kutusuna parola girilecekse, girilen                                                                                                                                                        |

|                        |                        |                                                                                                                                                                                                                     |
|------------------------|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                        |                        | karakterlerin hangi karakter olarak görüneceğini belirler.                                                                                                                                                          |
| <b>WordWrap</b>        | <b>Boolean</b>         | Metin kutusuna girilen değerlerin, satır sonlandığında bir alt satira geçilmesini belirtir. Eğer <b>MultiLine</b> özelliği <b>False</b> ise, alt satırlar tanımlı olmayacağı için bu özelliğin bir etkisi görülmez. |
| <b>MaxLength</b>       | <b>Integer</b>         | Metin kutusunun alabileceği maksimum karakter sayısını belirtir.                                                                                                                                                    |
| <b>ReadOnly</b>        | <b>Boolean</b>         | Metin kutusunun yazmaya karşı korumalı olduğunu belirtir.                                                                                                                                                           |
| <b>CharacterCasing</b> | <b>CharacterCasing</b> | Metin kutusuna karakterler girilirken büyük veya küçük harfe çevrilmesini sağlar. <b>Upper</b> değeri büyük, <b>Lower</b> değeri küçük harfe çevrimi sağlar.                                                        |

## TextBox Olayları

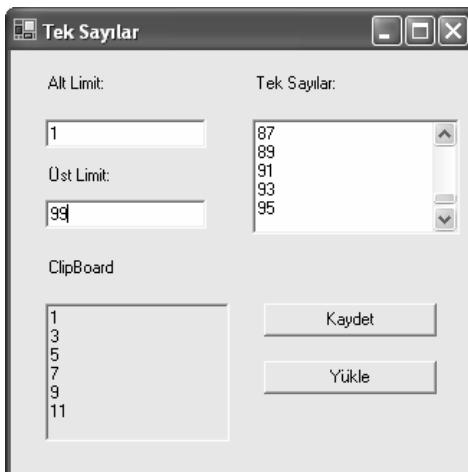
| Olay               | Açıklama                                            |
|--------------------|-----------------------------------------------------|
| <b>TextChanged</b> | Metin kutusundaki yazı değiştiği zaman gerçekleşir. |

## TextBox Metotları

| Metot        | Açıklama                                         |
|--------------|--------------------------------------------------|
| <b>Cut</b>   | Seçilen karakterleri siler ancak hafızada tutar. |
| <b>Copy</b>  | Seçilen karakterleri kopyalar                    |
| <b>Paste</b> | Hafızaya alınan karakterleri metin               |

|                  |                                    |
|------------------|------------------------------------|
|                  | kutusuna yapıştırır                |
| <b>Clear</b>     | Metin kutundaki yazıları temizler  |
| <b>SelectAll</b> | Metin kutusundaki tüm yazıyı seçer |

Örnek: Form üzerinde girilen değerlere göre tek sayıların hesaplanması ve görüntülenmesi işlemi için **TextBox** kontrolünün birçok olayından ve özelliğinden yararlanılır.



```

private void Form1_Load( System.Object sender,
System.EventArgs e ) {
    // Form yüklenirken kontrollerin ayarlanması:
    txtAltSayi.MaxLength = 2;
    txtUstSayi.MaxLength = 4;

    txtSayilar.Multiline = true;
    txtSayilar.ScrollBars = ScrollBars.Vertical;

    txtClipboard.ReadOnly = true;
    txtClipboard.Multiline = true;
}

// Bu olay hem txtUstSayi hem de txtAltSayi kontrolünün
// TextChanged olayında gerçekleşir.
// Handles ifadesinden sonra kontroller virgülle ayrılmıştır
private void txtUstSayi_TextChanged( System.Object
sender, System.EventArgs e ) {
    TekSayiYazdir();
}

public bool Kontrol() {
    // Metin kutularına sayı girildiyse
    if ( IsNumeric( txtUstSayi.Text ) & IsNumeric(
txtAltSayi.Text ) ) {
        // ve alt limit 0 dan büyük, ve üst limitten küçükse
        int ust = txtUstSayi.Text;
        int alt = txtAltSayi.Text;
        if ( ust > alt & alt > 0 ) {
            // giriş doğru yapılmıştır
            return true;
        }
    }
}

```

```
        }

    // Kod buraya gelirse, giriş yanlış yapılmıştır
    return false;
}

public void TekSayiYazdir() {
    if ( !(Kontrol()) ) { return; }

    txtSayilar.Clear();
    int alt = txtAltSayi.Text;
    int ust = txtUstSayi.Text;

    // Sayılar metin kutusuna, tek sayıların yazdırılması
    for ( i=alt; i<=ust; i++ ) {
        if ( i % 2 == 1 ) {
            txtSayilar.Text += i + Constants.vbCrLf;
        }
    }
}

// Sayıların txtClipboard isimli metin kutusuna
// kaydedilmesi:
private void btnKaydet_Click( System.Object sender,
System.EventArgs e ) {
    txtClipboard.Text = txtSayilar.Text;

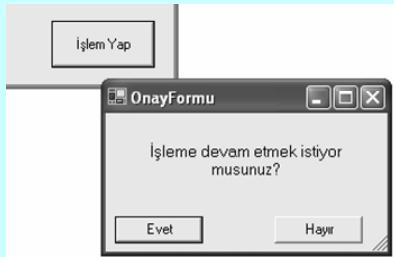
    // Sayıların kopyalanması için, önce seçilmesi gereklidir.
    txtSayilar.SelectAll();
    txtSayilar.Cut();
}

// Cut yordamı çağrııldıkten sonra veriler kopyalanır.
// Paste ile bu kopyalanan veriler geri yazdırılır.
private void btnYukle_Click( System.Object sender,
System.EventArgs e ) {
    txtSayilar.Clear();
    txtSayilar.Paste();
}
```

## Button

### Button

- Komut vermek için kullanılan düğmelerdir.



Windows uygulamalarında, form üzerinde komut düğmeleri olarak kullanılır.

## Button Özellikleri

| Özellik              | Değer Tipi           | Açıklama                                                                               |
|----------------------|----------------------|----------------------------------------------------------------------------------------|
| <b> DialogResult</b> | <b> DialogResult</b> | Ait olduğu form <b>ShowDialog</b> metodу ile çağrıldığı zaman, dönüş değerini belirler |
| <b> FlatStyle</b>    | <b> FlatStyle</b>    | Düğmeye basıldığından ve düğmenin üzerine gelindiğinde görünen formatı belirler        |

## Button Olayları

| Olay          | Açıklama                                   |
|---------------|--------------------------------------------|
| <b> Click</b> | Düğme üzerine tıklandığı zaman gerçekleşir |

Örnek: Bir formun üzerindeki düğmelerin DialogResult özellikleri değiştirilerek, özel bir mesaj kutusu tasarlabilir.



```
private void btnIslemYap_Click ( System.Object sender, System.EventArgs e ) {
    OnayFormu onay = New OnayFormu;

    onay.btnExitDialog.DialogResult = DialogResult.No;
    onay.FlatStyle = FlatStyle.Flat;

    onay.btnExit.DialogResult = DialogResult.Yes;
    onay.btnExit.FlatStyle = FlatStyle.Flat;

    if (onay.ShowDialog == DialogResult.Yes)
        // Kayıt işlemleri...
}
```

## CheckBox

- CheckBox**
- Kullanıcıya seçenekler sunmayı sağlar.
  - Birçok seçenek seçilebilir.



Kullanıcının birçok seçeneği birden seçmesi için kullanılır.

## CheckBox Özellikleri

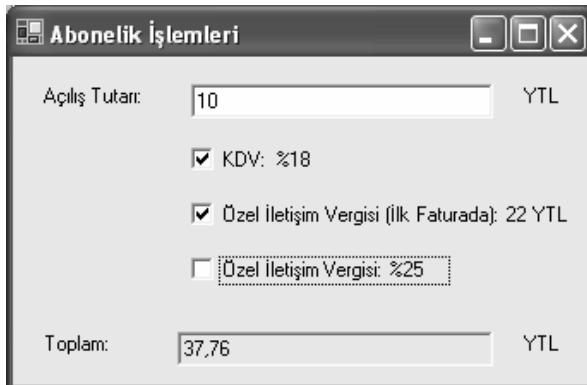
| Özellik            | Değer Tipi              | Açıklama                                                                                                                                                    |
|--------------------|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Checked</b>     | <b>Boolean</b>          | Kontrolün seçili olup olmadığını belirler                                                                                                                   |
| <b>CheckAlign</b>  | <b>ContentAlignment</b> | Seçme kutusunun ve üzerinde yazan metnin birbirlerine göre konumlarını belirler                                                                             |
| <b>Appearance</b>  | <b>Appearance</b>       | Kontrolün seçme kutusu ya da düğme şeklinde olmasını belirler                                                                                               |
| <b>ThreeState</b>  | <b>Boolean</b>          | Seçili olup olmaması dışında, <b>Intermediate</b> durum da eklenir. Eğer kontrol <b>Intermediate</b> durumundaysa <b>Checked</b> özelliği <b>True</b> olur. |
| <b>AutoChecked</b> | <b>Boolean</b>          | Kontrole basıldığı                                                                                                                                          |

|  |  |                                                                                                                                                                                        |
|--|--|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  |  | zaman seçili duruma geçileceğini belirtir. Eğer bu özellik <b>False</b> ise, kontrolün durumunu değiştirmek için, <b>Click</b> olayında, <b>Checked</b> özelliğini güncellemek gerekir |
|--|--|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## CheckBox Olayları

| Olay                | Açıklama                                            |
|---------------------|-----------------------------------------------------|
| <b>CheckChanged</b> | Seçme kutusunun durumu değiştiği zaman gerçekleşir. |

Örnek: Bir GSM şebekesinden faturalı hat açılışında toplam tutar hesaplanırken, bazı seçenekler **CheckBox** kontrolleri ile sunulabilir.



```
// Form üzerindeki tüm seçme kutularının durumu
// değiştiği zaman, toplam fiyat tekrar hesaplanır

        double toplam = txtAcilisTutari.Text;

        // İlk faturada 22 YTL açılış bedeli eklenir
        if ( cbozeliletimisilkfatura.Checked ) {
            toplam += 22;
        }

        // KDV eklenir
        if ( cbKDV.Checked ) {
            toplam *= 1.18;
        }

        // Özel iletişim vergisi eklenir
        if ( cbozeliletimis.Checked ) {
            toplam *= 1.25;
        }

        txtToplam.Text = toplam;
```

## RadioButton

- RadioButton
  - Sunulan seçeneklerin bir tanesini seçmeyi sağlar.
  - GroupBox kontrolü ile gruplanır.
- GroupBox
  - Kontrollerin düzenlenmesini sağlar.
  - Başlık yazısı bulunur.
- Panel
  - Yatay – Dikey kaydırma çubukları bulunur.

**RadioButton** kontrolleri, kullanıcıya sunulan seçeneklerden sadece bir tanesinin seçilmesine izin verir. Form üzerinde birden fazla **RadioButton** konulduğunda bu kontrollerin sadece bir tanesi seçili olabilir. Fakat bazı durumlarda, farklı seçenek grupları kullanılarak kullanıcının birden fazla seçim yapması istenebilir. Bu durumda, bazı seçenekler **GroupBox** kontrolü ile gruplanmalıdır.

Bu kontrolün özellikleri ve olayları **CheckBox** kontrolü ile aynıdır. Sadece bir seçenek seçilebildiği için, kontrollerin yapılması **CheckBox** kontrolüne göre daha kolaydır.

## GroupBox

Bu kontrol kontrollerin mantıksal bir düzende gruplanması için kullanılır. İçinde bulunan kontrollerin işleyişlerinde bir farklılık görünmez. Bir grup **RadioButton** kontrolünün, diğer **RadioButton** kontrollerinden etkilenmemesi için kullanılır.

## Panel

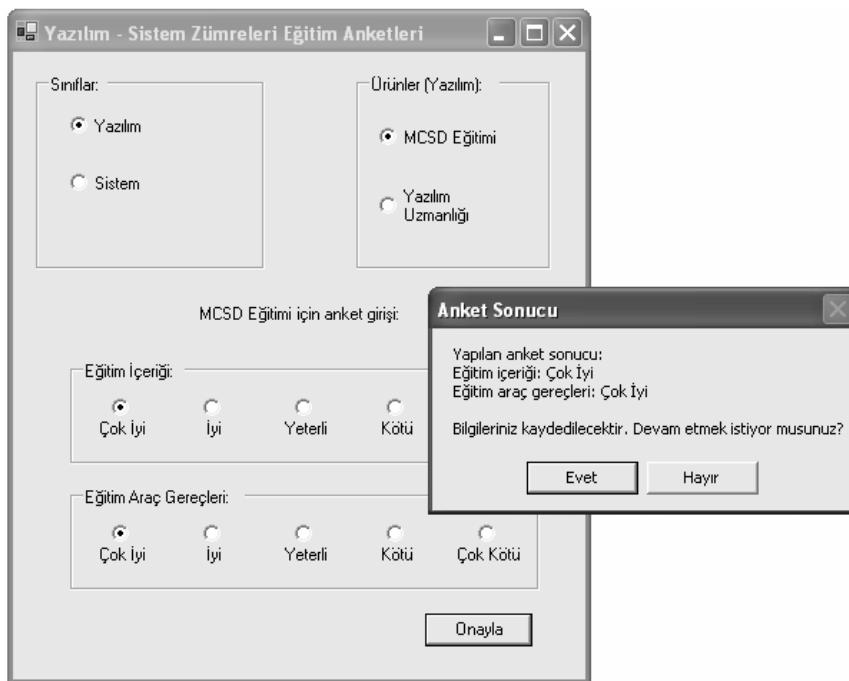
**GroupBox** kontrolü gibi, kontrollerin belli bir düzende gözükmekini sağlamak için kullanılır. **GroupBox** kontrolünden farklı olarak yatay ve dikey kaydırma çubuklarının bulunur, ancak Panel üzerinde başlık yazısı bulunmaz.

## Panel özellikleri

| Özellik           | Değer Tipi     | Açıklama                                            |
|-------------------|----------------|-----------------------------------------------------|
| <b>AutoScroll</b> | <b>Boolean</b> | Panelde kaydırma çubuklarının görünürüğünü belirler |

Paneller, seçeneklere göre bir grup kontrolün gizlenmesi veya görüntülenmesi aşamasında etkili bir rol oynar.

Örnek: **RadioButton**, **GroupBox** ve **Panel** kontrolleri, BilgeAdam eğitim anketi formunun tasarımda kullanılabilir. Anket, bir eğitimin ürünlerini hakkında yapılır. Anket bilgileri eğitim araç gereçleri ve eğitim içeriği üzerinde “çok iyi” den “çok kötü” ye kadar bir değer verilmesiyle oluşturulur. Sonuç olarak elde edilen anket bilgileri kullanıcıya gösterilerek onaylaması beklenir.



- Global değişkenlerin oluşturulması:
- ```
// Ozet bilgilerinin tutulduğu değişken
private string AnketOzet;
```
- ```
// Onaylama düğmesinin aktif hale gelmesi için
// tüm oylamaların yapılmış olması gerekir
private bool Icerikoyusecildi, Aracoyusecildi;
```
- Formun yüklenmesi sırasında, kontroller üzerinde yapılan ayarlar:
- ```
private void Form1_Load( System.Object sender,
System.EventArgs e ) {
    // Sistem ve yazılım seçeneklerini tutan
    // GroupBox kontrolleri gizlenir:
```

```

        grpSistem.Visible = false;
        grpYazilim.Visible = false;

        // Anketleri tutan Panel kontrolü gizlenir
        pnlAnket.Visible = False;

        // Onayla düğmesi oylamadan önce pasif haldedir
        btnOnayla.Enabled = False;
    }
}

```

- Eğitimler seçildiklerinde, ilgili alt seçeneklerin görüntülenmesi sağlanır. Alt seçenekler, ayrı **GroupBox** kontrollerinde tutulur.

```

private void rbYazilim_CheckedChanged( System.Object
sender, System.EventArgs e ) {
    // GroupBox kontrollerini görünümleri, eğitimleri
    // seçili olmasına doğru orantılıdır.
    grpYazilim.Visible = rbYazilim.Checked;
    grpSistem.Visible = rbSistem.Checked;

    UrunTemizle();
    pnlAnket.Visible = False;
}

// Ürünler başlangıç değerlerine çevrilir
void UrunTemizle() {
    rbMCSD.Checked = False;
    rbMCSE.Checked = False;
    rbsistemUzmanligi.Checked = False;
    rbyazilimUzmanligi.Checked = False;
}

```

- Alt ürünler seçildiklerinde, anket paneli görüntülenir ve panelin karşılama mesajında, ilgili ürünün ismi gösterilir.

```

private void rbsistemUzmanligi_CheckedChanged(
System.Object sender, System.EventArgs e ) {
    string panelMesaji;

    // Bu olayı tetikleyen RadioButton kontrolü alınır
    RadioButton basilan = sender;

    lblKarsilamaMesaji.Text = basilan.Text + " iin
anket girişi:";
    pnlAnket.Visible = true;
}

```

- Anketlerde, ilgili konularda oylama yapıldığı zaman, oylama düğmesi aktif hale getirilir ve anket mesajı oluşturulur.

```

// Eğitim içeriği için verilen oy
private void rbCokIyi_Icerik_CheckedChanged(
System.Object sender, System.EventArgs e ) {
    IcerikOyuSecildi = true;

    RadioButton basilan = sender;
    AnketOzetiCikar( "Eitim ierii: " + basilan.Text
);
}

```

```

        }

// Eğitim araç gereçleri için verilen oy

    private void rbCokIyi_Arac_CheckedChanged(
System.Object sender, System.EventArgs e ) {
    AracOyuSecildi = true;

        RadioButton basilan = sender;
        AnketOzetiCikar( "Eitim ara gereleri: " +
basilan.Text );
    }

    public void AnketOzetiCikar( string ozet ) {
        AnketOzet += ozet + Constants.vbCrLf;

        if ( IcerikOyuSecildi & AracOyuSecildi ) {
            btnOnayla.Enabled = true;
        }
    }
}

```

Anket bilgileri oluşturulduktan sonra, onay düğmesi aktif hale gelir. Bu düğmeye basıldığı zaman kullanıcıya girdiği bilgiler mesaj kutusu ile gösterilir. Kullanıcı onayladıktan sonra kayıt işlemleri gerçekleşir.

```

private void btnOnayla_Click( System.Object sender,
System.EventArgs e ) {
    string mesaj;
    mesaj = "Yaplan anket sonucu: " +
Constants.vbCrLf + AnketOzet + Constants.vbCrLf;
    mesaj += "Bilgileriniz kaydedilecektir. Devam
etmek istiyor musunuz?";

    if ( MessageBox.Show( mesaj, MsgBoxStyle.YesNo,
"Anket Sonucu" ) == DialogResult.No ) {
        return;
    }
    else {
// Anket kayıt işlemleri...

```

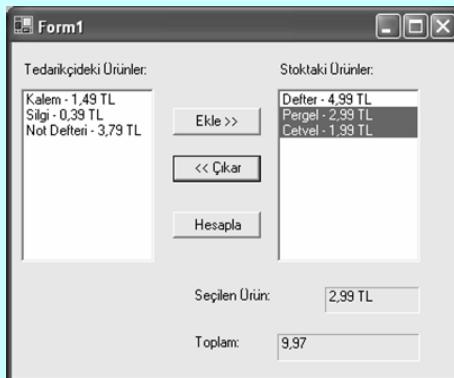
```

    }
}

```

## ListBox

- Nesnelerin listelenmesini sağlar.
- İstenen sayıda öğe seçilebilir.



Kullanıcıya sunulan seçeneklerin bir liste halinde görünmesini sağlar. Liste kutusundan istenen sayıda öğe seçilebilir.

## ListBox Özellikleri

Özellik	Değer Tipi	Açıklama
<b>Items</b>	<b>ListBox.ObjectCollection</b>	Liste kutusuna eklenen öğelerin tutulduğu koleksiyon nesnesidir.
<b>SelectedItem</b>	<b>Object</b>	Liste kutusundan seçilen öğeyi alır.
<b>SelectedItems</b>	<b>SelectedObjectCollection</b>	Liste kutusundan seçilen öğeleri alır. Seçilen öğeler dinamik bir dizide tutulur.
<b>SelectedIndex</b>	<b>Integer</b>	Liste kutusundan seçilen öğenin indisini

		alır.
<b>SelectedIndices</b>	<b>SelectedIndexCollection</b>	Liste kutusundan seçilen öğelerin indislerini bir koleksiyon nesnesinde tutar.
<b>DataSource</b>	<b>Object</b>	Listenin öğelerinin tutulduğu veri kaynağıdır. Veri kaynağı boş geçilirse <b>Items</b> koleksiyonuna eklenen öğeler görüntülenir.
<b>DisplayMember</b>	<b>String</b>	Veri kaynağından gelen öğelerin, kullanıcıya gösterilecek özelliğidir.
<b>ValueMember</b>	<b>String</b>	Veri kaynağından gelen öğelerin, dönüş değerini belirleyen özelliğidir.
<b>SelectedValue</b>	<b>Object</b>	Seçilen ögenin, liste kutusunun <b>valueMember</b> ile belirtilen özelliğidir.
<b>SelectionMode</b>	<b>SelectionMode</b>	Liste kutusundan kaç tane öğe seçilebileceğini belirtir. <b>None</b> değeri 0, <b>One</b> değeri 1, <b>MultiSimple</b> ve <b>MultiExtended</b> değerleri birden fazla ögenin seçilebileceğini belirtir.
<b>MultiColumn</b>	<b>Boolean</b>	Liste kutusundaki öğelerin biden fazla kolonda görüntülenmesini belirler.

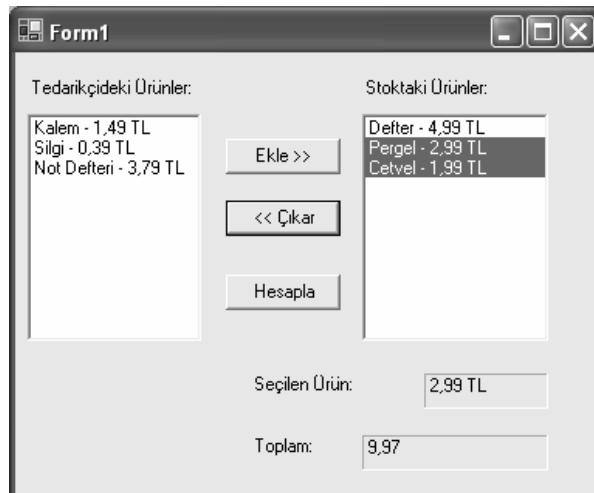
## ListBox Olayları

Olay	Açıklama
<b>SelectedIndexChanged</b>	Liste kutusunda bir öğe seçildiği zaman gerçekleşir.

## ListBox Metotları

Metot	Açıklama
<b>GetItemText</b>	Parametre olarak verilen nesnenin liste kutusunda gösterilen yazısını döndürür.
<b>GetSelected</b>	Parametre olarak verilen indisteki ögenin seçili olup olmadığını döndürür.
<b>FindString</b>	Parametrededeki <b>String</b> ifadesini liste kutusunda arayarak, bulduğu ilk ögenin indisini döndürür

Örnek: Tedarikçiden alınacak ve stokta bulunan ürünler listelemek ve alım satım işlemi yapmak için **ListBox** kontrolleri kullanılabilir.



- Ürünlerin tutulması için bir **Struct** oluşturulur. Bu ürün yapısının **ToString** metodu tekrar yazılmıştır. Bunun nedeni, **ListBox** kontrolünde listelenen nesnelerin görüntülentiği değer **ToString** metodu çağrılarak belirlenir. Dolayısıyla liste kutularında istenen formatta değerin gözükmekini sağlamak için **ToString** metodunun tekrar yazılması gereklidir.

```
public struct Urun {
    public string Ismi;
```

```

        public double Fiyat;

        public Urun( string UrunIsim, double UrunFiyat )
{
    Ismi = UrunIsim;
    Fiyat = UrunFiyat;
}

        public override string ToString() {
            return string.Format( "{0} - {1:C}", Ismi,
Fiyat );
}

```

- Liste kutularının özellikleri ayarlanır ve içine eleman doldurulur.

```

private void Form1_Load( System.Object sender,
System.EventArgs e ) {
    lbTedarikci.SelectionMode =
SelectionMode.MultiExtended;
    lbStok.SelectionMode =
SelectionMode.MultiExtended;

    UrunEkle();
}

```

```

public void UrunEkle() {
    Urun u = new Urun();

    u = new Urun( "Kalem", 1.49 );
    lbTedarikci.Items.Add( u );
    u = new Urun( "Silgi", 0.39 );
    lbTedarikci.Items.Add( u );
    u = new Urun( "Defter", 4.99 );
    lbTedarikci.Items.Add( u );
    u = new Urun( "Cetvel", 1.99 );
    lbTedarikci.Items.Add( u );
    u = new Urun( "Pergel", 2.99 );
    lbTedarikci.Items.Add( u );
    u = new Urun( "Not Defteri", 3.79 );
    lbTedarikci.Items.Add( u );
}

```

- Tedarikçi liste kutusundan, stok liste kutusuna öğe aktarılması için, seçilen değerler önce liste kutusuna eklenir. Daha sonra bu seçilen değerler, diğer listede olmayacağı için tek tek çıkartılır.

```

private void btnEkle_Click( System.Object sender,
System.EventArgs e ) {
    // Tedarikçiden alınan ürünler stok listesine eklenir
    foreach ( object item in
lbTedarikci.SelectedItems ) {
        lbStok.Items.Add( item );
    }
    // Stok listesine eklenen tüm ürünler
    // tedarikçi listesinden çıkartılır
    foreach ( object item in lbStok.Items ) {
        lbTedarikci.Items.Remove( item );
    }
}

```

```

        btnCikar.Enabled = true;
        btnHesapla.Enabled = true;
    }

```

- Stok listesinden öğe çıkarmak için, ekleme işlemine benzer kodlar çalıştırılır.

```

private void btnCikar_Click( System.Object sender,
System.EventArgs e ) {
    // Tedarikçiden alınan ürünler stok listesine eklenir
    foreach ( object item in lbStok.SelectedItems )
    {
        lbTedarikci.Items.Add( item );
        // Stok listesine eklenen tüm ürünler
        // tedarikçi listesinden çıkartılır
        foreach ( object item in lbTedarikci.Items ) {
            lbStok.Items.Remove( item );
        }

        if ( lbStok.Items.Count == 0 ) {
            btnCikar.Enabled = false;
            btnHesapla.Enabled = false;
        }
    }
}

```

- Stoktaki toplam fiyatın hesaplanması işlemi, ürünlerin fiyatlarının alınıp toplanması ile gerçekleşir.

```

private void btnHesapla_Click( System.Object sender,
System.EventArgs e ) {
    double toplam = 0;

    for ( int i=0; i<=lbStok.Items.Count - 1; i++ ) {
        Urun urun = ( Urun )( WindowsApplication8.Form1.Urun )( lbStok.Items[ i ] );
        toplam += urun.Fiyat;
    }
    lblToplam.Text = System.Convert.ToString( toplam );
}
}

```

- Stok listesindeki bir öğenin seçildiği durumda, bu öğenin fiyatı görüntülenir.

```

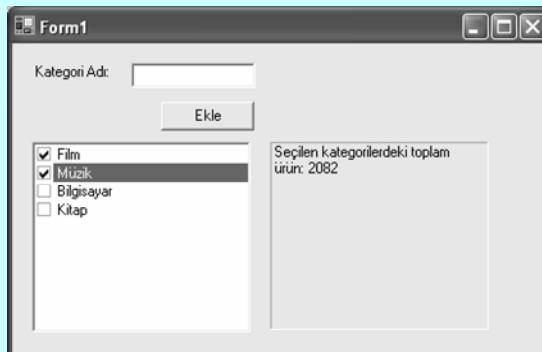
private void lbStok_SelectedIndexChanged(
System.Object sender, System.EventArgs e ) {
    Urun secilen = new
WindowsApplication8.Form1.Urun();
    secilen = ( Urun )( lbStok.SelectedItem );

    lblUrunFiyat.Text = string.Format( "{0:c}", secilen.Fiyat );
}

```

## CheckedListBox

- CheckedListBox**
- ListBox yapısındadır.
  - Öğeler işaret kutusu ile gösterilir.



Liste kutusunun tüm özellik, metot ve olaylarını alır ve listedeki öğelerin işaret kutusu ile gösterilmesini sağlar.

### CheckedListBox Özellikleri

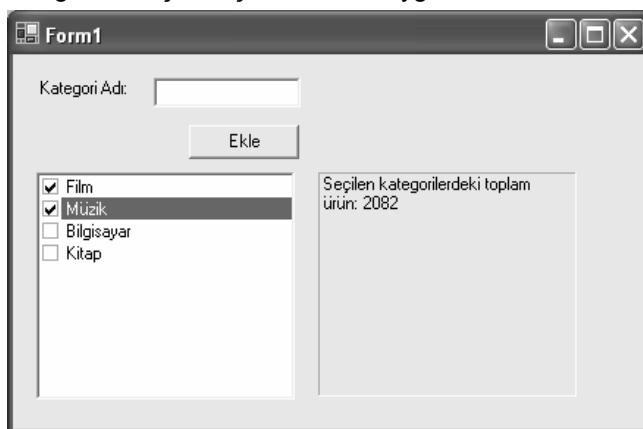
Özellik	Değer Tipi	Açıklama
<b>CheckedItems</b>	<b>CheckedItemCollection</b>	Liste kutusunda işaretlenmiş öğeleri tutar
<b>CheckedIndices</b>	<b>CheckedIndexCollection</b>	Liste kutusunda işaretlenmiş öğelerin indislerini tutar
<b>CheckOnClick</b>	<b>Boolean</b>	Liste kutusunda öğeye tıklandığı zaman işaretlenmesini belirler. <b>False</b> ise ilk tıklamada öğe seçilir, ikinci tıklamada seçme kutusu işaretlenir.

### CheckedListBox Metotları

Metot	Açıklama

<b>GetItemSelected</b>	Parametre olarak verilen indisteki öğenin seçili olup olmadığını döndürür
<b>SetItemSelected</b>	İlk parametrede verilen indisteki elemanın seçili olup olmadığını, ikinci parametrede verilen <b>Boolean</b> değeri ile belirler

Örnek: Kategori başına stoktaki toplam ürünlerin gösterildiği bir uygulamada, listelenen kategorileri seçmek için bu kontrol uygun olur.



- Listede bir öğe seçildiği zaman, seçilen tüm kategorilerin ürün stok durumu alınır ve toplam ürün sayısı kullanıcıya gösterilir.

```

private void chlistKategoriler_SelectedIndexChanged(
System.Object sender, System.EventArgs e) {
    int toplam;

    toplanır. // Listedeki seçilen öğelerin ürün adeti
    i++ ) {
        for ( i=0; i<chlistKategoriler.Items.Count - 1;
        {
            if ( chlistKategoriler.GetItemChecked( i ) )
                object secilen = null;
                secilen = chlistKategoriler.Items( i );

                // Stok durumunu gösteren fonksiyon arılır
                toplam += StokDurumu( secilen.ToString() );
        }
    }

    lblToplamUrun.Text = "Seçilen kategorilerdeki
    toplam ürün: " + toplam;
}

// Kategoriye göre, stoktaki ürünlerin belirlenmesi
public int StokDurumu( string kategori ) {
    switch ( kategori ) {

```

```

        case "Film":
            return 1100;
        case "Mzik":
            return 982;
        case "Bilgisayar":
            return 302;
        case "Kitap":
            return 1222;
        default:
            return 10;
    }

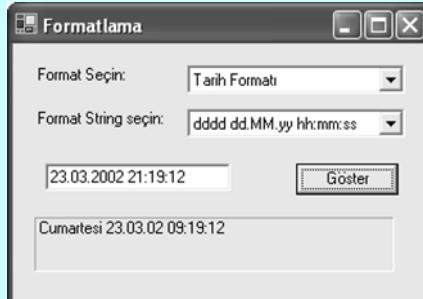
// Ekleme işlemi
private void btnKategoriEkle_Click( System.Object
sender, System.EventArgs e ) {
    chlistKategoriler.Items.Add( txtKategoriAdi.Text
);
}

```

## ComboBox

### ComboBox

- Listelenen öğeler açılan kutuda görüntülenir.
- Listeden bir tane öğe seçilebilir.

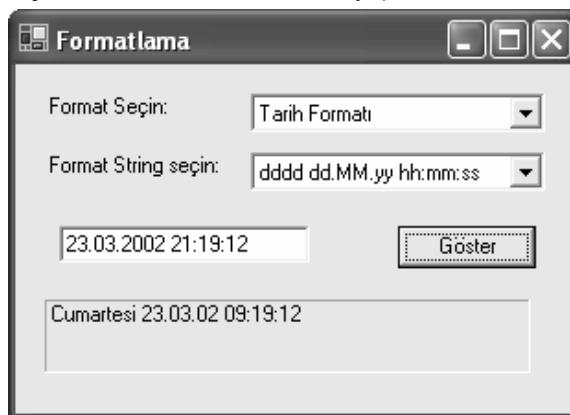


Liste kutusu ile aynı özelliklere sahiptir. Ancak listelenen öğeler açılan bir kutuda görüntülenir ve listeden en fazla bir tane öğe seçilebilir. Liste kutusuna göre bir başka farklılığı ise, isteğe bağlı olarak, kullanıcının açılan kutu üzerinde değer girebilmesidir. Dolayısıyla bir **TextBox** kontrolü gibi de davranışabilir.

## ComboBox Özellikleri

Özellik	Değer Tipi	Açıklama
<b>DropDownStyle</b>	<b>ComboBoxStyle</b>	Kontrolün listeleyme stilini belirler. <b>Simple</b> stil, listedeki sadece bir öğeyi görüntüler. <b>DropDown</b> stili, listenin tüm elemanlarını görüntüleyerek seçilmelerini ve kullanıcının değer girmesini sağar. <b>DropDownList</b> kullanıcının değer girmesini engeller.
<b>DropDownWidth</b>	<b>Integer</b>	<b>ComboBox</b> kontrolünün açılan listesinin genişliğini belirler.
<b>MaxDropDownItems</b>	<b>Integer</b>	Kontrole eklenebilecek maksimum öğe sayısını belirler.
<b>MaxLength</b>	<b>Integer</b>	Kullanıcının girebileceği maksimum karakter sayısını belirler.
<b>SelectedText</b>	<b>String</b>	Seçilen öğenin görüntülenen yazısını belirler.

Ömek: Tarih ve sayı formatlarını, kullanıcının seçimine bırakarak bir sayı veya tarih yazdırma işlemi **ComboBox** kontrolleri ile yapılabilir.



- **ComboBox** kontrollerinin özelliklerinin ayarlanması ve format tiplerine öğe eklenmesi

```
private void Form1_Load( System.Object sender,
System.EventArgs e ) {
```

```

        cmbFormat.DropDownStyle =
ComboBoxStyle.DropDownList;
        cmbFormatString.DropDownStyle =
ComboBoxStyle.DropDownList;

        cmbFormat.Items.Add( "Tarih Formatı" );
        cmbFormat.Items.Add( "Sayı Formatı" );
    }

```

- Tarih ya da sayı formatlarından biri seçildiği zaman, ikinci **ComboBox** kontrolüne değişik format seçenekleri eklenir.

```

private void cmbFormat_SelectedIndexChanged(
System.Object sender, System.EventArgs e ) {
    cmbFormatString.Items.Clear();

    switch ( cmbFormat.SelectedIndex ) {
        case 0:
            cmbFormatString.Items.Add( "dd - MM -
yyyy" );
            cmbFormatString.Items.Add( "yyyy*MM*dd
hh:mm" );
            cmbFormatString.Items.Add( "ddd
dd.MM.yy hh:mm:ss" );
            break;
        case 1:
            cmbFormatString.Items.Add( "C" );
            cmbFormatString.Items.Add( "P" );
            cmbFormatString.Items.Add( "N" );
            break;
    }
}

```

- Format seçildikten sonra metin kutusuna girilen değer alınır ve ilgili formatta gösterilir

```

private void btnGoster_Click( System.Object sender,
System.EventArgs e ) {
    switch ( cmbFormat.SelectedIndex ) {
        case 0:
            DateTime d = txtYazi.Text;
            lblSonuc.Text = d.ToString(
cmbFormatString.Text );
            break;
        case 1:
            int i = txtYazi.Text;
            lblSonuc.Text = i.ToString(
cmbFormatString.Text );
            break;
    }
}

```

## NumericUpDown

### NumericUpDown

- Sayısal değerlerin yukarı aşağı okları ile seçilmesini sağlar



### DomainUpDown

- Object tipinde nesnelerin seçilmesini sağlar.



Bu kontrol kullanıcının, sayısal bir değeri girmesini veya yukarı aşağı okları ile seçmesini sağlar.

## NumericUpDown Özellikleri

Özellik	Değer Tipi	Açıklama
<b>HexaDecimal</b>	<b>Boolean</b>	Sayıların on altılık tabanda görüntülenmesini belirler.
<b>Increment</b>	<b>Decimal</b>	Aşağı yukarı oklar kullanıldığında, sayıların artma ve azalma adımlarını belirler.
<b>Maximum</b>	<b>Decimal</b>	Kontrolde gösterilen sayıların alabileceği maksimum değeri belirler.
<b>Minimum</b>	<b>Decimal</b>	Kontrolde gösterilen sayıların alabileceği minimum değeri

		belirler.
<b>ThousandSeparators</b>	<b>Boolean</b>	Sayıların basamak ayracını gösterilmesini belirler.
<b>Value</b>	<b>Decimal</b>	Kontrolün gösterdiği sayı değerini belirler.
<b>ReadOnly</b>	<b>Boolean</b>	<b>True</b> değerini alırsa kullanıcının giriş yapmasını engeller.

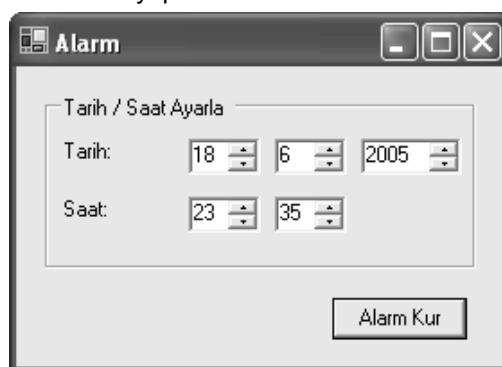
## NumericUpDown Olayları

Olay	Açıklama
<b>ValueChanged</b>	Kontrolün sayı değeri değiştiği zaman gerçekleşir

## NumericUpDown Metotları

Metot	Açıklama
<b>DownButton</b>	Aşağı düğmesine basar ve sayı değerini düşürür.
<b>UpButton</b>	Yukarı düğmesine basar ve sayı değerini artırır.

**Örnek:** Alarm kurarken, tarih ve zaman değerlerinin ayarlanması **NumericUpDown** kontrolü ile yapılabilir.



- Tarih ve zaman değerlerinin alabileceği maksimum ve minimum değerler ayarlanır.

```
private void Form1_Load( System.Object sender,
System.EventArgs e ) {
    nYil.Minimum = 1;
    nAy.Minimum = 1;
```

```

nGun.Minimum = 1;
nYıl.Maximum = 2099;
nAy.Maximum = 12;
nGun.Maximum = 31;

nSaat.Minimum = 0;
nDakika.Minimum = 0;

nSaat.Maximum = 23;
nDakika.Maximum = 59;

nYıl.Value = DateTime.Now.Year;
nAy.Value = DateTime.Now.Month;
nGun.Value = DateTime.Now.Day;
nSaat.Value = DateTime.Now.Hour;
nDakika.Value = DateTime.Now.Minute;
}

```

- Bu değerlerden herhangi biri değiştiği zaman, doğru tarih ve zaman değerinin girilmesi kontrol edilir

```

private void nGun_ValueChanged( System.Object
sender, System.EventArgs e ) {
    string tarih;
    tarih = nGun.Value + "." + nAy.Value + "." +
nYıl.Value;

    if ( !( IsDate( tarih ) ) ) {
        MessageBox.Show( tarih );
    }

    string zaman;
    zaman = nSaat.Value + ":" + nDakika.value;

    if ( !( IsDate( zaman ) ) ) {
        MessageBox.Show ( zaman );
    }
}

```

## DomainUpDown

**NumericUpDown** kontrolü ile aynı yapıdadır ancak sayısal değerler yerine Object tipinde değerler tutar. Bu değerler kontrolün **Items** koleksiyonunda tutulur. Kontrol, bu özelliği ile liste kutusuna benzemektedir.

### DomainUpDown Özellikleri

Özellik	Değer Tipi	Açıklama
<b>Items</b>	<b>DomainUpDownItemCollection</b>	Kontrolün öğelerinin tutulduğu dinamik bir listedir.
<b>SelectedItem</b>	<b>Object</b>	Kontrolde seçilen öğeyi tutar.

<b>Wrap</b>	<b>Boolean</b>	Liste sonuna gelindiğinde baştaki veya sondaki öğeye geri dönülmesini belirler.
-------------	----------------	---

## DomainUpDown Olayları

Olay	Açıklama
<b>SelectedIndexChanged</b>	Kontrolde seçilen öğe değiştiği zaman gerçekleşir.

Örnek: Metin kutularının değiştirilmek istenen yazı tipleri **DomainUpDown** kontrolünde tutulabilir.



```

private void Form1_Load( System.Object sender,
System.EventArgs e ) {
    for ( i=0; i<=10; i++ ) {
        dFont.Items.Add(
System.Drawing.FontFamily.Families[ i ].Name );
    }

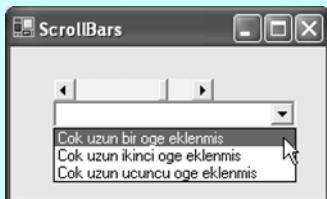
    dFont.Wrap = true;
}

private void dFont_SelectedIndexChanged(
System.Object sender, System.EventArgs e ) {
    if ( dFont.SelectedIndex >= 0 ) {
        TextBox1.Font = new Font(
dFont.SelectedItem.ToString(), 15 );
    }
}

```

## HScrollBar / VScrollBar

- HScrollBar – VScrollBar**
- Sayısal değer taşıyan kaydırma çubuklarıdır.



**Horizontal – Vertical ScrollBar** kontrolleri, sayısal bir değer taşıyan kaydırma çubuklarıdır. Tuttukları değerlerin sayısal olması bakımından **NumericUpDown** kontrolüne benzer. Bu kontroller, üzerinde kaydırma çubukları olmayan kontroller üzerinde kullanılabilir. Örneğin bir **ListBox**, **Panel** gibi kontrollerin kendi **ScrollBar** kontrolleri vardır. **TextBox** kontrolünün de ilgili özellikleri ayarlanarak yatay ve dikey **ScrollBar** kontrolleri gösterilebilir.

### ScrollBar Özellikleri

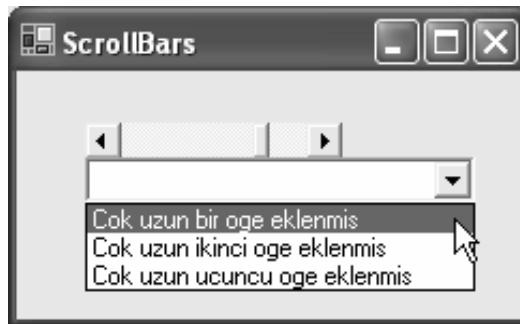
Özellik	Değer Tipi	Açıklama
<b>Value</b>	<b>Integer</b>	Kaydırma çubuğuuna pozisyonuna göre alınan değeri tutar.
<b>SmallChange</b>	<b>Integer</b>	Kontrolü, üstündeki oklar ile kaydırıldığı zaman eklenenek ya da çıkartılacak değeri tutar.
<b>LargeChange</b>	<b>Integer</b>	Kontrolü, kaydırma çubuğuundaki boşluğa tıklanarak kaydırıldığında zaman eklenenek ya da çıkartılacak değeri tutar.
<b>Minimum</b>	<b>Integer</b>	<b>Value</b> özelliğinin alabileceği maksimum değeri tutar

<b>Maximum</b>	<b>Integer</b>	<b>value</b> özelliğinin alabileceği minimum değeri tutar
----------------	----------------	---

## ScrollBar Olayları

Olay	Açıklama
<b>Scroll</b>	Çubuklar kaydırıldıkları zaman gerçekleşir.
<b>ValueChanged</b>	Kod ile ya da çubuklar kaydırılınca <b>value</b> özelliği değiştiği zaman gerçekleşir.

Örnek: Bir **ComboBox** kontrolünün öğelerini listelemek için, aşağıya doğru bir kaydırma çubuğu görüntülenir. Ancak listedeki bazı elemanların kontrole sağlamiyorsa, çalışma anında bu kontrolün genişliği artırılabilir.



```

private void Form1_Load( System.Object sender,
System.EventArgs e ) {
    hsGenislik.Maximum = ComboBox1.width * 2;
    hsGenislik.value = ComboBox1.width;
}

private void hsGenislik_Scroll( System.Object
sender, System.Windows.Forms.ScrollEventArgs e ) {
    ComboBox1.width = hsGenislik.value;
}

```

## TrackBar

### TrackBar

- Kaydırma çubuğu pozisyonunu görsel olarak takip edilir.
- Pozisyon, klavye tuşları ile değiştirilebilir.

Bu kontrol, **ScrollBar** kontrollerine benzer yapıdadır ancak kontrol, bir cetvel biçiminde olduğu için, üzerinde durulan pozisyon görsel olarak takip edilebilir. Kontrolün, kaydırma çubuklarından bir farkı da üzerine odaklanabilir olmasıdır. Dolayısıyla kontrolün **value** değeri klavyede bulunan yukarı, aşağı, sağ, sol okları ve **PageUp**, **PageDown** düğmeleri ile değiştirilebilir.

### TrackBar Özellikleri

**TrackBar** kontrolünün birçok özelliği **ScrollBar** kontrollerinin özellikleriyle aynıdır. Fakat kontrolü daha esnek hale getiren birkaç özelliği vardır.

Özellik	Değer Tipi	Açıklama
<b>TickStyle</b>	<b>TickStyle</b>	Kontrolün değerini gösteren çizgilerin pozisyonunu belirler
<b>TickFrequency</b>	<b>Integer</b>	Çizgiler arasında kalan değerlerin sayısını belirler
<b>Orientation</b>	<b>Orientation</b>	Kontrolün yönünün yatay veya dikey olmasını sağlar.

## TabControl

- TabControl**
- Sekme sayfa yapısı sunar.
  - TabPage nesnelerinden oluşur.



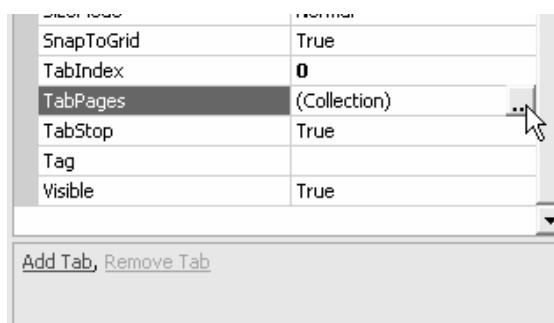
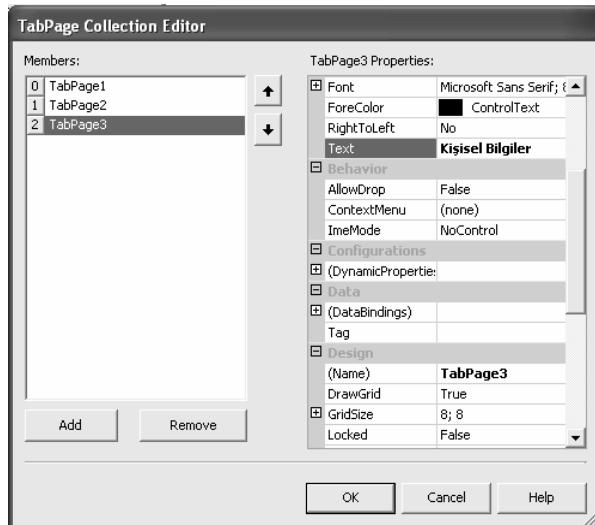
**TabControl** nesnesi, içinde sekme sayfaları tutan yapıdır. Bu sayfalar, **TabPage** nesneleri olarak oluşturulup yapılandırıldıktan sonra **TabControl** nesnesinin **TabPages** koleksiyonuna eklenir. Ekleme işlemi, **Properties** paneli ile tasarım alanında da yapılabilir.

### TabControl Özellikleri

Özellik	Değer Tipi	Açıklama
<b>HotTrack</b>	<b>Boolean</b>	Fare ile sekme sayfalarının üzerine gelindiğinde, isimlerinin görsel olarak değişmesini belirler
<b>ItemSize</b>	<b>Size</b>	Sekme sayfalarının boyutunu belirler
<b>Multiline</b>	<b>True</b>	Eklenen sekmelerin birden fazla satırda üst üste gözükmesini belirler
<b>ShowToolTips</b>	<b>Boolean</b>	Fare sekme sayfalarının üzerindeyken bilgi mesajının gösterilmesini belirler
<b>SelectedTab</b>	<b>TabPage</b>	Seçilen sekme sayfasını belirler

<b>SelectedIndex</b>	<b>Integer</b>	Seçilen sekme sayfasının indisini belirler
<b>TabCount</b>	<b>Integer</b>	Sekme sayısını belirler
<b>TabPage</b>	<b>TabPageCollection</b>	Kontrolün içinde bulunduğu sekme sayfalarının koleksiyonudur.

**TabControl** nesnesine **TabPage** sayfaları eklemek için tasarım anında **TabPage Collection Editor** penceresinden yararlanılabilir.



## TabPage Özellikleri

Sekme sayfaları, normal form tasarımları gibi kontroller eklenecek yapılır.

**TabPage** kontrolü **Panel** kontrolünden türer ve **Panel** kontrolünün tüm özelliklerini alır.

Özellik	Değer Tipi	Açıklama
<b>ToolTipText</b>	<b>String</b>	Bu özelliğin değeri, fare sayfanın üzerindeyken, bilgi mesajı olarak gösterilir. Ait

		olduğu nesnesinin özelliği <b>True</b> olmalıdır.	<b>TabControl</b> <b>ShowToolTip</b>
--	--	---	---

**Örnek:** Bir kullanıcı kaydının tek bir formda görüntülenmesi isteniyorsa, bu form **TabControl** ile küçük sayfalara bölünebilir.



## DatePicker

### DatePicker

- Takvimden zaman değeri seçilmesini sağlar.
- Takvim yapısı açılan kutu şeklindedir.



Bir açılan kutudan zaman değeri seçmeyi sağlar. Seçilen değer **Date** tipinde olur.

## DatePicker Özellikleri

Özellik	Değer Tipi	Açıklama
<code>CalendarTrailingForeColor</code>	<code>Color</code>	Bir önceki ve bir sonraki ayın günlerinin görüntülenme rengi
<code>CalendarTitleForeColor</code>	<code>Color</code>	Takvim başlığının önalan rengi
<code>CalendarTitleBackColor</code>	<code>Color</code>	Takvim başlığının arka plan rengi
<code>CalendarMonthBackgroundColor</code>	<code>Color</code>	Takvim arka plan rengi
<code>CalendarForeColor</code>	<code>Color</code>	Takvimdeki yazıların ön plan rengi
<code>CalendarFont</code>	<code>Font</code>	Takvimin gösterileceği yazı tipi ayarları
<code>ShowCheckBox</code>	<code>Boolean</code>	Tarih değerinin yanında seçme kutusunun gösterilmesi.
<code>Checked</code>	<code>Boolean</code>	Seçme kutusu görüntülendiği zaman, tarihin seçili olup olmadığını gösterir
<code>Format</code>	<code>DateTimePickerFormat</code>	Kontrolün görüntüleneceği formatı belirler. <code>Long</code> , <code>Short</code> değerleri uzun ve kısa tarih formatını, <code>Time</code> sadece zamanı gösterir. <code>Custom</code> değeri, <code>CustomFormat</code> özelliğine girilen formatta gösterileceğini belirler
<code>CustomFormat</code>	<code>String</code>	Tarihin hangi formatta gösterileceğini belirler.
<code>Value</code>	<code>Date</code>	Seçilen tarih değerini belirler
<code>MaxDate</code>	<code>Date</code>	Kontrolün alabileceği maksimum tarih değeri
<code>MinDate</code>	<code>Date</code>	Kontrolün alabileceği minimum tarih değeri
<code>ShowUpDown</code>	<code>Boolean</code>	Kontrolün formunu açılan kutu ya da yukarı aşağı okları formatında gösterir. Bu özellik <code>True</code> olduğunda, kontrolün formatı, <code>NumericUpDown</code>

		kontrolünün formatında olur.
--	--	------------------------------

Örnek: Verit tabanından bir kaydın belli tarih aralıkları ile sorgulanması sırasında, kullanıcının başlangıç ve bitiş tarihlerini seçmesi için bu kontrol kullanılır.



```

private void Form1_Load( System.Object sender,
System.EventArgs e ) {
    TarihAyarla( dtBaslangic );
    TarihAyarla( dtBitis );
}

public void TarihAyarla( DateTimePickerer dtTarih ) {
    dtTarih.Format = DateTimePickerFormat.Custom;
    dtTarih.CustomFormat = "dd - MM - yyyy";
    dtTarih.MaxValue = DateAndTime.Now.AddYears( 2 );
    dtTarih.MinValue = DateAndTime.Now.AddYears( -2 );
}

private void btnAra_Click( System.Object sender,
System.EventArgs e ) {
    DateTime basTarih, sonTarih;
    basTarih = dtBaslangic.Value;
    sonTarih = dtBitis.Value;

    if ( DateTime.Compare( basTarih, sonTarih ) == 1
) { return; }

    string Sql;
    Sql = "Select * From Siparisler where
SiparisTarih Between ";
    Sql += basTarih + " And " + sonTarih;

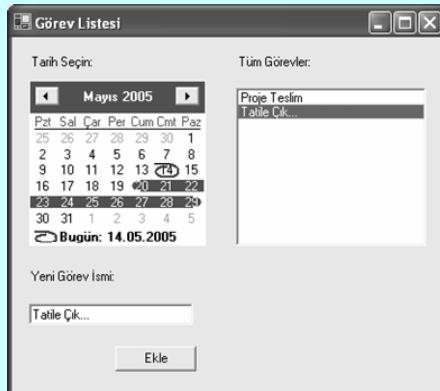
    // komutunu çalıştır
}

```

## MonthCalendar

### MonthCalendar

- Takvimden bir zaman aralığı seçilmesini sağlar.



**DateTimePicker** kontrolünün açılan takvimi biçimindedir. Bu kontrol kullanıcıya, tarih alanları üzerinde daha esnek çalışmayı sağlar.

### MonthCalendar Özellikleri

**DateTimePicker** kontrolünün birçok özelliğini almasına rağmen, bazı özelliklerinde değişiklikler görülür. Örneğin **value** özelliği bu kontrolde yoktur. Bu kontrolden seçilen değerler, bir tarih aralığıdır. Dolayısıyla tek bir **Date** tipini tutan bir özellik yoktur.

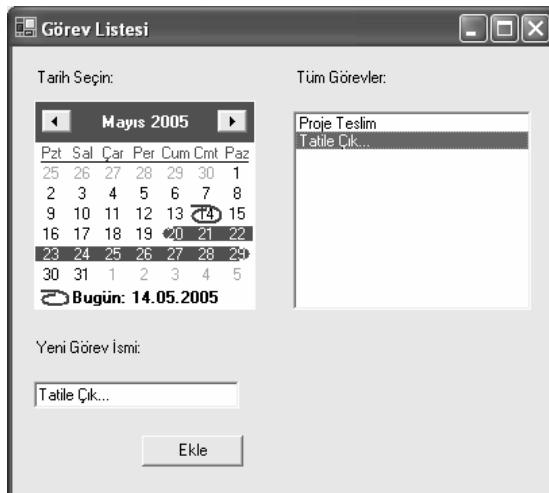
Özellik	Değer Tipi	Açıklama
<b>MaxSelectionCount</b>	<b>Integer</b>	Bir seferde maksimum kaç gün seçileceğini belirler.
<b>SelectionRange</b>	<b>SelectionRange</b>	Başlangıç ve bitiş tarihlerinden oluşan bir seçim aralığı nesnesidir.
<b>SelectionBegin</b>	<b>Date</b>	Seçilen tarih aralığının hangi tarihten itibaren başladığını belirler
<b>SelectionEnd</b>	<b>Date</b>	Seçilen tarih aralığının hangi tarihte bittiğini belirler
<b>ScrollChange</b>	<b>Integer</b>	İleri geri düğmelerine

		basıldığı zaman kaç ay atlanacağını belirler
<b>MonthlyBoldedDates</b>	<b>Date()</b>	Takvimde hangi günlerin kalın yazı tipinde gösterileceğini belirler. İşaretlenen günler, her ay için kalın gösterilir.
<b>ShowToday</b>	<b>Boolean</b>	Takvimin alt kısmında, sistem takvimine göre hangi günde olduğunu gösterir
<b>ShowTodayCircle</b>	<b>Boolean</b>	Takvimde, o günün seçili olmasını belirler
<b>ShowWeekNumbers</b>	<b>Boolean</b>	Takvimin sol tarafında, yılın hafta numaralarını gösterir

## MonthCalendar Olayları

Olay	Açıklama
<b>DateChanged</b>	Seçilen tarihten farklı bir tarih seçildiğinde gerçekleşir
<b>DateSelected</b>	Yeni bir tarih seçildiği zaman gerçekleşir. <b>DateChanged</b> olayı gerçekleşikten hemen sonra bu olay gerçekleşir.

Örnek: Yapılacak görevlerin tutulduğu bir Windows uygulamasında, görevin başlangıç ve bitiş tarihleri tek bir **MonthCalendar** kontrolünden kolaylıkla seçilebilir.



- Görevlerin tanımlanması için bir Görev sınıfı oluşturulur.

```
public class Gorev {
    public string GorevIsmi;
    public DateTime BaslangicTarihi;
    public DateTime BitisTarihi;

    // Liste kontrollerinde görevin isminin görüntülenmesi
    // için, ToString metodunu tekrar yazmak gereklidir.
    public override string ToString() {
        return GorevIsmi;
    }

    public Gorev( string Isim, DateTime basTarihi,
    DateTime bitTarihi ) {
        this.GorevIsmi = Isim;
        this.BaslangicTarihi = basTarihi;
        this.BitisTarihi = bitTarihi;
    }
}
```

- Görevler ekleneceği zaman, yeni bir görev nesnesi oluşturulur ve görevin başlangıç-bitiş tarihleri ayarlanır.

```
private void Form1_Load( System.Object sender,
System.EventArgs e ) {
    // Maksimum iki hafta seçilsin
    MonthCalendar1.MaxSelectionCount = 14;
}

private void btnEkle_Click( System.Object sender,
System.EventArgs e ) {
    DateTime baslangicTarihi =
MonthCalendar1.SelectionStart;
    DateTime bitisTarihi =
MonthCalendar1.SelectionEnd;
    string gorevIsmi = txtYeniGorev.Text;

    Gorev yeniGorev = new Gorev( gorevIsmi,
baslangicTarihi, bitisTarihi );
    ListBox1.Items.Add( yeniGorev );

}

private void ListBox1_SelectedIndexChanged(
System.Object sender, System.EventArgs e ) {
    Gorev secilen;
    secilen = ListBox1.SelectedItem;

    MonthCalendar1.SelectionStart =
secilen.BaslangicTarihi;
    MonthCalendar1.SelectionEnd =
secilen.BitisTarihi;
    txtYeniGorev.Text = secilen.GorevIsmi;
}
```

## Timer

### Timer

- Zaman değeri ayarlanabilen sayaçtır.
- Interval özelliği ile, kaç milisaniyede bir çalışacağı belirlenir.

Windows uygulamalarında sayaç görevini görür.

### Timer Özellikleri

Özellik	Değer Tipi	Açıklama
<b>Enabled</b>	<b>Boolean</b>	Kontrolün aktif olup olmadığını belirler.
<b>Interval</b>	<b>Integer</b>	Sayacın hangi zaman aralığında bir çalışması gerektiğini belirler. Milisaniye cinsindedir.

### Timer Olayları

Olay	Açıklama
<b>Tick</b>	<b>Interval</b> özelliğinde belirtilen zaman değeri geçtiğinde gerçekleşir.

### Timer Metotları

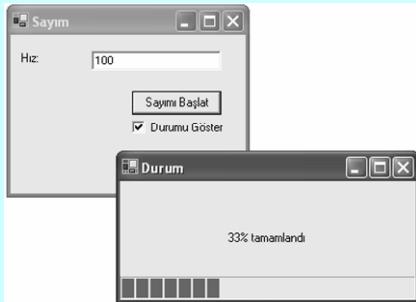
Metot	Açıklama

<b>Start</b>	Sayacı başlatır
<b>Stop</b>	Sayacı durdurur

## ProgressBar

### ProgressBar

- Yapılan işlemlerin ilerleyişini gözlemeyi sağlar.
- Maksimum ve minimum değerleri arasındaki pozisyonu gösterir.

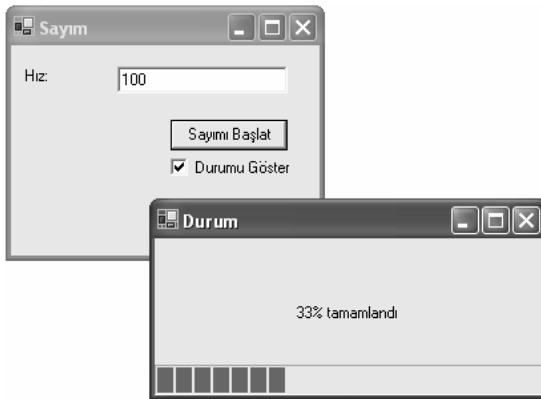


**ProgressBar**, belli bir andaki değerinin, alabileceği değer aralığına göre yüzdesini gösterir. Yapılan bir işlemin ilerleyişini göstermesi açısından oldukça kullanışlı bir kontroldür.

### ProgressBar Özellikleri

Özellik	Değer Tipi	Açıklama
<b>Minimum</b>	<b>Integer</b>	Kontrolün alabileceği minimum değer belirler
<b>Maximum</b>	<b>Integer</b>	Kontrolün alabileceği minimum değer belirler
<b>Value</b>	<b>Integer</b>	Kontrolün verilen değer aralığındaki pozisyonunu belirler

Ömek: **ProgressBar** bir sayılm işleminde kalan durumu göstermek için kullanılabilir.



- **ProgressBar** ile durumun gösterileceği ayrı bir form eklenir. Burada sayma işleminin hızı için bir **Timer** bulunur. Sayaç her işlediğinde yeni değer **ProgressBar** kontrolünde gösterilir.

```
public int kalan;

private void Durum_Load( System.Object sender,
System.EventArgs e ) {
    kalan = ProgressBar1.Maximum;
    Timer1.Start();
}

private void Timer1_Tick( System.Object sender,
System.EventArgs e ) {
    if ( kalan == 0 ) {
        Timer1.Stop();
        this.Close();
    }

    int aralik;
    aralik = ProgressBar1.Maximum -
ProgressBar1.Minimum;

    int oran = ( aralik - kalan ) / aralik * 100;
    Label1.Text = oran + "% tamamlandı";

    ProgressBar1.Value = ProgressBar1.Maximum -
kalan;
    kalan -= 1;
}
```

- Oluşturulan bu form, başlangıç formundan çağrılarak durum gösterilir.

```
private void Form1_Load( System.Object sender,
System.EventArgs e ) {
    CheckBox1.Checked = true;
}

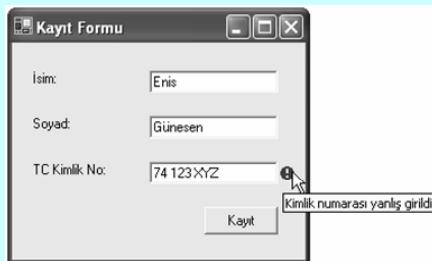
private void btnBaslat_Click( System.Object sender,
System.EventArgs e ) {
    Durum frmDurum = new Durum();
    frmDurum.Timer1.Interval = TextBox1.Text;

    if ( CheckBox1.Checked ) {
        frmDurum.ShowDialog();
```

```
}
```

## ErrorProvider

- Hata mesajlarını kontrollerin yanında gösterir.



Form üzerindeki kontrollerin yanında hata mesajları gösterilmesini sağlar.

### ErrorProvider Özellikleri

Özellik	Değer Tipi	Açıklama
<b>BlinkRate</b>	<b>Integer</b>	Hata simgesinin kaç milisaniyede bir yanıp söneceğini belirler
<b>BlinkStyle</b>	<b>ErrorBlinkStyle</b>	Hata simgesinin yanıp sönme stilini belirler. <b>AlwaysBlink</b> , her zaman, <b>BlinkIfDifferentError</b> farklı bir hata meydana geldiğinde yanıp söneceğini belirler. <b>NeverBlink</b> ise simgenin yanıp sönmeden görüntüleneceğini belirler
<b>Icon</b>	<b>Icon</b>	Hata mesajlarının gösterilmesi sırasında çıkan simgeyi belirler

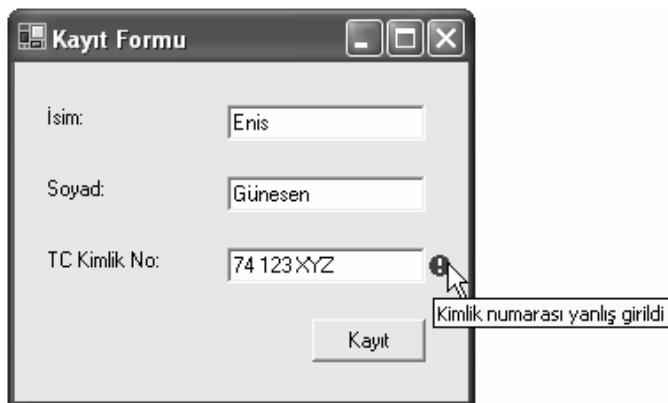
## ErrorProvider Metotları

Metot	Açıklama
<b>SetError</b>	Kontrollerin hata mesajlarının belirlenmesi için kullanılır

**ErrorProvider** kontrolü forma eklendiği zaman, **Properties** panelinde, kontrollerin ekstra özellikleri görünür. Bu özellikler, forma eklenen her **ErrorProvider** için oluşturulacaktır.

Özellik	Açıklama
<b>IconAignment On ErrorProviderİsmi</b>	Hata simgesinin, kontrolün üzerinde nerde bulunacağını belirler
<b>IconPadding On ErrorProviderİsmi</b>	Hata simgesinin, kontrolden kaç piksel uzakta duracağını belirler
<b>Error On ErrorProviderİsmi</b>	Varsayılan hata mesajını belirler

Örnek: Kayıt işlemlerinin yapıldığı sırada, isim soyadı ve TC kimlik numaralarının girişleri **ErrorProvider** kontrolü ile denetlenebilir.



- Metin kutularının **Validating** olayında, girilen verilerin kontrolleri yapılır ve gereki̇ği durumlarda **ErrorProvider** ile hata mesajları gösterilir.

```
private void txtIsim_Validating( object sender,
System.ComponentModel.CancelEventArgs e ) {
    if ( txtIsim.Text == "" ) {
        ErrorProvider1.SetError( txtIsim, "sim alan
bo girilemez" );
        // Bu komut olayın gerçekleşmesini engeller
        // Dolayısıyla veri girilmeden bu alandan çıkışlamaz
        e.Cancel = true;
    }
}
```

```
        else {
            // Eğer beri doğru girilmişse, Error simgesini
            // gizlemek için, hata mesajı boş girilir
            ErrorProvider1.SetError( txtIsim, "" );
        }

    }

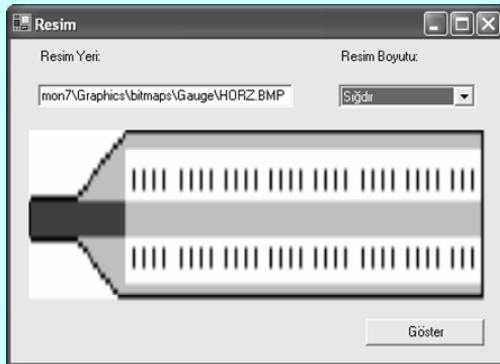
    private void txtSoyad_Validating( object sender,
System.ComponentModel.CancelEventArgs e ) {
    if ( txtSoyad.Text == "" ) {
        ErrorProvider1.SetError( txtSoyad, "Soyad
alan bo girilemez" );
        e.Cancel = true;
    }
    else {
        ErrorProvider1.SetError( txtSoyad, "" );
    }
}

private void txtTCKimlik_Validating( object sender,
System.ComponentModel.CancelEventArgs e ) {
    if ( !( IsNumeric( txtTCKimlik.Text ) ) ) {
        ErrorProvider1.SetError( txtTCKimlik,
"Kimlik numaras yanl girildi" );
        e.Cancel = true;
    }
    else {
        ErrorProvider1.SetError( txtTCKimlik, "" );
    }
}
```

## PictureBox

### PictureBox

- Resim görüntülemeyi sağlar.

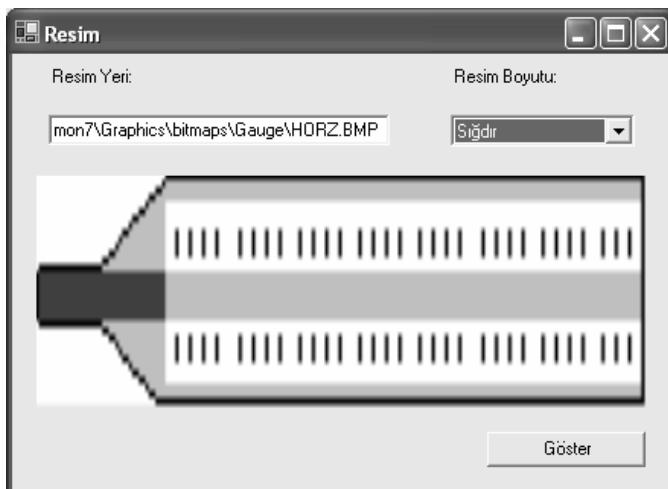


Form üzerinde bir resim görüntülemek için kullanılır.

### PictureBox Özellikleri

Özellik	Değer Tipi	Açıklama
<b>Image</b>	<b>Image</b>	Kontrolün resim kaynağını belirler
<b>SizeMode</b>	<b>PictureBoxSizeMode</b>	Kontrolün, resmi nasıl görüntüleyeceğini belirler. <b>AutoSize</b> değeri, kontrolün büyüklüğünü resmin büyüklüğüne göre ayarlar. <b>CenterImage</b> değeri, resmi kontrolün ortasına gelecek şekilde ayarlar. <b>Normal</b> değeri, kontrolün sol üst köşesine göre konumlandırır. <b>StretchImage</b> değeri, resmi kontrolün büyülüğüne göre boyutlandırır ve resmin tam görünmesini sağlar.

Örnek: Form üzerinde bir resmin değişik boyutlarda gösterilmesi için PictureBox kontrolü tercih edilir.



```
private void Form1_Load( System.Object sender,
System.EventArgs e ) {
    ComboBox1.Items.Add( "Normal" );
    ComboBox1.Items.Add( "Ortala" );
    ComboBox1.Items.Add( "Sdr" );
    ComboBox1.Items.Add( "Otomatik Boyutlandır" );
}

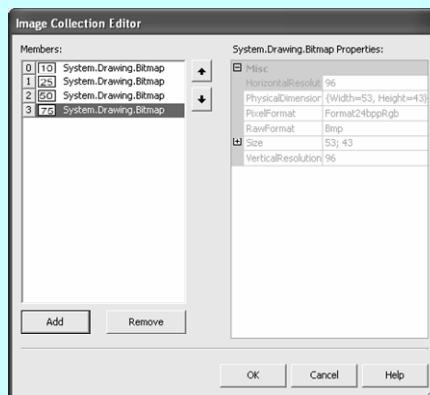
private void ComboBox1_SelectedIndexChanged(
System.Object sender, System.EventArgs e ) {
    switch ( ComboBox1.SelectedIndex ) {
        case 0:
            PictureBox1.SizeMode =
PictureBoxSizeMode.Normal;
            break;
        case 1:
            PictureBox1.SizeMode =
PictureBoxSizeMode.CenterImage;
            break;
        case 2:
            PictureBox1.SizeMode =
PictureBoxSizeMode.StretchImage;
            break;
        case 3:
            PictureBox1.SizeMode =
PictureBoxSizeMode.AutoSize;
            break;
    }
}

private void btnGoster_Click( System.Object sender,
System.EventArgs e ) {
    PictureBox1.Image = Image.FromFile(
txtResimYeri.Text );
}
```

## ImageList

### ImageList

- Resimleri liste halinde tutar.
- Kontrollerin öğelerine resim atanmasını sağlar.



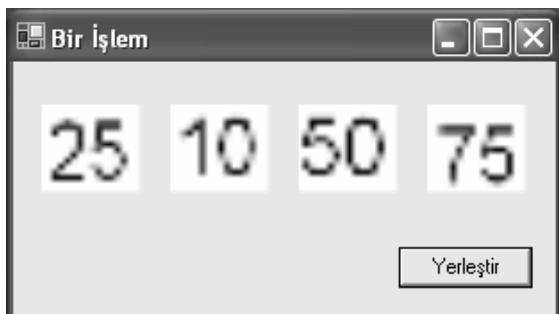
**ImageList** kontrolü, form kontrolleri ve içinde bulunan öğeleri için arka plan resmi sağlayan bir listesi görevini görür.

### ImageList Özellikleri

Özellik	Değer Tipi	Açıklama
<b>Images</b>	<b>ImageCollection</b>	Kontrolün içinde bulunan resimlerin listelendiği dinamik bir koleksiyondur. Bu özellik bir koleksiyon olduğu için, diğer liste kontrollerinin öğelerinin resmini belirleme işlemi büyük ölçüde kolaylaşır.
<b>ImageSize</b>	<b>Size</b>	Kontrolün tuttuğu resimlerin büyüklüğünü belirler
<b>TransparentColor</b>	<b>Color</b>	Listedeki resimlerin bu özellikte belirtilen renkteki bölgeleri saydam olur.

Windows uygulamalarında **ImageList** kontrolünün kullanımı, diğer kontrollerin **ImageList** özelliği olarak belirlendikten sonra gerçekleşir. Bu kontrollerin listelediği öğelerin arka plan resimleri **ImageList** kontrolü ile belirlenir.

Örnek: **ImageList** kontrolünde tutulan resimler bir sayı oyununda rasgele resim göstermek için kullanılabilir.



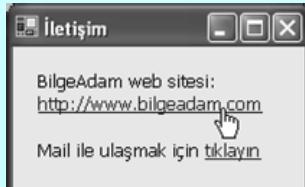
```
private void btnYerlestir_Click( System.Object sender, System.EventArgs e ) {
    int max = ImageList1.Images.Count - 1;

    Random r = new Random();
    PictureBox1.Image = ImageList1.Images( r.Next(
max ) );
    PictureBox2.Image = ImageList1.Images( r.Next(
max ) );
    PictureBox3.Image = ImageList1.Images( r.Next(
max ) );
    PictureBox4.Image = ImageList1.Images( r.Next(
max ) );
}
```

## LinkLabel

### LinkLabel

- Nesnelere bağlantı kurulmasını sağlar.
- Metin içinde birden fazla bağlantı tutabilir.



Bu kontrol, nesnelere bağlantı kurmak için kullanılır. **Text** özelliğinde birden fazla nesneye bağlantı kurulabilir. Bu durumda, kontrole tıklandığı zaman hangi bağlantının işleneceği **Click** olayında belirlenir.

## LinkLabel Özellikleri

Özellik	Değer Tipi	Açıklama
<b>LinkArea</b>	<b>LinkArea</b>	Bağlantının hangi karakterler arasında aktif olacağını belirler
<b>LinkBehavior</b>	<b>LinkBehavior</b>	Bağlantının yazısında bulunan çizginin ne zaman gösterileceğini belirler. <b>HoverUnderline</b> değeri fare üzerinde durduğu zaman, <b>AlwaysUnderline</b> değeri her zaman altı çizili olduğunu belirler. <b>NeverUnderline</b> değeri ise bağlantı yazısının altının çizilmeyeceğini belirler.
<b>LinkColor</b>	<b>Color</b>	Bağlantının <b>LinkVisited</b> özelliği <b>False</b> olduğu zaman gösterilecek rengini belirler
<b>LinkVisited</b>	<b>Boolean</b>	Bağlantının en az bir kere tıklandığını belirler
<b>VisitedLinkColor</b>	<b>Color</b>	Bağlantının <b>LinkVisited</b> özelliği <b>True</b> olduğu zaman gösterilecek rengini belirler
<b>Links</b>	<b>LinkLabel.LinkCollection</b>	Kontrolün <b>Text</b> özelliğinde bulunan bağlantıları tutar

## LinkLabel Olayları

Olay	Açıklama
<b>Click</b>	Kontrolün üzerine

	tıklandığı zaman gerçekleşir. Diğer kontrollerin tıklama olayından farklı olarak, <b>LinkLabel1</b> üzerinde hangi bağlantıya basıldığı anlaşılır.
--	--

Örnek: İletişim bilgi formunda e-posta ve internet adresleri gibi bağlantıları göstermek için **LinkLabel1** kullanılır.

```
private void Form1_Load( System.Object sender,
System.EventArgs e ) {
    string bilgi;

    bilgi = "BilgeAdam web sitesi:  

http://www.bilgeadam.com" + Constants.vbCrLf;
    bilgi += Constants.vbCrLf + "Mail ile ılamak iin
tklayn" + Constants.vbCrLf;
    LinkLabel1.Text = bilgi;

    // İnternet adresinin başladığı karakterden
    // itibaren link eklenir
    LinkLabel1.Links.Add( 22, 24,
"http://www.bilgeadam.com" );

    // Mail adresinin başladığı karakterden
    // itibaren link eklenir
    LinkLabel1.Links.Add( 72, 8,
"mailto:postakutusu@bilgeadam.com" );
}

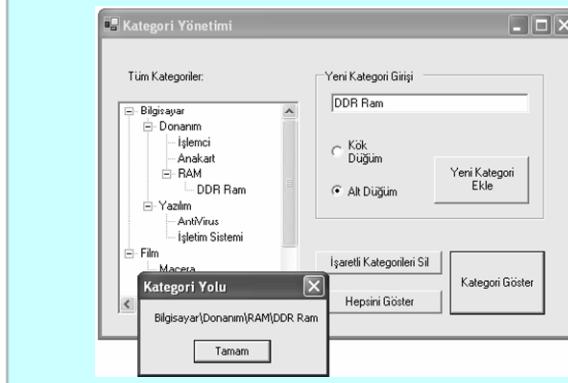
private void LinkLabel1_LinkClicked( System.Object
sender, System.Windows.Forms.LinkLabelLinkClickedEventArgs e )
{
    int tiklanan;
    tiklanan = LinkLabel1.Links.IndexOf( e.Link );
    //Tıklanan Linkin ziyaret edildiği belirtilir
    LinkLabel1.Links( tiklanan ).Visited = true;

    // Linki çalıştırmak için ilgili işlem gerçekleştirilir
```

```
System.Diagnostics.Process.Start( e.Link.LinkData );  
}
```

## TreeView

- Öğelerin hiyerarşik yapıda görüntülenmesini sağlar.
- TreeNode nesnelerinden oluşur.



Bu kontrol, içinde bulunan öğeleri hiyerarşik bir yapıda görüntüler. Her eklenen öğe bir düğümü temsil eder. Düğümler birleşerek ağaç yapısını oluştururlar. Düğümler kök ve alt düğüm olarak ikiye ayrılır. Kök düğümler, kontrolün ilk sırasında yer alır ve aynı seviyededir. Alt düğümler, kök düğümlerin ve diğer alt düğümlerin altına eklenebilir.

### TreeNode nesnesi

**TreeView** kontrolünde gösterilen öğeler, özelliklerini **TreeNode** sınıfından alır. Kök ve alt düğümlerin tümü **TreeNode** tipindedir. Her düğümün bir **Nodes** özelliği vardır. Bu özellik, düğümün, alt düğümlerinin tutulduğu koleksiyondur. Alt düğümler oluşturulup bu özelliğe eklenebilir.

**TreeNode** düğümleri oluşturulup, özellikleri atandıktan sonra **TreeView** kontrolünde gösterilmesi için, **TreeView** nesnesinin **Nodes** koleksiyonuna eklenmesi gereklidir.

### TreeNode Özellikleri

Özellik	Değer Tipi	Açıklama
<b>Text</b>	<b>String</b>	Düğümden üstünde gösterilen yazıyı belirler

<b>Nodes</b>	<b>TreeNodeCollection</b>	Düğümün alt düğümlerini tutan koleksiyondur
<b>Checked</b>	<b>Boolean</b>	<b>TreeView</b> kontrolünde seçim kutuları gösteriliyorsa, düğümün işaretli olup olmadığını belirler
<b>NextNode</b>	<b>TreeNode</b>	Aynı seviyedeki bir sonraki düğümü gösterir
<b>PrevNode</b>	<b>TreeNode</b>	Aynı seviyedeki bir önceki düğümü gösterir
<b>LastNode</b>	<b>TreeNode</b>	Alt düğümlerinin en sonuncusunu gösterir
<b>FirstNode</b>	<b>TreeNode</b>	Alt düğümlerinin ilkini gösterir
<b>NodeFont</b>	<b>Font</b>	Düğümün yazı tipini belirler
<b>FullPath</b>	<b>String</b>	Düğümün, kökten kendisine kadar olan tüm düğümlerin <b>Text</b> özelliklerini sıralar
<b>Parent</b>	<b>TreeNode</b>	Düğümün ait olduğu <b>TreeNode</b> nesnesini belirtir

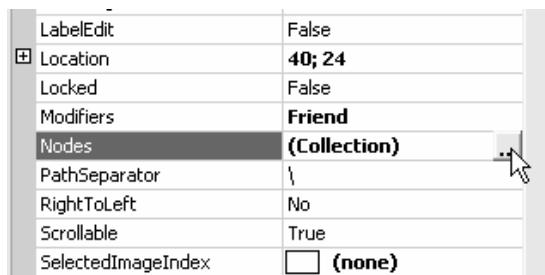
### TreeNode Metotları

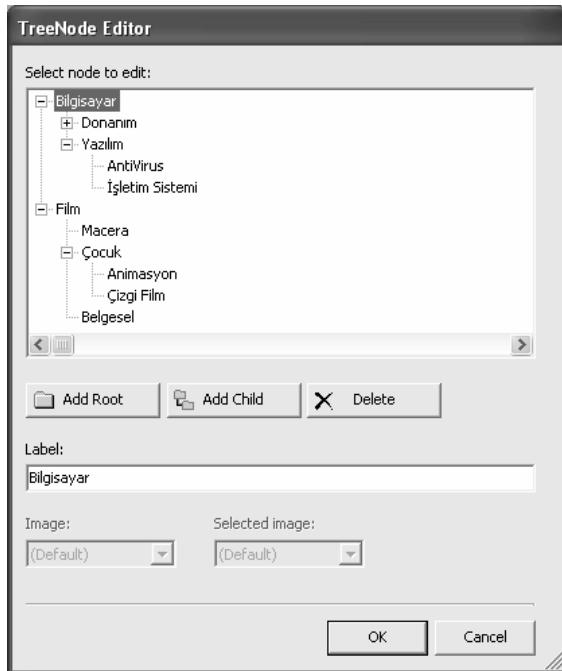
Metot	Açıklama
<b>Collapse</b>	Düğümün ilk seviyedeki alt düğümlerini gizler. Eksi işaretine basılması ile aynı görevi görür.
<b>Expand</b>	Düğümün ilk seviyedeki alt düğümlerini gösterir. Artı işaretine basılması ile aynı görevi görür.
<b>ExpandAll</b>	Düğümün alt düğümlerini son seviyeye kadar gösterir.
<b>Toggle</b>	Düğümün durumu açıksa kapalı, kapalıysa açık duruma getirir
<b>GetNodeCount</b>	Verilen parametre True ise tüm alt düğümlerin, False ise sadece ilk seviyedeki düğümlerin sayısını verir.

## TreeView Özellikleri

Özellik	Değer Tipi	Açıklama
<b>Checkboxes</b>	<b>Boolean</b>	Düğümlerin yanında işaret kutularının gösterilmesini belirler
<b>ImageIndex</b>	<b>Integer</b>	Kontrolün tüm öğeleri için varsayılan resmin, <b>ImageList</b> içindeki indisini belirler. Bu özelliğin kullanılması için, kontrolün <b>ImageList</b> özelliğinin belirlenmesi gereklidir.
<b>SelectedImageIndex</b>	<b>Integer</b>	Öğenin üzerine gelip seçildiğinde gösterilecek resmin, <b>ImageList</b> içindeki indisini belirler
<b>SelectedNode</b>	<b>TreeNode</b>	Seçilen düğümü belirler
<b>TopNode</b>	<b>TreeNode</b>	Kontrolün ilk kök düğümünü gösterir
<b>ShowLines</b>	<b>Boolean</b>	Düğümler arasında çizgilerin gözükmemesini belirler
<b>ShowPlusMinus</b>	<b>Boolean</b>	Alt düğümleri gösterip gizlemek için kullanılan artı ve eksi işaretlerinin gözükmesini belirler
<b>ShowRootLines</b>	<b>Boolean</b>	Kök düğümlerinin çizgilerinin ve artı eksi işaretlerinin gözükmesini belirler
<b>PathSeparator</b>	<b>String</b>	Bir düğümün <b>FullPath</b> özelliğinde gösterilen düğümleri ayıran karakterleri belirler

**TreeView** kontrolüne kod ile düğüm eklenebildiği gibi, tasarım anında Visual Studio TreeNode Editor penceresini kullanarak da düğüm eklenebilir.





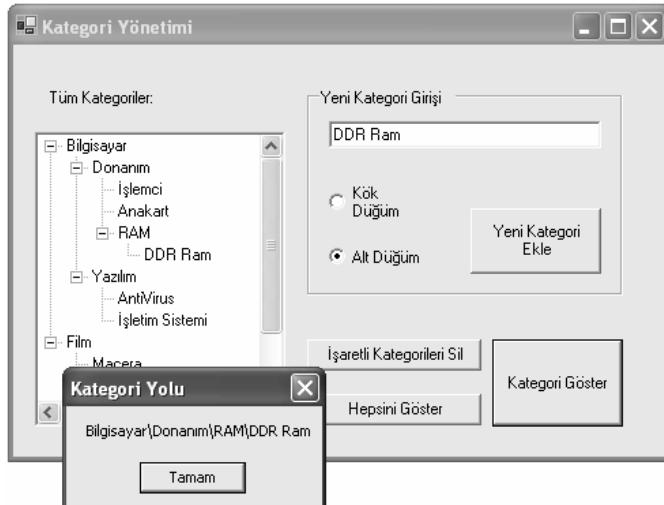
## TreeView Metotları

Metot	Açıklama
<b>CollapseAll</b>	Kontrolün tüm düğümlerini gizler
<b>ExpandAll</b>	Kontrolün tüm düğümlerini gösterir

## TreeView Olayları

Olay	Açıklama
<b>BeforeSelect</b>	Düğüm seçilmeden önce gerçekleşir
<b>AfterSelect</b>	Düğüm seçildikten sonra gerçekleşir
<b>BeforeCollapse</b>	Düğüm kapanmadan önce gerçekleşir
<b>AfterCollapse</b>	Düğüm kapanıktan sonra gerçekleşir
<b>BeforeExpand</b>	Düğüm açılmadan önce gerçekleşir
<b>AfterExpand</b>	Düğüm açıldıktan sonra gerçekleşir

Örnek: Ürün kategorileri, genelde tek kategori olarak ele alınsa da, aslında hiyerarşik bir yapıda incelenmeleri gereklidir. Her kategorinin sonsuz sayıda alt kategorisi olabilir. Bu tip kategoriler, en iyi şekilde **TreeView** kontrolü ile görüntülenebilir.



- Yeni kategori ekleme işlemi kök düğüm ve alt düğüm olarak yapılabilir. Eğer **RadioButton** kontrollerinde kök düğüm seçilmişse ana kategori; alt düğüm seçilmişse, seçilen kategorinin altına bir alt kategori eklenir.

```
private void btnYeniKategoriEkle_Click(
System.Object sender, System.EventArgs e ) {
    TreeNode secilen;
    secilen = TreeView1.SelectedNode;

    if ( RadioButton1.Checked ) {
        // Kök düğüm eklenir
        TreeView1.Nodes.Add( txtYenikategori.Text );

    }
    else if ( RadioButton2.Checked ) {
        // Seçilen kategoriye alt kategori eklenir
        secilen.Nodes.Add( txtYenikategori.Text );
    }
}
```

- Seçilen bir kategorinin silinme işlemi için, o düğümün hangi ana düğüme ait olduğu bulunmalıdır.

```
private void btnSil_Click( System.Object sender,
System.EventArgs e ) {
    TreeNode secilen = TreeView1.SelectedNode;

    if ( !( secilen.Parent == null ) ) {           // 
        // Seçilen düğüm, Parent düğümünün Nodes
        // koleksiyonundan çıkarılır.
        secilen.Parent.Nodes.Remove( secilen );
    }
    else {
        // Eğer Parent yok ise Kök düğümdür.
        Treeview1.Nodes.Remove( secilen );
    }
}
```

- ```

        }
    }

    • Tüm düğümlerin gösterilmesi ve seçilen düğümün hiyerarşik yapısının
      gösterilmesi

```

```

private void btnGoster_Click( System.Object sender,
System.EventArgs e ) {
    TreeView1.ExpandAll();
}

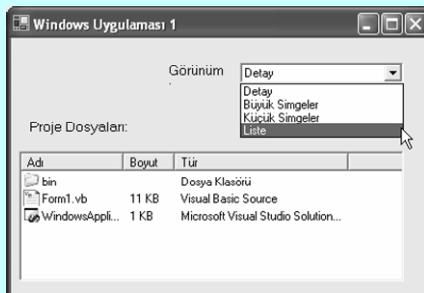
private void btnKategoriGoster_Click( System.Object
sender, System.EventArgs e ) {
    TreeNode secilen = TreeView1.SelectedNode;

    MessageBox.Show ( secilen.FullPath,
MsgBoxStyle.OKOnly, "Kategori Yolu" );
}

```

## ListView

- ListView**
- Öğelerin değişik şekillerde listelenmesini sağlar.
  - ListViewItem nesnelerinden oluşur.
  - Her öğe, ListViewSubItem alt öğelerinden oluşur.



Kullanıcıya değişik listeleme seçenekleri sunan bir kontroldür. İçinde bulunan öğeler, tek bir nesne olarak veya detayları ile gösterilebilir. Dolayısıyla öğeler **ListViewItem** nesnesi, detayları ise **ListViewSubItem** nesnesi olarak tanımlanır.

## ListView Özellikleri

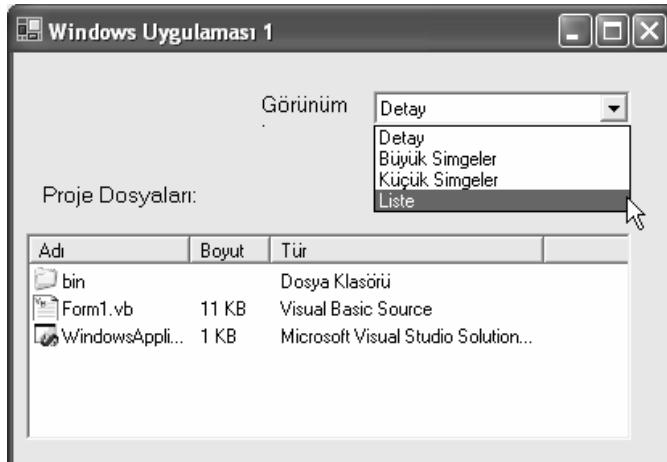
| Özellik                   | Değer Tipi                    | Açıklama                                                                                                                                                                                                                                                                                                             |
|---------------------------|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>View</b>               | <b>View</b>                   | Listenin görünümünü belirler. <b>LargeIcons</b> değeri listedeki öğelerin büyük resimle, <b>SmallIcons</b> küçük resimle görünmesini sağlar. <b>List</b> değeri, öğeleri küçük resimle fakat alt alta görünmesini sağlar. <b>Details</b> değeri, alt öğelerin kolonlar altında görüntülendiği detay görünümü sağlar. |
| <b>AllowColumnReorder</b> | <b>Boolean</b>                | Detay görünümünde, kolonların kullanıcı tarafından düzenlenebilmesini belirler                                                                                                                                                                                                                                       |
| <b>Activation</b>         | <b>ItemActivation</b>         | Öğelerin ne zaman etkinleştirileceğini belirler. <b>OneClick</b> değeri, ögenin tek tıklamaya, <b>Standard</b> değeri, ögenin çift tıklamaya aktif hale geleceğini belirler. <b>TwoClick</b> değeri seçili iken, ilk tıklandığında öge seçilir, daha sonra ikinci defa tıklandığında ise öge aktif hale gelir.       |
| <b>Checkboxes</b>         | <b>Boolean</b>                | Öğelerin yanında seçme kutularının bulunmasını belirler                                                                                                                                                                                                                                                              |
| <b>Columns</b>            | <b>ColumnHeaderCollection</b> | Detay görünümünde iken, öğelerin alt öğelerinin gösterileceği kolonları tutan koleksiyondur                                                                                                                                                                                                                          |
| <b>FußRowSelect</b>       | <b>Boolean</b>                | Detay görünümde,                                                                                                                                                                                                                                                                                                     |

|                  |                |                                                                                                                                                                              |
|------------------|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                  |                | ögenin tüm detay satırının seçilmesini belirler                                                                                                                              |
| <b>GridLines</b> | <b>BooTean</b> | Kolonlar ve satırlar arasında ayırcı çizgilerin gözükmesini belirler                                                                                                         |
| <b>LabelEdit</b> | <b>BooLead</b> | Çalışma anında, kullanıcın, liste öğelerinin yazısını değiştirmesini belirler. Bu özelliğin kullanılması için, <b>Activation</b> özelliği <b>Standard</b> olması gereklidir. |

## ListView Olayları

| Olay                   | Açıklama                                   |
|------------------------|--------------------------------------------|
| <b>AfterLabelEdit</b>  | Ögenin yazısı değişikten sonra gerçekleşir |
| <b>BeforeLabelEdit</b> | Ögenin değişmeden önce gerçekleşir         |

Örnek: Windows Explorer ile dosya görünümleri **ListView** ile gerçekleştirilir.



- Form yüklenirken **ListView** kontrolüne kolon ve öğeler eklenir. Ayrıca **ComboBox** kontrolüne görünüm seçenekleri eklenir.

```
private void Form1_Load( System.Object sender,
System.EventArgs e ) {
```

```

        ComboBox1.Items.Add( "Detay" );
        ComboBox1.Items.Add( "Büyük Simgeler" );
        ComboBox1.Items.Add( "Küçük Simgeler" );
        ComboBox1.Items.Add( "Liste" );
        ComboBox1.DropDownStyle = ComboBoxStyle.DropDownList;

        ListView1.Columns.Add( "Ad", 100,
HorizontalAlignment.Left );
        ListView1.Columns.Add( "Boyut", 50,
HorizontalAlignment.Left );
        ListView1.Columns.Add( "Tür", 170,
HorizontalAlignment.Left );

        ListView1.View = View.Details;

        ListViewItem oge = new ListViewItem( "bin" );
        oge.SubItems.Add( "" );
        oge.SubItems.Add( "Dosya klasör" );
        oge.ImageIndex = 0;
        ListView1.Items.Add( oge );

        oge = new ListViewItem( "Form1.vb" );
        oge.SubItems.Add( "11 KB" );
        oge.SubItems.Add( "Visual C# Source" );
        oge.ImageIndex = 2;
        ListView1.Items.Add( oge );

        oge = new ListViewItem( "windowsApplication1.sln" );
        oge.SubItems.Add( "1 KB" );
        oge.SubItems.Add( "Microsoft Visual Studio
Solution Object" );
        oge.ImageIndex = 1;
        ListView1.Items.Add( oge );
    }

```

**ComboBox** kontrolünde seçilen değer değiştiği zaman, **ListView** görünümü değişir.

```

private void ComboBox1_SelectedIndexChanged(
System.Object sender, System.EventArgs e ) {
    switch ( ComboBox1.SelectedIndex ) {
        case 0:
            ListView1.View = View.Details;
            break;
        case 1:
            ListView1.View = View.LargeIcon;
            break;
        case 2:
            ListView1.View = View.SmallIcon;
            break;
        case 3:
            ListView1.View = View.List;
            break;
    }
}

```

{}

## Dinamik Kontroller

s

### Dinamik Kontroller

- Çalışma anında oluşturulup forma eklenir.
- AddHandler ile kontrolün olaylarına erişilir.

```
Void Yordam1()
{
    Button b = new Button();
    b.Click += new EventHandler(ButonaBasildi);
}

Private Void ButonaBasildi(Object sender ,EventArgs e)
{
}
```

Kontroller tasarım anında eklenip ayarlanıldığı gibi, çalışma anında da oluşturulup forma eklenebilir. Kontrollerin, **Properties** panelinde gözüken tüm özelliklere kod tarafında ulaşılabilen için çalışma anında önceden oluşturulmuş bir kontrolün özelliği değiştirilebilir. Bununla birlikte, yeni bir form oluşturup gösterme işlemi gibi, çalışma anında yeni bir kontrol oluşturup, özellikleri atanıp form üzerinde gösterilebilir.

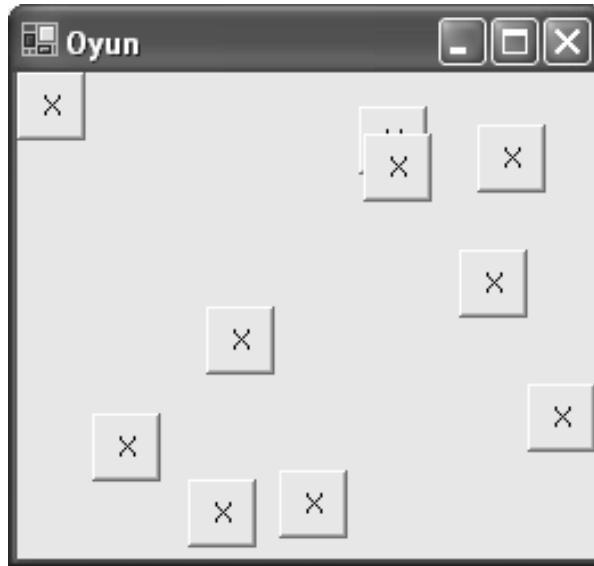
Yeni eklenen kontrollerin olaylarına erişmek için **EventHandler** nesnesi kullanılır. Olay gerçekleştiği zaman çalıştırılacak kodların bulunduğu yordam ise **EventHandler** nesnesi oluşturulurken, parametre olarak geçilmelidir.

```
button1.Click+=new EventHandler(button1_Click);
```

Bu şekilde tanımlanan yordamların, kontrolün olay tanımlayıcısı ile aynı parametrelere sahip olmalıdır.

```
private void button1_Click(object sender, EventArgs e)
{
}
```

Örnek: Form üzerinde sürekli düğme eklenen ve düğmelerin, basıldığı zaman yok edildiği bir oyun yapılması için, bu düğmelerin dinamik bir şekilde oluşturulması gereklidir.



- Form üzerindeki bir **Timer** kontrolü, iki saniyede bir düğme oluşturup forma ekler.

```
private void Timer1_Tick( System.Object sender,
System.EventArgs e ) {
    // Yeni bir düğme oluşturulur.
    Button b = new Button();
    b.Height = 30;
    b.Width = 30;
    b.Text = "X";

    int maxLocation_Y, maxLocation_X;

    // Yeni düğmenin yeri form dışında bir yerde olamaz
    maxLocation_X = this.Width - b.Width;
    maxLocation_Y = this.Height - b.Height;

    Random r = new Random();

    // Düğmenin bulunacağı yer rasgele ayarlanır.
    b.Location = new Point( r.Next( maxLocation_X ),
    r.Next( maxLocation_Y ) );

    b.Click +=new EventHandler(ButonaBasildi );

    // Oluşturulan kontrol, Formun kontroller
    // listesine eklenmelidir.
    this.Controls.Add( b );
}
```

- Oluşturulan kontrollere tıklandığı zaman çalıştırılacak yordam yazılır.

```
private void ButonaBasildi( System.Object sender,
System.EventArgs e ) {
    // Kontrolün, üzerine basıldığı zaman yok edilmesi
    sender.Dispose();
}
```

- Form yükleniği zaman **Timer** nesnesi çalışmaya başlar

```
private void Form1_Load( System.Object sender,
System.EventArgs e ) {
    Timer1.Interval = 500;
    Timer1.Start();
}
```

## Lab 1: Internet Tarayıcısı

Bu labda, Windows altında bulunan Microsoft Web Tarayıcısı kontrolünü projeye ekleyerek Internet tarayıcısı gerçekleştirilecektir.

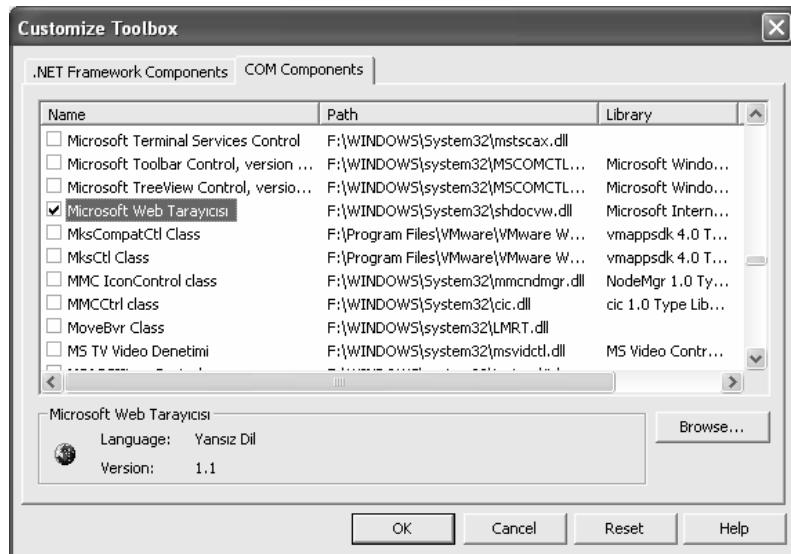
Bu labda kullanılan kontroller ve teknikler:

- **LinkLabel** – Ana sayfaya bağlantı sağlar.
- **RadioButton** – Bağlantıların yeni ya da aynı pencerede açılması seçeneğini sunar.
- **GroupBox** – **RadioButton** kontrollerini grüplamak için kullanılır.
- **TabControl** – Tarayıcıların farklı pencerelerde gözükmekini sağlar.
- Microsoft Web Tarayıcısı – Internet sitelerini görüntülenmesini sağlar.
- **Dispose** Metodu – **TabPage** sayfalarının silinmesi için kullanılır.
- **foreach** – Sayfaların tümünün kapanması için kullanılır.
- Dinamik Kontroller – Bağlantılar yeni bir sayfada açıldığı zaman, yeni bir **TabPage** oluşturulur. Bu sayfanın içine yeni bir tarayıcı kontrolü oluşturulup eklenir. Daha sonra bu sayfa **TabControl** nesnesine eklenir.

## Kontrollerin eklenmesi

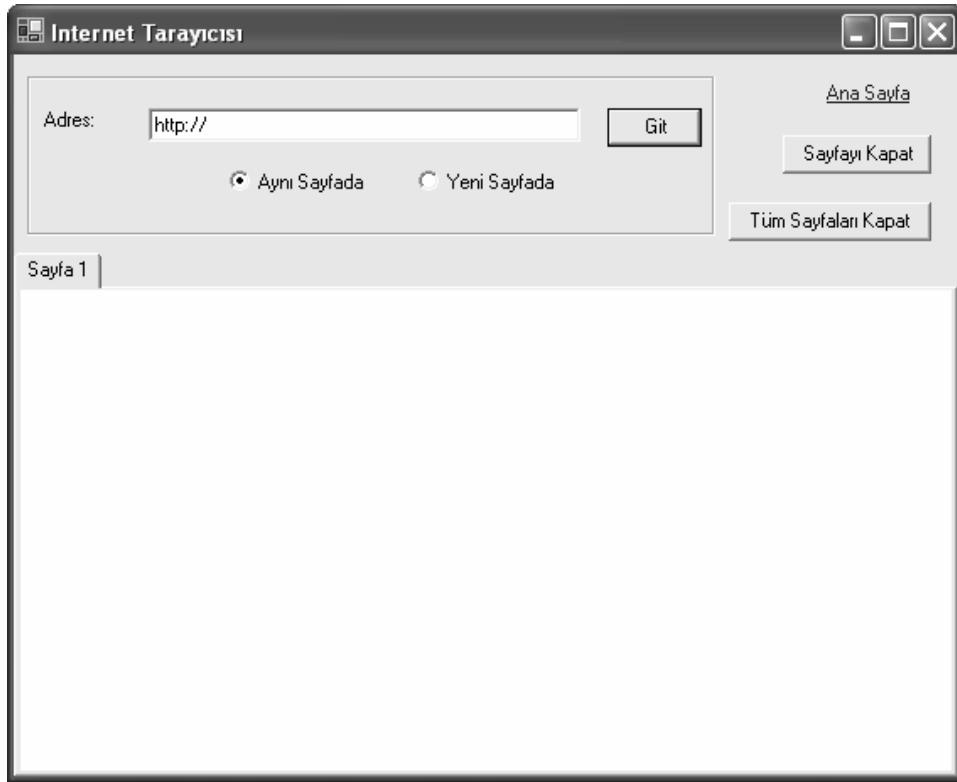
Yeni bir Windows projesi açın ve **ToolBox** paneline Microsoft Web Tarayıcısını ekleyin.

**Not:** ToolBox paneline kontrol ekleme işlemleri için Modül 3 e bakın



Form üzerine tablodaki kontrolleri ekleyin belirtilen özelliklerini ayarlayın.

| Kontrol – Kontrol İsmi           | Özellik         | Değer                  |
|----------------------------------|-----------------|------------------------|
| <b>TextBox</b> – txtAdres        | <b>Text</b>     | <b>http://</b>         |
| <b>RadioButton</b> - rbAyniSayfa | <b>Checked</b>  | <b>True</b>            |
| <b>RadioButton</b> - rbYeniSayfa |                 |                        |
| <b>GroupBox</b> – GroupBox1      | <b>Text</b>     |                        |
| <b>LinkLabel</b> – LinkLabel1    | <b>Text</b>     | Ana Sayfa              |
| <b>Button</b> – btnSayfaKapat    | <b>Text</b>     | Sayfayı Kapat          |
| <b>Button</b> – btnTumunuKapat   | <b>Text</b>     | Tüm Sayfaları Kapat    |
| <b>TabControl</b> – TabControl1  | <b>TabPages</b> | Yeni bir sayfa ekleyin |
|                                  | <b>Dock</b>     | <b>Bottom</b>          |
| <b>TabPage</b> – TabPage1        | <b>Text</b>     | Sayfa 1                |
| Tarayıcı – AxWebBrowser1         | <b>Dock</b>     | <b>F11</b>             |



## Kodların yazılması

1. Form yüklenirken **LinkLabel** kontrolünün göstereceği bağlantıyı ve formun **AcceptButton** özelliğini ayarlayın.

```
private void Form1_Load( System.Object sender,
System.EventArgs e ) {
    LinkLabel1.Links.Add( 0, 9,
    "http://www.bilgeadam.com" );
    this.AcceptButton = btnGit;
}
```

2. Yazılan Internet adresine gitmek için kullanıcı, aynı sayfayı veya yeni açılacak bir sayfayı kullanabilir. Seçilen duruma göre aynı sayfada ya da farklı sayfada Internet sitesini görüntüleyen kodları yazın.

```
private void btnGit_Click( System.Object sender,
System.EventArgs e ) {
    // Girilen bağlantının başında http ifadesi
    // bulunmuyorsa bu ifade eklenir
    if ( !( txtAdres.Text.StartsWith( "http://" ) )
) {
        txtAdres.Text = txtAdres.Text.Insert( 0,
"http://" );
    }

    // TabControl nesnesinde sayfa yoksa ya da Yeni Sayfa
    // seçeneği seçilmişse, adres yeni sayfada gösterilir.
    if ( rbYenisayfa.Checked ||
TabControl1.TabPages.Count == 0 ) {
```

```

        Yenisayfa( txtAdres.Text );
    }
    else {
        Aynisayfa( txtAdres.Text );
    }

    public void Yenisayfa( string link ) {
        // Dinamik kontroller oluşturulur.
       TabPage sayfa = new TabPage( link );
        AxSHDocVw.AxWebBrowser tarayici = new
AxSHDocVw.AxWebBrowser();
        tarayici.Dock = DockStyle.Fill;

        // Tarayıcı TabPage kontrolüne eklenir
        sayfa.Controls.Add( tarayici );

        // Oluşturulan sayfa TabControl nesnesine eklenir.
        TabControl1.TabPages.Add(sayfa);

        // Yeni açılan sayfa seçili olarak gösterilir
        TabControl1.SelectedTab = sayfa;

        // Tarayıcı, verilen bağlantıyı görüntüler
        tarayici.Navigate(link);
    }

    public void Aynisayfa( string link ) {
        // Internet sitesi, seçilen sayfada gösterilir.
       TabPage sayfa = null;
        sayfa = TabControl1.SelectedTab;

        AxSHDocVw.AxWebBrowser tarayici = null;
        // Tarayıcı, sayfanın kontrolleri içinde bulunur.
        // Sayfada başka kontrol bulunmadığı için, ilk
        // kontrol tarayıcıdır.
        tarayici = ( ( AxSHDocVw.AxWebBrowser )(
sayfa.Controls[ 0 ] ) );

        sayfa.Text = link;
        tarayici.Navigate( link );
    }
}

```

3. Ana sayfa bağlantısına tıklandığı zaman, BilgeAdam internet sitesinin yeni bir sayfada açılmasını sağlayan kodları yazın.

```

private void LinkLabel1_LinkClicked( System.Object
sender, System.Windows.Forms.LinkLabelLinkClickedEventArgs e
) {
    Yenisayfa( System.Convert.ToString(
e.Link.LinkData ) );
}

```

4. Seçilen sayfayı ve tüm sayfaları kapatın kodları yazın.

```

private void btnSayfaKapat_Click( System.Object
sender, System.EventArgs e ) {
   TabPage sayfa = null;
    sayfa = TabControl1.SelectedTab;

    if ( !( sayfa == null ) ) {

```

```

        sayfa.Dispose();
    }

    private void btnTumunuKapat_Click( System.Object
sender, System.EventArgs e ) {
    foreach ( System.Windows.Forms.TabPage sayfa in
TabControl1.TabPages ) {
        sayfa.Dispose();
    }
}

```

## Lab 2: 4 Haneli Sayı Bulma Oyunu

Bu labda, MasterMind oyunundan uyarlanmış 4 haneli sayı bulma oyunu programlanır. Oyunun işleyişi rakamları farklı ya da aynı olarak tutulan 4 haneli sayının tahmin edilmesidir. Tahmin edilen sayıyla ilgili ipucular verilir. Yerini tutan rakamlar için + ile, rakamları tutmayan ancak sayı içinde geçen rakamlar – ile belirtilir.

Örnek:

Tutulan sayı: 1980

Tahmin 1: 4952

İpucu: +1 (Sadece 9 rakamı yerini tuttu)

Tahmin 2: 9820

İpucu: +1 -2 (0 yerini tuttu, 9 ve 8 bulundu ancak yeri tutturulmadı)

Bu labda kullanılan kontroller ve teknikler:

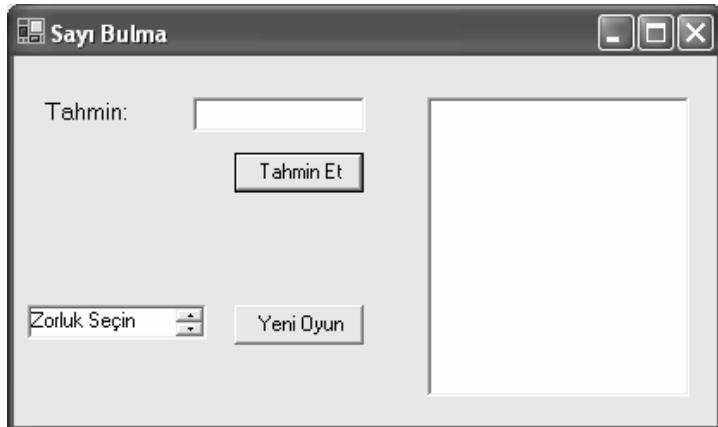
- **ListBox** – Yapılan tahminleri tutmayı sağlar
- **DomainUpDown** – Oyunun zorluk derecesinin seçilmesini sağlar
- **ErrorProvider** – Kullanıcının, tahminleri düzgün formatta girip girmedğini kontrol eder.
- **İç İçe Döngüler** – Farklı rakamları olan sayılar üretmek ve tahminleri kontrol etmek için kullanılır.

## Kontrollerin eklenmesi

Form üzerine tablodaki kontrolleri ekleyin belirtilen özelliklerini ayarlayın.

| Kontrol – Kontrol İsmi              | Özellik      | Değer                              |
|-------------------------------------|--------------|------------------------------------|
| <b>TextBox</b> – txtTahmin          |              |                                    |
| <b>ListBox</b> – ListBox1           |              |                                    |
| <b>DomainUpDown</b> – DomainUpDown1 | <b>Items</b> | Farklı Sayılar<br>Tekrarlı Sayılar |

|                      | Text | Zorluk Seçin |
|----------------------|------|--------------|
| Button – btnTahminEt | Text | Tahmin Et    |
| Button – btnYeniOyun | Text | Yeni Oyun    |
| Label – lblMesaj     |      |              |



## Kodların Yazılması

Sistem tarafından tutulacak sayılar, **DomainUpDown** kontrolünde yapılan seçime göre farklı ya da aynı rakamlara sahip olacaktır.

```
private int Bulunacaksayı;
public int sayiuret() {
    int sayı = DörtHaneliSayı();
    // Sayıdaki rakamlar tekrar edilebilirse
    if ( DomainUpDown1.SelectedIndex == 1 ) {
        return sayı;
    }
    // Sayının rakamları birbirinden farklı
    // olana kadar sayı üretir
    while ( !( SayıKontrol( sayı ) ) ) {
        sayı = DörtHaneliSayı();
    }
    return sayı;
}
```

- Rakamları birbirinden farklı dört haneli sayı üretir.

```
public int DörtHaneliSayı() {
    Random r = new Random();
    int sayı = r.Next(10000);
    // Sayı 4 haneli olana kadar tekrar üretir
    while ( sayı < 1000 ) {
```

```

        sayı = r.Next(10000);
    }

    return sayı;
}

```

- Sayının rakamlarının birbirinden farklı olmasını kontrol eder.

```

public bool SayiKontrol( int sayı ) {
    char[] rakamlar =
sayi.ToString().ToCharArray();

edilir      // Rakamlar tek tek bir birleriyle kontrol
            // Tekrarlanan rakam varsa False döner
for (int i=0; i<=rakamlar.Length - 2; i++ ) {
    for (int j=i + 1; j<=rakamlar.Length - 1;
j++ ) {
        if ( rakamlar[ i ] == rakamlar[ j ] ) {
            return false;
        }
    }
}
return true;
}

```

- Yeni oyun düğmesine tıklandığı zaman sayı üretilir ve oyun başlar

```

private void btnYeniOyun_Click( System.Object
sender, System.EventArgs e ) {
    BulunacakSayı = SayiUret();
    lblMesaj.Text = "Yeni oyun! Sayı üretildi...";
}

```

- Metin kutusunun **Validating** olayında, girilen değerler kontrol edilir.

```

private void txtTahmin_Validating( object sender,
System.ComponentModel.CancelEventArgs e ) {
    if ( txtTahmin.Text.Length == 4 ) {
        ErrorProvider1.SetError( txtTahmin, "" );
    }
    else {
        ErrorProvider1.SetError( txtTahmin, "Sayı
yanlış girildi" );
        e.Cancel = true;
    }
}

```

- Tahmin edilen sayının hangi rakamlarının tutuğu kontrol edilir

```

public string TahminKontrol( int sayı ) {
    string sonuc ="";

// Sonuç kümesindeki artı ve eksı sayıları
byte artı = 0;
byte eksı = 0;
byte i, j;

char[] sdizi;
sdizi = sayı.ToString().ToCharArray();

```

```

        char[] sBulunacak;
        sBulunacak =
BulunacakSayi.ToString().ToCharArray();

        // Yerleri tutan sayılar bulunur
        for ( i=0; i<=3; i++ ) {
            if ( sdizi[ i ] == sBulunacak[ i ] ) {
                arti += 1;
            }
        }

        // Yerleri tutmayan sayıların kontrol
        for ( i=0; i<=3; i++ ) {
            for ( j=0; j<=3; j++ ) {
                if ( i != j ) {
                    if ( sdizi[ i ] == sBulunacak[ j ] )
{
                        eksi += 1;
                        break;
                    }
                }
            }
        }

        if ( arti == 0 & eksi == 0 ) {
            sonuc = "0";
        }
        else if ( arti == 4 ) {
            sonuc = "Tebrikler!";
        }
        else if ( arti != 0 && eksi != 0 ) {
            sonuc = "+" + arti + " - " + eksi;
        }
        else if ( arti == 0 ) {
            sonuc = "- " + eksi;
        }

        return sonuc;
    }

    private void btnTahmin_Click( System.Object sender,
System.EventArgs e ) {
    ListBox1.Items.Add( TahminKontrol(
int.Parse(txtTahmin.Text) ) );
}

```

### Lab 3: Hafıza Oyunu

Hafıza oyunu, belli sayıda kart içinden aynı resme sahip olanların bulunması ile gerçekleştiriliyor. Bu labda, form üzerine, seçilen seviye kadar kart ekleme işlemi yapılır. Kontroller çalışma anında ekleneceği için dinamik olarak oluşturulmalıdır.

Bu labda kullanılan kontroller ve teknikler:

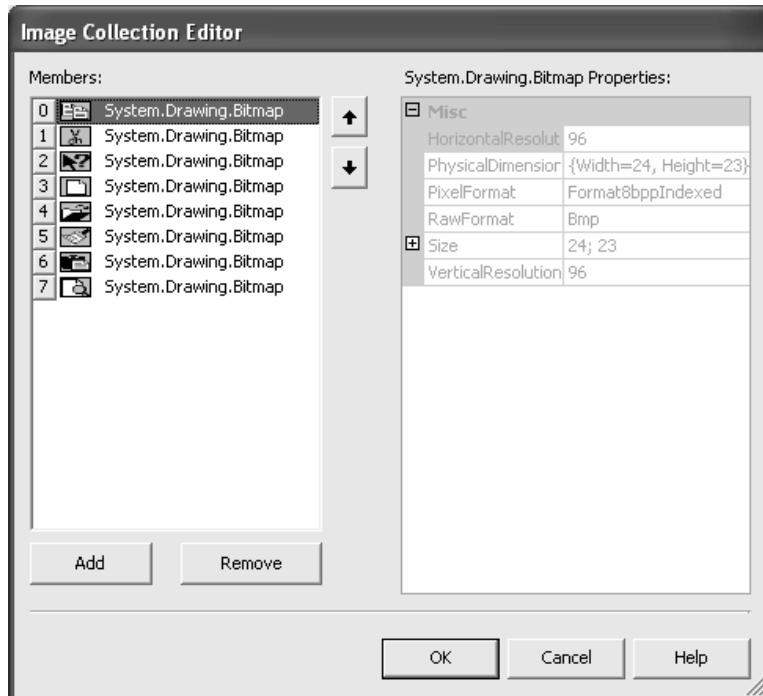
- **ComboBox** – Seviyenin seçilmesi için kullanılır
- **ImageList** – Eklenen kartların resimlerini tutar

- Dinamik kontroller – Kullanıcının seçtiği seviye kadar kart ekleme işlemi için kullanılır.
- **Tag** – Kontrollerin **Tag** özelliği, o kontrole ait bilgi tutmak için kullanılır. Bu labda, yeni eklenen kartların hangi resmi taşıyacağı kontrolü **Tag** özelliğinde tutulur.

## Kontrollerin eklenmesi

Form üzerine tablodaki kontrolleri ekleyin belirtilen özelliklerini ayarlayın.

| Kontrol – Kontrol İsmi        | Özellik       | Değer                       |
|-------------------------------|---------------|-----------------------------|
| <b>ComboBox</b> – ComboBox1   | <b>Items</b>  | 4 Kart<br>8 Kart<br>16 Kart |
| <b>ImageList</b> – ImageList1 | <b>Images</b> | 8 tane resim ekleyin        |



## Kodların Yazılması

1. **ComboBox** kontrolünden seviye seçildiği zaman, form üzerinde var olan tüm düğmelerin silinip, seçilen seviye kadar düğme eklenmesi gerekir. Bu işlem oyunu baştan başlatır.

```
public void KartYerlestir( int kartSayisi ) {
    DugmeleriSil();
    int x = 10;
    int y = 50;
```

```

        for (int i=1; i<=kartSayisi; i++ ) {
            // Dinamik bir düğme oluşturulur ve
            // ayarlanır
            Button kart = new Button();
            kart.Height = 30;
            kart.Width = 30;
            kart.Location = new Point( x, y );

            kart.Click += new System.EventHandler(
ButonaTiklandi );
                this.Controls.Add( kart );
            }

            // Bir sonraki eklenecek olan düğme
            // ilk kontrolün 70 piksel sağında
            x += 70;

            // Düğme Form sınırları içinde olması
            if ( x > this.width ) {
                x = 10;
                y += 50;
            }
        }
    KartResimYukle();
}

```

2. Düğmeleri silme işlemi, form üzerindeki tüm düğmelerin bir listeye atılıp daha sonra formun kontrollerinden kaldırılarak yapılır.

```

public void Dugmelerisil() {
    ArrayList silinecek = new ArrayList();

    // Form iindeki Button kontrolleri bir listede
    // tutulur
    foreach ( System.Windows.Forms.Control c in
this.Controls ) {
        if ( c is Button ) {
            silinecek.Add( c );
        }
    }

    for (int i=0; i<=silinecek.Count - 1; i++ ) {
        this.Controls.Remove( (Control)silinecek[ i ]
] );
    }
}

```

3. Kartlara resim yüklerken, her resim iki karta yüklenmesi gereklidir.

```

public void KartResimYukle() {
    // Düğmeler bir listeye alınır.
    ArrayList dugmeler = new ArrayList();
    foreach ( System.Windows.Forms.Control c in
this.Controls ) {
        if ( c is Button ) {
            dugmeler.Add( c );
        }
    }
}

```

```

        }

Random r = new Random();
int i = 0;

da
    // Kartlar ikişer ikişer ele alınır. İki karta
    // aynı resim atanır. ve bu iki kart düğmeler
    // listesinden çıkartılır.
    while ( dugmeler.Count > 0 ) {
        Button kart1 = null, kart2 = null;

        kart1 = (Button)dugmeler[ r.Next(
dugmeler.Count - 1 ) ];
        kart1.Tag = i;
        dugmeler.Remove( kart1 );

        kart2 = (Button)dugmeler[ r.Next(
dugmeler.Count - 1 ) ];
        kart2.Tag = i;
        dugmeler.Remove( kart2 );

        i += 1;
    }
}

```

4. Eklenen kartlara tıklandığı zaman, ilk seferde bir kart açılır ve resmi gösterilir. İkinci kart açıldığı zaman bu iki kartın resmi aynıysa kart formdan kaldırılır.

```

private Button Acikkart;
private bool acik = false;

private void ButonaTiklandi( object sender,
EventArgs e ) {
    Button kart = (Button)sender;

    // Eğer ilk kart açılıyorsa
    if ( !( acik ) ) {
        // Kart görüntüle
        kart.BackgroundImage = ImageList1.Images[
int.Parse(kart.Tag.ToString()) ];
        Acikkart = kart;
        acik = true;

        // Eğer ikinci kart açılıyorsa
    }
    else {
        // Açılmış kartın resmi, yeni alan kartın
        // resmi ile aynıysa, bu kartlar silinir
        if ( kart.Tag == Acikkart.Tag ) {
            this.Controls.Remove( kart );
            this.Controls.Remove( Acikkart );
        }
        else {
            Acikkart.BackgroundImage = null;
        }
        acik = false;
    }
}

```



## Lab 4: Hesap Makinesi

Bu labda, bir hesap makinesinde kullanılan genel fonksiyonlar gerçekleştirilecektir.

Bu labda kullanılan kontroller ve teknikler:

- **Button** – Hesap makinesindeki her işlem ve sayı için bir düğme kullanılır
- **Try Catch Finally** – Hesaplamlar yapılırken, kullanıcın yanlış bir değer girmesi durumunda çıkacak hataları yakalamak için kullanılır.

## Kontrollerin eklenmesi

Form üzerine tablodaki kontrolleri ekleyin belirtilen özelliklerini ayarlayın.

| Kontrol – Kontrol İsmi                                                                                         | Özellik     | Değer                                   |
|----------------------------------------------------------------------------------------------------------------|-------------|-----------------------------------------|
| <b>Button</b> – 0 – 9 arası her sayı için Button(Sayı) isminde bir düğme ekleyin. Örnek: 5 sayısı için Button5 | <b>Text</b> | Temsil ettikleri sayılar                |
| <b>Button</b> – Her işlem için bir düğme ekleyin: Çarpma, bölme, toplama çıkarma, eşitlik, temizleme           | <b>Text</b> | Temsil ettikleri işlemleri. * + / - = C |



## Kodların Yazılması

- İşlemin türünü ve seçildiğini belirleyen, girilen bir önceki sayıyı tutan global değişkenleri yazın.

```
private bool IslemSecildi = false;  
private double Sayi;  
private string Islem;
```

- Sayı düğmelerinden herhangi birine basıldığı zaman, metin kutusunun görünümünü değiştiren işlemi yazın.

```
private void Button1_Click( System.Object sender,  
System.EventArgs e ) {  
    if ( !( IslemSecildi ) ) {  
        txtSayi.Text += ((Button)sender).Text;  
    }  
    else {  
        txtSayi.Text = ((Button)sender).Text;  
        IslemSecildi = false;  
    }  
}
```

- İşlem seçildiği zaman, bir önceki girilen sayıyı tutan kodları yazın.

```
private void btnCarp_Click( System.Object sender,  
System.EventArgs e ) {  
    Islem = ((Button)sender).Text;  
    try {  
        Sayi = double.Parse(txtSayi.Text);  
        IslemSecildi = true;  
    }  
    catch ( Exception ex ) {  
        MessageBox.Show( "Sayı düzgün formatta  
girilmedi");  
    }  
    finally {  
        txtSayi.Text = "";  
        txtSayi.Focus();  
    }  
}
```

- Eşittir düğmesine basıldığı zaman aritmetik operasyonu yapan kodları yazın.

```
private void btnEsit_Click( System.Object sender,  
System.EventArgs e ) {  
    switch ( Islem ) {  
        case "*":  
            Sayi *= double.Parse(txtSayi.Text);  
            break;  
        case "/":  
            Sayi /= double.Parse(txtSayi.Text);  
            break;  
        case "-":  
            Sayi -= double.Parse(txtSayi.Text);  
            break;
```

```

        case "+":
            Sayi += double.Parse(txtSayi.Text);
            break;
    }

    txtSayi.Text = Sayi.ToString();
}

```

5. C (temizle) tuşuna basıldığı zaman, metin kutusunu temizleyen ve global değişkenleri başlangıç değerlerine getiren kodları yazın.

```

private void btnTemizle_Click( System.Object sender,
System.EventArgs e ) {
    Sayi = 0;
    IslemSecildi = false;
    txtSayi.Text = "";
    txtSayi.Focus();
}

```

## Modül Sonu Soruları & Alıştırmalar

### Özet

- Listeleme Kontrolleri
  - ListBox, TreeView, ComboBox
- Resim Kontrolleri
  - PictureBox, ImageList
- Düzenleme Kontrolleri
  - TabControl, Panel, HScrollBar, VScrollBar
- Zaman ve Tarih Kontrolleri
  - DateTimePicker, MonthCalendar
- Dinamik Kontroller
  - Çalışma anında eklenen kontroller

1. Formun kapanmasını şeffaflığını yavaşça azaltarak sağlamak için, formun hangi olay, özellik ve metodlarından faydalanan? Uygulamasını yazın.
2. Fiziksel olarak bulundukları yerlerin bir dizide tutulduğu resimlerin, slayt gösterisi şeklinde gösterilmesi hangi kontroller ile sağlanır? Uygulamasını yazın.

3. Kurumsal bir şirketin elemanlarının bağlı oldukları departmanları ve müdürleri hiyerarşik olarak hangi kontrol ile gösterilebilir? Her müdür ve departman başka bir müdür ve departmana bağlıdır. Uygulamasını structure yapısını kullanarak ve ilgili kontroller ile birlikte yazın.
4. Microsoft Excel ile oluşturulan sayfalar, aynı pencerede tutulur. Bir Windows uygulamasında sınırsız sayıda sayfanın aynı form üzerinde tutmayı hangi kontrol sağlar? Bu sayfalar çalışma alanında oluşturulmak istenirse, kontrolün hangi özelliklerinden faydalанılır.

## Modül 10: Menü Tasarımı ve MDI Formlar

### Hedefler

- ↳ Menüler
  - ↳ MainMenu, ContextMenu
- ↳ ToolBar
- ↳ ToolTip
- ↳ StatusBar
- ↳ NotifyIcon
- ↳ RichTextBox

### Konu 1: Menü Tasarımı

Windows uygulamalarında en çok kullanılan tasarım araçları menülerdir. Dosya, düzen, görünüm gibi menüler neredeyse tüm Windows uygulamalarında, belli başlı işlerin yapılmasında kullanıcıya kolay erişim sağlar. Uygulamalarda, menülerde tanımlanan işlemlere görsel kısa yollar sunulur. Bu işlem araç kutuları ile sağlanır.

Bu bölüm tamamlandıktan sonra:

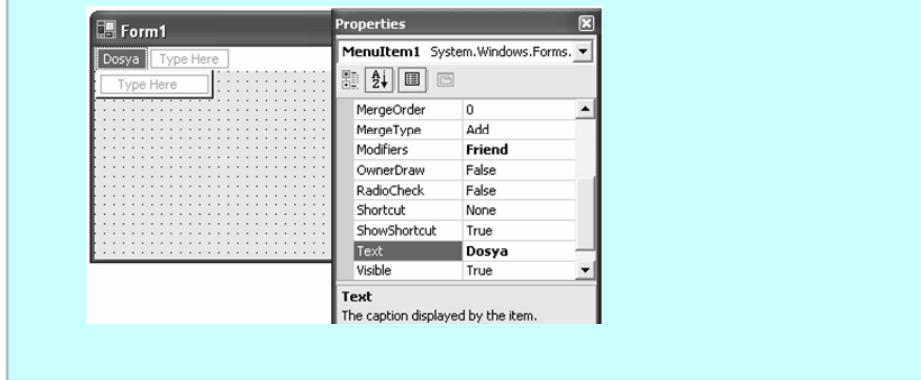
- **MainMenu**, **ContextMenu** kontrolleri ile menü tanımı yapabilecek,
- **ToolBar** kontrolü ile tasarımında araç çubuklarını kullanabilecek,
- **ToolTip** kontrolü ile menü araçlarının kullanımı hakkında bilgi sağlayacak,
- **StatusBar**, **NotifyIcon** kontrolleri ile uygulamaların tasarımını zenginleştireceksiniz.

### Menüler

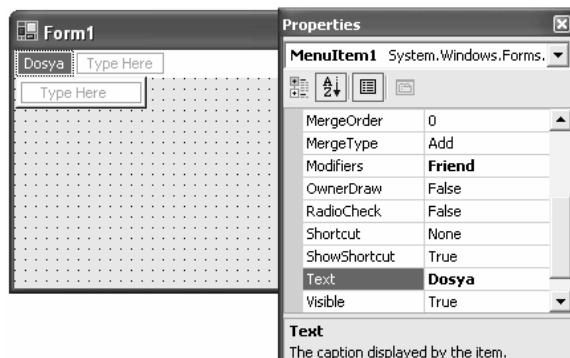
Windows uygulamalarında kullanılan iki tip menü vardır. **MainMenu**, formların başında duran sabit menüdür. **ContextMenu**, fare ile sağ tıklandığında çıkan menüdür.

## MainMenu

- # MainMenu
- Formların başında duran menüdür.
  - MenuItem nesnelerinden oluşur.
  - Menü öğelerine kısa yollar atanabilir.



Windows uygulamasına bir menü eklemek için, **Toolbox** panelinden bir **MainMenu** kontrolünü forma sürükleşin. Eklenen menü bir bileşen olarak formun alt bölümünde gözükecektir. Ancak üstüne gelindiğinde formun başlığının hemen altında belirir. Menü ögesi eklemek veya ismini değiştirmek için üstüne gelinir ve başlık yazısı yazılır. **Properties** panelinde bu menünün **MenuItem** olarak eklendiği görülür.

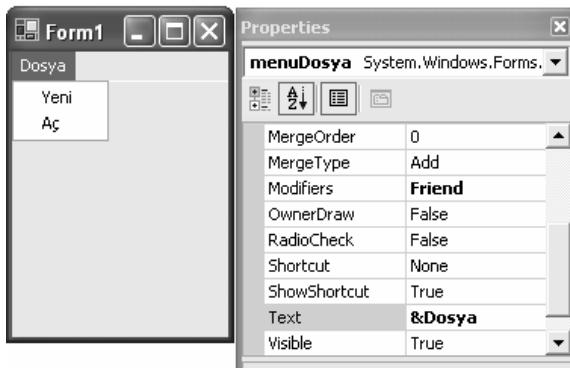


Menüye **MenuItem** eklendiğinde hemen altında ve yanında, menü eklemek için bir yer açılır. Bu açılan yere de menü ismi girip, alt menü öğeleri oluşturulabilir.

Menü öğelerine basıldığı zaman bir işlemin gerçekleşmesi için, kontrole çift tıklanarak bu öjenin **Click** olayına geçilir. Çalıştırılmak istenen kodlar buraya yazılır.

```
private void menuYeni_Click(System.Object sender,
System.EventArgs e) {
}
```

Menü öğelerine isim verirken & işaretin kullanılarak, kullanıcının klavyenin **ALT** tuşuyla bu öğeyi çalıştırmasını sağlanabilir. & işaretin hangi karakter ile kullanılırsa, kısa yol olarak o karakter kullanılır.



### MenuItem özellikleri

| Özellik             | Değer Tipi                | Açıklama                                                                        |
|---------------------|---------------------------|---------------------------------------------------------------------------------|
| <b>Checked</b>      | <b>Boolean</b>            | Menü öğesinin yanında seçili olduğuna dair bir işaretin gözükmemesini sağlar    |
| <b>Enabled</b>      | <b>Boolean</b>            | Menü öğesinin aktif durumda olup olmadığını belirler                            |
| <b>RadioCheck</b>   | <b>Boolean</b>            | Öğenin seçilme stilinin <b>RadioButton</b> düğmesi olarak gözükmemesini sağlar. |
| <b>ShortCut</b>     | <b>ShortCut</b>           | Menüye ulaşım için bir kısa yol tanımlar.                                       |
| <b>ShowShortcut</b> | <b>Boolean</b>            | Menünün kısa yolunun, isminin yanında gözükmemesini belirler                    |
| <b>MenuItems</b>    | <b>MenuItemCollection</b> | Alt menülerin tutıldığı koleksiyondur.                                          |

Örnek:

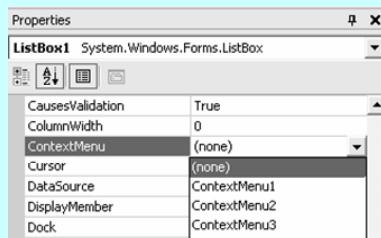
```
private void Form1_Load( System.Object sender,
System.EventArgs e ) {
    MenuItem dosya = new MenuItem( "D&osya" );
```

```
// Yeni işleminin yapılması için bir menü eklenir.  
MenuItem yeni = new MenuItem( "&Yeni" );  
yeni.Shortcut = Shortcut.CtrlN;  
yeni.ShowShortcut = true;  
yeni.Select += new System.EventHandler(  
Yeniclick );  
dosya.MenuItems.Add( yeni );  
  
// Açma işleminin yapılması için bir menü eklenir.  
MenuItem ac = new MenuItem( "&Aç" );  
ac.Shortcut = Shortcut.CtrlO;  
ac.ShowShortcut = false;  
ac.Select += new System.EventHandler( AcClick );  
dosya.MenuItems.Add( ac );  
  
MainMenu1.MenuItems.Add( dosya );  
}  
  
private void AcClick( System.Object sender,  
System.EventArgs e ) {  
}  
  
private void Yeniclick( System.Object sender,  
System.EventArgs e ) {  
}
```

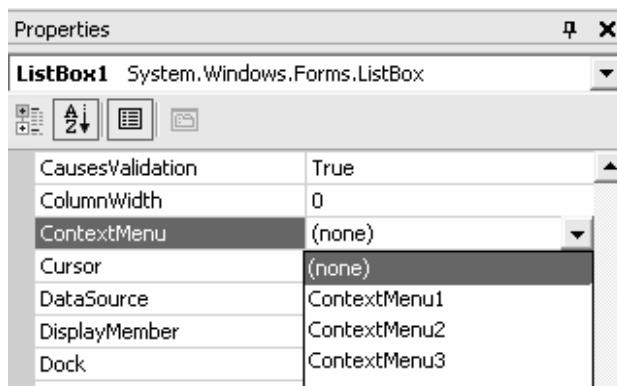


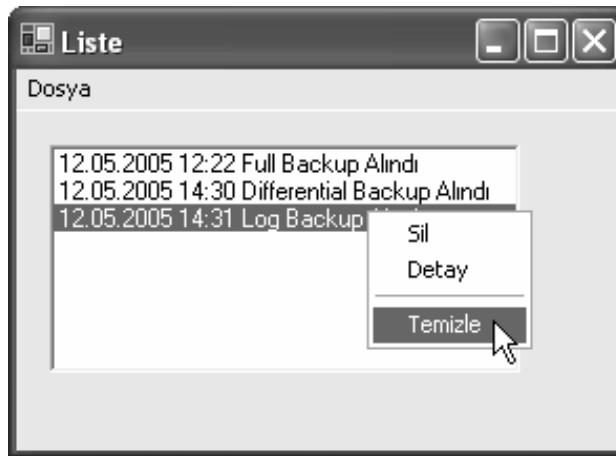
## ContextMenu

- ### ContextMenu
- Kontrollerin ContextMenu özelliğine atanır.
  - Kontollere sağ tıklandığı zaman çıkan menüdür.



**ContextMenu**, bir kontrolün üstüne sağ tıklandığı zaman çıkan menüdür. Bu menü uygulamaya eklendiği zaman **Properties** panelinde, kontrollerin **ContextMenu** özelliği olarak bu menü atanabilir.

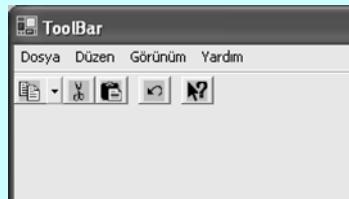




## ToolBar

### ToolBar

- Menülerin işlevlerine görsel kısa yollar sunar.
- ToolBarButton nesnelerinden oluşur.
- ImageList kontrolü ile kullanılır.
- Hangi düğmeye basıldığı ButtonClick olayı ile anlaşılır.



**ToolBar** kontrolü menülerin altında kullanıcıya kısa yollar, kullanım kolaylığı sunan bir kontroldür. Kontroldeki öğeler çoğu zaman **ImageList** kontrolünün sağladığı resimler ile gösterilir. Resim yerine yazı da gösterilebilir ancak yazı ile işlem listelemek menüler ile sağlanır.

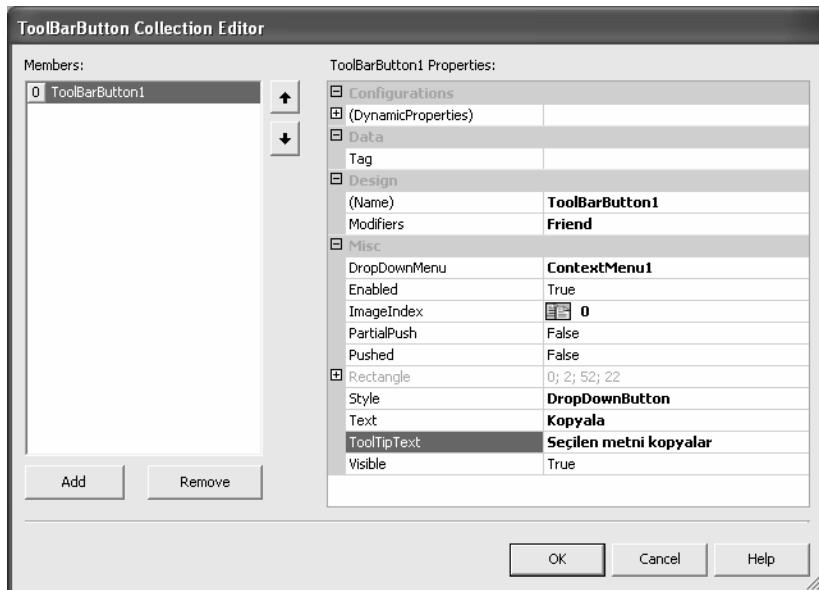
**ToolBar** kontロlünde yapılacak işlemler bir **ToolBarButton** olarak gösterilir.

#### ToolBar Özellikleri

| Özellik | Değer Tipi | Açıklama |
|---------|------------|----------|
|---------|------------|----------|

|                       |                                |                                                                                                                  |
|-----------------------|--------------------------------|------------------------------------------------------------------------------------------------------------------|
| <b>Buttons</b>        | <b>ToolBarButtonCollection</b> | Kontrolün düğmelerinin tutulduğu koleksiyon                                                                      |
| <b>ButtonSize</b>     | <b>Size</b>                    | Kontroldeki düğmelerin boyutunu belirler. Düğmelerin boyutları ayrı ayrı belirlenemez.                           |
| <b>DropDownArrows</b> | <b>Boolean</b>                 | Stili <b>DropDownButton</b> olarak seçilmiş <b>ToolBarButton</b> düğmelerinin alt menüsünün görünmesini belirler |

**ToolBar** kontrolüne **ToolBarButton** düğmeleri eklemek için kontrolün **Buttons** özelliğinde faydalanyılır. Tasarım anında **Properties** panelinden **Buttons** özelliğine basıldığı zaman çıkan pencerede, kontrole düğme eklenir.

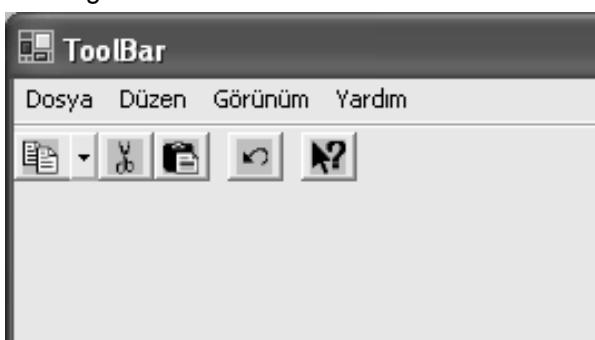


### ToolBarButton özellikleri

| Özellik      | Değer Tipi                | Açıklama                                                                                                                                                                                  |
|--------------|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Style</b> | <b>ToolBarButtonStyle</b> | Düğmenin görünüm stilini belirler. <b>PushButton</b> değeri standart bir düğmeyi, <b>ToggleButton</b> basıldığı zaman basılı kalan bir düğmeyi, <b>Separator</b> değeri düğmeler arasında |

|                     |                |                                                                                                                                                 |
|---------------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
|                     |                | bir ayrıacı temsil eder.<br><b>DropDownButton</b> değeri düğmenin yanında bir menünün açılacağını belirler.                                     |
| <b>DropDownMenu</b> | <b>Menu</b>    | Kontrolün stili <b>DropDownButton</b> olarak seçilmişse, yanında çıkacak menüyü belirler. Bu menü sadece <b>ContextMenu</b> cinsinden olabilir. |
| <b>Pushed</b>       | <b>Boolean</b> | Düğmenin basılı olup olmadığını belirler                                                                                                        |
| <b>Text</b>         | <b>String</b>  | Düğmenin üzerinde yazan yazıyı belirler                                                                                                         |
| <b>ImageIndex</b>   | <b>Integer</b> | <b>ToolBar</b> kontrolüne bir <b>ImageList</b> bağlanmışsa, bu özellik düğmenin hangi resmi göstereceğini belirler.                             |
| <b>ToolTipText</b>  | <b>String</b>  | Düğmenin üzerinde zaman gösterilecek ipucunu belirler.                                                                                          |

Düğmelere tıklandığı zaman çalışması istenen kodlar, **ToolBar** kontrolünün **ButtonClick** olayına yazılır. Ancak burada hangi düğmeye basıldığı kod yazarak bulunması gereklidir.



```
private void ToolBar1_ButtonClick( System.Object sender, System.Windows.Forms.ToolBarButtonEventArgs e )
{
    switch ( ToolBar1.Buttons.Indexof( e.Button ) )
    {
        // Ayraçlar da bir ToolBarButton olduğu için
        // indisler kontrol edilirken buna dikkat
        edilmelidir
        case 0:
            // Kopyala
```

```

        case 1:
        // Kes
        case 2:
        // Yapıştır
        case 4:
        // Geri Al
        case 6:
        // Yardım
        break;
    }
}

```

## ToolTip

### ToolTip

- Kontrollerin üzerine gelindiğinde bilgi mesajı verir.
- Mesaj, kontrollerin “ToolTip on ToolTip1” özelliğine yazılır.

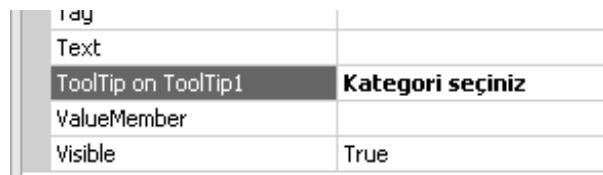


Bu kontrol, form üzerindeki kontrollerin üzerine gelindiği zaman ipucu göstermek için kullanılır. **ToolTip** forma eklendiği zaman, kontrollerin özelliklerinde **ToolTip on [ToolTip kontrolünün ismi]** şeklinde bir özellik belirir. Bu özelliğe verilen yazılar, çalışma anında kontrollerin ipucunu belirler.

### ToolTip Özellikleri

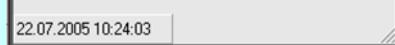
| Özellik               | Değer Tipi     | Açıklama                                                                                                      |
|-----------------------|----------------|---------------------------------------------------------------------------------------------------------------|
| <b>Active</b>         | <b>Boolean</b> | Kontrolün aktif olup olmadığını belirler. <b>False</b> değerini alırsa, form üzerinde ipucular görüntülenmez. |
| <b>AutomaticDelay</b> | <b>Integer</b> | <b>AutoPopDelay</b> ,                                                                                         |

|                     |                |                                                                                                                  |
|---------------------|----------------|------------------------------------------------------------------------------------------------------------------|
|                     |                | <b>InitialDelay</b> ,<br><b>ReshowDelay</b> değerleri için otomatik süreleri ayarlar.                            |
| <b>AutoPopDelay</b> | <b>Integer</b> | İpucunun görüntülenme süresini belirler.                                                                         |
| <b>InitialDelay</b> | <b>Integer</b> | İpucunun gözükmesi için, fare imlecinin kontrol üzerinde durması gereken süreyi belirler.                        |
| <b>ReshowDelay</b>  | <b>Integer</b> | Yeni bir kontrolün üzerinde gelindiği zaman, bu kontrole ait ipucunun gösterilmesi için gereken süreyi belirler. |
| <b>ShowAlways</b>   | <b>Boolean</b> | Seçilen kontrol aktif olmadığı zamanlarda dahi ipucunun gösterilmesini sağlar.                                   |



## StatusBar

- ### StatusBar
- Windows formlarının durum çubuğuudur.
  - ShowPanels birden fazla panelin gözükmekini sağlar.
  - Paneller birden fazla durum mesajı gösterilmek için kullanılır.



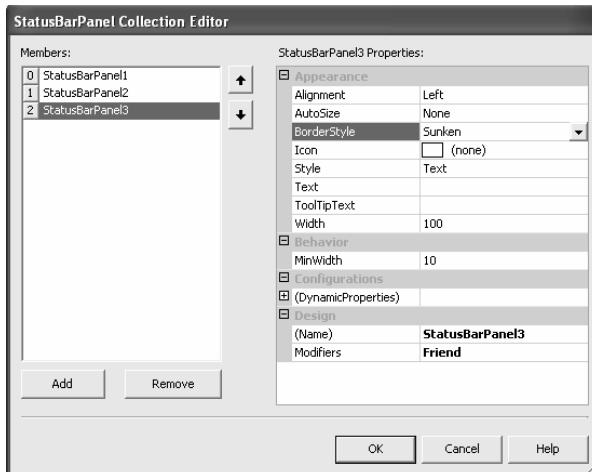
Windows uygulamalarında formların altında bulunan durum çubuğunu temsil eder. Durum çubuklarında sadece bir yazının görüntülenebilediği gibi, içindeki paneller ile birden fazla durum yazısı görüntülenebilir.

### StatusBar Özellikleri

| Özellik           | Değer Tipi                      | Açıklama                                                                                                                    |
|-------------------|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>Panels</b>     | <b>StatusBarPanelCollection</b> | Kontrolün içinde birden fazla yazı görüntülemek için kullanılan panelleri tutar.                                            |
| <b>ShowPanels</b> | <b>Boolean</b>                  | Birden fazla panelin gözükmekini belirler.                                                                                  |
| <b>SizingGrip</b> | <b>Boolean</b>                  | <b>StatusBar</b> kontrolünün yanında, formun boyutunu değiştirmek için kullanılan simgenin gözükmekini belirler             |
| <b>Text</b>       | <b>String</b>                   | <b>StatusBar</b> üzerinde yazan yazıyı belirler. Eğer <b>ShowPanels</b> özelliği <b>True</b> ise, bu özellikte yazılan yazı |

|  |  |           |
|--|--|-----------|
|  |  | gözükmez. |
|--|--|-----------|

**Statusbar** kontrolüne panel eklemek için kontrolün **Panels** özelliğinden yararlanılır.



### Panel özellikleri

| Özellik            | Değer Tipi                       | Açıklama                                                                                                                                                                                                                                                                       |
|--------------------|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>AutoSize</b>    | <b>StatusBarPanelAutoSize</b>    | Panelin bazı durumlara göre otomatik boyutlandırmasını sağlar. <b>None</b> değeri panelin boyutunun değişimeyeceğini, <b>Contents</b> değeri, panelin içerdeği yazıya göre değişecekini belirler. <b>Spring</b> değeri, durum çubuğundaki boş alanların paylaşılmasını sağlar. |
| <b>BorderStyle</b> | <b>StatusBarPanelBorderStyle</b> | Panelin kenarlık stilidir. <b>Raised</b> değeri, panelin bir düğme gibi gözükmesini, <b>Sunken</b> değeri, panelin basık gözükmesini sağlar. <b>None</b> değeri, kenarların gözükmesini engeller.                                                                              |
| <b>Alignment</b>   | <b>HorizontalTAignment</b>       | Panelin yazısının hizalanmasını belirler.                                                                                                                                                                                                                                      |

|                 |                            |                                                                                                                                                                                  |
|-----------------|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Text</b>     | <b>String</b>              | <b>Panel1</b> üzerinde yazan yazıyı belirler                                                                                                                                     |
| <b>Width</b>    | <b>Integer</b>             | Panelin genişliğini belirler                                                                                                                                                     |
| <b>Minwidth</b> | <b>Integer</b>             | <b>Panel1</b> boyutunun minimum değerini belirler.                                                                                                                               |
| <b>Style</b>    | <b>StatusBarPanelStyle</b> | Panelin üzerindeki yazıların stilini belirler. <b>Text</b> değeri, normal yazı gözükmesci sağlar. <b>OwnerDraw</b> , değişik font ve renklerde yazıların görüntülenmesini sağlar |
| <b>Icon</b>     | <b>Icon</b>                | Panel üzerinde görüntülenen simgeyi belirler                                                                                                                                     |

```
private void Form1_Load( System.Object sender,
System.EventArgs e ) {
    StatusBarPanel p = new StatusBarPanel();
    p.Minwidth = 100;
    p.AutoSize = StatusBarPanelAutoSize.Contents;
    p.Alignment = HorizontalAlignment.Left;
    p.BorderStyle =
StatusBarPanelBorderStyle.Raised;
    p.Style = StatusBarPanelstyle.Text;

    StatusBar1.Panels.Add( p );
}

Timer1.Interval = 1000;
Timer1.Start();
}
```

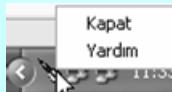
```
private void Timer1_Tick( System.Object sender,
System.EventArgs e ) {
    StatusBarPanel panel = new StatusBarPanel();
    panel = StatusBar1.Panels[ 0 ];
    panel.Text = System.Convert.ToString(
DateTime.Now );
}
```



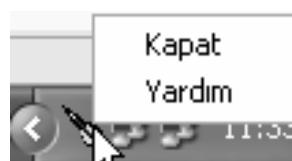
## NotifyIcon

### NotifyIcon

- Windows görev çubuğunda görüntülenen simgedir.



Windows uygulamalarının, Windows görev çubuğunda görüntünlendiği simgesi belirler.

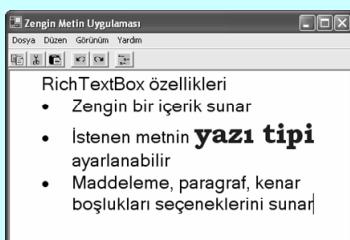


### NotifyIcon Özellikleri

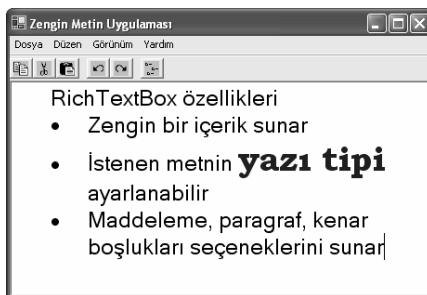
| Özellik            | Değer Tipi    | Açıklama                                                   |
|--------------------|---------------|------------------------------------------------------------|
| <b>Icon</b>        | <b>Icon</b>   | Görev çubuğunda gözükecek simgeyi belirler                 |
| <b>ContextMenu</b> | <b>Menu</b>   | Simgeye sağ tıklandığı zaman açılacak menü                 |
| <b>Text</b>        | <b>String</b> | Simge üzerine gelindiğinde görüntülenecek yazıyı belirler. |

## RichTextBox

- TextBox kontrolünden daha gelişmiş özelliklere sahiptir.
  - Seçilen yazının rengi, yazı tipi değiştirilebilir
  - Madde işaretleri kullanılabilir.
  - Satır başlarındaki boşluklar ayarlanabilir.



Normal bir metin kutusundan daha gelişmiş özelliklere sahip bir kontroldür. **TextBox** kontrolünde yazının yazı tipi, büyülüğu gibi ayarlar yapılabilir. Ancak sadece seçilen yazının rengi, yazı tipi, satır başı genişliği, madde işaretleri kullanımı gibi ayarlar yapmak mümkün değildir. **RichTextBox** kontrolü, bu tip zengin özelliklerin kullanılmasını sağlar.



### RichTextBox Özellikleri

**RichTextBox** kontrolü kullanıcıya birçok seçenek sunar, dolayısıyla tasarım ve çalışma anında erişilebilen birçok özelliği bulunur.

Tasarım anında ulaşılabilcek özellikler:

| Özellik           | Değer Tipi    | Açıklama                                   |
|-------------------|---------------|--------------------------------------------|
| <b>ZoomFactor</b> | <b>Single</b> | Metnin büyüğünü belirler. 1 – 64 arası bir |

|                            |                 |                                                                                                                                       |
|----------------------------|-----------------|---------------------------------------------------------------------------------------------------------------------------------------|
|                            |                 | değer alır.                                                                                                                           |
| <b>WordWrap</b>            | <b>Boolean</b>  | Uzun yazıların bir sonraki satıra geçerek görüntülenmesini sağlar                                                                     |
| <b>DetectUrls</b>          | <b>Boolean</b>  | Bağlantı olarak girilen yazıların <b>LinkLabel</b> şeklinde algılanmasını belirler                                                    |
| <b>Lines</b>               | <b>String()</b> | Satırları <b>string</b> dizisi olarak tutar                                                                                           |
| <b>BulletIntend</b>        | <b>Integer</b>  | Satırların madde işaretinden kaç piksel açıkta duracağını belirler                                                                    |
| <b>AcceptsTab</b>          | <b>Boolean</b>  | <b>Tab</b> tuşunu bir karakter olarak algılanmasını, dolayısıyla bu tuşa basıldığında kontrolden çıkışmasının engellenmesini belirler |
| <b>ShowSelectionMargin</b> | <b>Boolean</b>  | Satır başındaki boşluğun gösterilmesini belirler                                                                                      |
| <b>RightMargin</b>         | <b>Integer</b>  | Satırların maksimum uzunluğunu piksel cinsinden belirler.                                                                             |

Çalışma anında ulaşılabilecek özellikler:

| Özellik                | Değer Tipi     | Açıklama                                                                                       |
|------------------------|----------------|------------------------------------------------------------------------------------------------|
| <b>Capture</b>         | <b>Boolean</b> | Kontrol içine yazı yazarken farenin gizlenmesini belirler                                      |
| <b>UndoActionName</b>  | <b>String</b>  | En son yapılabilecek <b>Undo</b> işleminin tipini tutar                                        |
| <b>RedoActionName</b>  | <b>String</b>  | <b>Undo</b> işlemi yapıldıktan sonra, en son yapılabilecek <b>Redo</b> işleminin ismini tutar. |
| <b>SelectedText</b>    | <b>String</b>  | Seçilen metni belirler                                                                         |
| <b>SelectionBullet</b> | <b>Boolean</b> | Seçilen satırın madde                                                                          |

|                           |                |                                                      |
|---------------------------|----------------|------------------------------------------------------|
|                           |                | işaretli olarak görüntülenmesini belirler            |
| <b>SelectionAlignment</b> | <b>Boolean</b> | Seçilen satırın hizalanmasını belirler               |
| <b>SelectionColor</b>     | <b>Color</b>   | Seçilen metnin rengini belirler                      |
| <b>SelectionFont</b>      | <b>Font</b>    | Seçilen metnin yazı tipini belirler                  |
| <b>SelectionIntend</b>    | <b>Integer</b> | Seçilen satırın, sol kenara olan uzaklığını belirler |
| <b>SelectionLength</b>    | <b>Integer</b> | Seçilen metnin uzunluğunu belirler                   |

### **RichTextBox Metotları**

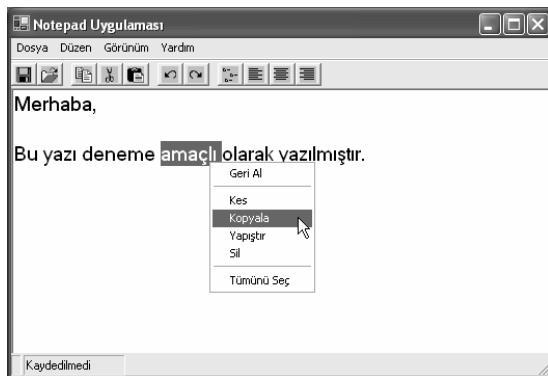
| Metot           | Açıklama                                                                                                                                                                     |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Find</b>     | Metin kutusu içinde, parametre olarak verilen bir yazıyı arar. Yazıyı ilk gördüğü yerin indisini döndürür.                                                                   |
| <b>LoadFile</b> | Bir dosyadan alınan metni yükler                                                                                                                                             |
| <b>SaveFile</b> | Parametre olarak verilen konumdaki dosyaya, metni yazar. Dosyanın <b>rtf</b> veya <b>doc</b> uzantılarında kaydedilmesi, zengin içeriğin görüntülenmesi açısından önemlidir. |
| <b>Undo</b>     | Yapılan işlem geriye alınır                                                                                                                                                  |
| <b>Redo</b>     | Geri alınan işlem tekrar yapılır                                                                                                                                             |

### RichTextBox olayları

| Olay               | Açıklama                                                   |
|--------------------|------------------------------------------------------------|
| <b>TextChanged</b> | Metin kutusundaki yazı değiştiği zaman gerçekleşir         |
| <b>LinkClicked</b> | Metin içindeki bir bağlantıya tıklandığı zaman gerçekleşir |

## Lab 1: Notepad uygulaması

Bu labda, **RichTextBox** kontrolünün sağladığı kolaylıklarla bir metin editörü uygulaması geliştirilir. Bu uygulamanın kullanımını kolaylaştmak için menüler, araç çubuğu ve durum çubuğundan faydalanılır.



Bu labda kullanılan kontroller ve teknikler:

- **MainMenu** – Dosya, düzen, görünüm ve yardım işlemleri için kullanılır
- **ContextMenu** – Araç çubugunu gizlemek ve kopyala, yapıştır, kes gibi metin işlemleri için kullanılır
- **RichTextBox** – Yazılan metnin tutulması için kullanılır
- **NotifyIcon** – Uygulamanın simgesinin görev çubuğunda gözükmesini sağlar
- **ToolBar** – Kaydetme, dosya açma, hizalama gibi işlevlere kısa yollar sağlamak için kullanılır.
- **ImageList** – Araç çubuğundaki düğmeleri resimlerini belirlemek için kullanılır
- **SaveFileDialog** – Dosyaların kaydedilmesi sırasında kullanılır.
- **OpenFileDialog** – Dosyaları açmak için kullanılır.
- **FontDialog** – Yazı tipini değiştirmek için kullanılır.
- **StatusBar** – Dosyalar açıldığı zaman isimlerini ve kayıt durumlarını görüntülemek için kullanılır.

## Kontrollerin eklenmesi

Form üzerine tablodaki kontrolleri ekleyin belirtilen özelliklerini ayarlayın.

| Kontrol – Kontrol İsmi                   | Özellik           | Değer                                                                                                                                                                                                                                                                  |
|------------------------------------------|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ContextMenu</b> – ContextMenu1        |                   | Geri Al, Kes, Kopyala, Yapıştır, Sil, Tümünü Seç değerlerini içeren menü öğeleri ekleyin                                                                                                                                                                               |
| <b>ContextMenu</b> – ContextMenu2        |                   | Gizle değerini içeren bir menü öğesi ekleyin                                                                                                                                                                                                                           |
| <b>ToolBar</b> – ToolBar1                | <b>Buttons</b>    | Kaydet, Aç, Kopyala, Kes, Yapıştır, <b>Undo</b> , <b>Redo</b> , Madde İşaretle, Sola Hizala, Sağa Hizala, Ortala komutları için düğmeler ekleyin. Her düğmenin <b>ImageIndex</b> özelliğine, <b>ImageList</b> içinde bulunan resimlerden uygun olanın indisini atayın. |
| <b>ImageList</b> – ImageList1            | <b>Images</b>     | Araç çubuğundaki öğeleri temsil eden resimler ekleyin                                                                                                                                                                                                                  |
| <b>OpenFileDialog</b><br>OpenFileDialog1 | -                 |                                                                                                                                                                                                                                                                        |
| <b>SaveFileDialog</b><br>SaveFileDialog1 | -                 |                                                                                                                                                                                                                                                                        |
| <b>FontDialog</b> - FontDialog1          |                   |                                                                                                                                                                                                                                                                        |
| <b>StatusBar</b> - StatusBar1            | <b>ShowPanels</b> | <b>True</b>                                                                                                                                                                                                                                                            |
|                                          | <b>Panels</b>     | İki tane panel ekleyin. İlk panelin <b>AutoSize</b> özelliğini <b>Contents</b> olarak belirleyin.                                                                                                                                                                      |
| <b>NotifyIcon</b> - NotifyIcon1          | <b>Icon</b>       | Uygulamanız için bir simge seçin                                                                                                                                                                                                                                       |
|                                          | <b>Text</b>       | “Notepad Uygulaması”                                                                                                                                                                                                                                                   |
| <b>RichTextBox</b> – RichTextBox1        | <b>Dock</b>       | <b>True</b>                                                                                                                                                                                                                                                            |

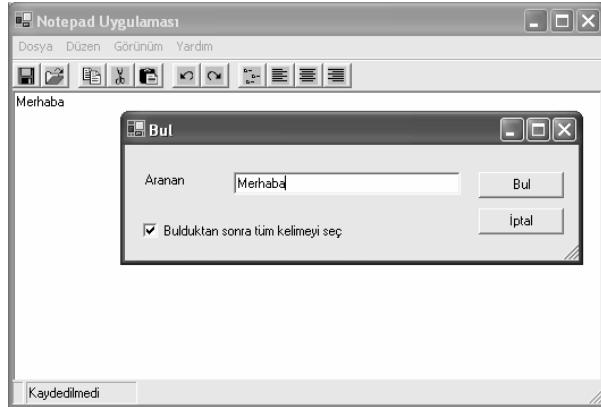
Uygulamaya son olarak bir **MainMenu** ve ilgili alanlara **MenuItem** öğelerini ekleyin. Parantez içinde belirtilen tuşlar, menü öğelerine erişmek için kullanılacak kısa yollardır. Bu değerleri, menü öğelerinin **ShortCut** özelliğine ekleyin.

- Dosya
  - Yeni (**Ctrl- N**)
  - Açı (**Ctrl- O**)
  - Kaydet (**Ctrl- S**)
  - Farklı Kaydet
  - Çıkış
- Düzen
  - Geri Al (**Ctrl- Z**)
  - Kes (**Ctrl- X**)
  - Kopyala (**Ctrl- C**)
  - Yapıtır (**Ctrl- V**)
  - Sil
  - Bul
  - Yazı Tipi
  - Tümünü Seç
- Görünüm
  - Sola Hizala
  - Sağa Hizala
  - Ortala
  - Madde İşaretle
  - Araç çubuğu gizle
- Yardım
  - Hakkında

Uygulamaya **frmBu1** isminde yeni bir form ekleyin. Bu form, metin kutusunda aranan değeri bulmak için kullanılacaktır.

Forma, arama işlemleri için gereken kontrolleri ekleyin.

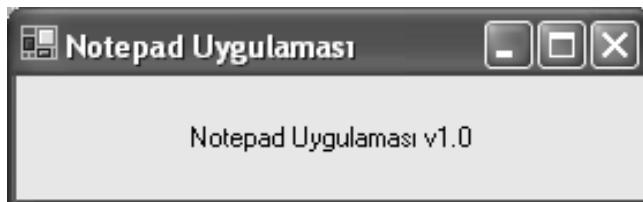
| Kontrol – Kontrol İsmi             | Özellik             | Değer                              |
|------------------------------------|---------------------|------------------------------------|
| <b>Button</b> – Button1            | <b>DialogResult</b> | DialogResult.OK                    |
|                                    | <b>Text</b>         | “Bul”                              |
| <b>Button</b> – Button2            | <b>Text</b>         | “İptal”                            |
| <b>CheckBox</b> – cbTumKelimeyiSec | <b>Text</b>         | “Bulduktan sonra tüm kelimeyi seç” |
|                                    | <b>Checked</b>      | True                               |
| <b>TextBox</b> – txtAranan         |                     |                                    |



Kontrolleri ekledikten sonra Button1 ve Button2 düğmelerinin **Click** olayına, formu kapatılan kodları yazın:

```
private void Button1_Click( System.Object sender,
System.EventArgs e ) {
    this.Close();
}
```

Uygulamaya **frmHakkında** isminde yeni bir form ekleyin. Bu form, uygulama bilgilerini içerir.



## Kodların yazılması

- Dosya ismini ve dosyanın kaydedildiip edilmediğini tutan değişkenleri tanımlayın

```
private bool Kaydedildi = true;
private string Dosyaİsmi;
```

Menü öğelerine kod eklemeden önce, yapılacak işlemler yordamlar içine yazılır. Böylece kodun karmaşıklığı azalır ve değişiklik yapmak kolaylaşır.

- Durum çubuğuunda değişiklik yapma işlemlerini yazın. Durum çubuğu, dosya açma kaydetme gibi işlemler sonunda değişecektir

```
public void DurumDegistir() {
    StatusBar1.Panels[ 0 ].Text = Dosyaİsmi;
    if ( Kaydedildi ) {
        StatusBar1.Panels[ 1 ].Text = "Kaydedildi";
    }
    else {
        StatusBar1.Panels[ 1 ].Text =
"Kaydedilmedi";
    }
}
```

- Dosyaya kaydetme ve farklı kaydetme işlemlerini yazın.

```
// Kaydetme işlemi
public void Kaydet() {
    if ( DosyaIsmi == "" ) {
        Farklikaydet();
    }
    else {
        RichTextBox1.SaveFile( DosyaIsmi );
        Kaydedildi = true;
    }
    DurumDegistir();
}

// Farklı kaydetme işlemi
public void Farklikaydet() {
    string dosya;
    // Kaydedilecek yeri seçmek için
    // SaveFileDialog kutusu gösterilir
    // Dosya yoksa otomatik olarak oluşturulması
    // sağlanır
    SaveFileDialog1.CreatePrompt = true;

    if ( SaveFileDialog1.ShowDialog() ==
DialogResult.OK ) {
        dosya = SaveFileDialog1.FileName;
        RichTextBox1.SaveFile( dosya );
        DosyaIsmi = dosya;
        Kaydedildi = true;
    }
    DurumDegistir();
}
```

- Yeni bir dosya veya var olan bir dosyayı açma işlemlerini tanımlayın.

```
public void DosyaAc( bool yeniDosya ) {
    if ( !( Kaydedildi ) ) {
        switch ( MessageBox.Show( "Dosya kaydedilsin
mi?", "", MessageBoxButtons.YesNoCancel ) ) {
            case DialogResult.OK:
                // Kaydetme işlemi yapılır
                Kaydet();
                break;
            case DialogResult.Cancel:
                // İşlem iptal edildi
                return;
        }
    }

    if ( !( yeniDosya ) ) {
        // Varolan bir dosya alır.
        string dosya = null;
        if ( OpenFileDialog1.ShowDialog() ==
DialogResult.OK ) {
            dosya = OpenFileDialog1.FileName;
            RichTextBox1.LoadFile( dosya );
            DosyaIsmi = dosya;
        }
    }
    else {
```

```

    // Yeni bir dosya açılır
    richTextBox1.Clear();
    DosyaIsmi = "";
}

Kaydedildi = true;
DurumDegistir();
}

```

- Bulma işlemlerini gerçekleştiren kodları yazın. Burada yeni bir form açılıp, orda girilen değerlere göre arama işlemi yapılır.

```

public void Bul() {
    // Bulma formu görüntülenir, iptal tuşuna basıldığında
    // çıkıştır
    frmBul bul = new frmBul();
    if ( !bul.ShowDialog() == DialogResult.OK ) 
    { return; }

    string aranan = bul.txtAranan.Text;
    if ( aranan == "" ) { return; }

    // Bulduktan sonra kelimenin tümünü işaretlenmesi
    // bilgisi alınır
    bool TumKelimeyisec =
    bul.cbTumKelimeyisec.Checked;

    // Bulunan ilk indis alınır.
    int start = richTextBox1.Find( aranan );

    if ( !( TumKelimeyisec ) ) {
        // Sadece aranan kelime seçilir.
        richTextBox1.Select( start, aranan.Length );
    }
    else {
        int son = start;
        int bas = start;

        while ( son < richTextBox1.Text.Length - 1
&& richTextBox1.Text.Substring( son, 1 ) != " " ) {
            son += 1;
        }

        while ( bas > -1 &&
RichTextBox1.Text.Substring( bas, 1 ) != " " ) {
            bas -= 1;
        }

        richTextBox1.Select( bas + 1, son - bas - 1
);
    }
}

```

- ToolBar düğmelerine basıldığı zaman gerçekleşecek kodları yazın.

**Dikkat:** Bu kodda belirtilen indis numaraları, uygulamanızda **ToolBar** kontrolüne eklediğiniz düğmelerin indis numaralı ile farklılık gösterebilir. Yapılan işlemler yorum satırı olarak geçilmiştir. Bu işlemleri, düğmelerin indislerine göre tekrar düzenleyin. Düğmelerin indislerini öğrenmek için **ToolBar** kontrolünün **Buttons** özelliğine bakın.

```

        private void ToolBar1_ButtonClick( System.Object
sender, System.Windows.FormsToolBarButtonClickEventArgs e )
{
    // Basılan düğmenin indisine göre işlem yapılır.
    switch ( ToolBar1.Buttons.IndexOf( e.Button ) )
    {
        case 0:
            // Kaydet
            Kaydet();
            break;
        case 1:
            // Ac
            DosyaAc(False);
            break;
        case 3:
            // Kopyala
            RichTextBox1.Copy();
            break;
        case 4:
            // Kes
            RichTextBox1.Cut();
            break;
        case 5:
            // Yapıtır
            RichTextBox1.Paste();
            break;
        case 7:
            // Geri Al
            RichTextBox1.Undo();
            break;
        case 8:
            // Tekrarla
            RichTextBox1.Redo();
            break;
        case 10:
            // Madde işaretle
            RichTextBox1.SelectionBullet = Not
RichTextBox1.SelectionBullet;
            break;
        case 11:
            // Sola Hizala
            RichTextBox1.SelectionAlignment =
HorizontalAlignment.Left;
            break;
        case 12:
            // Ortala
            RichTextBox1.SelectionAlignment =
HorizontalAlignment.Center;
            break;
        case 13:
            // Sağ'a Hizala
            RichTextBox1.SelectionAlignment =
HorizontalAlignment.Right;
            break;
    }
}

```

- Dosya içinde bulunan bir bağlantıya tıklandığı zaman, bu bağlantıyı ilgili tarayıcıda açan kodları yazın.

// Linke git

```

    private void RichTextBox1_LinkClicked( object
sender, System.Windows.Forms.LinkClickedEventArgs e ) {
    System.Diagnostics.Process.Start( e.LinkText
);
}

```

- Dosya içine yazılan yazı değiştiği zaman gereken kodları yazın

```

    private void RichTextBox1_TextChanged( System.Object
sender, System.EventArgs e ) {
    Kaydedildi = false;
    DurumDegistir();
}

```

- Uygulama kapanırken dosyanın kaydedilmesini soran kodları yazın.

```

// Kapanırken dosyanın kaydedilmesi kontrol edilir.
private void Form3_Closing( object sender,
System.ComponentModel.CancelEventArgs e ) {
    if ( !( Kaydedildi ) ) {
        switch ( MessageBox.Show ("Dosya
kaydedilsin mi?", "", MessageBoxButtons.YesNoCancel) ) {
            case DialogResult.OK:
                // Kaydetme işlemi yapılır
                Kaydet();

                break;
            case DialogResult.Cancel:
                // İşlem iptal edildi
                e.Cancel = true;
                break;
        }
    }
}

```

- Her menü öğesinin altına, ilgili işlemleri yazın. Burada dikkat edilmesi gereken nokta, bazı **ContextMenu** öğelerinin ve **MainMenu** öğelerinin aynı işlemi yaptığıdır. Örneğin “Geri Al” komutu, her iki menüde de vardır. Bu kodları farklı yordamlar yerine, aynı yordamın içine yazarak **Handles** ifadesine iki menü öğesinin **Click** olayı yazılır.

Örnek:

```

private void MenuItem19_Click( System.Object sender,
System.EventArgs e ) {
    RichTextBox1.Undo();
}

```

**Dikkat:** Bu kodda belirtilen menü isimleri, uygulamanızda **MainMenu** veya **ContextMenu** kontrolüne eklediğiniz menülerin isimleri ile farklılık gösterebilir. Yapılan işlemler yorum satırı olarak geçmiştir. İlgili menü öğesine çift tıklayarak **Click** olayında, burada belirtilen işlemleri yazın.

```

// Yeni Dosya aç
private void MenuItem13_Click( System.Object sender,
System.EventArgs e ) {

```

```
        DosyaAc(True);
    }

    // Dosya Aç
    private void MenuItem14_Click( System.Object sender,
System.EventArgs e ) {
    DosyaAc(False);
}

    // Kaydet
    private void MenuItem15_Click( System.Object sender,
System.EventArgs e ) {
    Kaydet();
}

    // Farklı Kaydet
    private void MenuItem16_Click( System.Object sender,
System.EventArgs e ) {
    Farklikaydet();
}

    // Çık
    private void MenuItem18_Click( System.Object sender,
System.EventArgs e ) {
    Application.Exit();
}

    // Geri al
    private void MenuItem19_Click( System.Object sender,
System.EventArgs e ) {
    RichTextBox1.Undo();
}

    // Kes
    private void MenuItem21_Click( System.Object sender,
System.EventArgs e ) {
    RichTextBox1.Cut();
}

    // Kopyala
    private void MenuItem22_Click( System.Object sender,
System.EventArgs e ) {
    RichTextBox1.Copy();
}

    // Yapıştır
    private void MenuItem23_Click( System.Object sender,
System.EventArgs e ) {
    RichTextBox1.Paste();
}

    // Yazı sil
    private void MenuItem24_Click( System.Object sender,
System.EventArgs e ) {
    // silinecek kelime RichTextBox kontrolünde seçilen
    // kelimedir
    string silinecek = RichTextBox1.SelectedText;

    // secilen kelimenin indisini bulunur
    int i = RichTextBox1.SelectionStart;

    RichTextBox1.Text = RichTextBox1.Text.Remove(i,
    silinecek.Length);
```

```
}

    // Tüm yazıyı seç
    private void MenuItem28_Click( System.Object sender,
System.EventArgs e ) {
    RichTextBox1.SelectAll();
}

    // Yazı tipini seç
    private void MenuItem36_Click( System.Object sender,
System.EventArgs e ) {
    // Font seçerken, renklerin de görünmesi sağlanır.
    FontDialog1.ShowDialog = True;

    if (FontDialog1.ShowDialog == DialogResult.OK)
    {
        RichTextBox1.SelectionFont = FontDialog1.Font;
    }
}

    // Sola Hizala
    private void MenuItem29_Click( System.Object sender,
System.EventArgs e ) {
    RichTextBox1.SelectionAlignment =
HorizontalAlignment.Left;
}

    // Sağa Hizala
    private void MenuItem30_Click( System.Object sender,
System.EventArgs e ) {
    RichTextBox1.SelectionAlignment =
HorizontalAlignment.Right;
}

    // Ortala
    private void MenuItem32_Click( System.Object sender,
System.EventArgs e ) {
    RichTextBox1.SelectionAlignment =
HorizontalAlignment.Center;
}

    // Madde işaretle
    private void MenuItem33_Click( System.Object sender,
System.EventArgs e ) {
    RichTextBox1.SelectionBullet = Not
RichTextBox1.SelectionBullet;
}

    // Hakkında formunun gösterilmesi
    private void MenuItem34_Click( System.Object sender,
System.EventArgs e ) {
    frmHakkında_hakkında = New frmHakkında();
    hakkında.ShowDialog();
}

    // Araç çubuğuunun gizlenmesi, MainMenu ve Toolbar
kontrolüne
    // atanın ContextMenu yapılır.
    private void MenuItem37_Click( System.Object sender,
System.EventArgs e ) {
    ToolBar1.Visible = MenuItem37.Checked;
    MenuItem37.Checked = Not MenuItem37.Checked;
```

```
}

// Dosya bulunması
private void MenuItem26_Click( System.Object sender,
System.EventArgs e ) {
    Bul();
}
```

## MDI Formlar

### MDI Formlar

- Multiple Document Interface – Bir çok alt formu barındıran formlardır.
- Bu formların **IsMdiContainer** özelliği **True** yapılır.
- Alt formun **MdiParent** özelliği, ait olduğu ana formu belirler
- **MdiChildren** özelliği alt form dizisini verir.

Multiple Document Interface formları, içinde birden fazla form barındıran formlardır. **MDIChild** olarak eklenen bu formlar birbirinden tamamen bağımsızdır. Örneğin bir Excel dosyası içinde birden fazla sayfa olabilir. Bu sayfalar ana forma bağlıdır. Ana form kapandığı zaman bu sayfalar da kapanır. **MDIParent** olarak nitelendirilen bu ana formların, **MDIChild** formlarını açmak ve yönetmek için menülere ihtiyaçları vardır.

Formları MDI olarak tanımlamak için **IsMdiContainer** özelliğinin **True** olarak ayarlanması gereklidir.

|                       |             |
|-----------------------|-------------|
| HelpButton            | False       |
| Icon                  | (Icon)      |
| ImeMode               | NoControl   |
| <b>IsMdiContainer</b> | <b>True</b> |
| KeyPreview            | False       |
| Language              | (Default)   |
| Localizable           | False       |

MDI formlara alt formlar eklemek için, form oluşturma işlemleri bilinen şekilde yapılır. Ancak formun **MDIParent** özelliği belirlenmelidir.

```
AltForm f = New AltForm();

// Oluşturulan form, ana forma bağlanır.
f.MdiParent = this;
f.Show();
```

Bir formun sahip olduğu alt formlara ulaşmak için, **MDIChildren** özelliğinden yararlanılır. Bu özellik tek boyutlu bir Form dizisidir.

```
// Tüm formları kapatır.
// Alt formlar kapandığı zaman, dizi otomatik olarak
// yeniden boyutlandırılır.
while (this.Mdichildren.Length > 0)
{
    this.Mdichildren[0].Close();
}

// Tüm formları Minimize eder
for (int i = 0; i < Me.Mdichildren.Length; i++)
{
    this.Mdichildren[i].WindowState =
FormWindowState.Minimized;
}
```



Alt formlar genişletildiklerinde, form üzerinde yazan başlığı ana forma taşınır. Alt formda tanımlı bir menü, ana formun menüsü ile birleşir. Bu menü bireşim işlemine **Merge** denir. Menü öğeleri varsayılan olarak, ana formdaki menülerin yanına eklenir. Ancak menü öğelerinin **MergeType** özelliği ile varsayılan değer değiştirilebilir.

- **MergeType.Add**

Varsayılan değerdir. Bu değeri alan menü öğeleri, bireleşme sonucunda menüye eklenir.

- **MergeType.MergeItem**

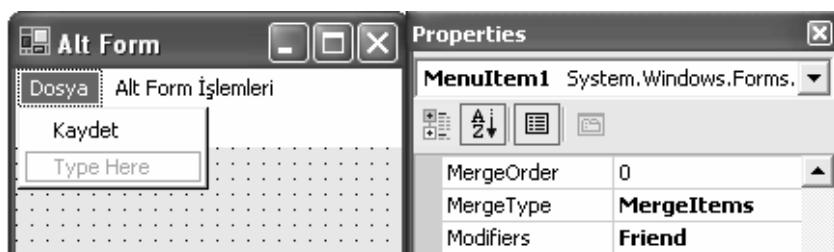
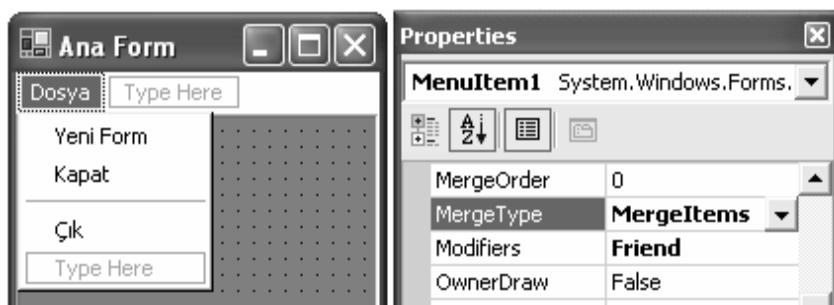
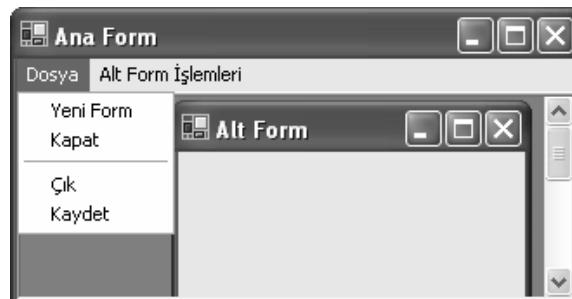
Bu değeri alan menüler, sonuç menüsünde aynı **MergeOrder** değerindeki menülerle birleşir.

- **MergeType.Replace**

Bireleşme sonucunda bu menü, aynı **MergeOrder** değerinde olan öğe ile değiştirilir.

- **MergeType.Remove**

Birleşme sonucunda bu menü çıkartılır.



Alt formlar, ana forma basamak şeklinde eklenir. Birçok alt form ile çalışılıyorsa bu formların düzenlenmesine ihtiyaç duyulur. Alt formları düzenlemek için formun **LayoutMdi** metodu kullanılır.

```
this.LayoutMdi(MdiLayout.TileHorizontal)
this.LayoutMdi(MdiLayout.TileVertical)
this.LayoutMdi(MdiLayout.Cascade)
this.LayoutMdi(MdiLayout.ArrangeIcons)
```

MDI Form içindeki alt formlardan seçili olana ulaşmak için, formun **ActiveMdiChild** özelliği kullanılır.

```
if (! this.ActiveMdiChild == null)
{
    this.Text = this.ActiveMdiChild.Text;
}
```

## Fare Olayları

### Fare olayları

- MouseEventArgs, olayla ilgili parametreleri tutar.
- MouseDown
  - Düğmeye basıldığı zaman gerçekleşir.
- MouseUp
  - Basılan düğme kaldırılınca gerçekleşir.
- MouseMove
  - Kontrolün üzerinde hareket edince gerçekleşir.

Fare olayları, formlar üzerinde farenin bir tuşunun tıklanması, üzerine gelmesi gibi olaylardır. Bu olayla ilgili parametreler, olay gerçekleştiği zaman **MouseEventArgs** nesnesi ile kullanıcıya bildirilir.

**MouseEventArgs** özellikleri:

- **Button**

Hangi fare düğmesine basıldığını gösterir.

- **Click**

Olay gerçekleşene kadar, düğmeye kaç defa basıldığını belirler. Örneğin fareye çift tıklanmışsa 2 değerini alacaktır.

- **Delta**

Farenin ortadaki düğmesinin dönme oranını gösterir.

- **X**

Kontrole göre, farenin tıklandığı pozisyonun x koordinatını gösterir.

- **Y**

Kontrole göre, farenin tıklandığı pozisyonun y koordinatını gösterir.

**NOT:** Fare olayları MDI formlar üzerinde gerçekleşmez.

### MouseDown olayı

Farenin herhangi bir düğmesi basıldığı zaman gerçekleşir. Kontrolün **Click** olayında önce çalışır.

## MouseUp olayı

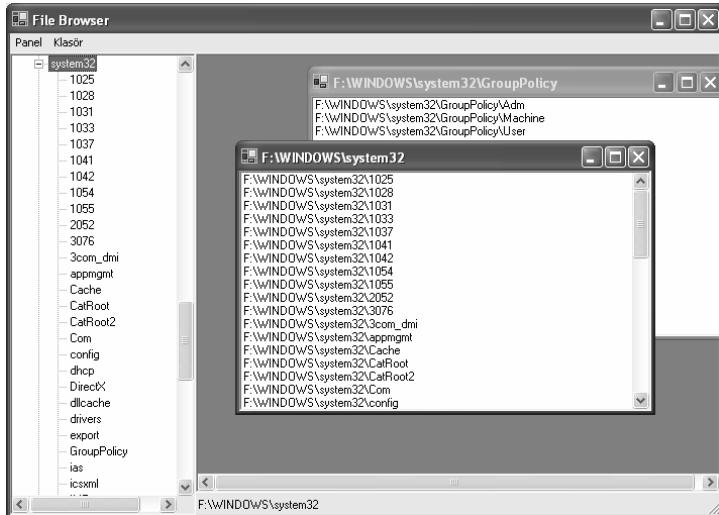
Farenin basılan düğmesi kaldırıldığı zaman gerçekleşir.

## MouseMove olayı

Farenin, kontrol üzerinde hareket etmesi ile gerçekleşir.

## Lab 2: File Browser

Bu labda, verilen bir konumdaki klasörlerin listelenmesi, seçilen klasörün bilgilerinin alt formlarda görünmesi uygulaması gerçekleştirilecektir.



Bu labda kullanılan kontroller ve teknikler:

- **MainMenu** – Klasörlerin görüntüleneceği konumu belirlemek, yeni klasör eklemek, klasör silmek gibi işlemler için kullanılır.
- **ContextMenu** – Seçilen klasörün alt klasörlerini listelemek, klasörü listeden kaldırmak için kullanılır.
- **TreeView** – Belirtilen konumdaki klasörleri ve alt klasörleri listelemeyi sağlar.
- **StatusBar** – Seçilen dosyaların konumlarını görüntülemeyi sağlar.
- **ListBox** – Alt klasörlerin listelenmesi için kullanılır.

## Kontrollerin eklenmesi

Form üzerinde tablodaki kontrolleri ekleyin belirtilen özelliklerini ayarlayın.

| Kontrol – Kontrol İsmi            | Özellik               | Değer                                                         |
|-----------------------------------|-----------------------|---------------------------------------------------------------|
| <b>Form</b>                       | <b>isMDIContainer</b> | <b>True</b>                                                   |
| <b>ContextMenu</b> – ContextMenu1 |                       | Alt Klasörler, Kaldır değerlerini içeren menü öğeleri ekleyin |
| <b>MainMenu</b> – MainMenu1       |                       | Yeni Konum, Dosya                                             |

|                               |  |                                                     |
|-------------------------------|--|-----------------------------------------------------|
|                               |  | Bilgileri değerlerini içeren menü öğelerini ekleyin |
| <b>StatusBar</b> – StatusBar1 |  |                                                     |
| <b>TreeView</b> – TreeView1   |  |                                                     |

Uygulamaya DosyaBilgileri isminde yeni bir form ekleyin. Form içine tablodaki kontrolleri ekleyin ve özelliklerini ayarlayın.

| Kontrol – Kontrol İsmi      | Özellik     | Değer                                                      |
|-----------------------------|-------------|------------------------------------------------------------|
| <b>MainMenu</b> – MainMenu1 |             | Yeni, Sil, Kapat değerlerini içeren menü öğelerini ekleyin |
| <b>ListBox</b> – ListBox1   | <b>Dock</b> | <b>F11</b>                                                 |

## Kodların yazılması

### Ana Form

- Belirtilen konumdaki klasörleri listeleyen kodları yazın.

```
public string[] KlasorleriAl( string konum ) {
    string[] klasorler = System.IO.Directory.GetDirectories( konum + @"\");
    for (int i=0; i<klasorler.Length - 1; i++ )
    {
        klasorler[ i ] = klasorler[ i ].Remove(
0, konum.Length + 1 );
    }

    return klasorler;
}
```

- Form üzerinde görüntülenecek klasörlerin bulunduğu yeri tutan değişkeni ve yeni formun açılmasını yazın.

```
private string YeniKonum;
```

```
public void FormBilgileri() {
    DosyaBilgileri f = new DosyaBilgileri();
    f.MdiParent = this;
    f.Text = YeniKonum +
    TreeView1.SelectedNode.FullPath;
    f.KlasorleriListele();
    f.Show();
}
```

- Yeni konumu seçilen menü altına, **TreeView** kontrolünde alt klasörleri listeleyen kodları yazın

```
// Yeni konum seçilmesi
private void MenuItem2_Click( System.Object sender,
System.EventArgs e ) {
```

```

YeniKonum =
Microsoft.VisualBasic.Interaction.InputBox( "Konum girin:",
"Yeni Konum", @"C:\", -1, -1 );

string[] klasorler = KlasorleriAl( YeniKonum );

for (int i=0; i<=klasorler.Length - 1; i++ ) {
    TreeView1.Nodes.Add( klasorler[ i ] );
}
TreeView1.SelectedNode = TreeView1.Nodes[ 0 ];
}

```

- **TreeView** kontrolünde bir klasör seçildiği zaman durum çubuğuunda klasörün ismini görüntüleyen kodları yazın.

```

private void TreeView1_AfterSelect( System.Object
sender, System.Windows.Forms.TreeViewEventArgs e ) {
    StatusBar1.Text = YeniKonum +
TreeView1.SelectedNode.FullPath;
}

```

**ContextMenu** içinde tanımlanan işlemleri yazın.

- Alt klasörlerin listelenmesi

```

// Alt klasörler
private void MenuItem3_Click( System.Object sender,
System.EventArgs e ) {
    TreeNode secilen = Treeview1.SelectedNode;
    secilen.Nodes.Clear();

    string konum = YeniKonum + secilen.FullPath;

    string[] altKlasorler = KlasorleriAl( konum );
    for (int i=0; i<=altKlasorler.Length - 1; i++ )
    {
        secilen.Nodes.Add( altKlasorler[ i ] );
    }
}

```

- Klasörün kaldırılma işlemi

```

// Seçilen klasörün listeden kaldırılma işlemi
private void MenuItem4_Click( System.Object sender,
System.EventArgs e ) {
    TreeNode secilen = Treeview1.SelectedNode;
    if ( secilen == null ) { return; }

    if ( secilen.Parent == null ) {
        Treeview1.Nodes.Remove( secilen );
    }
    else {
        secilen.Parent.Nodes.Remove( secilen );
    }
}

```

- Dosya bilgilerini görüntüleyen kodları yazın

```

// Dosya bilgileri - MainMenu öğesine tıklandığında
private void MenuItem5_Click( System.Object sender,
System.EventArgs e ) {
    FormBilgileri();
}

```

```
// Dosya bilgileri - TreeView öğesine çift tıklandığında
private void TreeView1_MouseDown( object sender,
System.Windows.Forms.MouseEventArgs e ) {
    if ( e.Clicks == 2 ) {
        FormBilgileri();
    }
}
```

- Farenin ortadaki tekerleğinin döndürülmesi işleminde, **TreeView** içinde seçilen öğeden bir önceki veya bir sonraki öğeye gidilmesi için gereken kodları yazın.

```
private void TreeView1_MouseWheel( object sender,
System.Windows.Forms.MouseEventArgs e ) {
    if ( TreeView1.SelectedNode == null ) { return; }

    if ( e.Delta < 0 ) {
        TreeNode sonraki =
TreeView1.SelectedNode.NextNode;
        if ( !( sonraki == null ) ) {
            TreeView1.SelectedNode = sonraki;
        }
    }
    else {
        TreeNode onceki =
TreeView1.SelectedNode.PrevNode;
        if ( !( onceki == null ) ) {
            TreeView1.SelectedNode = onceki;
        }
    }
}
```

DosyaBilgileri formunda yapılacak kodlar:

- Alt klasörlerin listelendiği kodları yazın

```
public void KlasorleriListele() {
    ListBox1.Items.Clear();
    string[] klasorler =
System.IO.Directory.GetDirectories( this.Text + @"\");
    for (int i=0; i<klasorler.Length -1; i++ ) {
        ListBox1.Items.Add( klasorler[ i ] );
    }
}
```

- Yeni klasörün eklenmesi için gereken kodları yazın.

```
private void MenuItem3_Click( System.Object sender,
System.EventArgs e ) {
    string yeniKlasor = Interaction.InputBox( "Yeni
klasör ismi girin:", "", "", -1, -1 );
    yeniKlasor = yeniKlasor.Insert( 0, this.Text +
@"\" );
    System.IO.Directory.CreateDirectory( yeniKlasor
);

    KlasorleriListele();
}
```

- Seçilen klasörün silinmesini sağlayan kodları yazın.

```
private void MenuItem2_Click( System.Object sender,
System.EventArgs e ) {
    string silinecek;
```

```
    silinecek = ListBox1.SelectedItem.ToString();  
    System.IO.Directory.Delete( silinecek, true );  
    KlasorleriListele();  
}
```

## Modül Sonu Soruları & Alıştırmalar

### Özet

- ➔ Menüler
  - ➔ MainMenu, ContextMenu
- ➔ ToolBar
- ➔ ToolTip
- ➔ StatusBar
- ➔ NotifyIcon
- ➔ RichTextBox

1. **MainMenu** ve **ContextMenu** nesnelerini ve kullanım alanlarını açıklayınız.  
Kontrolleri içeren bir uygulama geliştirin.
2. **ImageList** kontrolünün kullanım amacını ve kullanımını açıklayınız.  
Kontolu içeren bir uygulama geliştirin.
3. **SDI** ve **MDI** form yapılarını açıklayınız ve her iki tür için birer örnek uygulama geliştirin.

## **Modül 11: Veri Yapıları**

### **Hedefler**

- ➔ Access ortamı
- ➔ Veri tipleri
- ➔ Veri modelleme teknikleri

Birçok şirket, kurum ve kayıtlarını tutan yapılar için verinin önemi çok büyüktür. Verilerin kâğıt üzerinde tutulması hem aramaların yapılması hem de kayıt düzeni açısından çok zor bir yöntemdi. Bilgisayarların iş yaşamında kullanılmaya başlanması ile verilen yönetimi daha da kolaylaştı. Ancak bu teknoloji ilerledikçe kullanılması zorlaşmaya başladı. Verilerin tutulması metin dosyalarından tablolara aktarıldı. Günümüzde veri ve tablo yapılarının yönetimi artık veritabanı yöneticilerin eline bırakılmış durumdadır.

Windows ve Web uygulamaların çoğu veri üzerine yoğunlaşır. Uygulamalarda veriye hızlı bir şekilde ulaşmak ve veriyi yönetmek için tablo yapılarının iyi bir şekilde modellenmesi gerekmektedir. Bu modülde Microsoft Access veritabanı üzerinde veri yapılarının kullanılması işlenir.

Bu modülü tamamladıktan sonra:

- Microsoft Access ortamını tanıyacak,
- Veritabanlarında kullanılan değişik veri tiplerini tanıyacak,
- Veri modelleme tekniklerini öğreneceksiniz.

## Konu 1: Access' e Giriş

Access Microsoft'un ilişkisel veritabanıdır. İçindeki birçok sihirbaz yardımını ile kullanım kolaylığı ve hızlı bir şekilde tablo tasarımının yapılmasını sağlar. Access tasarım görünümlerinde, tabloların yapısını analiz etmek için sorguları kolay bir şekilde oluşturma işlemini kolaylaştırır. Karmaşık bir dosya yapısı olmaması taşınabilirliğini kolaylaştırır ve her platformda çalışmasını sağlar.

### Access Ortamı

#### Access ortamı

- Görev Bölmesi
  - Başlangıç
  - Yardım
  - Arama Sonuçları
  - Dosya Arama
  - Yeni Dosya
- Tablo oluşturmak
  - Tasarım görünümünde tablo
  - Sihirbaz ile tablo
  - Veri girerek tablo

Access ortamı, veritabanı geliştirirken kullanıcıya birçok kolaylık sunar. Access açıldığı zaman sağ panelde “Görev Bölmesi” çıkar. Bu panel birçok işleme kısa yol sağlar.

- Başlangıç

Access Office Online başlangıç sayfasıdır. Microsoft haber sitelerine bağlantıları ve en son açılan veritabanlarını listeler.

- Yardım

Online yardım seçeneklerini sunar

- Arama sonuçları

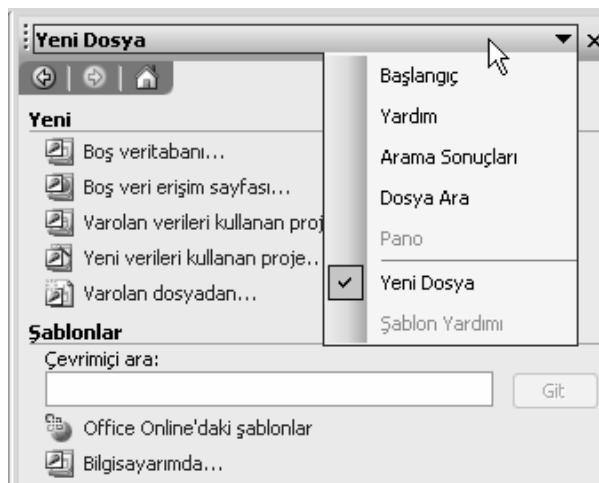
Online yardımda bulunan sonuçları listeler

- Dosya Arama

Belirtilen yerde, belli tipte dosyaları aramayı sağlar.

- Yeni Dosya

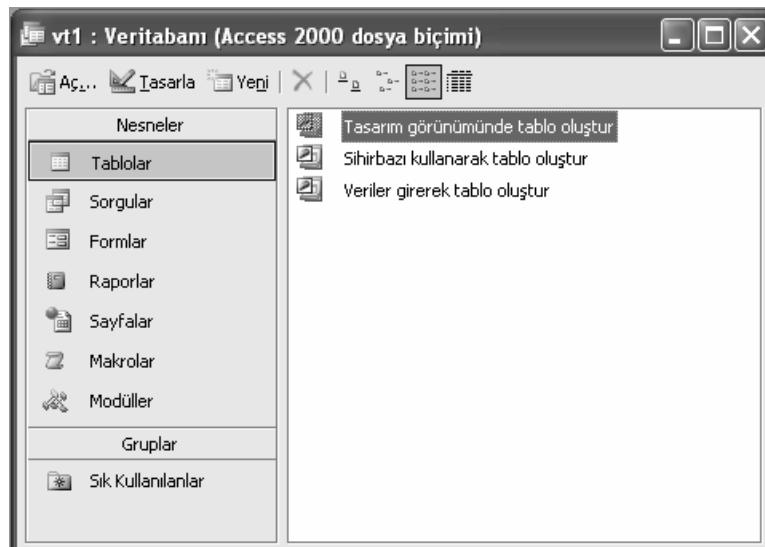
Yeni bir veri tabanı dosyası veya veri erişim dosyası açmak için kullanılır.



Boş veritabanı komutu verildiği zaman “Yeni Veritabanı Dosyası” diyalog kutusunda, dosya ismi girilip yeni veritabanı oluşturulur. Oluşturulan veritabanı dosyalarının uzantısı **mdb** olur.

Daha önceden oluşturulmuş bir veritabanını açmak için Dosya menüsünden Açı komutu verilir. **Ctrl + O** kısa yolu da dosyaları açmak için kullanılabilir.

Veritabanı açıldığı zaman, veritabanı üzerinde yapılabilecek tüm işlemleri sunan bir pencere çıkar. Veritabanı nesnelerini yönetilmesi bu pencere ile yapılır. Sol panelde bulunan nesneler sekmesinde, veritabanında bulunabilecek tüm nesneler listelenmiştir. Bir nesne tipi seçildiğinde, veritabanında bulunan bu tipteki tüm öğeler görüntülenir. Örneğin Tablolar sekmesine gelindiğinde veritabanı üzerindeki tablolar görüntülenir, yeni tablo oluşturmak için seçenekler sunulur.

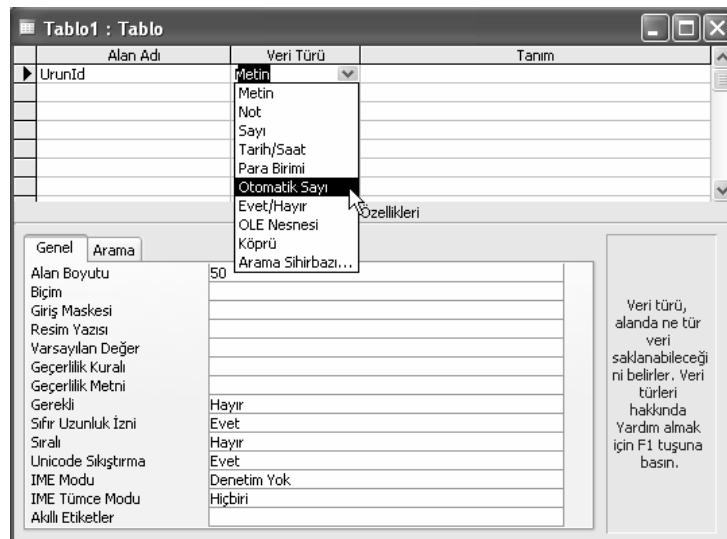


## Veritabanı Nesnesi oluşturmak

Veritabanı penceresinde nesneleri oluşturmak için farklı yollar sunulmuştur. Tabloları oluşturmak için bu kısa yollardan yararlanılabilir.

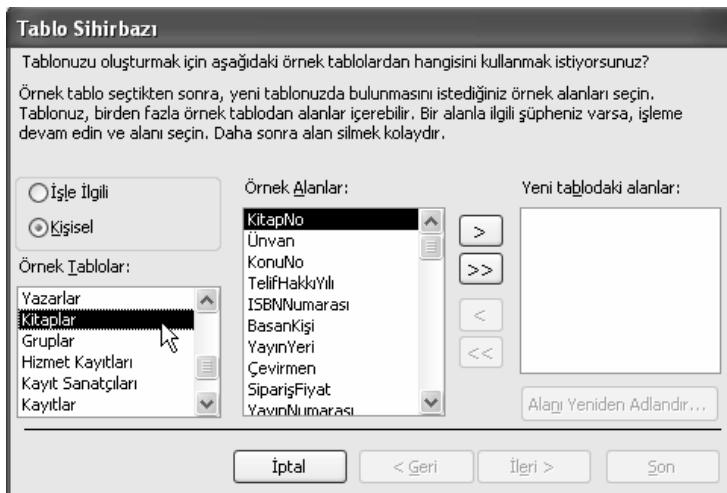
- Tasarım görünümünde tablo oluştur

Bu seçenek ile tablodaki verilerin tasarımını tamamen kullanıcıya bırakılmıştır. Kullanıcı alan adlarını kendisi girip, ilgili veri tipini ve gerekli ayarları seçebilir.



- Sihirbaz kullanarak tablo oluştur

Access içinde çok sık karşılaşılan, kullanıcıya büyük hız sağlayan sihirbaz yardımı ile tablo oluşturulur. Sihirbaz, hangi tipte tablo oluşturulacağını, önceden hazırlanmış zengin şablonları kullanıcıya sunarak belirler.



- Veriler girerek tablo oluştur

Bu seçenek ile tablolar veri girişi ile oluşturulur. Access kullanıcının girdiği verilere göre alan sayısını ve tipi belirler. Ancak alan adlarını daha sonradan değiştireilmelidir.

The top part shows the 'Table1 : Tablo' properties dialog with columns for 'Alan Adı' (Column Name), 'Veri Türü' (Data Type), and 'Tanım' (Description). The bottom part shows the 'Table1 : Tablo' data grid with six columns labeled Alan1 through Alan6, containing sample data like 'Enis', 'Günesen', etc.

| Alan Adı | Veri Türü     | Tanım |
|----------|---------------|-------|
| Kimlik   | Otomatik Sayı |       |
| Alan1    | Metin         |       |
| Alan2    | Metin         |       |
| Alan3    | Tarih/Saat    |       |
| Alan4    | Evet/Hayır    |       |
| Alan5    | Para Birimi   |       |

| Alan1 | Alan2   | Alan3      | Alan4 | Alan5   | Alan6 |
|-------|---------|------------|-------|---------|-------|
| Enis  | Günesen | 27.03.2005 | Evet  | 1000 TL |       |
|       |         |            |       |         |       |
|       |         |            |       |         |       |
|       |         |            |       |         |       |
|       |         |            |       |         |       |
|       |         |            |       |         |       |

Oluşturulan tablolar tasarım ve veri sayfası görüntülerinde incelenebilir. Veri sayfası görünümü kullanıcıya veri girmesi için büyük kolaylıklar sağlar.

Örneğin Evet/Hayır veri tipindeki bir alan veri girilmesi için bir **CheckBox** görüntülenir. Ayrıca tablonun ilişkide olduğu tablolar bulunur ve alt tablo olarak kullanıcı sunulur.

This screenshot shows a hierarchical view of three tables: Categories, Products, and Orders. The Categories table has one record for 'Beverages'. The Products table has several records under 'Beverages', including 'Chai'. The Orders table lists purchases for 'Chai', with details like Order ID, Unit Price, Quantity, and Discount.

| Category ID | Category Name |
|-------------|---------------|
| 1           | Beverages     |

| Product ID | Product Name |
|------------|--------------|
| 1          | Chai         |

| Order ID | Unit Price | Quantity | Discount |
|----------|------------|----------|----------|
| 10285    | \$14,40    | 45       | 20%      |
| 10294    | \$14,40    | 18       | 0%       |
| 10317    | \$14,40    | 20       | 0%       |
| 10348    | \$14,40    | 15       | 15%      |
| 10354    | \$14,40    | 12       | 0%       |

Tablolar oluşturulduktan sonra aralarındaki ilişkilerin kurulması ve görüntülenmesi için araç çubuğuunda "İlişkiler" düğmesi kullanılır.



## Konu 2: Veri Yapılarına Giriş

### Veri Yapıları

- Metin Veri Tipleri
  - Text, Memo
- Sayısal Veri Tipleri
  - Byte, Integer, Long Integer
  - Single, Double, Decimal
- Tarih Veri Tipi
  - Genel Tarih, Uzun Tarih, Kısa Tarih
  - Orta Uzunlukta Tarih, Uzun Saat, Kısa Saat
- Yes/No Veri Tipi
- OLE Veri Tipi

Veritabanlarında veriler aynı tipinde tutulmaz. Bu durum küçük veriler için fazla yer alanları açmayı engellediği gibi değişik formatlardaki verilerin yönetilebilirliğini artırır. Örneğin kategori tablosunda tutulan verilerin sayısı genellikle azdır ve çok fazla artmaz. Dolayısıyla bu verilerin tekil alanında tutulan sayının çok büyük veri tipinde olması gerekmekz. Ancak makalelerin tutulduğu bir alanın kapasitesinin çok büyük olması gerekir.

### Metin Veri Tipleri

- Metin (Text)

Metin bilgilerini tutmak için tanımlanan veri tipidir. Bu değere girilebilecek maksimum karakter sayısı 255 tır. Bir alana belirtilen uzunluktan küçük bir değer girildiğinde, kalan boş yerler için kaynak ayrılmaz. Metin veri tipi sayısal değerler de içerebilir.

- Not (Memo)

Maksimum 65535 karakter tutar. Büyük metinsel veriler için tercih edilmelidir.

## Sayısal Veri Tipleri

Sayı veri tipinin birden fazla alan boyutu vardır.

- Bayt (**Byte**)

0 – 255 arasında bir sayı

- Tamsayı (**Integer**)

- 32,768 ile 32,767 arasında bir sayı

- Uzun Tamsayı (**Long Integer**)

- 2,147,483,648 ile 2,147,483,647 arasında bir sayı

- Tek (**Single**)

Negatif sayı aralığı: -3.402823E+38 ile -1.401298E-45

Pozitif sayı aralığı: 1.401298E-45 ile 3.402823E38

- Çift (**Double**)

Negatif sayı aralığı: 1.79769313486231E+ 308

-4.94065645841247E-324

Pozitif sayı aralığı: 1.94065645841247E-324

1.79769313486231E+308

- Ondalık (**Decimal**)

-10^38-1 ile 10^38-1 arasında sayı

Otomatik Sayı (**AutoNumber**) veri tipi, alana veri girildiği zaman otomatik olarak belirlenen sayıları ifade eder. Sayılar rasgele ya da birden başlayarak girilir.

## Tarih Veri Tipi

Tarih alanları için değişik boyutlarda depolama seçenekleri sunar.

- Genel Tarih

Kısa Tarih ve Uzun Saat birleşimi bir görünümündür.

- Uzun Tarih

12 Aralık 2004 Pazar formatında görünür

- Orta Uzunlukta Tarih

12 Ara 2004 formatında görünür

- Kısa Tarih

12.12.2004 formatında görünür

- Uzun Saat

15:11:19 formatında görünür

- Kısa Saat

15:11 formatında görünür

## Evet/Hayır Veri Tipi

Bir bit değerinde, evet ve hayır değerlerini alan veri tipidir. Veri sayfalarında veya sorgu sonucunda bir **CheckBox** ile ifade edilir. Eğer seçili ise bu alanın değeri -1, değilse 0 olur. Bu alan sorgulanırken -1 ve 0 değerleri kontrol edilmelidir.

## OLE Veri Tipi

Alana bir nesne eklemek veya bağlamak için kullanılan veri tipidir. Resimler, Excel dosyalarını veya bir dosyadan seçebileceğiniz herhangi bir nesne bağlanabilir.

## Konu 3: Veri Modelleme Gereksinimleri

### Veri Modelleme Gereksinimleri

- Normalizasyon
  - Birinci Normal Form
  - İkinci Normal Form
  - Üçüncü Normal Form
- Primary Key
- Foreign Key
- İlişkiler
  - Bire Bir
  - Bire Sonsuz
  - Sonsuza Sonsuz

Verileri tablolarda tutarken bazı modellemelere gereksinim duyulur. Örneğin yazılan bir verinin tekrarlamaması önemlidir. Ürünler tablosunda kategori isim olarak tutulursa, aynı kategorideki ürünleri için bu isim tekrardan yazılması gerekecektir. Bu durum, tabloya hem veri girişi zorlaştırmır, hem de değişiklik yapılmak istenirse her ürünün kategorisini değiştirmek gereklidir. Bu tip sorunlar normalizasyon kurallarını ortaya çıkarmıştır.

Normalizasyon, yer alanından kazanma, veri tutarlılığı ve ölçülebilirlik amacıyla tablolardan gereksiz verilerin çıkartılması işlemleridir. Bu işlemler, tabloların üç etapta normal formlara getirilmesi ile gerçekleşir.

## Birinci Normal Form

### Birinci Normal Form

- Yatay düzeyde gereksiz veri tekrarı yapılmaz.
- Bir kolonda sadece bir veri tutulur.
- Tekrarlanan veriler için ayrı bir tablo oluşturulur.

Bu işlem, yatay düzeyde gereksiz veya tekrarlanan verilerin çıkartılmasıdır. Satırlarda en az düzeyde veri tutulması ve bir bilginin sadece bir kolonda bulunması sağlanır.

Örnek:

Bu örnekte bir eğitmen grubunun yaptığı projeler bir veritabanında tutulur. Verilerin tek bir tabloda tutulması bazı problemlere yol açacaktır.

| Eğitmen1 | Eğitmen2 | Proje              | Konu         | Saat | Kurum     |
|----------|----------|--------------------|--------------|------|-----------|
| Ali      | Veli     | Uzmanlık Kitabı    | Windows      | 300  | BilgeAdam |
| Ali      | Veli     | Mühendislik Kitabı | Windows, Web | 350  | BilgeAdam |

Bu örnekte, projelerin eğitmenleri iki ayrı alanda tutulmuştur. Bu durum 1NF (birinci normal form) kuralını ihlal etmiştir. Yani bir satırda, bir verinin tekrar etmesi söz konusudur. Bu tabloda projeleri iki eğitmen ile sınırlanmış oluyor. Ancak bir kitabı birçok eğitmenin yazdığı durumlar da olabilir.

Ayrıca proje konularında birden fazla bilgi tutulur. Mühendislik Kitabı projesinin Windows ve Web olmak üzere iki tane konusu bulunur. Belli bir konuya göre arama yapmak zorlaşır.

Eğitmenler tek bir alanda toplanıp, konular kitaplara göre tekrar düzenlenebilir.

| Eğitmen | Proje              | Konu    | Saat | Kurum     |
|---------|--------------------|---------|------|-----------|
| Ali     | Uzmanlık Kitabı    | Windows | 300  | BilgeAdam |
| Veli    | Uzmanlık Kitabı    | Windows | 300  | BilgeAdam |
| Ali     | Mühendislik Kitabı | Web     | 350  | BilgeAdam |
| Veli    | Mühendislik Kitabı | Web     | 350  | BilgeAdam |

Yeniden düzenlenen bu tabloda ise bir kitap projesi için iki tane satır oluşuyor. Ayrıca Mühendislik kitabının sadece Web konusunda olduğu görülmüyor. Diğer konu için de ayrıca iki satır eklenmesi gereklidir.

| Eğitmen | Proje              | Konu    | Saat | Kurum     |
|---------|--------------------|---------|------|-----------|
| Ali     | Uzmanlık Kitabı    | Windows | 300  | BilgeAdam |
| Veli    | Uzmanlık Kitabı    | Windows | 300  | BilgeAdam |
| Ali     | Mühendislik Kitabı | Web     | 350  | BilgeAdam |
| Veli    | Mühendislik Kitabı | Web     | 350  | BilgeAdam |
| Ali     | Mühendislik Kitabı | Windows | 350  | BilgeAdam |
| Veli    | Mühendislik Kitabı | Windows | 350  | BilgeAdam |

Ancak bu şekilde verilerin gereksiz yere tekrarlandığı görülür. Veriler bu şekilde tekrar yazıldıkları zaman hataılma olasılığı artar. Dolayısıyla veri bütünlüğü bozulur. Örneğin Mühendislik Kitabı yerine Muhendis Kitabı gibi bir veri girildiği zaman, alınacak raporlarda çelişkiler meydana gelir.

Dolayısıyla bu tekrarlanan verilerin ayrı bir tabloda tutulması gereklidir.

| Eğitmen No | Eğitmen |
|------------|---------|
| 1          | Ali     |
| 2          | Veli    |

| Konu No | Konu    |
|---------|---------|
| 500     | Windows |
| 501     | Web     |

| Proje No | Proje              | Saat | Kurum     |
|----------|--------------------|------|-----------|
| 100      | Uzmanlık Kitabı    | 300  | BilgeAdam |
| 101      | Mühendislik Kitabı | 350  | BilgeAdam |

Eğitmenler ve konular tablosundaki verilerin birer numarası vardır. Bu verilere erişmek için konu veya eğitmenin ismiyle değil, numara ile ulaşılacaktır. Dolayısıyla tablolarda onlarca karakterin tekrarlanması yerine, verileri temsil eden numaralar tekrarlanacaktır. Bu durum hem veritabanının büyümesini engeller hem de tablo üzerinde kayıt aramalarını hızlandırır.

Tablolar birbirinden ayrıldıktan sonra projelerin hangi eğitmenler tarafından yapıldığı ve hangi konularda olduğu bilgileri kaybedilmiştir. Bu bilgilerin elde edilmesi için tablolar arasında ilişkiler kurmak gereklidir.

İlişkilerin kurulması için tabloların, birbirlerine referans vermesi gereklidir. Yani bir tablodan diğerine ulaşmak için bir bilgi gereklidir. Örneğin bir projenin hangi konuda olduğunu belirlemek için, konu numarasına ihtiyaç vardır. Bu numara, projenin hangi konuda olduğunu belirleyecektir.

Tablolar arasında ilişkileri kurmak için bu numaraların doğru biçimde kullanılması gereklidir. Bu numaralar davranışlarına göre ikiye ayrılır.

## Birincil Anahtar (Primary Key)

### Primary Key

- Bir ya da birden fazla alan Primary Key yapılabilir.
- Alanlardaki veriler tekrarlanamaz.

Tablonun bir ya da birden fazla alanı, tek bir veriyi temsil etmesi için **Birincil Anahtar** yapılır. Bu anahtar verinin bir daha tekrarlanmamasını sağlar ve ilişkiler kurulurken ana tabloyu belirler.

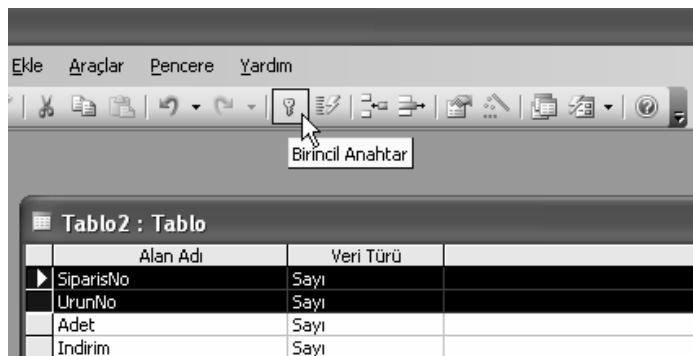
Örnekteki **Birincil Anahtar** olan alanlar Eğitmen No, Proje No ve Konu No alanlarıdır.

Birden fazla alanın **Birincil Anahtar** olarak tanımlanması, alanların tuttuğu verinin birleşik olarak tekliğini sağlar. Örneğin sipariş detayları tablosunda, sipariş numarası ile ürün numarasının beraber tekrarlanmaması gereklidir. Aksi halde bir siparişteki ürünün iki farklı adet, indirim vs. bilgileri olacaktır.

| Sipariş No | Ürün No    | Adet     | İndirim  |
|------------|------------|----------|----------|
| <b>100</b> | <b>680</b> | <b>1</b> | <b>0</b> |
| 100        | 679        | 2        | 10       |
| 102        | 680        | 1        | 15       |
| <b>100</b> | <b>680</b> | <b>2</b> | <b>5</b> |

Bu tabloda sipariş ve ürün numarası beraber **Birincil Anahtar** yapılmıştır. Dolayısıyla bu alanların herhangi birisinde bir veri tekrarı olabilir. Önemli olan bu iki alanın beraber aynı veri tutmamasıdır. Örneğin 100 numaralı siparişte 680 numaralı ürün kaydı iki defa geçmiştir. Yapılacak sorgularda, bu ürünün siparişte 1 adet olduğu ve 0 YTL indirim yapıldığını, aynı zamanda 2 adet olduğunu ve indirim 5 YTL olduğu görülür. Bu da verinin tutarlılığını bozar.

Access ile tablolarda **Birincil Anahtar** oluşturmak için, istenen alanlar seçilerek araç çubuğundaki **Birincil Anahtar** düğmesine basılır.



## Yabancı Anahtar (Foreign Key)

### Foreign Key

- Başka bir tablonun Primary Key alanına referans gösterir
- İlişkideki Primary Key üzerinde güncelleme ve silme işlemleri, bu alanda da yapılabilir.
  - Cascade Update
  - Cascade Delete
- İlişkilerde, Foreign Key alanındaki değer kontrol edilebilir
  - Enforce Referential Integrity

Bir tablo içinde başka bir tabloya referans vermek için, o tablonun numarası kullanılır. Yani o tablonun **Birincil Anahtar** alanına gönderme yapılır. Bu işlemin yapılması için, referans gönderen tabloda bu verinin tutulması gereklidir. Farklı bir tablonun birincil anahtarını tutan alana **Yabancı Anahtar** denir. Örneğin, şarkılının tutulduğu bir tabloda albüm numarası, albümler tablosundaki **Birincil Anahtar** olan alana referans verir.

| Alan Adı   | Veri Türü     |
|------------|---------------|
| Sarki_Id   | Otomatik Sayı |
| Sarki_Ismi | Metin         |
| Album_Id   | Sayı          |

| Alan Adı   | Veri Türü |
|------------|-----------|
| AlbumId    | Sayı      |
| Album_Ismi | Metin     |

Bu anahtarların kullanımı ilişkilerin tanımlanmasında büyük öneme sahiptir. Tabloların normalizasyonunun sağlanması için birbirleriyle ilişkilendirilmeleri gereklidir. Üç çeşit ilişki vardır.

1. Bire bir ilişki (**One to One**)
2. Bire sonsuz ilişki (**One to Many**)
3. Sonsuza sonsuz ilişki (**Many to Many**)

Access ile tablolar arasındaki ilişkiler, bir alanının sürüklendiğinde diğer tablodaki bir alanın üzerine bırakılması ile kurulur. Access bu alanların **Birincil Anahtar** olup olmadığına bakarak ilişkinin cinsini belirler.



İlişki tanımlanırken çıkan ilişkileri Düzenle penceresinde, tablolardaki hangi alanlar üzerinde ilişki kurulacağı gösterilir. Buradan ilişkinin türü davranışı hakkında özel ayarlamalar yapılır.

- Bilgi Tutarlılığına Zorla (**Enforce Referential Integrity**)  
Bir tablodaki verinin diğer tabloda var olup olmadığını kontrol eder.
- İlişkili Alanları Ardarda Güncelle (**Cascade Update**)  
**Birincil Anahtar** üzerinde bir değişiklik yapılmışsa, ilişkide olduğu tablolardaki **Yabancı Anahtar** alanları da değiştirir.
- İlişkili Kayıtları Ardarda Sil (**Cascade Delete**)  
Tabloda bir kayıt silindiği zaman, ilişkide olduğu tablolardaki veriler de silinir.



## Tekil Kısıtı (Unique Constraint)

### Unique Constraint

- Primary Key dışındaki alanların tekil olması
- Unique tanımlanırken alan indekslenir.

Bazı durumlarda, **Birincil Anahtar** olmayan alanların bazlarının da tabloda birden fazla geçmesi istenmez. Örneğin öğrenci tablosundaki bir numara başka bir öğrenci için geçerli değildir. Ya da sicil tablosundaki bir TC kimlik numarası da tekrarlanmaz. Bu alanların **Tekil** olarak tanımlanması gereklidir.

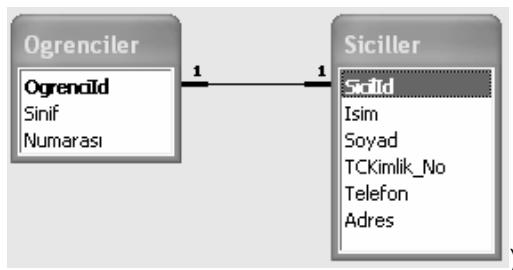
Access ile tablo tasarlarken, alanların **Tekil** olarak tanımlanması indekslemeyi gerektirir. Bir alanın indekslenmesi, tabloda aramaların o alan üzerinden daha hızlı yapılmasını sağlar. Ancak her alan üzerinde indeks kullanılmamalıdır. Bu durum sorguların performansını artırmak yerine düşürür. Üzerinde sıkça sorgu çalıştırılan alanlar indekslenebilir.

| Genel              | Arama                                                                                                                                                                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Alan Boyutu        | 50                                                                                                                                                                                                                                                          |
| Bigim              |                                                                                                                                                                                                                                                             |
| Giriş Maskesi      |                                                                                                                                                                                                                                                             |
| Resim Yazısı       |                                                                                                                                                                                                                                                             |
| Varsayılan Değer   |                                                                                                                                                                                                                                                             |
| Geçerlilik Kuralı  |                                                                                                                                                                                                                                                             |
| Geçerlilik Metni   |                                                                                                                                                                                                                                                             |
| Gerekli            |                                                                                                                                                                                                                                                             |
| Sıfır Uzunluk İzni |                                                                                                                                                                                                                                                             |
| Sıralı             |                                                                                                                                                                                                                                                             |
| Unicode Sıkıştırma |                                                                                                                                                                                                                                                             |
| IME Modu           |                                                                                                                                                                                                                                                             |
| IME Türkçe Modu    |                                                                                                                                                                                                                                                             |
| Akıllı Etiketler   |                                                                                                                                                                                                                                                             |
|                    | <input type="button" value="Hayır"/> <input type="button" value="Evet"/> <input type="button" value="Hayır"/><br><input type="button" value="Hayır"/> <input type="button" value="Evet (Yineleme Var)"/> <input type="button" value="Evet (Yineleme Yok)"/> |

## Bire bir ilişki

Bir tablodaki bir kayıt, diğer tablodaki bir veri için ancak bir kez kullanılabilir. Örneğin sicil tablosu, bir kişinin ismini, soyadını ve kimlik numarasını tutuyor. Öğrenci tablosu ise öğrencinin okul numarası, sınıfı gibi kayıt bilgilerini tutuyor. Öğrenci ile sicil arasında bire bir ilişki vardır. Öğrenci tablosundaki bir veri, sicil tablosunda sadece bir referans gösterebilir. Sicil tablosundaki bir veri de, öğrenci tablosundaki bir veri için kullanılabilir. Dolayısıyla bir öğrencinin bir sicili olabilir, bir sicil ise sadece bir öğrenciye ait olabilir.

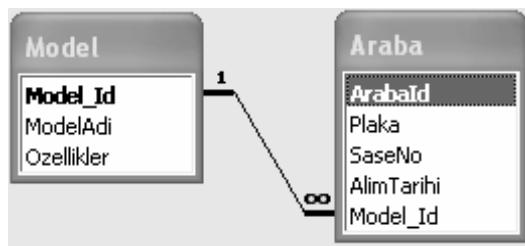
Tablolar arasındaki bu ilişkiler iki **Birincil Anahtar** üzerinden yapılır.



## Bire sonsuz ilişki

Tablodaki bir verinin, ilişkide olduğu tabloda birden fazla kullanılabilir. Örneğin bir araba ve model tabloları arasında bire sonsuz bir ilişki vardır. Araba tablosundaki bir veri, model tablosundaki bir veriyi bir kez kullanabilir. Ancak model tablosundaki bir veri, araba tablosunda birden fazla veri tarafından kullanılabilir. Yani bir arabanın sadece bir modeli olur ve bir model birden fazla arabanın modeli olabilir.

Tablolar arasında bire sonsuz bir ilişki oluşturmak için, birden fazla veride geçecek olan tabloda **Birincil Anahtar**, bu değerin bir kere tutulacağı tabloda **Yabancı Anahtar** olmak zorundadır.



## Sonsuza sonsuz ilişki

İki tablo arasında sonsuza sonsuz bir ilişkiyi temsil eder. Tablolardaki her veri diğerine için birden fazla kullanılıyorsa, iki taraflı sonsuz bir ilişki vardır. Örneğin bir film ve oyuncu tabloları arasındaki ilişki sonsuza sonsuzdur. Film tablosundaki bir veri, oyuncular tablosunda birden fazla veri için kullanılabilir. Aynı şekilde oyuncu tablosundaki bir veri, filmler tablosunda birden fazla veri için kullanılabilir. İlişki şu şekilde tanımlanabilir:

Bir oyuncu birden fazla filmde oynayabilir. Bir filmde birden fazla oyuncu bulunabilir.

Tablolarda sonsuza sonsuz bir ilişki kurmak için, ara tabloya ihtiyaç duyulur. Bunun nedeni, her iki tablodaki verilerin birden fazla eş bulunabilir olmasıdır. Yapılan ara tabloda, iki tablodan alınan **Birincil Anahtar** alanları konur. Bu alanlar ikili **Birincil Anahtar** yapılarak veri bütünlüğü sağlanmış olur.



Tablolardan birinci normal forma getirilmesi için ilişkilerin kurulması gereklidir. Bu durumda, ayrılan tablolardan birbirleri ile ilişkiler saptanması ve bunun sonucunda **Yabancı Anahtar** alanlarının eklenmesi veya ara tabloların oluşturulması gereklidir.

Örneğin, Proje ile konular arasında bir sonsuza sonsuz bir ilişki vardır. Bir projenin birden fazla konusu olabilir ve bir konuda birden fazla proje yapılabilir. Bunun için ara tablonun kurulması gereklidir.

| Konu No | Konu    |
|---------|---------|
| 500     | Windows |
| 501     | Web     |

| Proje No | Proje              | Saat | Kurum     |
|----------|--------------------|------|-----------|
| 100      | Uzmanlık Kitabı    | 300  | BilgeAdam |
| 101      | Mühendislik Kitabı | 350  | BilgeAdam |

| Proje No | Konu No |
|----------|---------|
| 100      | 500     |
| 101      | 500     |
| 101      | 501     |

Bu tablo ile 100 numaralı Uzmanlık Kitabı projesinin 500 numaralı Windows konusunda olduğu görülür. Bu tablo biçimini, belli konulardaki projelerin sorgulanmasını da destekler.

Eğitmenler ile projeler arasında da sonsuza sonsuz bir ilişki vardır. Bir eğitmen birden fazla projede bulunabilir. Bir projeyi birden fazla eğitmen yürütebilir. Dolayısıyla bu ilişki için de bir ara tablo yapılması gereklidir.

| Eğitmen No | Proje No |
|------------|----------|
| 1          | 101      |
| 2          | 101      |
| 1          | 100      |
| 2          | 100      |

## İkinci Normal Form

- ### İkinci Normal Form
- Kolon düzeyinde veri tekrarı yapılmaz.
  - Kolonlarda tekrar edilen veriler ayrı bir tabloda tutulur.

Birinci normal form satır bazında gereksiz verilerin çıkartılmasıydı. İkinci normal form ise kolon bazında veri tekrarını kontrol eder. Eğer bir kolonda bir veri, birden fazla kullanılıyorsa bu verilerin ayrı bir tabloda tutulması gereklidir.

Örnekte kurum ismi BilgeAdam, tüm satırlar için yazılmıştır. Dolayısıyla bu kolonda veri tekrarı yapılmıştır. Bu kurum ismi ayrı bir tabloda tutulup, ana tabloda numarası ile referans gösterilmelidir.

| Kurum No | Kurum İsmi | Şehir    | Adres                           |
|----------|------------|----------|---------------------------------|
| 221214   | BilgeAdam  | İstanbul | Barbaros<br>Bulvarı<br>Beşiktaş |

Bu durumda, projeler ve kurum tablosu arasında bire sonsuz bir ilişki olduğu için, projeler tablosuna hangi kuruma ait olduğunu belirtmek için bir **Yabancı Anahtar** eklenir.

| Proje No | Proje                 | Saat | Kurum No |
|----------|-----------------------|------|----------|
| 100      | Uzmanlık<br>Kitabı    | 300  | 221214   |
| 101      | Mühendislik<br>Kitabı | 350  | 221214   |

## Üçüncü Normal Form

### Üçüncü Normal Form

- Primary Key ile direkt ilişkisi bulunmayan alanlar ayrı bir tabloya alınır.

Üçüncü normal formda tablonun, **Birincil Anahtar** ile direkt ilişkisi bulunmayan, ancak diğer alanlara bağlı alanlar bulunur. Örneğin kurumlar tablosunda şehir ismi alanı, kurum ile doğrudan bağlantısı yoktur. Adres alanı ile daha çok bağlantılıdır. Bu alanların ayrı bir tabloya alınması üçüncü derece normalizasyondur.

Tablolar ayırdıktan sonra aralarındaki ilişkiler belirlenmelidir. Bu örnekte bir kurumun birden fazla adresi olabilir. Ancak bir adres, sadece bir kuruma aittir.

| Kurum No | Kurum İsmi |
|----------|------------|
| 221214   | BilgeAdam  |

| Adres No | Şehir    | Adres                        | Kurum No |
|----------|----------|------------------------------|----------|
| 17982    | İstanbul | Barbaros Bulvarı<br>Beşiktaş | 221214   |

Üçüncü normal forma getirilen tabloların diğer formların da kısıtlarını sağlaması gereklidir. Adres tablosundaki şehirler alanı, her adres için tekrarlanacaktır. Bu da ikinci normal form kuralının ihlali demek olur. Dolayısıyla şehir alanını ayrı bir tablo olarak ayırmak gereklidir.

| Şehir No | Şehir İsmi |
|----------|------------|
| 34       | İstanbul   |

| Adres No | Şehir No | Adres                        | Kurum No |
|----------|----------|------------------------------|----------|
| 17982    | 34       | Barbaros Bulvarı<br>Beşiktaş | 221214   |

(Sehirler Ornek\_Ilişkiler)

## Uygulama: Alışveriş Modeli

Bir e-ticaret internet sitesinin hedefi, ürünlerin büyük kitlelere satışını gerçekleştirmektir. İnternet kullanıcıları bu hedef kitleyi oluşturur. Satılan ürünler, bu kullanıcılarla çeşitli hizmetler sunularak pazarlanmalıdır. Veritabanında ürünlerin tutulması, stok durumlarının ve siparişlerin gözlenmesi kadar kullanıcı kayıtlarının tutulması, yeni kampanyaların açılması, ürünler hakkındaki yorumların tutulması gibi kavamlar da önemlidir. Veritabanının tasarılanmasında bu kavamlar tek tek ele alınıp incelenmelidir.

### Kaynak yönetimi modülü

E-ticaret firmasının ürünlerinin yönetimi, stok, sipariş ve firma bilgilerinden oluşur. Ürünlerin stoklardaki durumları takip edilmeli ve gerektiği zaman firmalardan tedarik edilmeleri gerekir. Dolayısıyla ürünler, firmalar, siparişler, stoklar bu modülde işlenmelidir.

#### Ürünler:

Bu tablo, ürünlerin detaylı bilgilerini tutar. Ürünün ismi, birim fiyatı, eklenme tarihi, özellikleri, üretimde olup olmadığı, incelenme sayısı gibi bilgilerin tutulur. Ürünlerin hangi kategoride oldukları ve sağlayıcı firma bilgileri de tutulmalıdır. Ancak kategori ismi kolon bazında birçok defa tekrarlanacağı için ikinci normalizasyon kuralına göre ayrı bir tabloya alınmalıdır. Sağlayıcı firma bilgileri de aynı şekilde ayrı bir tabloda tutulmalıdır. Bu durumda bire sonsuz bir ilişki oluşur. Yani bir firma birden fazla ürün sağlar, ancak bir ürün sadece bir firma tarafından sağlanır. Dolayısıyla bu iki alan **Yabancı Anahtar** olarak tanımlanmalıdır.

| Urunler : Tablo |               |           |
|-----------------|---------------|-----------|
|                 | Alan Adı      | Veri Türü |
| UrunId          | Otomatik Sayı |           |
| İsim            | Metin         |           |
| KategoriId      | Sayı          |           |
| FirmaId         | Sayı          |           |
| BirimFiyat      | Para Birimi   |           |
| EklemeTarihi    | Tarih/Saat    |           |
| Ozellikler      | Not           |           |
| UretiliyorMu    | Evet/Hayır    |           |
| IncelenmeSayisi | Sayı          |           |
|                 |               |           |

### Firmalar:

Firma bilgileri ayrı bir tablo olarak tutulur. Bilgi olarak adres, müşteri temsilcisi ismi, e-posta ve web sayfası tutulur.

|   | Alan Adı          | Veri Türü     |
|---|-------------------|---------------|
| 1 | FirmaId           | Otomatik Sayı |
|   | İsim              | Metin         |
|   | Adres             | Metin         |
|   | MüşteriTemsilcisi | Metin         |
|   | Email             | Metin         |
|   | WebSayfasi        | Köprü         |
|   |                   |               |
|   |                   |               |

|                  |                     |
|------------------|---------------------|
| Genel            | Arama               |
| Alan Boyutu      | Uzun Tamsayı        |
| Yeni Değerler    | Artan               |
| Biçim            |                     |
| Resim Yazısı     |                     |
| Sıralı           | Evet (Yineleme Yok) |
| Akıllı Etiketler |                     |

## Siparisler:

Ürünler satın alındıktan sonra, sipariş bilgisi olarak kayda geçer. Siparişlerin nakliye ücreti, sipariş verilme ve gönderilme tarihi, havale ile ödeme durumlarında son ödeme tarihi, gönderilecek adres, ödenip ödenmediği ve

siparişin iptal edilip edilmediği gibi bilgileri tutulur. Ayrıca siparişin hangi kayıtlı kullanıcının verdiği de tutmak gereklidir. Bir siparişi sadece bir kullanıcı verebilir ve bir kullanıcı birden fazla sipariş verebilir. Dolayısıyla bir sonsuz bir ilişki oluşturmak için kullanıcı numarası **Yabancı Anahtar** olarak tanımlanmalıdır.

|                  | Alan Adı      | Veri Türü |
|------------------|---------------|-----------|
| ▼ SiparisId      | Otomatik Sayı |           |
| KullaniciId      | Sayı          |           |
| NakliyeUcreti    | Para Birimi   |           |
| SiparisTarihi    | Tarih/Saat    |           |
| SonOdemeTarihi   | Tarih/Saat    |           |
| GonderilmeTarihi | Tarih/Saat    |           |
| Adres            | Metin         |           |
| Odendi           | Evet/Hayır    |           |
| IptalEdildi      | Evet/Hayır    |           |
|                  |               |           |
|                  |               |           |
|                  |               |           |

Siparişler ile ürünler arasında sonsuza sonsuz bir ilişki vardır. Yani bir siparişte birden fazla ürün bulunabilir. Bir kullanıcı aynı anda birden fazla ürün almak isteyebilir. Aynı şekilde bir ürün birden fazla siparişte bulunabilir. Yani bir ürün birden fazla kullanıcıya satılabilir. Bu durumda siparişler ile ürünler arasında ayrı bir tablo yapılması gereklidir.

Bu ara tablo, bir siparişteki bir ürün bilgisini tutacaktır. Dolayısıyla bu tabloyu daha etkin bir şekilde kullanılabılır. Örneğin belli bir siparişte bir üründen kaç tane alındığı ancak bu tabloda tutulabilir. Ve bu ürün, yapılan bir kampanyadan alınmış olabilir. Böylece bu kavıttı kampanya bilgilerinin de tutulması gereklidir.



Stok ile ürünler arasında sonsuza sonsuz bir ilişki vardır. Yani bir stok merkezinde birden fazla ürün bulunabilir ve bir ürün birden fazla stok merkezinde bulunabilir. Bu ilişki için ara bir tablo yapılmalıdır.

|   | Alan Adı | Veri Türü |
|---|----------|-----------|
| ? | StokId   | Sayı      |
| ? | UrunId   | Sayı      |
| ? | Adet     | Sayı      |
| ? | Reserve  | Sayı      |
|   |          |           |
|   |          |           |

## **Müşteri yönetim modülü**

Kaynak planlamaları yapıldıktan sonra, bu kaynakların müşteriye ne şekilde sunulacağına karar verilmelidir. Kullanıcılar internet sitesini kullanırken kendilerine bir hesap açabilirler. Ve siparişlerini bu hesap ile yaptıklarında, kendilerine ait istatistikleri kolayca elde edebilirler. Örneğin bir kullanıcı, en çok hangi kategoride ürünleri satın aldığı sorgulayabilir. Kullanıcılar, siparişlerini vermeden önce ürünlerle ilgili bilgi almak isteyebilir. Bu ürünleri daha önce alan kullanıcıların yazdıkları yorumlardan faydalananmaları için, ürün yorumlarının da tutulması gereklidir. Ayrıca kullanıcıya değişik tarihlerde açılan, belli süreli kampanyaların sunulması e-ticaret sitesinin kullanımını artıracaktır. Kullanıcılar ürünleri incelerken, satın almadan önce sepetlere ekleyebilirler. Böylece siteyi tekrar ziyaret edince, daha önceden inceledikleri ve sepete ekledikleri ürünleri görebilirler.

### Kullanıcılar:

Bu tabloda kullanıcı hakkında bilgiler tutulur. İsim, soyadı, e-posta, kayıt tarihi gibi bilgilerin yanı sıra siteye giriş yapmak için kullanıcı adı ve parolanın da tutulması gereklidir. Bu parolanın değişikliği durumda güvenlik sorusu ve cevabı da ayrıca tutulmalıdır.

|               | Alan Adı      | Veri Türü |
|---------------|---------------|-----------|
| KullanıcıId   | Otomatik Sayı |           |
| İsim          | Metin         |           |
| Soyad         | Metin         |           |
| Kullanıcıİsmi | Metin         |           |
| Parola        | Metin         |           |
| Email         | Metin         |           |
| KayıtTarihi   | Tarih/Saat    |           |
| ParolaSorusu  | Metin         |           |
| ParolaCevabı  | Metin         |           |
|               |               |           |
|               |               |           |
|               |               |           |

#### **Yorumlar:**

Kullanıcıların yaptıkları yorumların bir tabloda tutulması gereklidir. Ancak burada dikkat edilmesi gereken nokta, bir kullanıcının yorum yazması için sisteme giriş yapması gerekmektedir. Dolayısıyla burada kullanıcılar tablosuna bir referans göstermeye gerek yoktur. Yorumları yazan kişileri takma adları, yazdıkları yorumlar, tarih ve verdiği puan tutulmalıdır.

Ayrıca yorumun hangi ürün hakkında yapıldığını belirten ve ürünler tablosuna referans gösteren bir **Yabancı Anahtaralanının** tutulması gereklidir.

|          | Alan Adı      | Veri Türü |                  |
|----------|---------------|-----------|------------------|
| YorumId  | Otomatik Sayı |           |                  |
| UrunId   | Sayı          |           |                  |
| Rumuz    | Metin         |           |                  |
| Açıklama | Not           |           |                  |
| Rating   | Sayı          |           |                  |
| Tarihi   | Tarih/Saat    |           |                  |
|          |               |           | Alan Özellikleri |

**Sepetim:**

Kullanıcıların ürünleri inceledikten sonra sepetlerinde saklaması için oluşturulan bir tablodur. Bu tabloda ürün numarası ve kullanıcı numarasına referans gösterilmelidir. Bu ürünlerin eklenme tarihi ve adeti de tabloda tutulmalıdır.

Kullanıcılar ürünleri, sürekli sepete ekleyip çıkartabilir. Çıkarma işleminde, verinin tablodan silinmesi gereklidir. Ancak bir kaydın sürekli eklenip silinmesi performansı düşürür. Dolayısıyla ürünün sepetten çıkartıldığını belirleyen bir yes/no veri tipinde alan belirlenebilir. Bu alanın değeri evet ise ürün sepetteydi ve kullanıcıya gösterilir. Ürünün tekrar ekleme işleminde ise sadece bu alan güncellenir.

| Sepetim : Tablo |               |               |
|-----------------|---------------|---------------|
|                 | Alan Adı      | Veri Türü     |
| ▼               | SepetId       | Otomatik Sayı |
|                 | KullaniciId   | Sayı          |
|                 | UrunId        | Sayı          |
|                 | Adet          | Sayı          |
|                 | EklenmeTarihi | Tarih/Saat    |
|                 | UrunSepette   | Evet/Hayır    |
|                 |               |               |
|                 |               |               |
|                 |               |               |
|                 |               |               |
|                 |               |               |

Alan  
Ara

Genel    Arama

|               |              |
|---------------|--------------|
| Alan Boyutu   | Uzun Tamsayı |
| Yeni Değerler | Artan        |
| Birim         |              |
| Resim Yazısı  |              |

**Kampanyalar:**

Kullanıcıya sunulan kampanyalar e-ticaret kavramında önemli bir yer alır. Bu kampanyalar bir ya da birden fazla ürünün belli tarihler arasında toplam fiyatta belli bir indirim yapılmasıyla gerçekleşir. Kampanya tablosunda kampanyanın başlangıç bitiş tarihleri, devam edip etmediği, ve yapılan indirim birer alan olarak tutulmalıdır.

| Kampanyalar : Tablo |               |           |
|---------------------|---------------|-----------|
|                     | Alan Adı      | Veri Türü |
| • KampanyaId        | Otomatik Sayı |           |
| İsim                | Metin         |           |
| BitisTarihi         | Tarih/Saat    |           |
| BaslangicTarihi     | Tarih/Saat    |           |
| Indirim             | Para Birimi   |           |
| DevamEdiyor         | Evet/Hayır    |           |

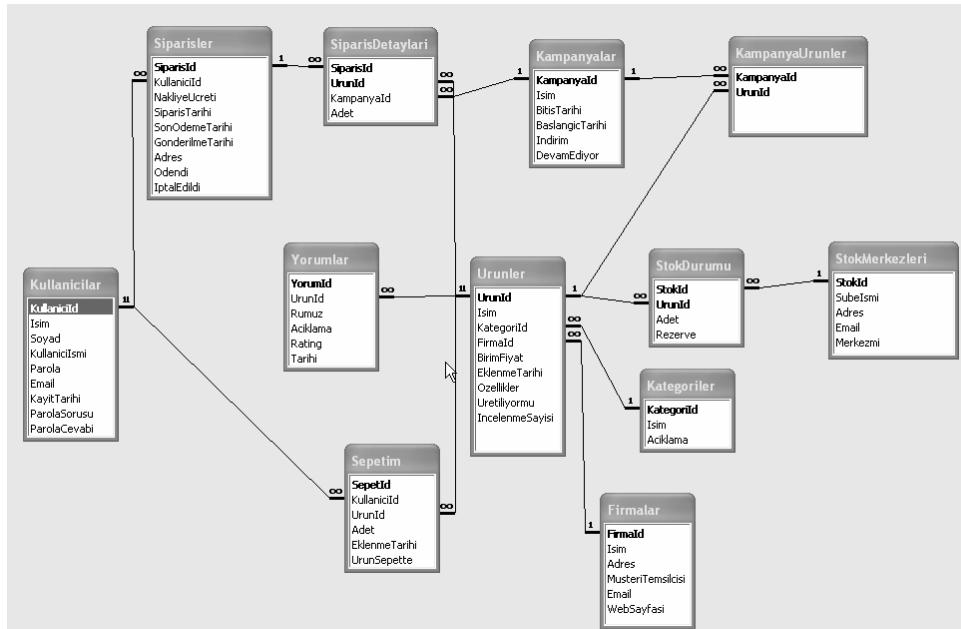
| Genel            | Arama               |
|------------------|---------------------|
| Alan Boyutu      | Uzun Tamsayı        |
| Yeni Değerler    | Artan               |
| Biçim            |                     |
| Resim Yazısı     |                     |
| Sıralı           | Evet (Yineleme Yok) |
| Akıllı Filtreler |                     |

Bu durumda bir kampanyada birden fazla ürün olabilir. Bir ürün ise birden fazla kampanya dâhilinde olabilir. Dolayısıyla ara tablo eklenerek sonsuza sonsuz bir ilişki kurulmalıdır.

| KampanyaUrunler : Tablo |          |           |
|-------------------------|----------|-----------|
|                         | Alan Adı | Veri Türü |
| • KampanyaId            | Sayı     |           |
| • UrunId                | Sayı     |           |

| Genel              | Arama        |
|--------------------|--------------|
| Alan Boyutu        | Uzun Tamsayı |
| Biçim              |              |
| Ondalık Basamaklar | Otomatik     |
| Giriş Maskesi      |              |
| Resim Yazısı       |              |
| Varsayılan Değer   |              |



## Modül Sonu Soruları & Alıştırmalar

### Özet

- ↳ Menüler
  - ↳ MainMenu, ContextMenu
- ↳ ToolBar
- ↳ ToolTip
- ↳ StatusBar
- ↳ NotifyIcon
- ↳ RichTextBox

1. Veritabanı yönetim sistemi kavramını ve bu sistemlere neden ihtiyaç duyulduğunu açıklayın.
2. Microsoft Access platformunun avantajlarını açıklayın.

3. Microsoft Access'te yer alan veri türlerini ve kullanım alanlarını açıklayın.
4. **Birincil Anahtar** ve **Yabancı Anahtar** yapılarını ve kullanım alanlarını açıklayın. Ömek bir veri tabanı geliştirin.

## Modül 12: SQL Giriş

### Hedefler

- ↳ Select cümlesi: Sorgulama
- ↳ Update cümlesi: Güncelleme
- ↳ Insert cümlesi: Veri Ekleme
- ↳ Delete cümlesi: Silme
- ↳ Join: Tabloları birleştirme

SQL dili (**Structured Query Language**), veritabanları üzerinde sorgu yapmak için kullanılan bir dilidir. Sorgular, analiz aşamalarında, veri eklerken güncellerken ve silerken kullanılır. Sorgular tek bir tablo üzerinde yapılabileceği gibi birçok tablodan veri okunmayı sağlar. Sorgular üzerinde konan kriterler, detaylı veri analizi yapmak için kullanılır.

Bu modülü tamamladıktan sonra

- **Select** cümleleri ile tablo sorgulayabilecek,
- Kriterler, hesaplama fonksiyonları kullanarak sorguları şekillendirebilecek,
- **Update** sorgusu ile tabloları güncelleyebilecek,
- **Insert** sorgusu ile tablolara veri ekleyebilecek,
- **Delete** sorgusu ile tablolardan veri silebilecek,
- **Join** ile birden fazla tabloyu birleştirip sorgu çalıştırabileceksiniz.

## Access ile Sorgu Oluşturmak

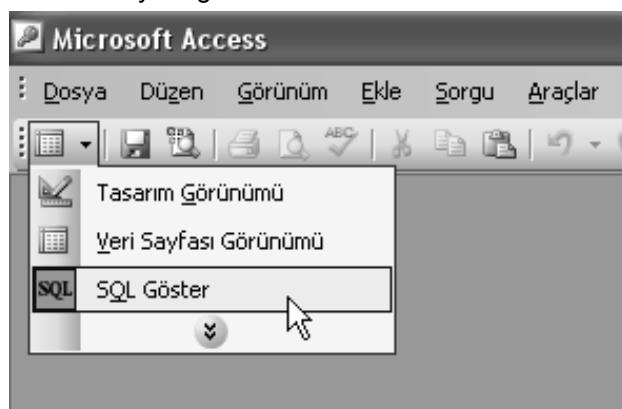
### Access ile sorgu oluşturmak

- Tasarım görünümünde sorgu
- Sihirbaz ile sorgu

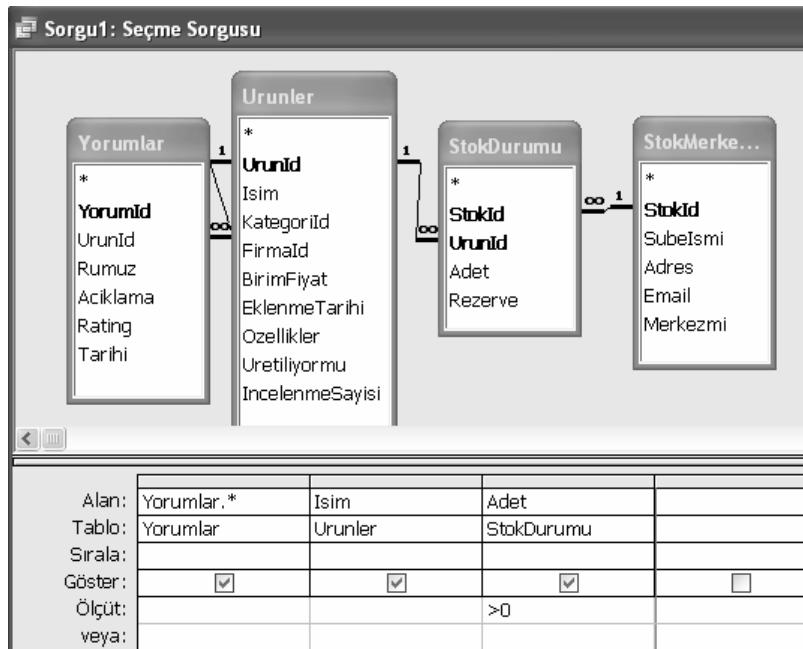
Access ile sorguları görüntülemek, oluşturmak için veritabanı penceresinden sorgular sekmesi seçilir. Sorgular iki şekilde oluşturulabilir.

- Tasarım görünümünde sorgu

Sorgular, istenen tablolar ve gerekli alanlar eklenecek oluşturulur. Burada sorgunun üç farklı görünüm şekli vardır. Tasarım görünümü, SQL görünümü ve Veri sayfası görünümü.



Tasarım görünümünde sorgular, tabloların görsel olarak eklenip, alanlarının seçilmesi ile oluşturulur. Tabloları bağlama işlemleri, kriterler ve alan isimlerinin SQL diline çevrilmesi Access tarafından yapılır.



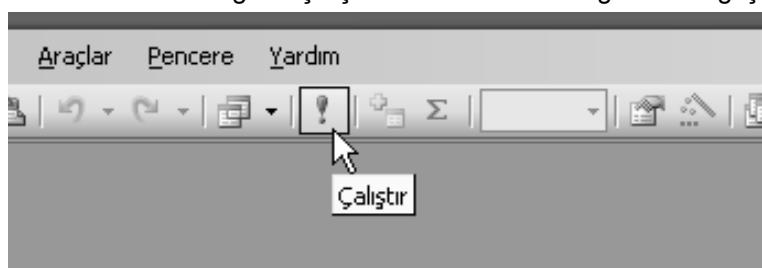
SQL görünümünde sorgular, SQL cümlesinin kullanıcı tarafından yazılarak oluşturulur. Bu modülde sorgular, bu görünümde oluşturulacaktır.

```

SELECT Yorumlar.* , Urunler.Isim , StokDurumu.Adet
FROM StokMerkezleri INNER JOIN ((Urunler INNER JOIN Yorumlar ON Urunler.UrunId = Yorumlar.UrunId) INNER JOIN StokDurumu ON Urunler.UrunId = StokDurumu.UrunId) ON StokMerkezleri.StokId = StokDurumu.StokId
WHERE ((StokDurumu.Adet)>0);

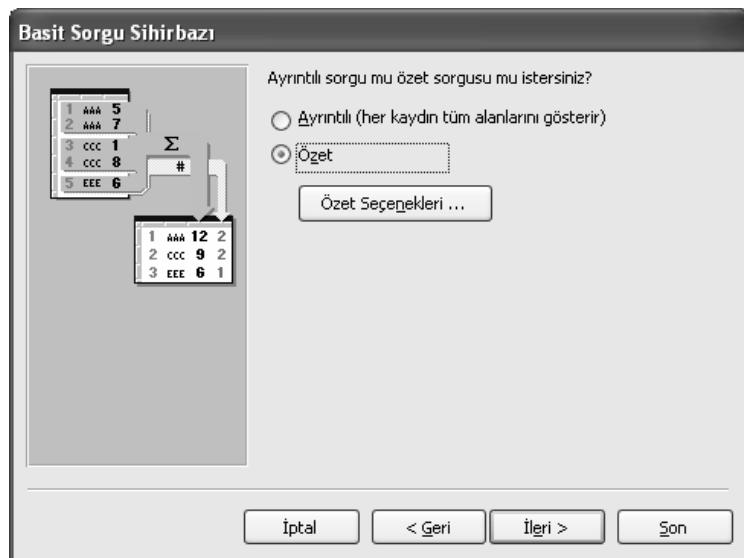
```

Veri sayfası görünümünde, SQL sorgusunu çalıştırıldıktan sonra verilerin görünümüdür. SQL sorguları çalıştırıldıktan sonra da bu görünümde geçilir.



- Sihirbaz ile sorgu

Access sihirbazı, tablolar üzerinde yapılacak sorguların kolay ve hızlı bir şekilde oluşturulmasını sağlar.



## Select From Where

### Select Sorgusu

- Tablolardan veri çekmek için kullanılır.
- From ile tablolar belirtilir.
- Where ifadesinden sonra kriterler yazılır.

```
SELECT Alanlar
FROM Tablo İsmi
WHERE Kriterler
```

```
SELECT Urunler.Isim, Urunler.BirimFiyat
FROM Urunler
WHERE Urunler.Isim LIKE '*Studio*'
```

**Select** sorgusu tablolardan veri kümesi çekmek için kullanılan sorgudur. Sorgunun yapısı **Select** Alanlar **From** Tablo İsmi **Where** Kriterler şeklindedir. Bu cümlede **Select** kelimesinden sonra gelen alanlar, tabloları oluşturulan kolonlardır. Sonuç kümesinde, tablonun hangi alanları olacağını gösterir. Burada yapılan kolon bazında filtrelemedir.

**From** ifadesi, sorgunun hangi tablo veya tablolar üzerinde yapılacağını gösterir. **Where** ifadesinden sonra, sorgu kümesinde, verilen kriterle uygun satırlar görüntülenir. Bu yapılan satır bazında filtrelemedir.

```
SELECT * FROM Urunler
```

Buradaki yıldız ifadesi, tüm alanların listeleneceği anlamına gelir.

```
SELECT
Urunler.Isim,
Urunler.BirimFiyat,
Urunler.EklenmeTarihi
FROM Urunler
```

**Select** ifadesinde alanların ismi verilirken, hangi tabloya ait olduğu da yazılır. Ancak bu durum tek tablo üzerinden yapılan işlemler için gerekli değildir. Birkaç tablo üzerinde sorgu yapıldığı zaman, alanları tablo ismiyle belirtmek gereklidir.

```
SELECT
Isim,
BirimFiyat,
EklenmeTarihi
FROM Urunler
```

|   | Isim                   | BirimFiyat  | EklenmeTarihi |
|---|------------------------|-------------|---------------|
| ▶ | Visual Studio.NET 2002 | 1.150,00 TL | 10.09.2002    |
|   | Visual Studio.NET 2003 | 1.300,00 TL | 20.10.2003    |
|   | Yazılım Uzmanlığı 2004 | 0,00 TL     | 09.02.2004    |
|   | Yazılım Uzmanlığı 2005 | 0,00 TL     | 30.05.2005    |
| * |                        |             |               |

**Where** ifadesinden sonra yazılan kriterler mantıksal karşılaştırmalardır. Bu karşılaştırmalar alanlardaki değerler üzerinde yapılır. Karşılaştırmalar aritmetik olabildiği gibi metinsel de olabilir.

- Büyük

Alandaki değerin verilen bir değerden veya başka bir alandan büyük olduğunu kontrol eder.

```
SELECT Urunler.*
FROM Urunler
WHERE Urunler.Incelenmesayisi > 100
```

- Büyük Eşit

Verilen bir alanın veya değerin, kontrol edilen alandan büyük veya alana eşit olduğunu kontrol eder.

```
SELECT Urunler.*
```

```
FROM Urunler
WHERE Urunler.IncelenmeSayisi >= 100
```

- Küçük

Alandaki değerin verilen bir değerden veya başka bir alandan büyük olduğunu kontrol eder.

```
SELECT Urunler.*
FROM Urunler
WHERE Urunler.IncelenmeSayisi < 100
```

- Küçük Eşit

Verilen bir alanın veya değerin, kontrol edilen alandan küçük veya alana eşit olduğunu kontrol eder.

```
SELECT Urunler.*
FROM Urunler
WHERE Urunler.IncelenmeSayisi <= 100
```

- Between - And

Alandaki değerin iki değer arasında olduğunu kontrol eder. Değerlere eşit oldukları durumlar da sonuç kümесine dâhil edilir.

```
SELECT Urunler.*
FROM Urunler
WHERE Urunler.IncelenmeSayisi BETWEEN 100 AND 200
```

- Not

Verilen kriterde **uymayan** kayıtları döndürür.

```
SELECT Urunler.*
FROM Urunler
WHERE NOT Urunler.IncelenmeSayisi = 0
```

- Like

Alandaki değerin belirli bir metin biçimde olduğunu kontrol eder.

```
SELECT Alanlar FROM Tablo WHERE Alanismi LIKE 'Pattern'
```

**Pattern** ifadesinde yazılan karakterler, alanların içinde kesin olarak gelecek karakterlerdir. Örneğin **İsim LIKE 'Enis'**. Ancak bazı özel karakterler farklı anlam ifade ederler. Örneğin \* karakteri sıfır veya daha fazla karakteri temsil eder. **İsim LIKE '\*ni\*** ifadesi sıfır veya daha fazla karakter ile başlayan, ni ile devam eden ve yine sıfır veya daha fazla karakter ile biten kelimeleri kontrol eder. Örneğin Deniz, Nil, Seni, Ni değerleri bu biçimde uyacaktır.

| Pattern | Örnek | True değeri döndüren örnek | False değeri döndüren örnek |
|---------|-------|----------------------------|-----------------------------|
|         |       |                            |                             |

|                                    |            |                           |                             |
|------------------------------------|------------|---------------------------|-----------------------------|
| Sıfır veya birden fazla karakter * | Nu*        | Nuray, Nuri               | Banu                        |
| Özel karakterlerin kullanımı       | Beş [*]    | Beş *                     | Beşiktaş                    |
| Tek karakter ?                     | Ç?n        | Çan, Çin                  | Çiban, Çanak                |
| Tek Sayı #                         | Versiyon # | Versiyon 5,<br>Versiyon 1 | Versiyon 10,<br>Versiyon Üç |
| Karakter Aralığı                   | [a-z]      | a, b, c                   | 43, 2                       |
| Aralık Dışı                        | [!0-9]     | a, b, c                   | 1, 2, 3                     |

Örnek: Microsoft Studio ürünlerin listelenmesi

```
SELECT Urunler.Isim
FROM Urunler
WHERE Urunler.Isim Like '*Studio*'
```



- **Is Null**

Bazı alanların değerleri boş bırakılmış olabilir. Boş bırakılan alanların değerleri **Null** olarak geçer. Sorgularda boş alanların kontrolü **Is Null** ifadesi ile yapılır.

```
SELECT Urunler./*
FROM Urunler
WHERE Urunler.Ozellikler Is NULL
```

Bir sorguda birden fazla kriter kullanılabilir. Ancak bu kriterlerin **AND** veya **OR** ifadeleri ile ayrılmaları gereklidir. **AND** ifadesi ile ayrılan kriterlerin hepsinin sağlandığı satırlar sonuca dahil edilir. **OR** ifadesi ile ayrılan kriterlerin herhangi biri sağlandığı satırlar sonuca dahil edilir.

Örnek: 12.12.2002 den sonra kaydolmuş, ismi E ile başlayan kullanıcılar.

```
SELECT *
FROM Kullanicilar
WHERE Kullanicilar.KayitTarihi > #12/12/2002# AND
Kullanicilar.Isim Like 'E*';
```

E-posta adresi veya web adresi olan firmalar.

```
SELECT Firmalar.Isim, Firmalar.Email, Firmalar.WebSayfasi
FROM Firmalar
```

```
WHERE ((Not (Firmalar.Email) Is Null)) OR ((Not (Firmalar.WebSayfasi) Is Null));
```

## Hesaplama Fonksiyonları

- ### Hesaplama Fonksiyonları
- Sum – Toplam
  - Avg – Ortalama
  - Max – Maksimum
  - Min – Minimum
  - Count – Sayma
  
  - AS anahtar kelimesi ile sonuç alanına mantıksal isim verilir.

```
SELECT Count(KullaniciId) AS [Toplam Kullanıcı Sayısı]  
FROM Kullanicilar;
```

Alanlar üzerinde sayma, toplama, ortalama alma gibi aritmetik işlemlerin yanı sıra minimum maksimum değerlerin alınması gibi işlemler de yapılabilir. Bu işlemlerin sonucunda sayısal bir sonuç ortaya çıkar. Bu sayı, sonuç tablosunda gösterilirken herhangi bir alan ismi ifade etmez. Dolayısıyla sonuç tablosunda sayısal değerleri gösterilirken mantıksal bir isim verilmesi gereklidir. Bu ifade ise **AS** anahtar kelimesi ile belirtilir.

### Sum

Kriterlerin sağlandığı alanlar üzerinde toplama işlemi yapar.

```
SELECT Sum(IncelenmeSayisi) AS [Toplam Incelenme Sayısı]  
FROM Urunler WHERE UretiliyorMu = -1;
```

| Sorgu1: Seçme Sorgusu |                         |
|-----------------------|-------------------------|
|                       | Toplam İncelenme Sayısı |
| ►                     | 4319                    |

## Avg

Kriterlerin sağlandığı alanların ortalama değerini alır.

```
SELECT Avg(Urunler.BirimFiyat) AS [Ortalama Fiyat]
FROM Urunler WHERE UretiliyorMu = -1;
```

| Sorgu1: Seçme Sorgusu |                |
|-----------------------|----------------|
|                       | Ortalama Fiyat |
| ►                     | 816,6667 TL    |

## Max

Kriterlerin sağlandığı alanların maksimum değerini alır. Metinsel değerlerde alfabetik olarak sıralama yapar.

```
SELECT Max(Isim) AS [En son geçen kullanıcı]
FROM Kullanıcılar;
```

```
SELECT Max(KayıtTarihi) AS [En son kaydolan kullanıcı]
FROM Kullanıcılar;
```

| Sorgu1: Seçme Sorgusu |                           |
|-----------------------|---------------------------|
|                       | En son kaydolan kullanıcı |
| ►                     | 13.05.2005                |

## Min

Kriterlerin sağlandığı alanların minimum değerini alır.

```
SELECT Min(Isim) AS [En başta geçen kullanıcı]  
FROM Kullanıcılar;
```

```
SELECT Min(KayıtTarihi) AS [İlk kaydolan kullanıcı]  
FROM Kullanıcılar;
```

| Sorgu1: Seçme Sorgusu |                          |
|-----------------------|--------------------------|
|                       | En başta geçen kullanıcı |
| ▶                     | Ali                      |

## Count

Değeri **Null** olmayan satırların kaç tane olduğunu verir. Genellikle tablolardaki satır sayısı istendiğinde bu fonksiyon kullanılır. Ancak bu tip bir sorguda, sayılan alanın boş bir değer almaması gereklidir. **Primary Key** alanının üzerinden bir sayımla yapılabilir.

```
SELECT Count(KullaniciId) AS [Toplam Kullanıcı Sayısı]  
FROM Kullanıcılar;
```

| Sorgu1: Seçme Sorgusu |                         |
|-----------------------|-------------------------|
|                       | Toplam Kullanıcı Sayısı |
| ▶                     | 5                       |

## Insert

### Insert Sorgusu

- Tablolara veri eklemek için kullanılır.

```
INSERT INTO Tablo (Alan1, Alan2,...)  
VALUES (Değer1, Değer2...)
```

```
INSERT INTO Siparisler (KullaniciId, NakliyeUcreti,  
SiparisTarihi, SonOdemeTarihi, Adres )  
VALUES (1, 3, '20.05.2005', '25.05.2005', 'Beşiktaş  
İstanbul')
```

- Insert Select cümlesi ile birden fazla satır tabloya eklenir.

**Insert** sorguları tablolara kayıt eklemek için kullanılır. Bu kayıtlar eklenirken tablo isimi, alan adı ve hangi değerlerin ekleneceği belirtilmelidir. **Insert** sorgularında dikkat edilmesi gereken nokta, gerekli olan (**Null** kabul etmeyen) alanlara değer eklenmesi unutulmamalıdır.

Sözdizimi:

```
INSERT INTO Tablo (Alan1, Alan2,...) VALUES (Değer1, Değer2...)
```

**Values** ifadesinde verilen değerler, tablonun yazılan alanlarıyla aynı sırada olması gereklidir.

```
INSERT INTO Siparisler ( KullaniciId, NakliyeUcreti,  
SiparisTarihi, SonOdemeTarihi, Adres )  
VALUES (1, 3, '20.05.2005', '25.05.2005', 'Beşiktaş  
İstanbul')
```

Bu tip **Insert** sorgularında sadece tek bir değer girilebilir. Ancak bazı durumlarda birden fazla verinin girilmesi istenebilir. Bu durumda, girilecek değerler **Select** cümlesiyle başka bir tablodan alınır.

Örnek: Ödenen siparişlerin tutulduğu ayrı bir tablo oluşturulur. Sipariş tablosundan bu tabloya tüm ödenen kayıtların aktarılması işlemi **Insert Select** cümlesi ile yapılır.

```
INSERT INTO Odenensiparisler ( SiparisId, KullaniciId,  
NakliyeUcreti, SiparisTarihi, GonderilmeTarihi, Adres)
```

```
SELECT SiparisId, KullaniciId, NakliyeUcreti, SiparisTarihi,  
GonderilmeTarihi, Adres  
FROM Siparisler  
WHERE Odendi = -1;
```

## Update

### Update Sorusu

- Tablolarda veri güncellemek için kullanılır.

```
UPDATE Tablo  
SET Alan1 = Değer1, Alan2 = Değer2, ...
```

```
UPDATE Kullanicilar  
SET ParolaSoru = 'Yeni Soru', ParolaCevabi = 'Yeni Cevap'  
WHERE KullaniciId = 23
```

- Sorgu yazılırken Where kriterinin unutulmaması gereklidir.

**Update** sorguları tablolarda var olan kayıtların belirli alanlarının güncellenmesi işlemini yapar. Bu sorguda da tablo, alan ve yeni değerlerin belirtilmesi gereklidir.

Sözdizimi:

```
UPDATE Tablo SET Alan1 = Değer1, Alan2 = Değer2, ...
```

Bu sorguda dikkat edilmesi gereken en önemli nokta, belli kayıtlarda güncelleme işlemi yapılmırsa **WHERE** kriterinin unutulmaması gereklidir. Aksi halde tablodaki tüm kayıtlar, sorguda belirlenen değerleri olacaktır.

Örnek:

```
UPDATE Kullanicilar  
SET ParolaSoru = 'Yeni Soru', ParolaCevabi = 'Yeni Cevap'  
WHERE KullaniciId = 23
```

## Delete

### Delete Sorgusu

- Tablolardan veri silmek için kullanılır

```
DELETE FROM Tablo
```

```
DELETE FROM Sepetim WHERE KullaniciId = 12
```

Tablodan veri silmek için kullanılır. Bu sorguda alan isimleri belirtilmez, ancak **WHERE** kriterinin unutulmaması gereklidir.

Sözdizimi:

**DELETE FROM Tablo İsmi**

Örnek:

**DELETE FROM Sepetim Where KullaniciId = 12**

## INNER JOIN

### Inner Join

- Tabloları birleştirmek için kullanılır.
- Primary Key ve Foreign Key alanları üzerinden birleştirme yapılır.

```
SELECT Alanlar
FROM Tablo1 AS isim1 INNER JOIN Tablo2 AS isim2
ON isim1.Alan = isim2.Alan
```

```
SELECT StokDurumu.Adet, Urunler.Isim
FROM
Urunler INNER JOIN
StokDurumu ON Urunler.UrunId = StokDurumu.UrunId;
```

Birden fazla tablodan kayıt çekilmek istendiğinde, bu tabloların **Primary Key** ve **Foreign Key** alanları üzerinden birleştirilmeleri gereklidir. Tabloları birleştirmek, birçok bilgiyi sonuç kümesinde tek bir tablo olarak göstermeyi sağlar. Örneğin bir ürünün hangi kategoride olduğu bilgisi ürünler tablosunda vardır. Ancak bu değer o kategori numarasını belirttiği için, son kullanıcıya bir şey ifade etmez. Kategori ismi ise, kategoriler tablosunda durur. Sonuç kümesinden kategori ismini görüntülemek için bu tabloların birleştirilmesi gereklidir.

Sözdizimi:

```
SELECT Alanlar
FROM Tablo1 AS isim1 INNER JOIN Tablo2 AS isim2
ON isim1.Alan = isim2.Alan
```

Burada tablo isimlerine birer takma isim verilmiştir. Bu isimler alanların seçiminde yazım kolaylığı sağlar. Bazı alanlar birbirleriyle aynı isimde oldukları için bu alanın hangi tabloya ait olduğu belirtilmelidir. Alan isimleri

```
SELECT isim1Alan1, isim1.Alan2, ..., isim2.Alan1, isim2.Alan2
FROM Tablo1 AS isim1 INNER JOIN Tablo2 AS isim2
ON isim1.Alan = isim2.Alan
```

İki tablonun birleştirme işlemi, **ON** ifadesinden sonra belirtilen alanlar üzerinden yapılır. Burada, iki tablo arasında ilişki kurulan alanlar belirtilmelidir.

Örnek: Ürünlerin stoklardaki miktarını öğrenmek için stok ve ürünler tablolarını birleştirmek gerekir.

```
SELECT
StokDurumu.Adet,
Urunler.Isim

FROM
Urunler INNER JOIN
StokDurumu ON Urunler.UrunId = StokDurumu.UrunId;
```

İkiden fazla tablodan bilgi çekmek için, önce iki tablo birleştirilir. Sonuç olarak çıkan tablo ile de diğer tablolar tek tek birleştirilir. Birleştirme işlemi ((Tablo1 + Tablo2) + Tablo3) + Tablo4... şeklindedir. **Inner Join** kullanılırken parantezlerin unutulmaması gereklidir.

Örnek: Bir kullanıcının sepetindeki ürünlerin birim fiyatları sorgulanmak istediği zaman kullanıcılar, sepetim, ürünler tabloları ilişkide oldukları alanlar üzerinden birleştirilmelidir.

```
SELECT
k.Isim,
k.Soyad,
s.Adet,
u.BirimFiyat

FROM
(Urunler u INNER JOIN
Sepetim AS s ON u.UrunId = s.UrunId) INNER JOIN
Kullanicilar AS k ON k.KullaniciId = s.KullaniciId

WHERE k.KullaniciId = 1
```

## Group By

- ## Group By
- Aynı verilerin gruplanmasıdır.
  - Hesaplama fonksiyonları ile kullanılır.
  - Hesaplama fonksiyonunda kullanılmayan alanlar gruplanmalıdır.

```
SELECT k.Isim, k.KategoriId,
       SUM(u.BirimFiyat) AS [Toplam Fiyat],
       COUNT(UrunId) AS [Ürün Sayısı]

  FROM
Urunler u INNER JOIN
Kategoriler k ON u.KategoriId = k.KategoriId

 GROUP BY k.KategoriId, k.Isim
```

Hesaplama fonksiyonlarının kullanıldığı sorgularda **Select** ifadesinden sonra sadece hesaplanan alan sonuç kümesine eklenmiştir. Ancak çoğu zaman, hesaplanan bir alanlar ile birlikte diğer alanların da sonuç kümesinde olması istenir.

Örneğin belli bir kategoride kaç tane ürünün bulunduğu, kategori numarası sonuç kümesinde olacak şekilde isteniyor. Bu durumda, ürünler tablosundaki kayıtların sayma işleminin gerçekleştirilmesi için önce kategori numarasına göre gruplanması gereklidir.

```
SELECT
k.KategoriId,
COUNT(UrunId) AS [Ürün Sayısı]
  FROM
Urunler u INNER JOIN
Kategoriler k ON u.KategoriId = k.KategoriId
 GROUP BY k.KategoriId
```

Tablodan kategori numarası dışında başka herhangi bir alan daha isteniyorsa, bu alan **Group By** ifadesine ya da bir hesaplama fonksiyonunun içine alınmalıdır.

```
SELECT
k.Isim,
k.KategoriId,
SUM(u.BirimFiyat) AS [Toplam Fiyat],
```

```
COUNT(UrunId) AS [Ürün Sayısı]
FROM
Urunler u INNER JOIN
Kategoriler k ON u.KategoriId = k.KategoriId
GROUP BY k.KategoriId, k.Isim
```

| Sorgu1: Seçme Sorgusu |         |            |              |             |
|-----------------------|---------|------------|--------------|-------------|
|                       | İsim    | KategoriId | Toplam Fiyat | Ürün Sayısı |
| ▶                     | Kitap   | 1          | 0,00 TL      | 2           |
| ▶                     | Yazılım | 2          | 2.450,00 TL  | 2           |

## Aritmetiksel İşlemler

### Aritmetiksel İşlemler

- Toplama, Çıkarma, Bölme, Çarpma

```
SELECT Urunler.Isim, BirimFiyat * 1.18 AS [KDV Dahil Fiyat]
FROM Urunler
```

Sorgular sırasında, alanlar üzerinde toplama, çıkarma, çarpma, bölme gibi aritmetiksel işlemler yapılabilir. Bu işlemler sabit değerler ile yapılabildiği gibi başka alanlardaki değerler ile de yapılabilir.

Örnek: Birim fiyatlarının KDV eklenmiş halini gösteren sorgu.

```
SELECT
Urunler.Isim,
BirimFiyat * 1.18 AS [KDV Dahil Fiyat]
FROM Urunler
```

```
SELECT  
Sum(BirimFiyat) * 1.18 AS [Toplam Ürünler Fiyatı KDV Dahil]  
FROM Urunler
```

Örnek: Stoklarda, rezerve edilmemiş toplam ürün sayısı

```
SELECT Urunler.Isim, Sum(StokDurumu.Adet -  
StokDurumu.Rezerve) AS [Açık Ürün Sayısı - Tüm Stoklar]  
FROM Urunler INNER JOIN StokDurumu ON Urunler.UrunId =  
StokDurumu.UrunId  
GROUP BY Urunler.Isim;
```

Toplama işlemi, sayılar üzerinde yapılabildiği gibi metinsel değerler üzerinde de birleştirme görevi görür.

Örnek: Kullanıcıların isim ve soyadlarının beraber görüntülenmesi

```
SELECT  
Kullanicilar.Isim + ' ' + Kullanicilar.Soyad AS [İsim Soyad]  
FROM Kullanicilar
```

## Modül Sonu Soruları & Alıştırmalar

### Özet

- ➔ Menüler
  - ➔ MainMenu, ContextMenu
- ➔ ToolBar
- ➔ ToolTip
- ➔ StatusBar
- ➔ NotifyIcon
- ➔ RichTextBox

1. **Select** ifadesinin kullanım alanını açıklayın ve bir örnek SQL cümlesi geliştirin.

2. **Insert** ifadesinin kullanım alanını açıklayın ve bir örnek SQL cümlesi geliştirin.
3. **Update** ifadesinin kullanım alanını açıklayın ve bir örnek SQL cümlesi geliştirin.
4. **Delete** ifadesinin kullanım alanını açıklayın ve bir örnek SQL cümlesi geliştirin.
5. **Delete** ve **Update** ifadelerini kullanırken dikkat etmemiz gereken noktaları açıklayın.
6. **Cascade Delete** ve **Cascade Update** ifadelerini içeren bir veri tabanı uygulaması geliştirin ve silme durumunu gözlemleyin.