YMH 219 Nesne Tabanlı Programlama

C # Temelleri

Prof.Dr. Erkan TANYILDIZI

Neler Öğreneceğiz

- Nesneye yönelik programla nedir?
- (Metot nedir?, sınıf nedir?, nesne nedir?, nesneler nasıl tü retilir?.....)
- Nesneye yönelik programlamanın üstünlükleri nelerdir ?
- .NET teknolojisi nedir? Neleri bizlere sağlar ve avantajl arı nelerdir?

Kaynaklar

- C#'ı Kavramak

 - Tom Archer, Arkadaş, 2002

- Her yönüyle C#

 -Sefer Algan, Pusula, 2004

- C# Temel Başlangıç Kılavuzu
- – BradleyJones, Sistem, 2003
- www.bilgeadam.com

C# Nedir?
.Net Framework Nedir?

C# Nedir?

Bilgisayar dillerini düzeylerine göre şu şekilde sınıflandırabiliriz:

Script Dilleri	Javascript,VBScript, Perl Script
Yüksek Düzeyli Diller	Vbasic, Delphi
Orta Düzeyli Diller	C# ve Java
Düşük Düzeyli Diller	C/C++
Assebmly(makine dili)	Assembly

C# Nedir?

- C# Programlama Dili Microsoft'un son zamanlarda geliştirdiği .NET platformunun bir öğesidir.

- Eski programlama dilleri ile yeni dillerin harmanlanması ile oluşmuştur.

- Ayrıca C, C++, Java, Visual Basic dillerinin bir türevi niteliğindedir.

C# Nedir?

- **C#** , C/C++ ve Java dillerinden türetilmiş,bu dillerin dezavantajlarının elenip iyi yönlerinin alındığı, güçlü basit, esnek, tip güvenli (type safe) Net platformu için sıfırdan geliştirilmiş %100 nesne yönelimli bir dildir.

- (type-safe, tür dönüşümlerindeki önlemler, örn: byte=byte+byte olamaz, int=byte+byte).

C# Nedir?

- C#, eskiden beri programcıların yaygın bir biçimde kullandığı C/C++ve Java dillerine benzerliği ile tanınan bir programlama dilidir. İlk aşamada çok benzer bir dil olarak görünse de bu iki dilden farklı bir çok özelliğe sahiptir.

- C/C++ ve Java'nın güzel özelliklerini alıp bu dillerin tehlikeli olabilecek olabilecek özelliklerini dışarıda bırakan bir dildir.

C# nedir?

- C#, C/C++ dilinden farklı olarak tamamıyla nesneye yöneliktir. “int”, “double” gibi temel veri türleri dahi birer nesne olarak tanımlanmıştır.

- Java dilinden farklı olarak C# dilinde işaretçiler **(pointer) kullanılabilmektedir**.

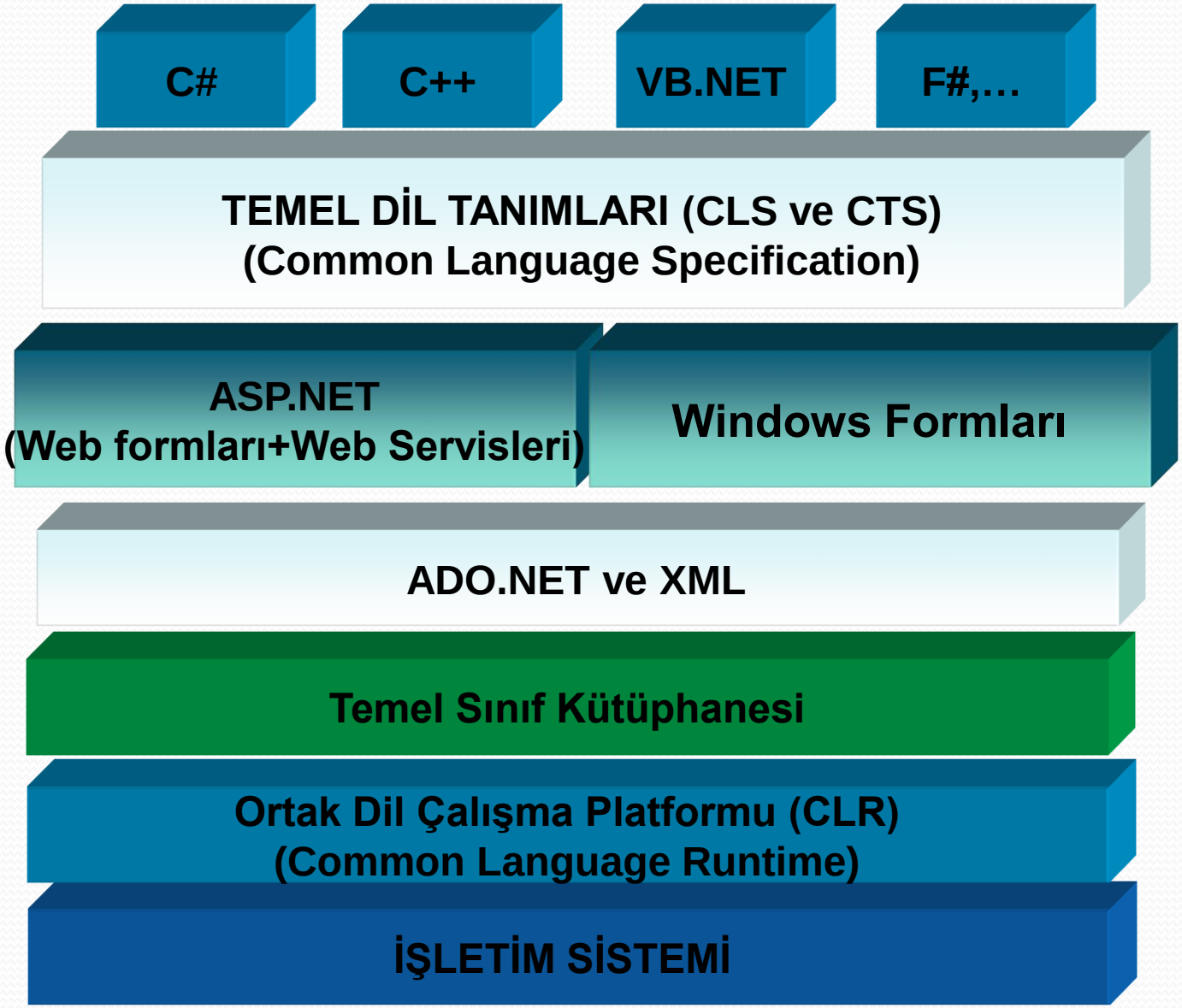
Neden C#'ı Tercih Edelim?

- Öğrenilmesi kolaydır.
- **Nesne yönelimli programlamaya tam destek vermektedir**.
- Yüksek verim.
- Güç ve kolaylık arasında dengededir.
- Xml desteği sunmaktadır.
- Windows Forms mantığını destekler.
- Internet teknolojilerinin gelişimine uyum sağlayabilmiş modern bir dildir.

C# Kullanım Alanları

- **Konsol tabanlı uygulama geliştirme**
- **Windows için program yazma**
  - *C# ile Windows için gelişmiş, güçlü, hızlı ve güvenli programlar yazılabilir. Fakat bunun için programın çalıştığı sistemde **.NET** platformunun yüklü olması gerekir.*
- **ASP.NET için en uygun dildir.**
  - *ASP.NET, ASP gibi script yorumlamalı olmayıp tamamen nesne yönelimli haldedir. C# bu konuda büyük kolaylıklar sağlamaktadır.*
- **C# ile Web Servisleri geliştirmek oldukça kolay ve hızlıdır**
- **Mobil uygulama geliştirme ve DLL yazabilme.**

.NET Bileşenleri



Visual Studio NET 2010

.NET Bileşenleri

- Önceden, geliştirilen yazılımlar direkt olarak makine koduna derlenir ve bu şekilde çalıştırılırlardı. Ayrıca bu programlar, her işletim sistemine özel olarak geliştirilmekte ve derleme işlemi işletim sistemine göre belirlenmekteydi.

- Böyle bir yapıda taşınabilirlikten söz etmek mümkün değildir.

- **Java'da ise, program önce byte code'a çevrilmektedir. Bu kodu JVM(J**ava **V**irtual **M**achine), işletim sisteminin istediği koda çevirmektedir.

- .NET platformunda da temel prensip Java ile benzerdir.

.NET Bileşenleri

- . NET kodu önce **IL** (**I**ntermediate **L**anguage-Ara dil)'ye derler ve bu IL kodu çalıştırılmak istendiği zaman **.NET CLR** (**C**ommon **L**anguage **R**untime-Ortak Çalışma Platformu), JIT (**J**ust **I**n **T**ime) derleyicilerini kullanarak makine diline çevirir.

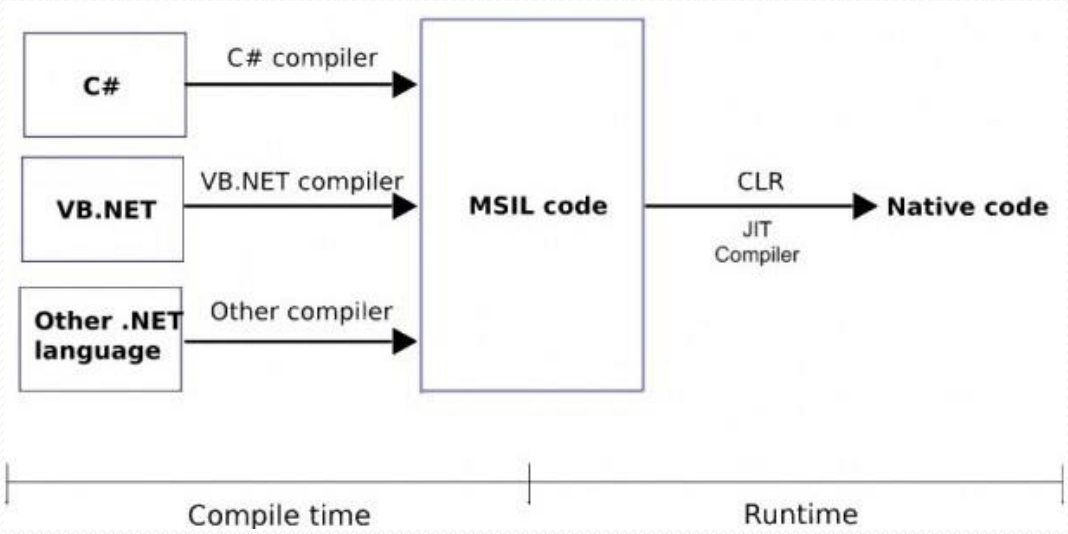
- CLR makine diline çevrilmiş bu kodu önbellekte tutar, bu performans artışına sebep olurken diğer taraftan sistem hafızasında küçümsenmeyecek yer işgal eder.

.NET Bileşenleri

- Temel Dil Tanımlamaları **CLS** (**C**ommon **L**anguage **S**pecifications) ve Ortak Tip Sistemi **CTS** (**C**ommon **T**ype **S**ystem) ile .NET uyumlu dillerin hepsi aynı değişkenleri ve benzer nesne yönelimli özellikleri taşır. Örn: C# ile yazılan programdaki temel veri tipleri, VB.NET ‘tekiler ile aynı özelliklere sahiptir. Böylece farklı dillerde yazılan bileşenler birbiri ile sorunsuz çalışırlar.

Ortak Dil Çalışma Platformu(CLR)

- **CLR**, .NET altyapısında programların çalışmasını kontrol eden ve işletim sistemi ile programımız arasında yer alan arabirimdir. (Normalde yazılan kodlar makine diline çevrilir ve işletim sistemi ile direkt bağlantı kurup çalışırdı.)



- IL kodu, CLR tarafından çağrılınca JIT derleyicileri tarafından makine diline çevrilir ve çalıştırılır.

Ortak Dil Çalışma Platformu(CLR)

- Eğer çok sayıda platform olduğunu düşünürsek, programların bunlar için ayrı ayrı yazılıp derlenmesi gerekir. Bu durum imkansız gibidir.

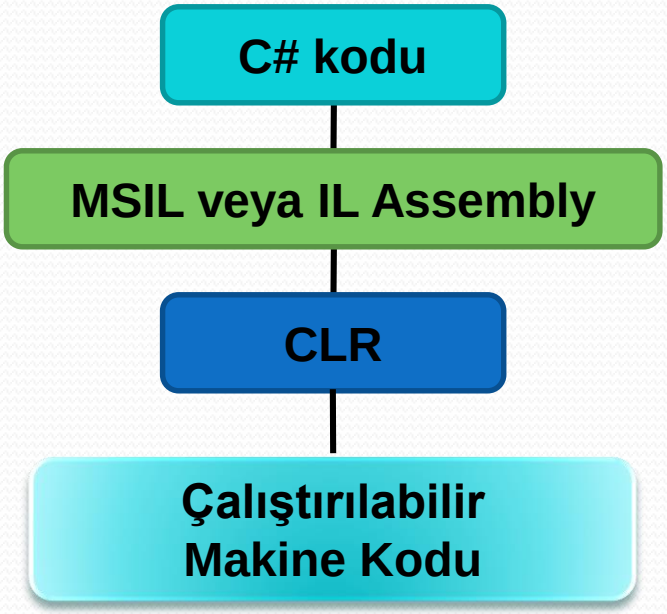
- Eğer platformdan bağımsız bir ortam istiyorsak, ihtiyaç duyulan şey CLR dir, hangi platformda iseniz (Linux,Mac,Windows) CLR bu noktada devreye girer ve .NET programlarının farklı platformlarda işletim sistemine göre çalıştırır.

Ortak Dil Çalışma Platformu(CLR)

- Managed Code(Yönetilen Kod): Yalnızca CLR yardımları altında çalışan koddur.

- Bir örnek vermek gerekirse ; Windows'ta çalışan farklı işlemlere sahibiz. Uygulamaların izlemesi gereken kural Windows genel kurallarına uymalarıdır. Managed kodda CLR tarafından Windows'un yaptığı şekilde çalıştırılan koddur.

Ortak Dil Çalışma Platformu(CLR)



.NET derleme ve çalıştırma

Aradil (IL veya MSIL) (Intermediate Language)

- Herhangi bir C++ veya Vbasic kodu direkt makine koduna çevrilirdi ve çalıştırılırdı. Makine diline çevrilen programlar, işlemciye ve işletim sistemine özel olarak derlenirdi.

  Örn: a ve b sayılarının toplamı için kullanılan bir C++ programı Intel işlemciler için farklı, SunSparc işlemciler için farklı derlenirdi.
- Fakat .NET ortamında kodumuzu derlediğimizde elde ettiğimiz IL (ara dil) kodu işlemciye bağlı olmaz.

Aradil (IL veya MSIL)

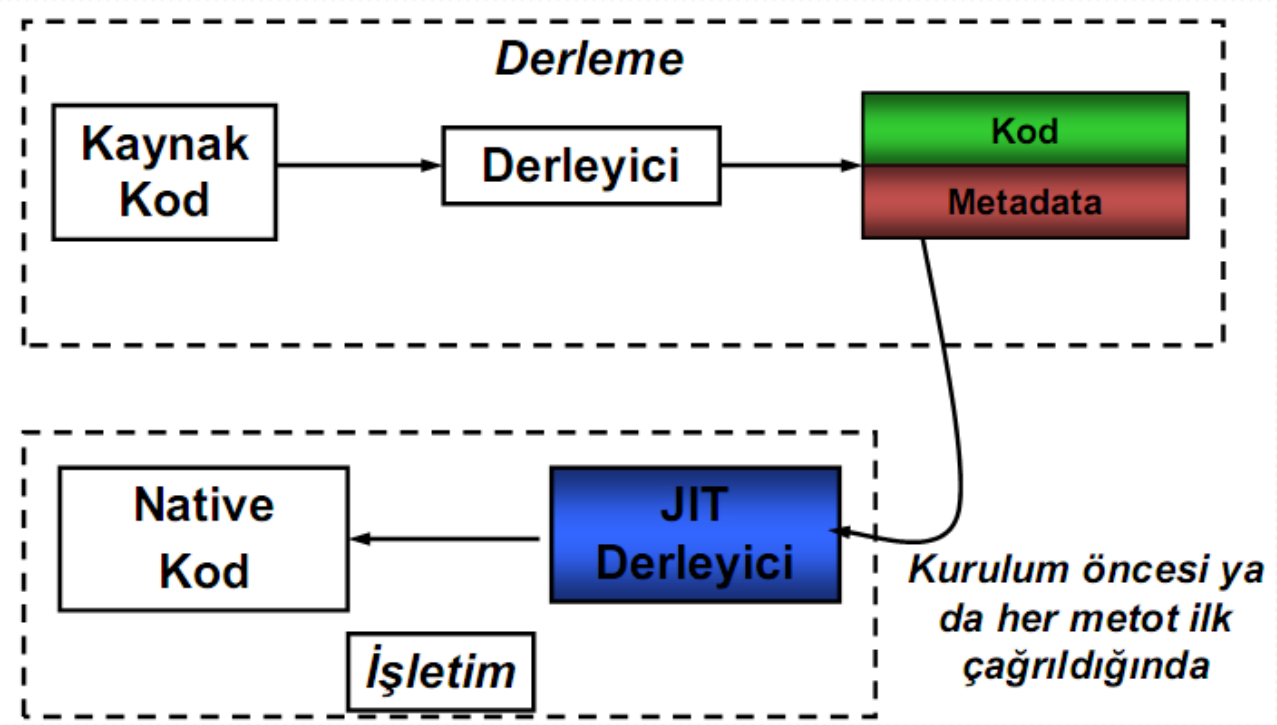
- **IL** içerisinde değişken tanımları, değişkenlerin nasıl saklanacağı, metotların nasıl çalıştırılacağı, aritmetik ve mantıksal işlemler, bellek kullanımı gibi birçok işin nasıl yapılacağı açıklanır.

- Bütün bunların yanında IL'de **Metadata** olarak adlandırılan bir birim daha vardır. Metadata, programda kullanılan verilerin tiplerinin yanında oluşturulan sınıfların metotlarını ve bunların özelliklerini ve diğer bilgileri içerir.

- Artık IL ile oluşturduğumuz kodumuzun çalıştırılabilir bir program olması için derlememiz gerekiyor. Bunun için JIT (Just in Time) derleyici kullanılır.

JIT Derleyiciler (Just in Time)

- Metadata'nın içeriği çalışma zamanında JIT derleyicileri tarafından kullanılır IL ve Metadata'sı oluşturulan kod parçası, çalıştırılabilir bir yapıdadır.



JIT Derleyiciler (Just in Time)

- C# ile IL'ye derlediğimiz programı çalıştırırken JIT derleyicileri devreye girerler. Bu derleyiciler programın çalıştırıldığı sistemin ve işlemcinin anlayabileceği makine kodunu oluştururlar.
- Windows ortamı için 3 çeşit JIT mevcuttur

1. Normal JIT
2. Pre-JIT
3. Eco-JIT

JIT Derleyiciler (Just in Time)

- Normal JIT : IL kodu makine koduna çevrilirken default(varsayılan) olarak kullanılan derleyicidir. IL kodunu orijinal makine koduna çevirir ve *önbellekte* tutar. Örneğin ; program içindeki bir derlenmiş bir metot program akışı içinde tekrar çağrılırsa önbellekten çekilir.
- Pre-JIT: Tüm program kodunu makine koduna çevirip sonra çalıştıran JIT. Fazla hafıza gerektirir. Programın daha hızlı çalışmasını sağlar.

JIT Derleyiciler (Just in Time)

- Eco JIT : Kısıtlı hafıza ve önbellekli sistemlerde .NET programlarının daha iyi çalışmalarını sağlamak için kullanılan derleyicidir.
- Derlenen ve çalıştırılan program parçaları normal JIT'de hemen hafızadan silinmiyordu. Fakat Eco-JIT'de kullanılabilir hafıza belli bir oranın altına düştüğünde, daha önceden derlenmiş ve çalıştırılmış kısımlar hafızadan silinirler.

CTS

- Bütün veri tiplerinin tanımlı olduğu bir sistem olarak düşünebiliriz. C# dilindeki veri türleri aslında CTS'deki veri türlerine karşılık gelen ara yüzlerdir.
- CTS sayesinde .NET platformu için geliştirilen bütün diller aynı veri tiplerini kullanırlar, tek değişen türlerin tanımlama yöntemi ve söz dizimidir. Geliştirilen bir nesnenin diğer dillerde de sorunsuz çalışmasını garanti eder.

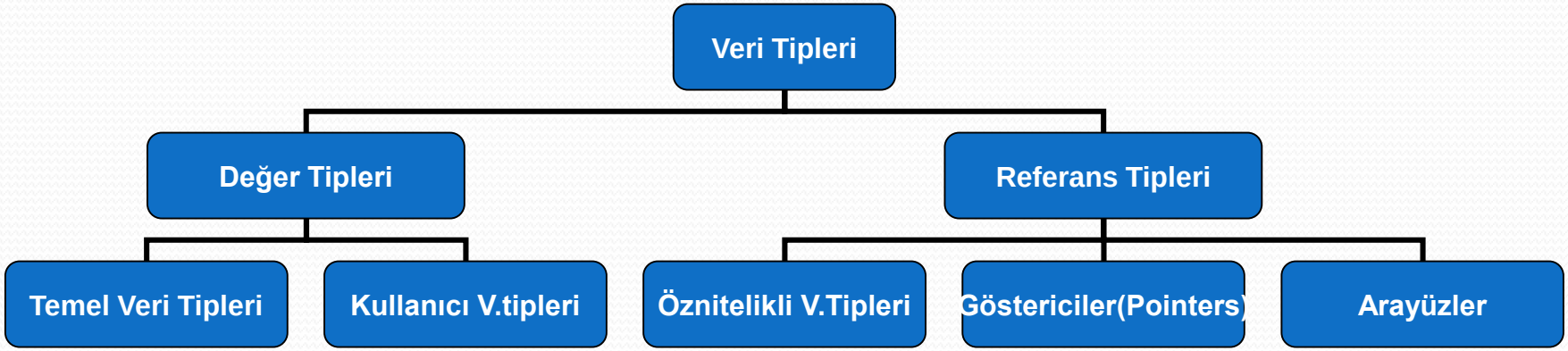
CTS

- **Örn:**
- Int tipi C++.NET ve VB.NET’teki tamsayı tiplerinin kapasiteleri aynıdır. Diğer bir deyişle VB’de geliştirilen bir dll C#’da rahatlıkla kullanılabilir.

CTS

- CTS sayesinde;
- .NET’de tip güvenli, yüksek performanslı ve kullanılan programlama dilinin diğer .NET uyumlu diller ile entegre bir şekilde çalışması sağlanır.
- Birçok programlama dilinin .NET mimarisinde tam nesne yönelimli olarak .NET için program yazma aracı olmasına imkan verir.
- Herhangi bir .NET uyumlu dilde geliştirilen nesne diğer dillerde de sorunsuz olarak çalıştırılabilir.

CTS



CTS veri tipleri şeması

CLS

- **CLS,** dil derleyicisinin uyması gereken kuralları içerir.
  - .NET platformunu paylaşan dillerin sadece CTS'yi desteklemeleri yetmemektedir. Bunun yanında **Ortak Dil Spesifikasyonu**' da **(CLS)** desteklemeleri gerekmektedir.

- **CLS**' ye uyan bir dille yazılmış kod ile diller arası iletişim sağlanmış olur.
  - CLS uyumlu bir dil ile geliştirilen bir program ile farklı diller arasında etkileşim sağlanabilir. .NET'in temel sınıf kütüphanelerinin içerisinde yer alan kodların büyük bir bölümü CLS uyumludur.

CLS

Desteklenen Programala Dilleri		
APL	Fortran	Pascal
C++	Haskell	Perl
C#	Java Language	Python
COBOL	Microsoft JScript®	RPG
Component Pascal	Mercury	Scheme
Curriculum	Mondrian	SmallTalk
Eiffel	Oberon	Standard ML
Forth	Oz	Microsoft Visual Basic®

Assembly

- .NET platformu için yazılan bütün kodların sonucunda oluşan .exe ve .dll uzantılı dosyalara genel olarak assembly denilmektedir.

- Derlenmiş kodlar ve metadata olarak adlandırlan özniteleyici kodlar Assembly içerisinde bulunurlar.

  - Assembly içersindeki metadata verileri, tür bilgileri ve başka kaynaklara olan bağlantıları saklar.

  - Assembly'de ayrıca versiyon bilgisi de tutulur.

  - Assembly sayesinde programlar register edilmeye gerek kalmadan direkt kopyalanarak kurulabilirler.

Application Domain

- Application domain sayesinde aynı anda çalışan birden fazla program veya process birbirinden izole edildiği halde sistemde herhangi bir aksaklığa yol açmadan aralarında veri alış-verişi yapabilirler

Namespaces and .NET Class Library (İsim Alanları ve .NET Sınıf Kütüphanesi)

- .NET Framework'ün programcılara sunduğu bir takım temel türler ve sınıflar mevcuttur.

- Bütün bu sınıfları ve türleri iyi organize edebilmek için .NET, isim alanı (**namespace**) kavramını kullanmaktadır.

- .NET'teki sınıf kütüphaneleri bir dilden bağımsız bir yapıdadır.

Namespaces and .NET Class Library (İsim Alanları ve .NET Sınıf Kütüphanesi)

- C# dilinde .NET Framework sınıf kütüphanesi içerisindeki veri türleri ve sınıflar "**using**" anahtar sözcüğü ile kullanılır. Diğer dillerde de bu isim alanları farklı şekillerde derleyiciye bildirilir.

- Program geliştirirken sınıfların birbiri ile ilgili olanlarını aynı isim alanı içine konulması programdaki hataları bulma ve anlaşılabilirlik açısından oldukça önemlidir.

Namespaces and .NET Class Library (İsim Alanları ve .NET Sınıf Kütüphanesi)

- .NET sınıf kütüphanesinde bulunan ve en sık kullanılan sınıf kütüphaneleri şunlardır:

- **System:** .NET ile çalışırken gerekli temel sınıfları içerir. Ayrıca diğer tüm sınıf kütüphaneleri bu isim alanı içinde kümelenmiştir.
  - Konsol temelli uygulamalarda temel giriş çıkış işlemleri için gerekli temel sınıf “Console”, bir çok matematiksel fonksiyonu içinde barındıran “Math” sınıfı da System isim alanı içersinde yer alır.
  - System hiyerarşinin tepesinde bulunur.

Namespaces and .NET Class Library (İsim Alanları ve .NET Sınıf Kütüphanesi)

- **System.Data:** Veritabanı işlemlerinin tamamı için hazır gelen sınıf kütüphanesine bu isim alanı ile erişilir.
  - Bu sınıf kütüphanesi içindeki SQL ile işlemler için “System.Data.SqlClient” isim alanı mevcuttur.

- **System.Xml:** Veri biçimlendirme ve internetten veri paylaşımı için en çok kullanılan teknolojilerden biri olan XML ile çalışmak için gerekli sınırları içerir.

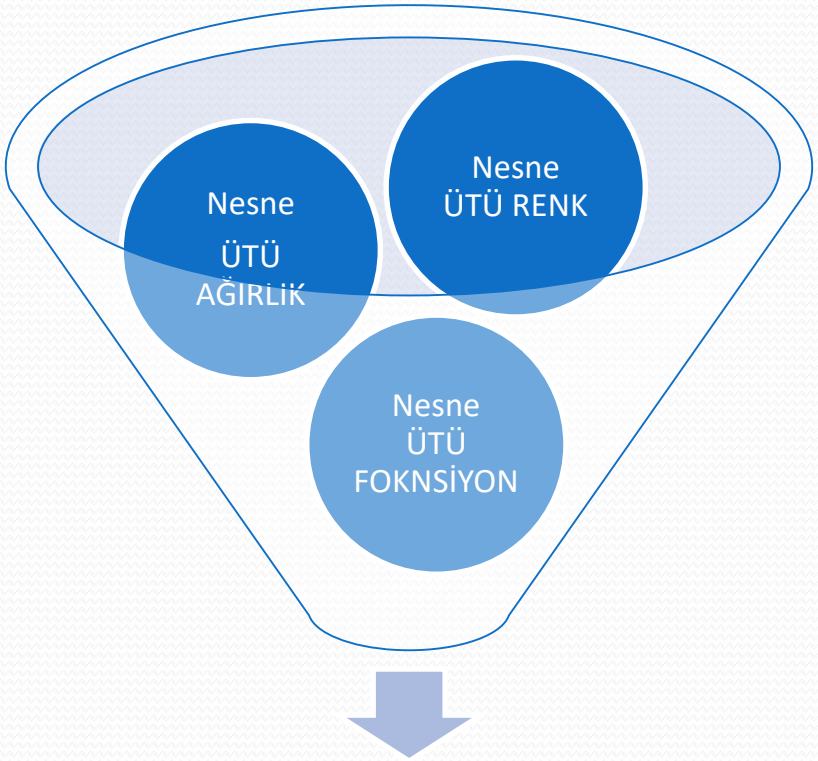
Namespaces and .NET Class Library (İsim Alanları ve .NET Sınıf Kütüphanesi)

- **System.Net:** Dağıtık uygulama geliştirmek için gerekli olan ağ bileşenlerini içeren isim alanıdır. HTTP ve ağ protokolleri için kullanılır.

- **System.IO:** Dosyalarla çalışmak (okuma/yazma) için gerekli işlemlerini içerir.

- **System.Windows.Forms:** Windows temelli uygulamalarda kullanılan görsel kontrolleri barındıran isim alanıdır

Nesne Yönelimli Programlama (C#)



ÜTÜ SINIFI

Bir C# programını Derlemek ve Çalıştırmak

- C# programını bir metin editörü kullanarak oluşturmalısınız.

- **1)** ornek.cs (uzantısı cs) olacak şekilde kaydediniz.

ornek.cs - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

```
using system;
class programım
{
        public static void Main()
        {
                Console.WriteLine("Bu benim ilk C# programım");
        }
} |
```

Bir C# programını Derlemek ve Çalıştırmak

- **2)** Başlat/Programlar/Microsoft Visual Studio.NET 200x/ Visual Studio.NET tools/ Visual Studio.NET Command Prompt seçeneğine tıklayarak komut satırını açınız.

- **3)** C:\>csc ornek.cs komut satırını kullanarak programınızı derleyeliniz.

- **4)** Son olarak sadece programın adını yazarak çalıştırınız.

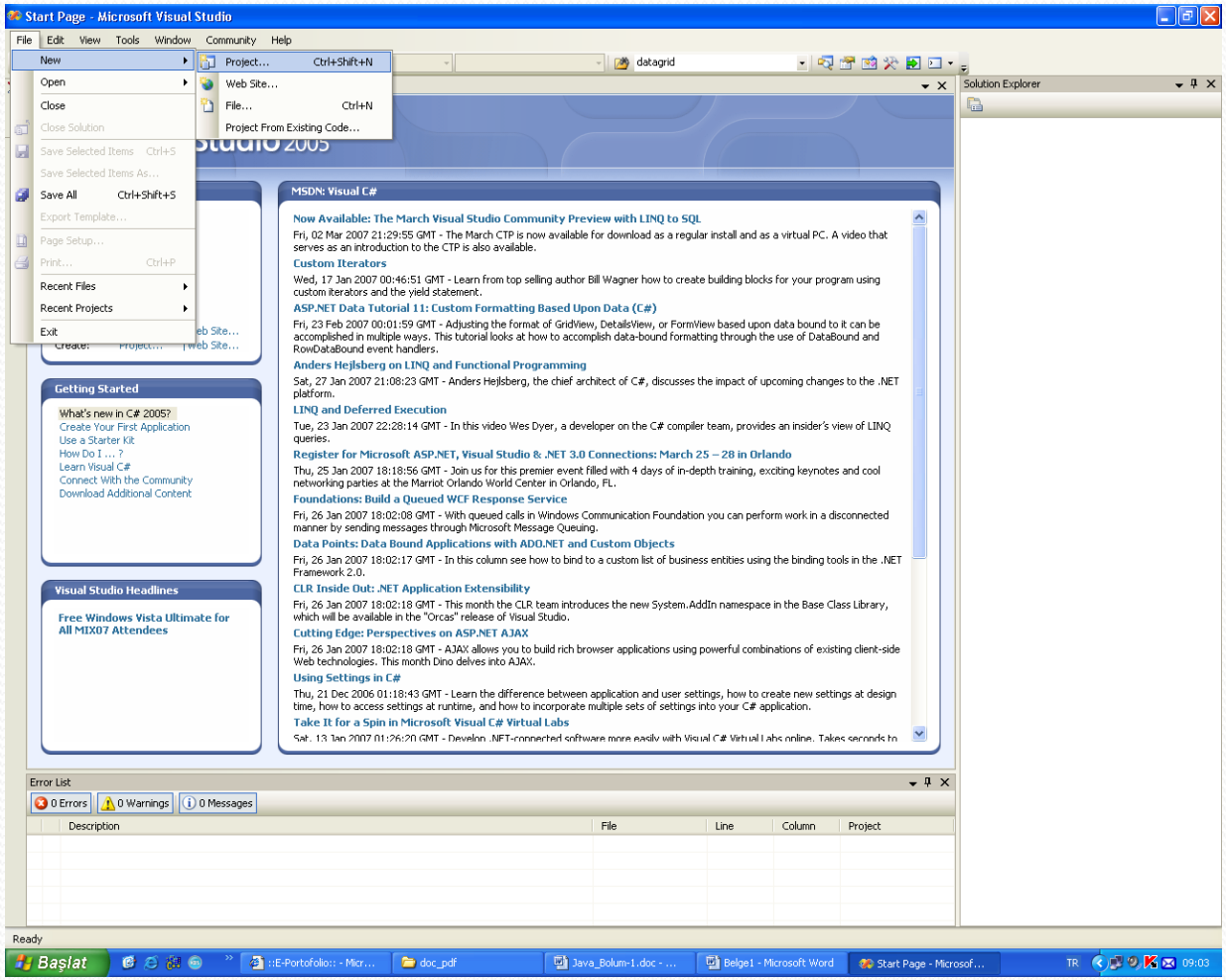
- C:\> ornek
- **Bu benim ilk C# programım**

Bir C# programını Derlemek ve Çalıştırmak

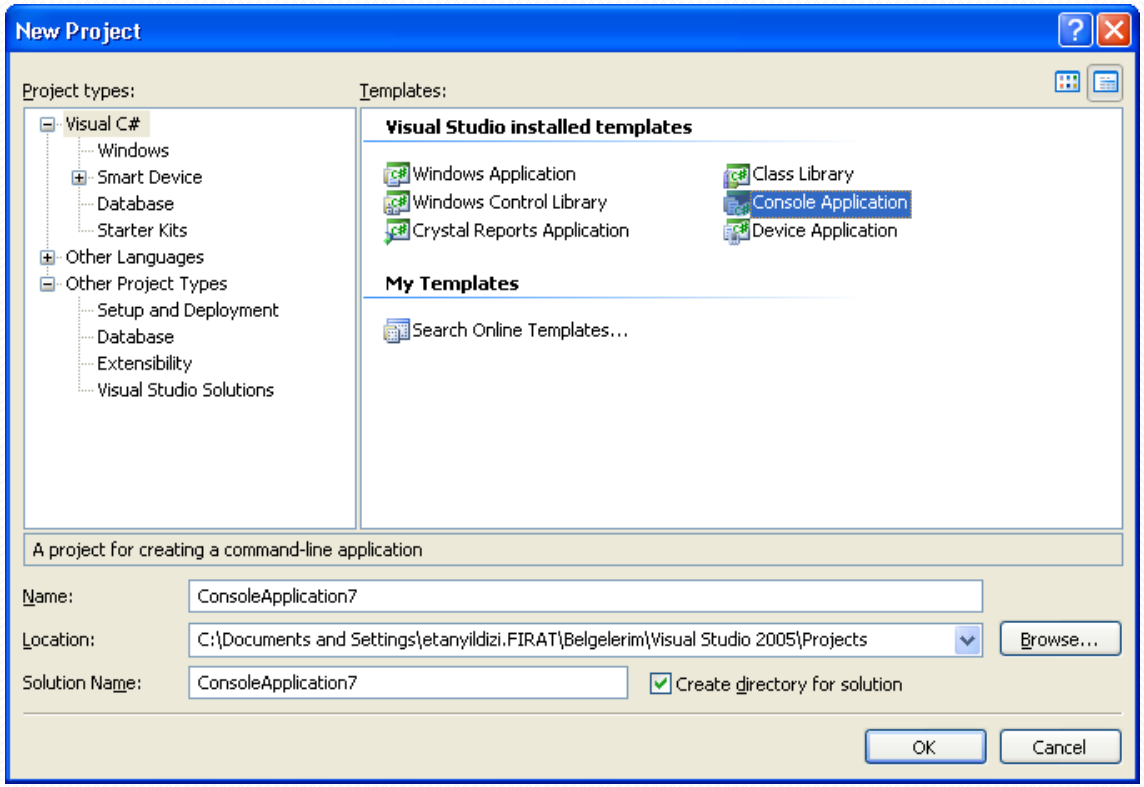
Visual Studio 2008 Command Prompt

```
C:\ornekler>csc ornek.cs
Microsoft (R) .NET Framework sürüm 3.5 için Microsoft (R) Visual C# 2008 Derleyi
cisi sürüm 3.5.30729.4926

Telif hakkı (C) Microsoft Corporation. Tüm hakları saklıdır.


C:\ornekler>ornek
Bu benim ilk C# programım

C:\ornekler>_
```

Bir C# programını Derlemek ve Çalıştırmak



Bir C# programını Derlemek ve Çalıştırmak



ConsoleApplication1 - Microsoft Visual Studio

File Edit View Refactor Project Build Debug Data Tools Test Window Help

Debug | Any CPU | connString

Toolbox

Program.cs* | Start Page

ConsoleApplication1.Program | Main(string[] args)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Bu benim ilk C# programım");
        }
    }
```

Solution Explorer - ConsoleApplication1

- Solution 'ConsoleApplication1' (1 project)
  - ConsoleApplication1
    - Properties
    - References
    - Program.cs

Properties

Error List

0 Errors | 0 Warnings | 0 Messages

Description	File	Line	Column	Project

Önemli

- class ilk_program1
- { static void Main()
- { System.Console.WriteLine("Merhaba C#"); }
- }
- C# dili daha önce de denildiği gibi %100 nesne yönelimli bir dildir. Nesne olmayan hiçbir şey yoktur. C ve C++ dillerinde programın çalışması main işlevinden başlar ancak main işlevi hiçbir zaman bir sınıf içerisinde olmamıştır.

- **C# dilinde her şey sınıflarla temsil edildiği için main işlevi de bizim belirlediğimiz bir sınıfın işlevi olmak zorundadır.**

Önemli

- Bütün C# programları en az bir sınıf içermelidir. Sınıf bildirimi içinde olmayan programlar derlenmez.

- **Main()** işlevi bizim için C ve C++ dillerinde olduğu gibi programımızın başlangıç noktasıdır

- **C#**'de diğer bazı dillerde olduğu gibi kaynak koddaki bütün satırlar ";" ile sonlandırılır. (Bazı durumlar hariç.)

Önemli

- Sınıflar ve işlevler açılan ve kapanan küme parantezler **{ }** içerisine yazılırlar.

- C# dilinde birçok kavram sınıf dediğimiz nesneler üzerine kurulmuştur. Her sınıfın iş yapan çeşitli elemanları vardır. İş yapan bu elemanlara **metot** ya da **işlev** denilmektedir.

Önemli

- .NET' i meydana getiren sınıf kütüphanesi hiyerarşik bir yapı sunmaktadır. Sınıflar isim alanı (namespace) dediğimiz kavramla erişilmesi kolay bir hale gelmiştir.

```
using System;

namespace Ornekalan1
{
   public class Program1
   {
      static void Main(string[] args)
      {

      }
   }

   public class Program2
   {
      public void foksiyon1()
      {

      }
```

```
      public void fonksiyon2()
      {

      }
   }

}

namespace Ornekalan2
{
   public class Program3
   {

   }

}
```

Önemli

```
using System;
class ilk_program2
{ static void Main()
    {
            Console.WriteLine("Merhaba C#");
    }
}
```

- **“using System”** deyimi ile System isim alanındaki bütün sınıflara doğrudan erişim hakkına sahip oluruz.

Önemli

```
using System;
class ilk_program2
{  static void Main()
    {
            Console.WriteLine("Bir tusa basin...");
            Console.ReadLine();
            Console.WriteLine("Bir tusa basin...");
    }
}
```

- **ReadLine** metodu da **WriteLine** gibi kullanılır ancak metodun parantezlerine herhangi bir şey yazılmaz.

Derleyici Parametreleri

- **csc programadi.cs* Programadi.exe şeklinde bir dosya oluşturur.
- **csc /out: yeniprogramadi.exe programadi.cs* yeniprogramadi.exe şeklinde bir dosya oluşturur.
- **csc /t:library programadi.cs* Programadi.dll şeklinde bir dosya oluşturur.
- **csc /t:module programadi.cs Programadi*.netmodule isimli derlenmiş .net modülü elde edilir.
- **csc /t:winexe programadi.cs* Program konu bir Windows uygulaması ise kullanılır.

Derleyici Parametreleri

- **csc /unsafe programadi.cs* Pointer kullanılan programları derlemek için kullanılır.
- **csc /bugreport:RaporDosyası.txt programadi.cs* Derlenen programda eğer hata varsa RaporDosyası.txt dosyasına kaydeder
- **csc /o:+ programadi.cs* Derleyicinin optimize edilmiş sonucu çıkarması için kullanılır.
- **csc /help veya /?* C# derleyicisinin parametrelerini verir.

Visual Studio.Net -C#
2. HAFTA
Temel Veri Türleri

Visual Studio.Net -C#
2. HAFTA
Temel Veri Türleri

C# Dilinde Temel Veri Türleri

- C#'da veri tipleri temel olarak 2'ye ayrılırlar. Bunlar önceden tanımlanmış veri türleri ve kullanıcı tarafından tanımlanmış veri türleridir. Önceden tanımlanmış olan veri türleri de kendi arasında **değer tipi (value type)** ve **referans tipi (reference type)** olarak 2'ye ayrılır.

C# Dilinde Temel Veri Türleri

- Verinin bellekte tutulması 6 bölgeden biri ile olmaktadır Bunlar:

- **Stack Bölgesi:** Program içerisinde bir tamsayı türünden nesnenin çalışma zamanında yüklendiği yer RAM' in stack bölgesidir.

- Tanımlı değişkenlerin tutulduğu bellek alanıdır. Derleyici tarafından değişkenlere yapılacak yer tahsisatı önceden bilinmelidir.

C# Dilinde Temel Veri Türleri

- **Heap Bölgesi:** Bütün C# nesneleri bu bölgede oluşturulur. Stack'ten farklı olarak bu bölgede tahsisatı yapılacak nesnenin derleyici tarafından bilinmesi zorunlu değildir. Bu bölgede bir nesneye alan ayırmak için **new** anahtar sözcüğü kullanılır.

- ***new*** ile tahsis edilen alanlar dinamiktir. Çalışma zamanında tahsisat yapılır, derleme zamanında bir yer ayrılmaz. **Stack**'e göre daha yavaştır.

- **(Değer** veri türleri **Stack, Referans** veri türleri **Heap'** te tutulurlar.)

C# Dilinde Temel Veri Türleri

- **Register Bölgesi:** Registerlar mikroişlemci üzerinde bulunan özel yapılardır Bu yapılarından dolayı diğer bölgelere göre veri transferi daha hızlı bir şekilde yapılabilmektedir.

- **Static Bölge:** Bellekteki herhangi bir bölgeyi temsil eder. Static alanlarda tutulan veriler programın bütün çalışma süresince saklanır. Bir nesneye bu özelliği kazandırmak için static anahtar sözcüğü kullanılır.

- **Sabit Bölge:** Program içerisinde, değerlerin değişmeden sürekli olarak aynı kaldığı bölümdür.

- **RAM Olmayan Bölge:** Bellek bölgesini temsil etmeyen disk alanlarını temsil eder.

Değişken Tanımlama

- Değişken tanımlama aşağıdaki gibidir:
- **<veritürü> <ismi>**

- C#'da bir değişkene herhangi bir değer atamadan onu kullanmak yasaktır. Eğer bir değişkeni kullanmak istiyorsak değişkenlere bir değer verilmesi zorunludur. Bu kural değer ve referans tipleri için de geçerlidir.

- Tanımlamalar ise programın istenilen bir yerinde yapılabilir. Bu konuda herhangi bir kural yoktur.

Değişken Tanımlama

- Değişken isimlendirme ile ilgili temel kuralları aşağıdaki şekilde özetlemek mümkündür:
- C#'da değişken isimlendirmede büyük ve küçük harf duyarlılığı vardır.
- Değişken isimleri nümerik bir karakter ile başlayamaz.
- Değişken isimlerinde boşluk karakteri olamaz.

Değişkenlerin Faaliyet Alanları (Scopes)

- Tanımlanan bir değişkene ancak tanımlandığı blok içerisinden ulaşılabilir. Bu blok aralığına değişkenin faaliyet alanı denir.

- Bir sınıfın üye elemanı olarak tanımlanmış değişken her zaman sınıfın faaliyet alanı içerisindedir.

- Yerel bir değişken, tanımlandığı blok arasında kaldığı sürece faaliyet alanındadır.

- Döngü bloklarında tanımlanan değişkenler döngünün dışına çıkılmadığı sürece faaliyet alanı içersindedirler.

Değişkenlerin Faaliyet Alanları (Scopes)

```
using System;
public class faaliyet_alani
{  static void Main()
   {
       { int x=29;
         Console.WriteLine(x);
       }
       { int x=19;
        Console.WriteLine(x);
       }
   }
}
```

```
C:\WINDOWS\system32\cmd.exe

F:\c#\bolum2\compiled>csc faaliyet_alani.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.1433
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

F:\c#\bolum2\compiled>faaliyet_alani
29
19

F:\c#\bolum2\compiled>
```

Değişkenlerin Faaliyet Alanları (Scopes)

- Faaliyet alanı devam eden bir değişkenin tekrar tanımlanması derleme esnasında hataya yol açar.

```
using System;
public class faaliyet_alani
{
   static void Main()
   {
            int x;
            { int x=20;
            }
   }
}
```

Sabitler

- Program boyunca değerinin değişmeyeceği düşünülen veriler sabit olarak tanımlanırlar. Bu tanımlamayı yapmak için tanımlama satırının başında const anahtar sözcüğünü kullanırız.

- Sabitlere ilk değer verilirken yine sabitler kullanılmalıdır. Değişken tanımlamada olduğu gibi sabitlerde de tanımlandıklarında mutlaka ilk değerleri verilmelidir.
- **const double pi=3.14**

Sabitler

```
using System;
public class faaliyet_alani
{
   static void Main()
   {
            int x=5,y;
            y=10;
            const int t=x+y;
   }
}
```

```
C:\WINDOWS\system32\cmd.exe

F:\c#\bolum2\compiled>csc sabitler.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.1433
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

sabitler.cs(8,14): error CS0133: The expression being assigned to 't' must be
        constant

F:\c#\bolum2\compiled>
```

Sabitler

```
using System;
public class faaliyet_alani
{
   static void Main()
   {
            const int x=5,y=10;
            const int t=x+y;
            Console.WriteLine(t);

   }
}
```

```
C:\WINDOWS\system32\cmd.exe
02/19/2008  10:31 PM    <DIR>          .
02/19/2008  10:31 PM    <DIR>          ..
02/19/2008  10:10 PM               181 faaliyet_alani.cs
02/19/2008  10:11 PM             3,072 faaliyet_alani.exe
02/19/2008  03:45 PM               103 ilk_program1.cs
02/19/2008  04:01 PM             3,072 ilk_program1.exe
02/19/2008  04:38 PM               111 ilk_program2.cs
02/19/2008  04:38 PM             3,072 ilk_program2.exe
02/19/2008  04:39 PM               182 ilk_program3.cs
02/19/2008  04:39 PM             3,072 ilk_program3.exe
02/19/2008  10:27 PM               130 sabitler.cs
02/19/2008  10:31 PM               153 sabitler2.cs
               10 File(s)         13,148 bytes
                2 Dir(s)  51,602,219,008 bytes free

F:\c#\bolun2\compiled>csc sabitler2.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.1433
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

F:\c#\bolun2\compiled>sabitler2
15

F:\c#\bolun2\compiled>_
```

Sabitler

```
using System;
public class faaliyet_alani
{   static void Main()
   {
            const int x=5,y=10;
            x+=2;
            const int t=x+y;
            Console.WriteLine(t);
   }
}
```

- Derleme gerçekleşmemektedir.
- Neden?

```
C:\WINDOWS\system32\cmd.exe
F:\c#\bolun2\compiled>csc sabitler3.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.1433
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

sabitler3.cs(7,2): error CS0131: The left-hand side of an assignment must be a
        variable, property or indexer

F:\c#\bolun2\compiled>
```

Sabitler

- Sabitlerle ilgili olarak 3 temel kural vardır: Bunlar:
- Sabitler tanımlandıklarında değerleri atanmalıdır. İlk değer verilmeyen değişkenler sabit olamazlar.
- Sabit ifadelere ancak sabit ifadelerle ilk değer atanabilir.
- Sabit ifadeler kendi yapılarından dolayı static bir nesne oldukları için ayrıca static anahtar sözcüğü kullanılmaz.

Değer ve Referans Tipleri

- Değer tipleri değişkenin değerini direkt bellek bölgesinden alırlar.

- Referans tipleri ise başka bir nesneye referans olarak kullanılırlar. Diğer bir değişler referans tipleri, heap alanında yaratılan nesnelerin adreslerini saklarlar.

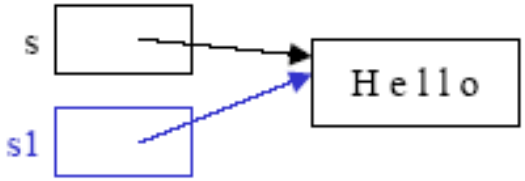
- Değer tipleri yaratıldıklarında stack bölgesinde oluşturulurlar. Referans tipleri ise kullanımı biraz daha sınırlı olan heap bellek bölgesinde saklanırlar.

Değer ve Referans Tipleri

- Temel veri tipleri (int,double, float ...) değer tipi; herhangi bir sınıf türü ise referans tipidir.
- İki değer tipi nesnesi birbirine eşitlenirken değişkenlerde saklanan değerler kopyalanarak eşitlenir ve bu durumda iki yeni bağımsız nesne elde edilmiş olur. Birinin değerini değiştirmek diğerini etkilemez.
- Fakat, iki referans tipi birbirine eşitlendiğinde bu nesnelerde tutulan veriler kopyalanmaz, işlem yapılan nesnelerin heap bölgesindeki adresleridir.
- İki nesne heap bölgesinde aynı yeri gösterdiği için, birinde yapılan değişiklik diğerini de etkileyecektir.

Değer ve Referans Tipleri

	Value Types	Reference Types
variable contains	value	reference
stored on	stack	heap
initialisation	0, false, '\0'	null
assignment	copies the value	copies the reference
example	int i = 17; int j = i;	string s = "Hello"; string s1 = s;

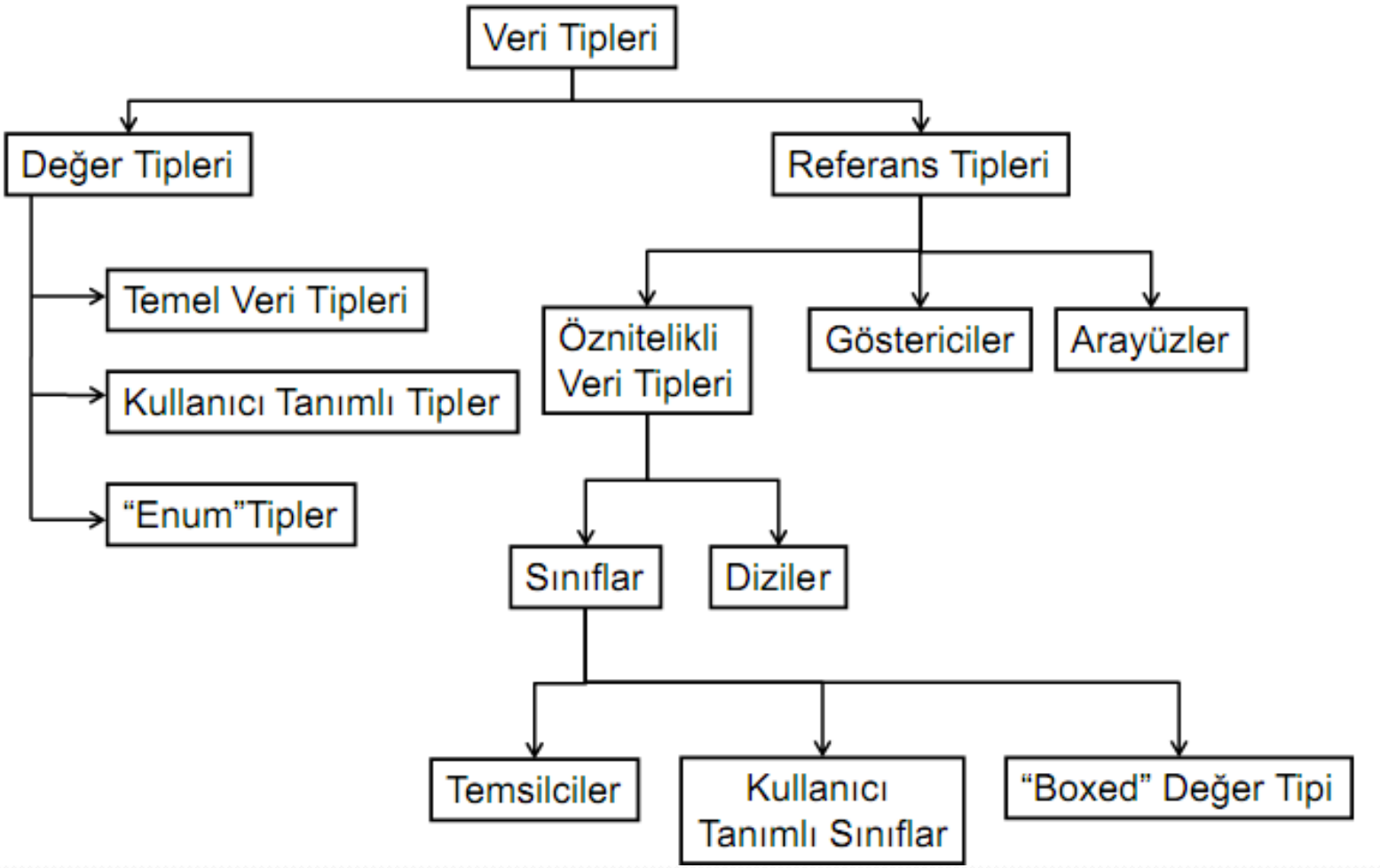


Değer ve Referans Tipleri

- CTS sayesinde, .NET platformu için geliştirilen bütün diller aynı veri tiplerini kullanırlar. Tek değişen veri türlerini tanımlama yöntemi ve sözdizimidir.

- C#'da önceden tanımlanmış temel veri tipleri 15 tanedir. (13 tanesi değer tipi, 2 tanesi ise referans tipi)

Değer ve Referans Tipleri



Değer Tipleri (Value Types)

- Değer tiplerinin tamamı **Object** denilen bir nesneden türemiştir. C#'da her nesne ya da veri tipi aslında **Object** tipidir.

- Değer tiplerinde bir nesnenin değeri direkt olarak saklıdır. Tanımlanan değer tiplerine aşağıdaki şekilde ilk değer ataması yapılabilir.

```
int a=3,b;
b=a;
```

Bu noktada a üzerindeki değişikliklerden b etkilenmeyecektir.

Değer Tipleri (Value Types)

- Değer tiplerine ilk değer verme;

  a=new int(); //yapıcı çalışır.(referans tip)

  a=0;
- Yukarıdaki iki satırda aynı işlemi yapar.

  float b; //derleyici hatası, atama yapılması gerekir.

  Error2 Use of unassigned local variable b'

  float b=new float(); //hata vermez yapıcı çalıştı

  b=3.21f //yeni atama yapılıyor

Değer Tipleri (Value Types)

Veri Tipi	Varsayılan Değer
bool	false
byte	0
char	'\0'
decimal	0.0M
double	0.0D
enum	enum sabiti tanımlamasındaki ilk değer
float	0.0F
int	0
long	0L
sbyte	0
short	0
struct	yapı içinde yer alan tüm değer tipleri varsayılan değere, referans tipler ise "null" değere atanır.
uint	0
ulong	0
ushort	0

Değer Tipleri (Value Types)

- Örn: Aşağıdaki programı bilgisayarınızda deneyin.

```
 using System;
public class varsayilan_degerler {
        static void Main() {
        bool a =new bool(); byte a1=new byte();
        char a2=new char(); decimal a3=new decimal();
        double a4=new double();
        float a5=new float();
        Console.WriteLine(a);
        Console.WriteLine(a1);                Console.WriteLine(a2);
        Console.WriteLine(a3);                Console.WriteLine(a4);
        Console.WriteLine(a5);
                        }
}
```

C# Tipi	.NET Framework	Tanım	Değer Aralığı
object	System.Object	Tüm CTS türleri için temel sınıf	-
bool	System.Boolean	Mantıksal Doğru/Yanlış	true ya da false
byte	System.Byte	8 bit işaretsiz tamsayı	0 ~ 255
sbyte	System.SByte	8 bit işaretli tamsayı	128 ~ 127
char	System.Char	Karakterleri temsil eder	16 Unicode karakterleri
decimal	System.Decimal	128 bit ondalıklı sayı	$\pm 1{,}5*10^{-28} \sim \pm 7{,}9*10^{28}$
double	System.Double	64 bit çift kayan sayı	$\pm 5*10^{-324} \sim \pm 1{,}7*10^{308}$
float	System.Single	32 bit tek kayan sayı	$\pm 1{,}5*10^{-45} \sim \pm 3{,}4*10^{38}$
int	System.Int32	32 bit işaretli tamsayı	-2.147.483.648 ~ 2.147.483.647
uint	System.UInt32	32 bit işaretsiz tamsayı	0 ~ 4.294.967.295
long	System.Int64	64 bit işaretli tamsayı	9.223.372.036.854.775.808 ~ -9.223.372.036.854.775.807
ulong	System.UInt64	64 bit işaretsiz tamsayı	0 ~ 18.446.744.073.709.551.615
short	System.Int16	16 bit işaretli tamsayı	-32.768 ~ 32.767
ushort	System.UInt16	16 bit işaretsiz tamsayı	0 ~ 65.535
string	System.String	Karakter Dizisi	Unicode Karakter Dizisi

Referans Tipleri (Reference Types)

- C# ‘ ta önceden tanımlı iki referans tipi vardır. **Object** ve **String**.

- Object türü C#’ta bütün türlerin türediği sınıftır. Diğer bir deyişle Object türünden bir nesneye herhangi bir veri türünden nesneyi atayabiliriz.

- Her nesne object türünden olduğu için bütün değerler ve nesneler object türünden bir değişkene atanabilir. Object türü özelleştirilerek farklı amaçlara yönelik kullanılabilirler. Object’e eşleştirme (Boxing) işlemi ve tersi, Object’i dönüştürme (Unboxing).

String Türü

- Referans türünden olan stringler, türü Unicode karakterlerden oluşan bir dizi gibi algılanmalıdır.

```
Strings1="Merhaba";
Strings2=".NET";
Strings3=s1+s2;
```

Stringleri arka arkaya eklemek için **+** operatörü kullanılır.

String Türü

- Özel anlamlar içeren karakterleri ifade etmek için \ ifadesini kullanırız (escape). Örn:

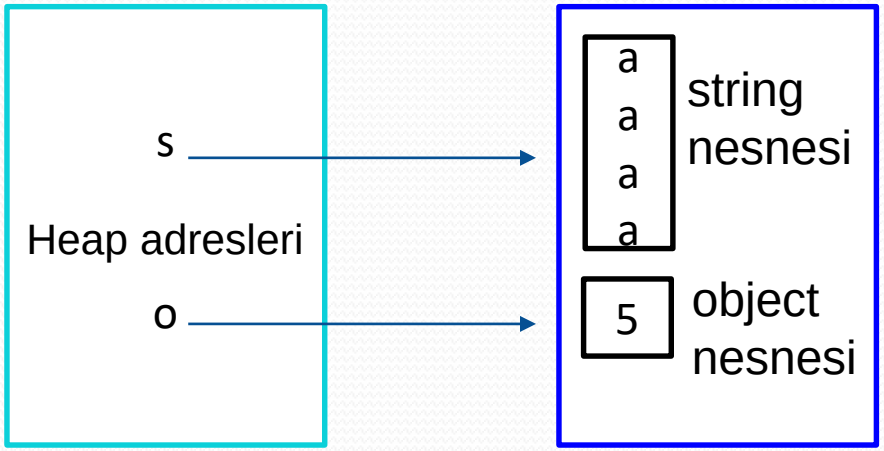
- **String path="C:\\WINDOWS\\assembly"**

- String içinde görünen ifadenin aynısını belirtmek için string ifadesinin önüne @ işareti konulur.Örn:

- **String path=@"C:\WINDOWS\assembly"**

Object ve String Veri Türü

- Object ve string değişkenler stack te oluşturulur değerleri heap te bulunur. Stack te ise heap adresleri saklanır.
- string s= new string('a', 4); object o =5;
- Stack Heap



Örnek

Aşağıdaki programı deneyiniz.

```
using System;
public class varsayilan_degerler
{
   static void Main()
   {
            object x;
            x=10;
            Console.WriteLine(x.GetType());
            x="B";
            Console.WriteLine(x.GetType());
            x=8.78F;
            Console.WriteLine(x.GetType());
            x=false;
            Console.WriteLine(x.GetType());
            x=5.489M;
            Console.WriteLine(x.GetType());
   }
}
```

var değişken yapısı (C# 3.0 dan sonra)

- **var,** anahtar sözcüğünü kullanarak bir veri tanımladığımızda, atadığımız ilk değerin tipini derleyici tarafından kendine değer tipi olarak belirlemektedir.
  - `var vr_Degisken = "string değer";` var tipinde tanımladığımız değişkenimize string tipinde değer ataması yapıyoruz. Derleyicimiz bu değer atamasından sonra değişkenimizin tipini string olarak belirleyecek ve RAM üzerinde string olarak tutulacaktır.
  - Console.WriteLine(vr_Degisken.GetType()); //bu şekilde ekrana tipini yazdırabilirsiniz.
  - **`vr_Degisken = 61;`** // tekrar aynı değişkenimize int tipinde değer atamaya çalıştığınızda derleyici hata verecektir. /* Cannot implicitly convert type 'int' to 'string'. (var için en küçük tamsayı tipi int dir).
  - Hata mesajı, veri tipi string olan değişkene int tipinde değer atamaya çalışıyorsunuz ya da dönüştürme yapılamıyor şeklinde de yorumlanabilir. Yani var tipindeki değişkenimize ilk atadığımız değerin tipini kendine tip olarak belirleyecek, sonraki atamalarda da aynı tipte değer isteyecektir.

var değişken yapısı

- object te ise durum farklıdır. Her gelen değer boxing olur yani gelen veri türüne göre object olarak RAM de saklanır.
  - **obj_Degisken = "string değer";** // değişkenimize string değer ataması yapıyoruz. object tipinden değişkenimize string değer ataması yaparken Boxing yani Kutulama işlemi gerçekleşiyor ve değişkenimiz RAM üzerinde değişime uğramadan object olarak kaydediliyor.
  - **obj_Degisken = 61;** //obj_Degisken değişkenimize şimdide int tipinde bir atama yapıyoruz. Değişkenimiz RAM üzerinde tip değişimine uğramadan Boxing işlemi gerçekleştirerek object olarak tutuluyor.

  - Not:var değişken tipi, diller arası, databaseler arası entegrasyonu sağlarken veri tipleri uyuşmazlığını gidermek için oluşturulmuş bir tiptir. Yani C#'ta int ile tanımlanan bir değişken Delphi 'de başka türlü tanımlanabilir. var değişken tipide bütün dillerde evrensellik özelliği taşımaktadır. Sizlere tavsiyem normal kodlamada verinizin tipine göre normal değişken tanımlayınız.

dynamic değişken yapısı (C# 4.0 dan sonra)

- **dynamic**, derleme zamanında (compile-time) gerçekleştirilen tip kontrollerinin pas geçilerek, bu kontrollerin tamamen çalışma zamanında (runtime) gerçekleştirilmesidir.
  - `dynamic str = "JavaBlog";`
  - `dynamic i = 2;`
  - `Console.WriteLine(str.GetType()); //System.String`
  - `Console.WriteLine(i.GetType()); //System.Int32`
- **dynamic,** dynamic kullanılarak oluşturulan değişkenler, çalışma zamanında kendilerine atanan değişkenleri dinamik bir şekilde çözümleyerek değişkenin tipi o an ne ise, o tip için işlemleri yapmasını sağlar.
- `dynamic str = "fatih";`
  - `string s = str + "javablog";`
  - `Console.WriteLine(s);`
  - `str = 12;`
  - `str = 4 + str;`
  - `Console.WriteLine(str);`

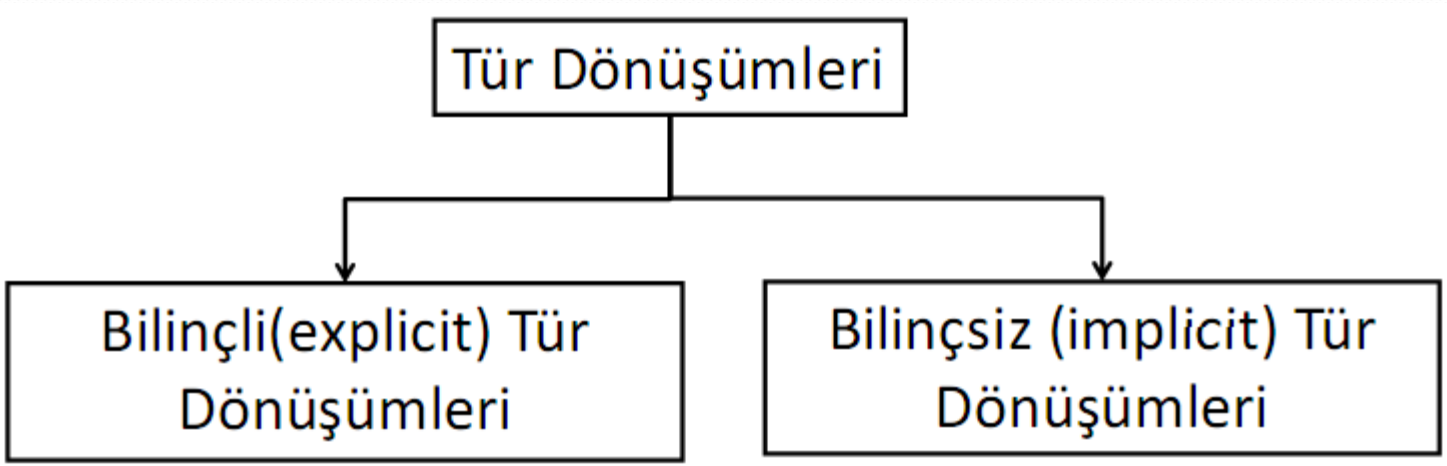
Çıktı:
fatihjavablog
16

object ve dynamic Arasındaki Farklar

- **dynamic,** object gibi davranır ancak aralarındaki en büyük fark, object tipine var olan tüm sınıf örnekleri atanabilse de, static tip yine System.Object'dir. Uygulamada gerçek tipi kullanabilmeniz için object tipi cast etmeniz gerekmektedir. dynamic'de ise buna gerek yoktur. .NET zaten runtimeda tipi otomatik olarak belirleyerek gerekli casting işlemini yapar.
  - `object o = "fatih";`
  - `int t = ((string)o).Length;// unboxing yapılıyor.`
  - `dynamic d = "fatih";`
  - `int k = d.Length;`
  - `object Rakam = 10;`
  - `Rakam = Rakam + 10;`
  - `/// Derleme zamanı hatası. Çünkü derleme anında Rakam değişkeni System.Object tipinde`
  - `Rakam = (int)Rakam + 10;`
  - `/// Hatasız atama için, unboxing yapmamız gerekiyor`
  - `dynamic sayi = 5;`
  - `sayi =sayi+ 5;`
  - `/// Hata vermez`

Tür Dönüşümleri

- Farklı türden değişkenlerin aynı ifade içinde işlem görmeleri için tür dönüşümü kullanılır. Tür dönüşümlerini aşağıdaki şekilde gruplara ayırmak mümkündür:



Bilinçsiz (Implicit) Tür Dönüşümleri

- Derleyici tarafından bir değişkeni tanımladığımız türün dışında geçici olarak başka bir türe çevirmeye **bilinçsiz tür dönüşümü** denir. Örn:

```
using System;
public class Tur_donusumu1
{
  static void Main()
  {
            int x=5;
            float a;
            a=x;
            Console.WriteLine(a);

  }
}
```

Bilinçsiz (Impicit) Tür Dönüşümleri

- Bilinçsiz yapılan tür dönüşümlerinde bir nesnenin türü asla kalıcı olarak değiştirilmez. Bilinçsiz yapılan tür dönüşümleri 2 şekilde gerçekleştirilebilir.
  - Küçük Türün Büyük Türe Dönüştürülmesi
  - Büyük Türün Küçük Türe Dönüştürülmesi

Tür Dönüşümleri

- **Küçük türün büyük türe dönüştürülmesi (Otomotik tip dönüşümü):**
- Küçük tür büyük türe dönüştürülürken fazla bitler sıfır ile doldurulur. Küçük türün yüksek anlamlı bitlerinin sıfırla beslenmesi değişkendeki değeri değiştirmediği için tür dönüşümünde herhangi bir veri kaybı olmaz.

Örn: byte a=12;

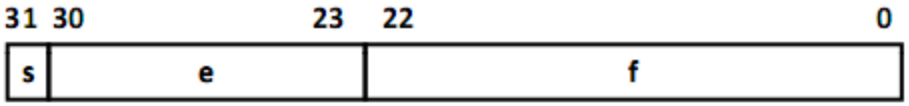
```
int b;
b=a;
```

Tür Dönüşümleri

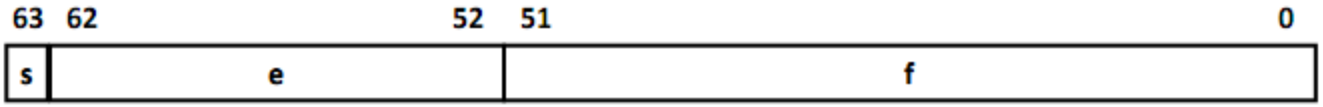
```
using System;
public class Tur_donusumu1
{
  static void Main()
  {
            byte a=20;
            int b;
            b=a;
            Console.WriteLine(b);

            float f=20f;
            double d;
            d=f;
            Console.WriteLine(d);

            char c='a';
            decimal m;
            m=c;
            Console.WriteLine(m);
  }
}
```

Tür Dönüşümleri

float sayı formatı



double sayı formatı



s : işaret biti

e : üsleri temsil eden bitler

f : noktalı (anlamlı) kısım

Tür Dönüşümleri

Tür	Bilinçsiz Dönüşüme İzin verilen Türler
sbyte	byte, ushort, uint, ulong, veya char
byte	sbyte veya char
short	sbyte, byte, ushort, uint, ulong, veya char
ushort	sbyte, byte, short, veya char
int	sbyte, byte, short, ushort, uint, ulong, veya char
uint	sbyte, byte, short, ushort, int, veya char
long	sbyte, byte, short, ushort, int, uint, ulong, veya char
ulong	sbyte, byte, short, ushort, int, uint, long, veya char
char	sbyte, byte, veya short
float	sbyte, byte, short, ushort, int, uint, long, ulong, char, veya decimal
double	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, veya decimal
decimal	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, veya double

Tip Dönüşümleri

```
using System;
class Otomatik_tip
{
        public static void Main()        {
                int a;
                float b=32.32f;
                double c;
                c=b;
Console.WriteLine("b'nin değeri="+b+"\nc'nin değeri="+c);
        }
}
```

Tür Dönüşümleri

```
using System;
public class Tur_donusumu1
{
  static void Main()
  {
            short b1=100;
            char c1=b1;
            bool b2=true;
            int i1=b2;
            double d1=10.2;
            int i2=d1;
            decimal m1=20.6M;
            double d2=m1;
            byte bt1=65;
            char c2=bt1;
            float f1=34.78F;
            decimal d3=f1;
  }
}
```

Tür Dönüşümleri

- Bazı türler arasında tür dönüşümü yapmak mümkün değildir. Bunlar :

a. Bool, decimal ve double türünden herhangi bir türe
b. Herhangi bir türden char türüne
c. Float ve decimal türünden herhangi bir türe (float türünden double türüne dönüşüm hariç)

Tür Dönüşümleri

- **Büyük türün küçük türe dönüştürülmesi:** Büyük türlerin küçük türlere otomatik dönüştürülmesi C#'da yasaklanmıştır. Eğer bu tür bir dönüştürme (bilinçsiz olarak) mümkün olsaydı birtakım veri kayıpları yaşanacaktır.

- İstenmeyen durum. Ancak "()" cast operatörü ile yapılır.

Bilinçli (Explicit) Tür Dönüşümleri

- Bilinçli (explicit) tür dönüşümü genellikle derleyicinin izin vermediği dönüşümlerde kullanılır.

- Bu tür dönüşümlerde de yine küçük türler büyük türe ya da tersi dönüşümler yapılabilir.

- Küçük türlerin büyük türlere çevrilmesi aynı bilinçsiz dönüşümde olduğu gibidir.

Tür Dönüştürme Operatörü

- Bilinçli tür dönüşümü yapılırken "tür dönüştürme operatörleri" kullanılır. Tür dönüştürme operatörü parantez içinde değişken ya da sabitten önce yazılır.

  - (dönüştürülecek tür) degişken_yada_sabit;

Tür Dönüştürme Operatörü

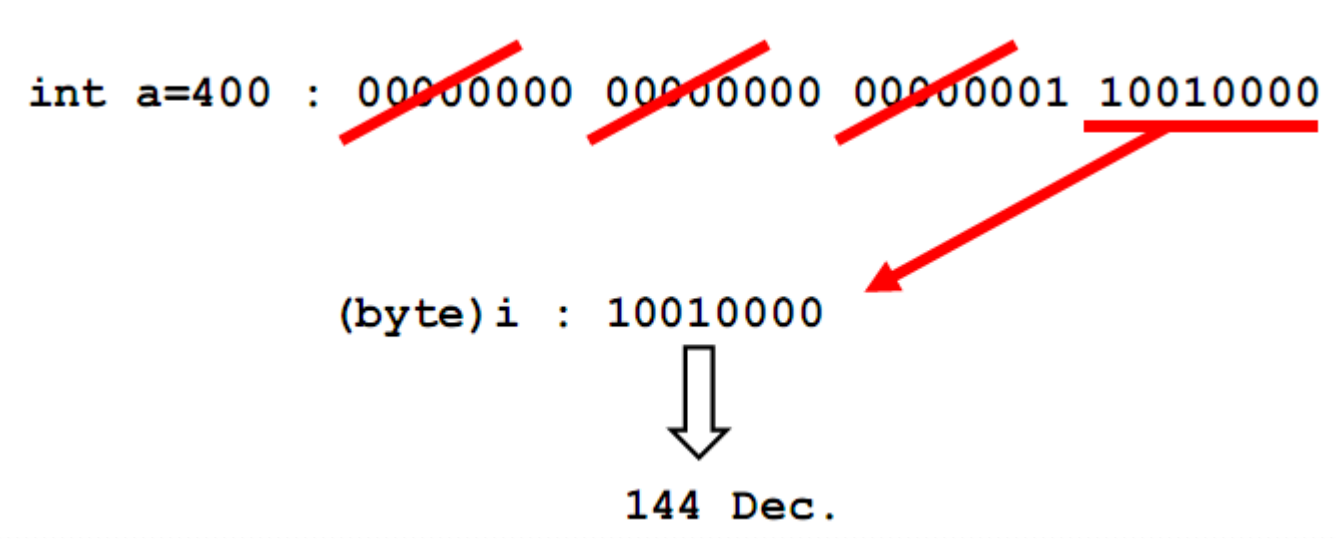
✓ Bilinçsiz yapılan tür dönüşümlerinde büyük türler, küçük türlere dönüştürülemiyordu. Eğer tür dönüştürme operatörü kullanılırsa bu işlem mümkün olur. Örn: Aşağıdaki programı deneyin:

```
using System;
public class Tur_donusumu1
{
  static void Main()
  {
        int a=400;
        byte b=(byte)a;
        Console.WriteLine(b);
  }
}
```

Tür Dönüştürme Operatörü

✓ Programı çalıştırdığımızda ekrana 144 yazdırdı.

Neden?



Tür Dönüşümleri

```
"C:\Documents and Settings\SoNDuRaK\ConsoleApplication15\...
Double integere çevrildi=1

i'nin değeri=123  iken  b'nin değeri=123
i'nin değeri=258  iken  b'nin değeri=2

u'nun değeri=32146  iken  s'nin değeri=32146
u'nun değeri=35000  iken  s'nin değeri=-30536

90sayısının char'a dönüştürürsek=Z  olur
Press any key to continue_
```

```
using System;
class Bilinçli_tip {
 public static void Main() {
    double d1,d2; int i;    byte b; char c;  uint u; short s;  d1=5.0;d2=4.0;
    //double int e dönüştü veri kaybı var,virgülden sonrası atılır
    i= (int) (d1/d2);Console.WriteLine("Double integere çevrildi="+i); Console.WriteLine();
   //int'i byte dönüştür, Veri kaybı yok.
    i=123;   b=(byte) i; Console.WriteLine("i'nin değeri="+i+"  iken  b'nin değeri="+b);
   //Veri kaybı var.
    i=258;   b=(byte) i; Console.WriteLine("i'nin değeri="+i+"  iken  b'nin değeri="+b);
   Console.WriteLine();
   //uint'i short'a dönüştür
    u=32146;   s=(short) u;   //Veri kaybı yok.
   Console.WriteLine("u'nun değeri="+u+"  iken  s'nin değeri="+s);
   u=35000;    s=(short) u;  //Veri kaybı var.
   Console.WriteLine("u'nun değeri="+u+"  iken  s'nin değeri="+s);  Console.WriteLine();
    //int'i char'a dönüştür.
   i=90; c=(char) i; Console.WriteLine(i+"sayısının char'a dönüştürürsek="+c+"  olur"); }
}
```

Checked ve Unchecked

- Tür dönüşümlerinde veri kayıplarında programa hata uyarısı verdirebilmek için checked deyimi kullanılır. Checked anahtar sözcüğü ile çalışma zamanında oluşabilecek veri kayıplarının olabileceği durumlarda hata vermesini sağlayabiliriz.

```
// unchecked checked işlemini ters çevirir.
using System;
class turdonusum   {
        static void Main() {
                int i=256;
                byte b;
                checked  //Taşma olduğundan program hata verir.
                {
            b=(byte) i;
             }
             Console.WriteLine(b);
        }
}
```

Checked ve Unchecked

- checked bir blok oluşturduğu için içinde yapılan değişken tanımlamaları dış bloklarda kullanılamaz.

```
using System;
class turdonusum    {
        static void Main() {
                int i=256;
                checked
                {
                 byte b=(byte) i;
                   }
                 Console.WriteLine(b); // Hata verir
        }
}
```

Checked ve Unchecked

- Normal şartlarda yapılan işlemler "**unchecked**"'dir Böyle bir ifadenin konmasının nedeni uzun "checked" blokların oluşturulması istenebilir.

- Bu durumlarda çok fazla blok oluşturmamak için "unchecked" ifadesi kullanılabilir.

Checked ve Unchecked

```
using System;
class TurDonusumu11
{
    static void Main()
    {
        int i1 = 255;
        int i2 = 500;
        byte b, c;

        checked
        {
            b = (byte)i1;
            Console.WriteLine(b);

            unchecked
            {
                c = (byte)i2;
            }
            Console.WriteLine(c);
        }
    }
}
```

- **Referans ve Değer Türleri Arasındaki Dönüşüm**

Object Türü ve ToString() Metodu

- Temel veri türleri de dahil olmak üzere bütün veri tipleri object denilen bir referans türünden türemiştir. Türeme, kalıtım yolu ile olduğu için var olan özellikler her zaman korunur.

- C# herşey nesne(object) referans türünden türetilmiştir. Temelde bir sınıf vardır. Örneğin object sınıfının ToString() metodu bütün temel veri ve referans türlerinde kullanılır.

ToString() Metodu

- **.ToString()** metodu bütün temel türlerde ya da referans türlerde kullanılabilir. Amacı ise string'e dönüşüm işlemdi yapmaktır.

- string str = 345.59f.ToString()

  - – **56.ToString();**
  - – **12.34F.ToString();**

ToString() Metodu

Örn:

…..

```
Static void Main() {
int a=5;
int b=7;
string a1=a.ToString();
string b1=b.ToString();
Console.WriteLine(a+b);
Console.WriteLine(a1+b1); }
```

Sonuç: 12

57

Boxing İşlemi

- Günümüzdeki popüler dillerde referans ve değer tipleri arasında dönüşüm yapılmamaktadır.

- Böyle bir çevrime ihtiyaç duyulduğunda “Boxing” kutulama yapılır. Bu yöntem değer tipindeki verileri “object” nesnesine çevirir.

- Bir değer tipini referans tipe atadığımızda stack’teki bilgi bit olarak heap’e kopyalanır ve stack’teki object türünden olan değişken heap’i gösterecek şekilde ayarlanır.

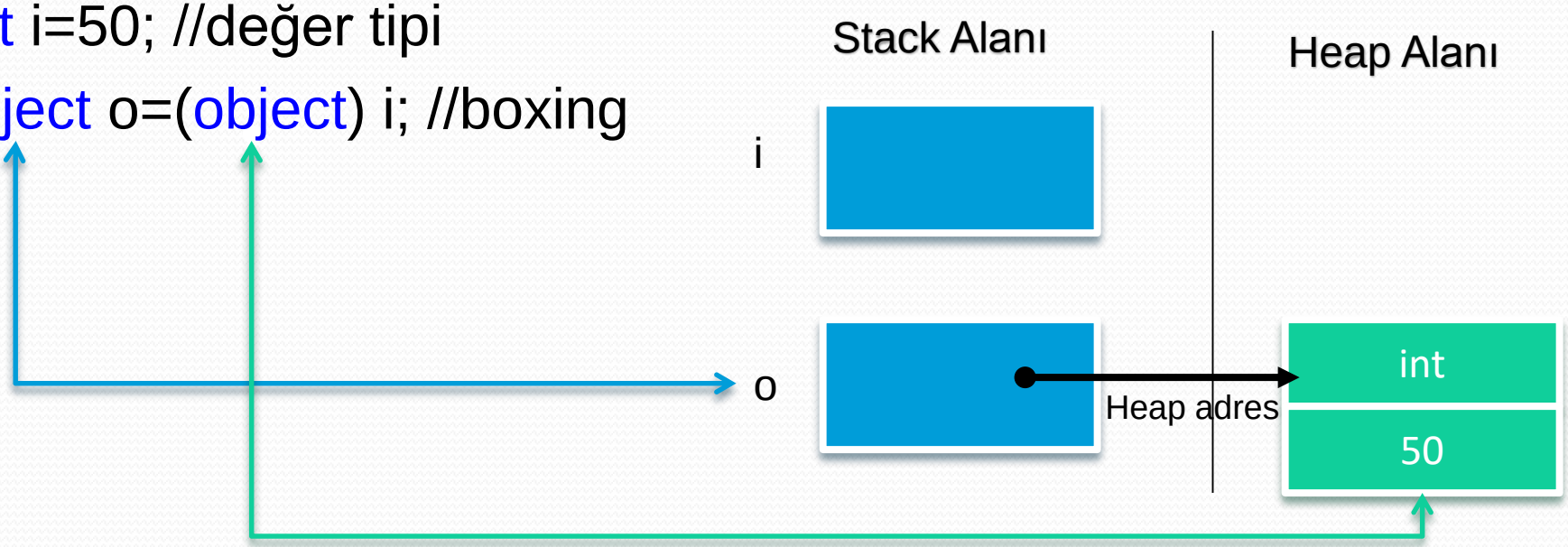
Örn: Bilinçsiz boxing işlemi.

```
int i=50; //değer tipi
object o=i; //boxing
```

Referans-Değer Dönüşüm(Boxing)

Örn: Bilinçli boxing işlemi.

```
int i=50; //değer tipi
object o=(object) i; //boxing
```



boxing işleminden sonra stack (yığın) üzerinde oluşturulan object türünden **o** değişkenine, stack (yığın)'da bulunan **i** değişkeninin kopyası heap (öbek)'te oluşturulup **o** değişkenine referans eder ve **o** değişkeni heap alanındaki adresi tutar.

Unboxing İşlemi

- Heap alanındaki nesnelerin değerlerinin bit olarak stack bölgesine kopyalanması işlemine "**unboxing**" adı verilir.

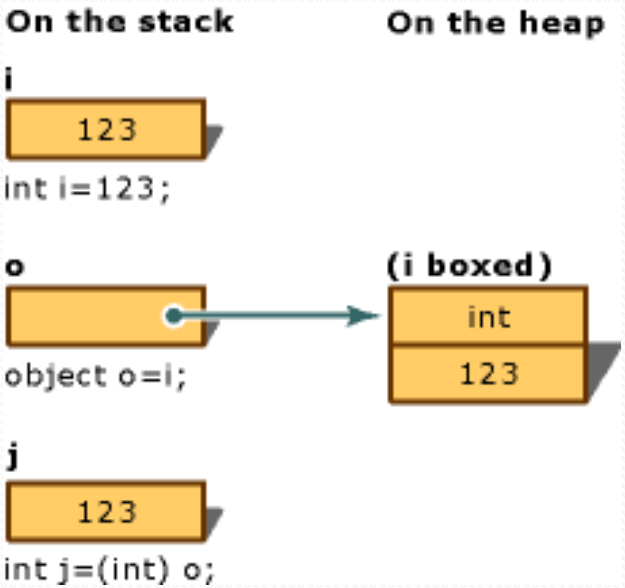
- Boxing işleminin tam tersidir. Bu işlem sonucunda referans türler değer türüne dönüştürülmüş olur. Aşağıdaki koşullara uyularak yapılmalıdır.

  - Unboxing işlemine tabi tutulacak nesnenin daha önceden boxing işlemine tabi tutulmuş olması gerekir.

  - Boxing işlemine tabi tutulmuş olan bu nesnenin unboxing işlemi sırasında doğru türe dönüştürülmesidir

Unboxing İşlemi

- Unboxing işlemi bilinçsiz bir biçimde yapılmaz, mutlaka tür dönüşüm operatörü kullanılmalıdır.
  - Örn:
  -
```
int i = 123;        // değer tipi (value type)
object o = i;       // boxing
int j = (int)o;     // unboxing
```



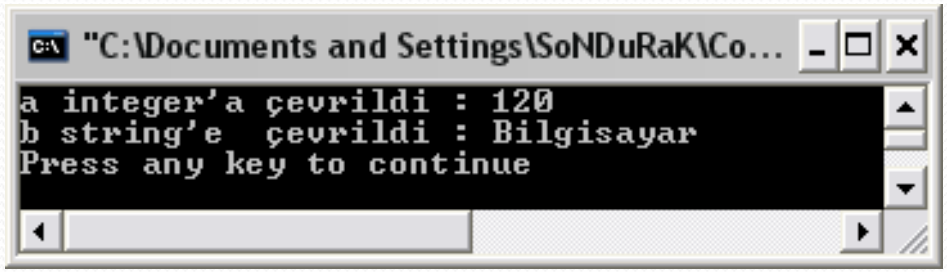
Boxing işlemi normal bir atama işleminden 20 kat daha uzun sürmektedir.
Unboxing işlemi ise yine normal bir atama işleminden 4 kat daha uzun sürmektedir.
Bu yüzden gerekli olmayan aşırı boxing ve unboxing kullanımı uygulamanızı gözle görülür bir biçimde yavaşlatacaktır.

Unboxing İşlemi

- Aşağıdaki kod parçasını bilgisayarınızda deneyiniz:

```
using System;

class TurDonusumu14
{
    static void Main()
    {
        int i = 10;
        object o = i;

        long l = (long)o;

        Console.WriteLine(i);
        Console.WriteLine(l);
    }
}
```

- **Örnek:**
- using System;
- class Boxing_UnBoxing_metodu{
- static void Main() {
- object a=120; //a'ya Boxing uygulandı
- object b="Bilgisayar"; //b'ye Boxing uygulandı
- int sayı=(int) a; //a'ya UnBoxing uygulandı
- string str=(string) b; //b'ye UnBoxing uygulandı
- Console.WriteLine("a integer'a çevrildi : "+sayı);
- Console.WriteLine("b string'e çevrildi : "+str);
- }
- }



System.Convert Sınıfı ile Tür Dönüşümü

- .NET sınıf kütüphanesinde yer alan “Convert” sınıfı string değerleri ve temel veri türlerini birbirine çevirmek için kullanılır. Her bir veri türü için ayrı bir çevrim fonksiyonuna sahiptir.

System.Convert Sınıfı ile Tür Dönüşümü

```
Convert.ToBoolean(str)
Convert.ToByte(str)
Convert.ToInt32(str)
Convert.ToChar(str)
…
int a=50;
byte b=Convert.ToInt32(a);//Yanlış tür dönüşümü
 string c="12.34";
a=float.Parse(c);
```

System.Convert İle Tür Dönüşümü

-

Metot	Açıklama
Convert.ToBoolean(str)	str nesnesini bool türüne çevirir.
Convert.ToByte(str)	str nesnesini byte türüne çevirir.
Convert.ToSByte(str)	str nesnesini Signed Byte türüne çevirir.
Convert.ToInt16(str)	str nesnesini short türüne çevirir.
Convert.ToUInt16(str)	str nesnesini ushort türüne çevirir.
Convert.ToInt32(str)	str nesnesini int türüne çevirir.
Convert.ToUInt32(str)	str nesnesini uint türüne çevirir.
Convert.ToInt64 (str)	str nesnesini long türüne çevirir.
Convert.ToUInt64(str)	str nesnesini ulong türüne çevirir.
Convert.ToSingle(str)	str nesnesini float türüne çevirir.
Convert.ToDouble(str)	str nesnesini double türüne çevirir.
Convert.ToDecimal(str)	str nesnesini decimal türüne çevirir.
Convert.ToChar(str)	str nesnesini char türüne çevirir.

System.Convert İle Tür Dönüşümü

```
using System;
Class convert_metodu {
        static void Main()
    {
                int a=50;
                string d=“50";
                int b=Convert.ToInt32(a);
                double c=Convert.ToDouble(d);
                Console.WriteLine("b : "+b+"\nc : "+c);
     }
}
```

System.Convert İle Tür Dönüşümü

```
using System;

class TurDonusumu15
{
    static void Main()
    {
        string s1, s2;
        int i1, i2, t;

        Console.Write("1.Sayıyı Girin:");
        s1 = Console.ReadLine();

        Console.Write("2.Sayıyı Girin:");
        s2 = Console.ReadLine();

        i1 = Convert.ToInt32(s1);
        i2 = Convert.ToInt32(s2);

        t = i1 + i2;

        Console.WriteLine("Toplam = " + t.ToString());
    }
}
```

System.Convert İle Tür Dönüşümü

- Dönüşüm işleminin sonucunda anlamlı bir sonuç elde edilemeyeceği durumlarda hata meydana gelir.

```
using System;

class TurDonusumu16
{
    static void Main()
    {
        char a = 'a';
        bool b = Convert.ToBoolean(a);
    }
}
```

```
int a=0;  int d = (int) 6.0; //float -> integer dönüşüm
object k="merhaba"+15; //object türü, hem karakter hem sayısal
float b=10.5f; //float tanımı
double c=20.1; //double tanımı
Double dd = new double(); //referans olarak double tanımı
const double pi = 3.14; //sabit tanımı
string[] isimler ={ "Ozlem","Nesrin", "Ozge", "Fulya" }; //string dizi tanımı
object[] isim ={ "Ozlem","Nesrin", "Ozge", "Fulya" }; //object dizi tanımı
string s = "true"; //string tanımı
string dd="“12.45f”;
b= float.Parse(dd); //string tip float’a çevriliyor
b=Convert.ToSingle(dd); //String float’a çevriliyor
a =Convert.ToInt32(b + c); //float -> integer
bool cevap = (Convert.ToBoolean(s)); //boolean tanımı
Console.Write((float)a/d+"\n");  //  () operatörü ile float dönüşümü
Console.WriteLine("cevap=" + cevap); // cevap = true  yazar
Console.WriteLine(k.GetType()); //bulunduğu sınıf,alanadını verir.
a = Convert.ToSingle(Console.ReadLine()); //girilen değer float’a çevriliyor
Console.WriteLine(“a={0} b= {1} c={2} d={3} ", a, b, c,d);
if (isimler[0].Equals("Ozlem")==true)  //eğer dizinin ilk elemanı Ozlem ise yazar
    Console.WriteLine("birinci isim Ozlem");
foreach (string ss in isimler)  // string dizi içindeki her bir eleman yazdırılıyor
{ Console.WriteLine(ss); }
```

Visual Studio.Net -C#

3. HAFTA

Operatörler,

Akış kontrol mekanizmaları,

Diziler

Visual Studio.Net -C#
3. HAFTA
Operatörler,
Akış kontrol mekanizmaları

Operatörler

- Operatörler, önceden tanımlanmış birtakım matematiksel ya da mantıksal işlemleri yapmak için kullanılan özel karakterler ya da karakterler topluluğudur.

- Operatörlerin işlem yapabilmek için ihtiyaç duydukları değerlere ise "**operand**" denir.

- **x+y -> "+" operatör, "x" ve "y" operand**
- **++z -> "++" operatör, "z" operand**

Operatörler

- Operatörler yapılarına ve işlevlerine göre sınıflandırılabilir

- ***Yapılarına göre***
  - ***Unary (tekil) operatörler***
    - **() , [], ++, _, +, _, !, ~, new, checked , unchecked , typeof, sizeof**
  - ***Binary (İkili) Operatörler***
    - *,/,%, + ,-,<,<=,>,>=, as, is, <<,>>,==,!=,&, ^, | ,&&, ||, = ,*=,/=,%=,+=,-=,<<=, >>=,&=,^=,| =
  - ***Ternary(üçlü) Operatörler***
    - ***? :***

Operatörler

- *İşlevlerine göre*
  - Aritmetik operatörler (++, --, *, /, +, -, %)
  - Karşılaştırma(<,>,!=, ==,as, is)
  - Mantıksal(|| ,&&, !)
  - Bitsel (>>,<<, ~(değil), &, |, ^ (xor-aynı 0,farklı 1))
  - Atama ve işlemli atama(*=, %=, -= vs)
  - Özel amaçlı (? : operatörü (a<b ? min=a : min=b))
    - new operatörü :yeni bir nesne oluşturur.
    - typeof : nesnelerin GetType() metodu gibi bağlı olduğu System türünü verir.
    - Sizeof, (), [],+,-,*,&,->, . , checked, unchecked

Operatör Önceliği

- Bazı ifadelerde birden fazla operatör kullanılmış olabilir. Bu gibi durumlarda operatörlerin belirli bir çalışma sırası olacaktır.
- **Büyükten küçüğe işlem önceliği;**
  - Parantez
  - Üs Alma
  - Çarpma , Bölme
  - Toplama, Çıkarma
    - X = 6 + 8*10
    - = 6 + (8 * 10) = 86
    - y=(7+(6+4)*5)*2 = 114

Operatör Önceliği

- Bazı operatörlerin öncelik sırası aynı olabilir. Çarpma-bölme ya da toplama- çıkarma gibi operatörler aralarında aynı önceliğe sahiptir. Böyle bir durumda sağdan sola ya da soldan sağa öncelik ilişkisi devreye girer.
- **Atama operatörleri hariç bütün binary operatörler soldan sağa önceliğe sahiptir, işlemler bu yönde yapılır.**

```
using System;

class Operatorler1
{
    static void Main()
    {
        int i = 5 * 4 / 6;
        Console.WriteLine(i);

        i = 5 * ( 4 / 6 );
        Console.WriteLine(i);
    }
}
```

Operatör Öncelik Tablosu

Birinci Öncelikliler	x.y (nesne erişim operatörü) , f(x) , dizi[x], x++ , x-- , new , typeof , checked , unchecked
Unary Operatörler	+ , - , ! , ~ , ++x , --x , (tür)x
Çarpma Bölme	* , / , %
Toplama ve Çıkarma	+ , -
Kaydırma Operatörleri	<< , >>
İlişkisel ve Tür Testi	< , > , <= , >= , is , as
Eşitlik Operatörleri	==, !=
Mantıksal VE (AND)	&
Mantıksal Özel Veya (XOR)	^
Mantıksal VEYA (OR)	|
Koşul VE	&&
Koşul VEYA	||
Koşul Operatörü	? :
Atama ve İşlemli Atama	= , *= , /= , %= , += , -= , <<= , >>= , &= , ^= , |=

- Yukarıdan aşağı doğru bir öncelik sıralaması vardır. Aynı satırda belirtilmiş olan operatörler ise aynı önceliğe sahiptirler. Bunlar arasında (atama operatörleri hariç) soldan sağa öncelik ilişkisi söz konusudur.

Operatör Önceliği

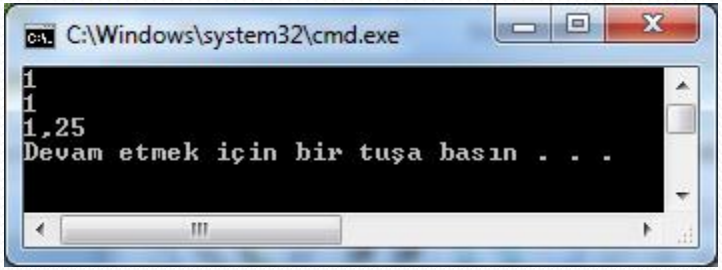
```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 4 +-6;
            Console.WriteLine(i);
        }
    }
}
```

Çıktı
-2

Aritmetik Operatörler

- **+ , -, * ve / operatörleri:**
- Toplama, çıkarma, çarpma ve bölme operatörleridir. İstenilen tipteki sayısal değerlerle kullanılabilir. Bölme işleminde kullanılan değerin tipine göre farklı sonuçlar elde edilebilir.

```
using System;

class Operatorler3
{
    static void Main()
    {
        int i = 5 / 4;
        float f1 = 5 / 4;
        float f2 = 5f / 4f;

        Console.WriteLine(i);
        Console.WriteLine(f1);
        Console.WriteLine(f2);
    }
}
```



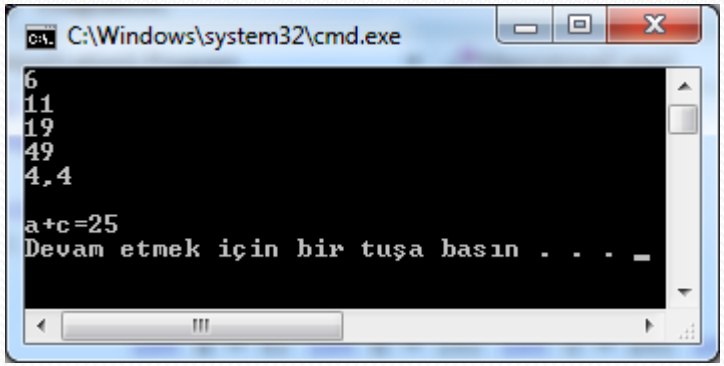
Aritmetik Operatörler

- **% operatörleri:**
  - Bölümden sonra kalanı bulmak yani “mod” almak amacıyla kullanılır Tüm sayısal türler için kullanılabilir.
  - x=10 % 3 -> x= 1
  - y= -7 % 4 -> y= -3
  - f=2.5f % 1.2f ->f = 0.1

- **++ ve -- Operatörleri:**
  - Önüne ya da sonuna geldiği değişkenin değerini 1 arttırır ya da 1 azaltır. İki şekilde kullanılırlar: Önek (prefix) ve Sonek (postfix). Bütün sayısal değerlerde kullanılabilir.

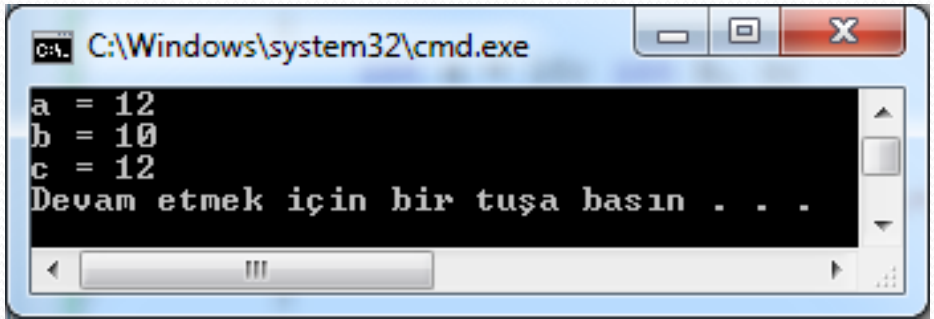
Aritmetik Operatörler

```
using System;

class Program
{
  static void Main(string[] args)
  {
   int a = 5; int b = 10; int c = 20;
   int d = 50; float f = 3.4f;
   a++; ++b; c--; --d;  f++;
   Console.WriteLine("{0}\n{1}\n{2}\n{3}\n{4}\n",a,b,c,d,f);
   Console.WriteLine("a+c={0}",a+c);
  }
}
```



Aritmetik Operatörler

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int a = 10; int b, c;
        b = a++; c = ++a;
        Console.WriteLine("a = {0}", a);
        Console.WriteLine("b = {0}", b);
        Console.WriteLine("c = {0}", c);
    }
}
```



Aritmetik Operatörler

- Arttırma ve Azaltma operatörlerinde dikkat edilmesi gereken durumlar şu şekilde özetlenebilir:

```
a = 4
b = a++
    a=5, b=4

a = 4
b = ++a
    a=5, b=5

a = 4
b = a--
    a=3, b=4

a = 4
b = --a
    a=3, b=3
```

Karşılaştırma Operatörü

- İki sayısal değerin birbirlerine göre büyüklüğünü, küçüklüğünü ve eşitliğini kontrol eder. Bu operatörlerin ürettiği değerler “true” ve “false” değerleridir.

= =	Eşittir	!=	Eşit Değildir
<	Küçüktür	>=	Büyük Eşittir
>	Büyüktür	<=	Küçük Eşittir

Karşılaştırma Operatörü

```
using System;
class ilişkisel_operatörler
{
    public static void Main()
    {
        int a = 10, b = 15;
        if(a == b) Console.WriteLine("a,b'ye eşittir");
        if(a < b) Console.WriteLine("a,b'denküçüktür");
        if(a > b) Console.WriteLine("a,b'denbüyüktür");
        if(a != b) Console.WriteLine("a,b'ye eşit değil");
        if(a >= b) Console.WriteLine("a,b'ye eşittir veya büyüktür");
        if(a <= b) Console.WriteLine("a,b'ye eşittir veya küçüktür");
    }
}
```

Operatörler

- **as operatörü** sık kullanılmayan bir operatördür. Uygun türler arası dönüşüm sağlar as operatörü sonucunda referans türde değer elde edilir.

- Genellikle object türündeki nesneleri farklı bir türe çevirmek için kullanılır. Çevirme işlemi başarısız ise null değeri üretir. Kullanımı:

- <referans tipi üretilecek ifade> as <referans tipi>

Örn:

- `object i="50";`
- `string s=i as string;`

- Not: **as** operatörü yalnızca referans türlü tipler ile kullanılabilir. Eşitliğin sağ tarafından geçerli bir tip gelmezse null atanmasını sağlar. Değer türlü tipler null alamadığından ötürü bu operatör ile kullanılamazlar.

Operatörler

- **is operatörü,** çalışma zamanında bir nesnenin türünün operand ile verilen türe uyumlu olup olmadığını kontrol eder. Sık kullanılmaz**.**

- Kullanımı:
- <ifade>is<tür>

- Örn:
- `int i=50;`
- `bool b1=i is int`

Çalışma sonucu “true” değerini verir.

Mantıksal Operatörler

&:	**AND**	**&&:**	**Kısa devre AND**
|:	**OR**	**||:**	**Kısa devre OR**
!:	**NOT**		

- **&& (VE) Operatörü:**
- "true" ya da "false" değerindeki operandları mantıksal VE ile işler. Operandlardan biri "false" ise "false" değeri üretir.
- **Kullanımı:** <ifade> && <ifade>
- Mantıksal VE işleminin doğruluk tablosu

1.Operand	**2.Operand**	**Sonuç**
True	True	True
True	False	False
False	True	False
False	False	False

- bool b1 = 25 < 15 && 5 == 50 **;**
- **false**
- bool b2 = 25 > 15 && 5 != 50 ;
- **true**

Mantıksal Operatörler

&:	**AND**	**&&:**	**Kısa devre AND**
|:	**OR**	**||:**	**Kısa devre OR**
!:	**NOT**		

- **|| (VEYA) Operatörü:**
- “true” ya da “false” değerindeki operandl arı mantıksal VEYA ile işler. Operandlardan biri “true” ise “true” değeri üretir.
  Kullanımı: <ifade> || <ifade>
- Mantıksal VEYA işleminin doğruluk tablosu:

1.Operand	**2.Operand**	**Sonuç**
True	True	True
True	False	True
False	True	True
False	False	False

- bool b1 = 25 < 15 || 5 == 50 ;
- **false**
- bool b2 = 25 > 15 || 5 != 50;
- **true**

Mantıksal Operatörler

&:	**AND**	**&&:**	**Kısa devre AND**
|:	**OR**	**||:**	**Kısa devre OR**
!:	**NOT**		

- **! (Değil) Operatörü:**
- Tek operand alır ve mantıksal Değil (NOT) işlemini uygular. Kullanımı: **! <ifade>**
- Mantıksal DEĞİL işleminin doğruluk tablosu
- **Operand** **Sonuç**
- True False
- False True
  - bool b1 = ! (25 < 10) ;
  - **True**
  - bool b2 = ! (10 != 20) ;
  - **false**

Mantıksal Operatörler

```
using System;
class mantıksal_operatörler
{
    public static void Main()
    {
        bool a = true;
        bool b = false;
        if (a ^ b) Console.WriteLine("a^b doğrudur");
        if (a && b) Console.WriteLine("a ve b doğrudur");
        if (a || b) Console.WriteLine("a veya b doğrudur");
    }
}
```

Bit Tabanlı (Bitwise) Operatörler

&:	Bit tabanlı AND	>> :	Sağa kaydırma (Right Shift)
|:	Bit tabanlı OR	<< :	Sola kaydırma (Left Shift)
^:	Bit tabanlı XOR	~ :	Birin tümleyeni (Ones Complement)

p	q	p & q	p | q	p ^ q	~p
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

- Bitsel operatörler ise sayıları ikilik olarak ele alır ve her bitlerinde işlem gerçekleştirirler. Bitsel operatörler tamsayılarda kullanılır, gerçek sayılarda kullanılamaz. Bitsel operatörlerde **bool** türünden ifadeler kullanılırsa mantıksal işlemler gerçekleştirir.
- Bu operatörler bitlerin durumlarını öğrenmek veya belirli bitlerin değerlerini değiştirmek için kullanılabilir.

Bit Tabanlı Operatörler

- **~ (Bitsel DEĞİL) Operatörü:**
- Tek operand alan **~** operatörü bir değer içindeki bitlerin teker teker tersini alır.
- **0000 1111** sayısının bitsel değili **1111 0000** olur.

```
using System;
class Operatorler
{
    static void Main()
    {
        byte b1 = 254;
        byte b2 = (byte)~b1;
        Console.WriteLine(b2);
    }
}
```

Bit Tabanlı Operatörler

- **& (Bitsel VE) Operatörü:**

- İki operandın bitlerini karşılıklı olarak VE işlemine tabi tutar.
- byte a = 153; **153 :1001 1001**
- byte b = 104; **104 :0110 1000**
- byte x = (byte)(a & b) ----------
- ->x=8 **VE 0000 1000**
-

Bit Tabanlı Operatörler

- **| (Bitsel VEYA) Operatörü:**

- İki operandın bitlerini karşılıklı olarak VEYA işlemine tabi tutar.
- byte a = 153; **153 :1001 1001**
- byte b = 104; **104 :0110 1000**
- byte x = (byte)(a | b) ----------
- ->x=249 **VE YA 1111 1001**
-

Bit Tabanlı Operatörler

- **^ (Bitsel Özel VEYA) Operatörü:**

- İki operandın bitlerini karşılıklı olarak Özel VEYA (XOR) işlemine tabi tutar.
- byte a = 153; **153 :1001 1001**
- byte b = 104; **104 :0110 1000**
- byte x = (byte)(a ^ b) ----------
- ->x=241 **XOR 1111 0001**
-

Bit Tabanlı Operatörler

- **^ (Bitsel Özel VEYA) Operatörü:**

```
using System;
class Operatorler
{
    static void Main()
    {
        int x = 195;
        int y = 26;
        x = x ^ y;
        y = y ^ x;
        x = x ^ y;
        Console.WriteLine("x={0}\ny={1}", x, y);
    }
}
```

x=26
y=195

Bit Tabanlı Operatörler

- **<< (Bitsel Sola Kaydırma) Operatörü:**
- Verilen bir tamsayının bitlerinin istenilen sayıda sola doğru ötelenmesini sağlar.
- İki operandı bulunur. İşlem yapılacak değer ve kaydırılacak bit sayısı. İşlem yapılırken en düşük değerlikli bit sıfır ile tamamlanırken en yüksek değerlikli bitten taşanlar atılır. Kullanımı:
- ifade << öteleme sayısı
- byte b = 0xFF // 1111 1111 (255)
- byte c = (byte)(b<<4) // 1111 0000

Bit Tabanlı Operatörler

- **>> (Bitsel Sağa Kaydırma) Operatörü:**
- Verilen bir tamsayının bitlerinin istenilen sayıda sağa doğru ötelenmesini sağlar. İki operandı bulunur. İşlem yapılacak değer ve kaydırılacak bit sayısı. İşlem yapılırken en yüksek değerlikli bit sıfır ile tamamlanırken en düşük değerlikli bitten taşanlar atılır. Kullanımı:
- ifade >> öteleme sayısı
- byte b = 0xFF // 1111 1111
- byte c = (byte)(b>>4) // 0000 1111

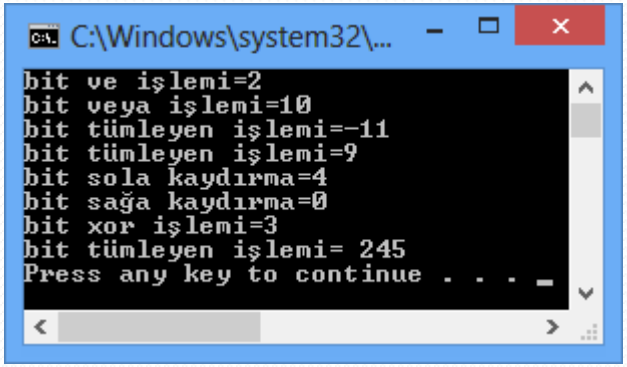
Bit Tabanlı Operatörler

- Örnek

```
using System;
class Operatorler
{
    static void Main()
    {
        byte b = 0xFF; //255
        b = (byte) (b << 4);
        Console.WriteLine(b); //240
        b = (byte)(b >> 3);
        Console.WriteLine(b); //30
    }
}
```

Bit Tabanlı Operatörler

```
using System;
class mantıksal_operatörler
{
    public static void Main()
    {
        int a = 10; int b = 2; int c=-10;
        Console.WriteLine("bit ve işlemi=" + (int)(a & b));
        Console.WriteLine("bit veya işlemi=" + (int)(a | b));
        Console.WriteLine("bit tümleyen işlemi=" + (int)(~a));
        Console.WriteLine("bit tümleyen işlemi=" + ~c);//(int) cast yapılmasa da olur.
        Console.WriteLine("bit sola kaydırma=" + (int)(1 << b));
        Console.WriteLine("bit sağa kaydırma=" + (1 >> b));
        Console.WriteLine("bit xor işlemi=" + (int)(1 ^ b));
        Console.WriteLine("bit tümleyen işlemi= "+(byte)(~a)); // ikiliğe dönüştürdü
    }
}
```



Atama ve İşlemli Atama Opreratörleri

- **= (Atama) Operatörü:**
- Bir değişkene herhangi bir değer atamak için kullanılır. Çok bilinen ve kullanılan bir operatördür.

- İşlemli atama operatörleri, işlem sonucunun işlenen operanda atandığı durumlarda kullanılır

- x = x + y; ifadesi yerine x+=y;
- a = a * b; ifadesi yerine a *= b;
- b = b << n; ifadesi yerine b<<=n;

Özel Amaçlı Operatörler

- **? : (Ternary) Operatörü:**
- Bir koşulu karşılaştırıp doğru ya da yanlış olduğu durumlar için farklı değerler üretmeyi sağlayan operatördür. Üç tane operanda sahiptir. Kullanımı:

- **koşul ? doğru değeri : yanlış değeri**

- sayi < 10 ? “10dan küçük” : “10 a eşit ya da büyük”

Özel Amaçlı Operatörler

- **() Operatörü (Tür Dönüştürme):**
  - İfade önüne yazıldığında tür dönüştürme operatörü olarak çalışır
- **[] Operatörü (İndeks):**
  - Dizilerde elemanın indeksini belirtmek için kullanılır
- **+, - (İşaret Operatörleri):**
  - Bir değişkenin içeriğindeki değerin negatif ya da pozitif olarak işlem görmesini sağlar. + aynı zamanda stringler de ekleme için de kullanılan bir operatördür.
- **&, *, -> ve sizeof Operatörleri:**
  - C/C++ dilindeki işaretçileri (pointer) C# dilinde de kullanmak mümkündür. Güvensiz (unsafe) kod yazarken bu operatörler de kullanılabilir.

Özel Amaçlı Operatörler

- **. Operatörü:**
  - Bir sınıfın ya da yapının elemanlarına ulaşmak için kullanılır
  - Console.WriteLine(); gibi…

- **new Operatörü:**
  - Yeni bir nesne oluşturmak için kullanılan operatördür.
  - Değer ve referans tiplerden yeni bir nesne oluşturmak için kullanılabilir
  - Örnek : Sinif s = new Sinif();

Özel Amaçlı Operatörler

- **checked ve unchecked Operatörleri**
  - Büyük türün küçük türe dönüşümde taşma meydana geldiğinde uyarı veren (checked) veya taşmayı göz ardı eden (unchecked) oparatörlerdir.

- **typeof Operatörü:**
  - Çalışma zamanında bir tür için System.Type sınıfı türünden bir nesne üretilmesini sağlar. Nesneler için kullanılan GetType() ifadesinin benzer işlemini tür ve ya sınıf tanımlamaları için gerçekleştirir

Özel Amaçlı Operatörler

- **Örnek**

```
using System;

class Operatorler13
{
    static void Main()
    {
        Type t;
        int i = 123;
        float f = 23.45F;

        t = typeof(int);
        Console.Write(t.ToString() + "  ");
        Console.WriteLine(i.GetType());

        t = typeof(float);
        Console.Write(t.ToString() + "  ");
        Console.WriteLine(f.GetType());
    }
}
```

Çıktı:
System.Int32
System.Int32
System.Single
System.Single

Akış Kontrol Mekanizmaları

- ✓ **Koşul ifadeleri**
  - if deyimi, switch deyimi
- ✓ **Döngü yapıları (loops)**
  - for döngüsü, while döngüsü, foreach döngüsü
- ✓ **Atlama(jump)**
  - break, continue, goto, return

Koşul İfadeleri

- Programlar içersinde bazı durumlarda programın akışının değişmesi ya da farklı işlemlerin yapılması gerekebilir. Bazen de seçimlerin yapılması ya da belirli işlemlerin belli sayıda tekrar edilmesi gerekebilir.

- Bu gibi işlemleri gerçekleştirmek için koşul ifadeleri ve döngü yapıları kullanılır.

- C# dilinde koşula bağlı olarak işlemler gerçekleştirmek için iki farklı deyim kullanılabilir:
  - **if deyimi**
  - **switch deyimi**

if Deyimi

- **Program akış kontrol deyimlerinin başında gelir. Koşula göre değişik işlemlerin yapılmasını sağlar. Basit kullanımları:**

```
if (koşul)
   deyim1;
else
  deyim2;
```

```
if (koşul)
  deyim1;
else if
  deyim2;
else
 deyim3;
```

- ***Eğer deyim kısmında 1 den fazla satır varsa { } program bloğu kullanmak zorunludur.***

if Deyimi

```
using System;
class if_yapısı
{
    public static void Main()
    {
        string s;
        int a, b, c;
        Console.Write("1.sayıyı giriniz : ");
        s = Console.ReadLine();
        a = Int32.Parse(s);
        Console.Write("2.sayıyı giriniz : ");
        s = Console.ReadLine();
        b = Int32.Parse(s);
        Console.Write("3.sayıyı giriniz : ");
        s = Console.ReadLine();
        c = Int32.Parse(s);
        if (a >= b && a >= c) Console.WriteLine("Enbüyük :" + a);
        else if (b >= c) Console.WriteLine("Enbüyük :" + b);
        else Console.WriteLine("Enbüyük :" + c);
    }
}
```

switch Deyimi

- Bir ifadenin alabileceği değişik değerlere göre işlemler gerçekleştirmek için kullanılan deyimdir.

- Karmaşık if ifadelerini daha sade bir şekilde ifade etmeyi de sağlayabilir.

Kullanımı:

```
switch(ifade)
{
case sabit1:
        deyim1;
        break;
case sabit2:
        deyim2;
        break;
default:
        deyim3;
        break;
}
```

switch Deyimi

- Her case ifadesinin bir atlama **(break, goto, return, continue)** ifadesi ile sonlandırmalısınız.

- **continue** deyimi switch ifadesi bir döngünün gövdesinde ise kullanılabilir.

- Bir case içinde hiçbir ifade verilmezse atlama deyimi kullanılması gerekmez.

  - case 2: // ifade yok
  - case 3:
  - case 4: ifade….;break;
  - case 5: ifade…; goto case 4;

- Son ifade (case veya default) boş olamaz. Son ifadeden sonra başka işlevi olan satır yok ise hata verir.

Switch Deyimi

- **switch ifadesi kullanırken dikkat edilecek bazı kurallar vardır:**
  - case sözcüğünden sonra gelen ifadeler sabit olmak zorundadır.
  - case ifadeleri tamsayı, karakter ya da string sabitler olabilir.
  - default ve case ifadeleri istenilen sırada yazılabilir.
  - Aynı switch bloğu içerisinde birden fazla aynı case ifadesi bulunamaz.
  - default kullanmak zorunlu değildir.
  - Akışı bir case ifadesinden bir başka case ifadesine yönlendirmek için mutlaka **goto** anahtar sözcüğü kullanılır.

Switch Deyimi

```
using System;
class switch_case_yapısı
{    public static void Main()
    {   string s;        int a;
        Console.Write("Kaçıncı Sınıftasınız : ");
        s = Console.ReadLine();
        a = Int32.Parse(s);
        switch (a)
        {   case (0):
            case (1): Console.WriteLine("Daha Yenisiniz");  break;
            case (2): Console.WriteLine("Mirasçısınız");    break;
            case (3): Console.WriteLine("Ev Sahibisiniz");  break;
            case (4): Console.WriteLine("Misafirsiniz");    break;
            default: Console.WriteLine("Siz okulu uzatmışsınız"); goto case 1;
        }
    }
}
```

Switch Deyimi

```
using System;
class switch_case_yapısı
{    public static void Main()
    { int a = 1;
        switch (a)
        {   case (0):
            case (1): Console.WriteLine("Daha Yenisiniz");  break;
            case (4): break;
            default:
         }
    }
}
```

Derleme zamanı hatası verir. (son ifade boş olamaz.)

Döngüler(Loops)

- Program içerisinde belirli işleri tekrar tekrar yapılmasını sağlayan kod bloklarıdır. Döngüler sonsuz olabileceği gibi belli bir koşul ile kontrol etmek de mümkündür.
- C# dilinde dört tip döngü yapısı bulunur:
  - for
  - while
  - do while
  - foreach

For Döngüsü

- for döngü tanımlamasında iki “;” ile ayrılmış üç ifade bulunur. Bunlardan bazıları boş olabilir fakat mutlaka “;” işaretleri kullanılmalıdır.

- İlk ifade bir defaya mahsus olmak üzere çalıştırılır. Genelde döngü değişkeninin tanımlanması ya da ilk değer ataması için kullanılır.

- İkinci ifade ise döngünün kontrol edildiği kısımdır Buradaki ifade “true” değer ürettiği sürece döngü devam eder.

- Son ifade ise genelde döngü değişkeninin değerinin değiştirildiği kısımdır.

For Döngüsü

- **for(ifade1; ifade2; ifade3)**
- **{**
- **deyimler;**
- **}**
- Kullanımı:
  - for(; ;) Sonsuz döngüdür.
  - for(int i=1; i<=100; i++);
  - for(int i=1, j=10; (i<=10) & (j>=1); i++, j--);

For Döngüsü

```
using System;

class for_1
{
    static void Main()
    {
        Console.Write("Bir sayı girin:");
        int n = Convert.ToInt32(Console.ReadLine());
        int toplam = 0;

        for (int i = 1; i <= n; i++)
            toplam = toplam + i;

        Console.WriteLine("1\'den {0}\'e kadar olan
            sayıların toplamı {1}\'dir.", n, toplam);
    }
}
```

While ve do-while Döngüsü

- while (koşul)
- {
- deyimler
- }
- Belirtilen koşul doğru olduğu sürece çalışmasına devam eder.

- do
- {
- deyimler
- } while(koşul);

- for ve while döngülerinde koşul sağlanmadığı durumlarda döngü bloğu hiç çalıştırılmaz. Bazı durumlarda ise döngünün en az bir kez çalıştırılması gerekebilir. Bu durumda do while kullanılabilir.

While ve do-while Döngüsü

```
using System;

class while_1
{
    static void Main()
    {
        int i=0;

        while (i < 1000)
        {
            i += 5;
            Console.Write("{0,5}",i);
            if (i % 50 == 0) Console.WriteLine();
        }
    }
}
```

Foreach Döngüsü

- Farklı bir döngü yapısıdır. Koleksiyon tabanlı nesneler içerisinde adım adım dolaşılmasını sağlar.

- Bu döngü elemanlara tek tek ulaşım sağlar fakat ulaşılan elemanlar sadece okunabilir özelliğe sahiptir.

Foreach Döngüsü

- string[ ] isimler ={ "Ozlem","Nesrin", "Ozge", "Fulya" };

//string dizi tanımı

```
foreach (string ss in isimler)  // elemanlar yazdırılıyor
 {
  Console.WriteLine(ss);
  }
```

- foreach (int i in dizi) // i sadece okunur(readonly)

```
 {  Console.WriteLine(i);
    /* i++;*/ //yasak kullanılamaz
  }
```

Jump (atlama) Deyimleri)

- **break*:*** Çalışan bir döngüden “break” sözcüğü kullanılarak çıkılabilir. Program akışı döngüden sonraki satırlardan devam eder. break sözcüğü sadece döngü ve switch ifadelerinde kullanılabilir.

- **continue*:*** Döngünün bir sonraki tekrarına geçilmesini sağlar.

- **goto*:*** Programın etiket ile belirlenmiş herhangi bir kısmına atlamak için kullanılır.
  - Nesneye yönelik programlamaya uygun bir yapı değildir. Switch ifadesindeki kullanımı dışında mümkün olduğunca kullanılmaktan kaçınılmalıdır.
- **return *:*** Metotların herhangi bir anda sonlandırılması için kullanılır. Metot sonlandırıldıktan sonra programın akışı metodu çağıran fonksiyondan devam eder.

Jump (atlama) Deyimleri)

```
using System;

class break_continue_1
{
    static void Main()
    {
        int sayi, t=0;

        while (true)
        {
            Console.Write("Negatif bir sayı girin:");
            sayi = Convert.ToInt32(Console.ReadLine());
            if (sayi == 0) break;
            if (sayi > 0)
            {
                Console.WriteLine("Pozitif sayı girdiniz,
   tekrar deneyin...");
                continue;
            }
            t = t + sayi;
        }
        Console.WriteLine("Toplam :{0}", t);
    }
}
```

Jump (atlama) Deyimleri)

```
public class Program_Goto
{
  public static void Main()
  {
   int m = 0;
   int n=10;
   bas:
   m++;
   if (m <= n) goto bas;
   else goto son;
   son: Console.WriteLine("m="+m);
   // Etiket sonda ise mutlaka kendinden sonra bir ifade gelmeli. Console ifadesi olmasaydı derleme hatası verirdi.
```

- Çıktı m=11;

System.Random Sınıfı

- Rastgele sayı üretmek için .NET Framework sınıf kütüphanesindeki **System.Random** sınıfı kullanılır. Rastgele sayı üretmek için öncelikle **Random** sınıfı türünden bir nesne oluşturmak gerekir. Kullanımı;

- `Random rnd=new Random();`
- **rnd** nesnesi oluşturulduktan sonra artık bu nesnenin metotlarına erişebiliriz.

System.Random Sınıfı

- `Random rnd=new Random();`

- **int rastgelesayı =rnd.Next(10,20);**
  - 10 ile 20 arasında bir sayı üretir. (10 dahil 20 hariç)

- **int rastgelesayı =rnd.Next(50);**
  - 0 ile 50 arasında bir sayı üretir.

- **int rastgelesayı =rnd.Next();**
  - Herhangi bir rasgele sayı üretir.

- **double rastgelesayı =rnd.NextDouble();**
  - 0.0 ile 1 arasında bir sayı üretir.

Visual Studio.Net -C#

4. HAFTA

Diziler

Metotlar

ÖRNEK UYGULAMALAR

Örnek 1: Operatör

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        int a = 2; int b = 3; int c = 6; int d = 1;
        /*  (a < b) = bu ifadenin doğru (true) olduğunu biliyoruz
            (c < d) = bu ifadenin yanlış (false) olduğunu biliyoruz   */
        Console.WriteLine(" (a<b)&&(c<d) --> " + ((a < b) && (c < d)));
        Console.WriteLine(" (a<b)||(c<d) --> " + ((a < b) || (c < d)));
        Console.WriteLine(" ! (a<b) -->" + (!(a < b)));
        Console.WriteLine(" (a<b)&(c<d) --> " + ((a < b) & (c < d)));
        Console.WriteLine(" (a<b)|(c<d) --> " + ((a < b) | (c < d)));
        Console.WriteLine(" (a<b)^(c<d) --> " + ((a < b) ^ (c < d)));
    }
}
```

Sonuç:

(a<b) && (c<d) --> False

(a<b) || (c<d) --> True

! (a<b) -->False

(a<b) & (c<d) --> False

(a<b) | (c<d) --> True

(a<b) ^ (c<d) --> True

Örnek 2: Operatör

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        int a = 2; int b = 3; int c = 6;
        Console.WriteLine(" (a & b) --> " + (a & b));
        Console.WriteLine(" (a | b) --> " + (a | b));
        Console.WriteLine(" (a ^ b) --> " + (a ^ b));
        Console.WriteLine(" ( ~a ) --> " + (~a));
        Console.WriteLine(" ( ~b ) --> " + (~b));
        Console.WriteLine(" ( ~c ) --> " + (~c));
    }
}
```

Sonuç:

```
(a & b) --> 2
(a | b) --> 3
(a ^ b) --> 1
( ~a ) --> -3
( ~b ) --> -4
( ~c ) --> -7
```

Örnek 3: while döngü

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        int i = 0; //döngü kontrol değişkeni
        while (i < 5)
        {
            Console.WriteLine("i = " + i);
            i++;
        }
    }
}
```

i = 0
i = 1
i = 2
i = 3
i = 4

Örnek 4: do-while döngü

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        int i = 0; //döngü kontrol değişkeni
        do
        {
            Console.WriteLine("i = " + i);
            i++;
        } while (i < 0);
    }
}
```

i = 0

Örnek 5 : do-while ve switch

```
using System;

class do_while_1
{
    static void Main()
    {
        int secim;

        do
        {
            Console.WriteLine("İşlemler\n========");
            Console.WriteLine("1 - Toplama");
            Console.WriteLine("2 - Çıkarma");
            Console.WriteLine("3 - Çarpma");
            Console.WriteLine("4 - Bölme");
            Console.WriteLine("0 - Çıkış\n");
            Console.Write("İşleminizi Seçin:");
            secim = Convert.ToInt32(Console.ReadLine());
            switch (secim)
            {
                case 1: Console.WriteLine("Toplama işlemi seçildi."); break;
                case 2: Console.WriteLine("Çıkarma işlemi seçildi."); break;
                case 3: Console.WriteLine("Çarpma işlemi seçildi."); break;
                case 4: Console.WriteLine("Bölme işlemi seçildi."); break;
                case 0: Console.WriteLine("Çıkış seçildi."); break;
                default: Console.WriteLine("Yanlış seçim."); break;
            }
        }
        while(secim != 0);
    }
}
```

Örnek 6: for döngü

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        for (int i = 0, j = 0; i < 20; i++, j++)
        {
            i *= j;
            Console.WriteLine("i = " + i + "  j = " + j);
        }
    }
}
```

Örnek 7: for döngü

```
using System;

class for_2
{
    static void Main()
    {
        string s;

        for (s = Console.ReadLine(); s != "Çıkış";
             s = Console.ReadLine())

        Console.WriteLine(s);
    }
}
```

Örnek 8: for döngü

```
using System;

class for_3
{
    static void Main()
    {
        int i=0, a, n;

        Console.Write("Bir Sayı Girin :");
        n = Convert.ToInt32(Console.ReadLine());

        Console.Write("Artım Miktarı  :");
        a = Convert.ToInt32(Console.ReadLine());

        for (; i < n; )
        {
            Console.Write("{0} ", i);
            i += a;
        }
    }
}
```

Örnek 9: for döngü

```
using System;

class for_4
{
    static void Main()
    {
        int k, t, toplam, n1, n2;

        Console.Write("Aralık başlangıcı :");
        n1 = Convert.ToInt32(Console.ReadLine());

        Console.Write("Aralık Sonu       :");
        n2 = Convert.ToInt32(Console.ReadLine());

        if (n1 <= n2)
        {
            for (k = n1; k <= n2; k++)
            {
                toplam = 0;
                for (t = 1; t <= k; t++) { if (k % t == 0) toplam = toplam + t; }
                if (toplam == k + 1) Console.WriteLine(k);
            }
        }
        else
        {
            Console.WriteLine("Geçerli bir aralık giriniz...");
        }
    }
}
```

Örnek 10: for döngü

```
using System;

class for_5
{
    static void Main()
    {
        int sayi;

        Console.Write("Bir tamsayı giriniz :");
        sayi = Convert.ToInt32(Console.ReadLine());

        for (int bit = 32; bit >= 1; bit--)
        {
            Console.Write("{0}", (sayi >> bit - 1) & 1);
        }
        Console.WriteLine();
    }
}
```

Visual Studio.Net -C#

4. HAFTA

Diziler

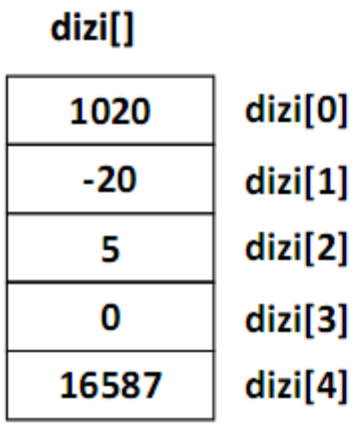
Metotlar

get-set yöntemi

Diziler (Arrays)

- Bellekte ard arda yer alan aynı türden nesneler kümesine dizi (array) denilir. Bir dizi içerisindeki bütün elemanlara aynı isimle ulaşılır. Yani dizideki bütün elemanların isimleri ortaktır. Elemanlar arasındaki ayırt edici özellik, bellekteki yeridir.

- 5 elemanlı bir tamsayı dizisi bellekte aşağıdaki gibi yerleştirilir.



Diziler (Arrays)

- C#'da diziler ayrı bir tür olarak tasarlanmıştır. Bütün diziler System.Array sınıfından türetilmiştir. Bir dizi tanımlandığında Array sınıfı türünden bir nesne tanımlanmış olur.

- Dizinin sınırlarını aşan bir indeks kullandığımızda derleme hatası oluşmaz, ancak çalışma zamanında hata oluşur.
- Dizi tanımlama iki türlü yapılabilir :

- `int[] dizi = new int[19];`
- veya
- `int[] dizi;`
- `dizi = new int[19];`

Diziler (Arrays)

- **new** anahtar sözcüğü ile, dizinin her elemanına temel veri türleri için varsayılan değer, ilk değer olarak verilmektedir.
- Bu ilk değer; referans türleri için null, nümerik türler için 0, bool türü için ise false'tır.

- Dizileri aşağıdaki gibi de tanımlamak mümkündür:

- `string[] dizi1={"Bilgisayar","Mühendisliği","Bölümü"};`
- `int [] dizi2= {5,9,12,56,23};`
- `float [] dizi3={8f,39f,324f,23f,2f};`

Diziler (Arrays)

- Dizilerin boyutu C ve C++ dillerinde derleme sırasında bilinmek zorundadır. Böylece derleyici, dizi için bellekte dizinin boyutuna göre yer ayırmaktadır. Ancak C# ‘ta ise diziler referans tipi olduğu için dizi boyutları çalışma zamanında belirlenebilir.

```
class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Dizinin boyutunu giriniz=");
            int boyut = Convert.ToInt32(Console.ReadLine());
            bool [] dizi = new bool [boyut];
            for (int i = 0; i < boyut; i++)
                Console.WriteLine(dizi[i]);
            Console.ReadLine();
        }
    }
```

- Bir dizinin boyutu bir kez belirlendikten sonra artık değiştirilemez. Yani dizinin boyutunu dinamik olarak değiştirmemiz mümkün değildir.

Diziler (Arrays)

```
using System;
class Length
{
    static void Main()
    {
        int k = 0;
        int[] dizi = new int[10];
        //diziye ait nesne oluşturuluyor
        k = dizi.Length;
        Console.WriteLine("Dizinin Uzunluğu : " + k);
    }
}
```

Diziler (Arrays)

- Diziye rastgele değer ataması; Random sınıfı, System içinde tanımlı.

```
Random rnd = new Random();
int[] dizi =new int[3];
dizi[0] = rnd.Next(2,10);
dizi[1] = rnd.Next(50);
dizi[2] = rnd.Next();
```

```
foreach(int i in dizi)
{
    Console.WriteLine(i);
}
```

```
for (k=0;k<3;k++)
{
Console.WriteLine(dizi[k]);
}
```

Çok Boyutlu Diziler (Multidimensional *Arrays)*

- İki türlü çok boyutlu dizi bulunur. Her boyutta eşit miktarda elemanın olduğu düzenli diziler (matrisler) ve her boyutta farklı sayıda eleman barındıran düzensiz diziler(jagged arrays). Aslında düzensiz diziler elemanları da dizi olan dizilerdir.

- Çok boyutlu dizi tanımı için tanımlama esnasında [ ] içine verilecek boyut kadar ‘,’ değeri eklenir. “[, ,]” şeklinde yazılır.

Matris Dizileri

- **int [,] dizi = {{1,2},{3,4},{5,6}};**
- Şeklinde tanımlanan bir 3x2'lik dizinin elemanları:

dizi[0,0]=1
dizi[0,1]=2
dizi[1,0]=3
dizi[1,1]=4
dizi[2,0]=5
dizi[2,1]=6

dizi

[0,0]	[0,1]
[1,0]	[1,1]
[2,0]	[2,1]

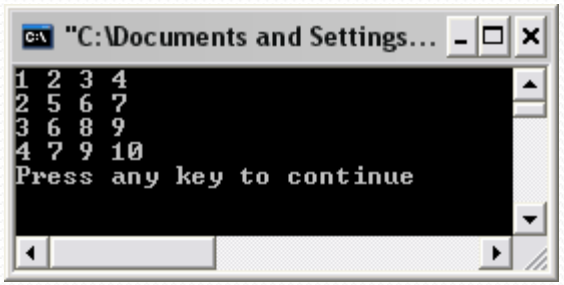
- int[,] dizi1=new int[2,3];

  dizi1[0,0]=1;

  dizi1[2,1]=7;

Çok Boyutlu Diziler

```
using System;
class diziler
{
  public static void Main()
  {    int[,] dizi = new int[4, 4]; int k=1;
       for (int i=0;i<4;i++)
          for(int j=i;j<4;j++) { dizi[i,j]=k;  dizi[j,i]=k;  k++;  }
       k=1;
       foreach(int x in dizi)
       {
          if (k%4==0)
          {  Console.Write(x+" ");  Console.WriteLine();    }
          else Console.Write(x+" "); k++;
       }
   }
}
```



Düzensiz Diziler(Jagged Arrays)

- Dizilerin elemanları da farklı diziler olabilir. Bu durumda tanımlanan dizinin birbirinden farklı boyutlarda alt elemanları olabilir. Bu gibi yapılara düzensiz diziler ismi verilir.

dizi[0][0]	dizi[0][1]	dizi[0][2]	dizi[0][3]	dizi[0][4]
dizi[1][0]	dizi[1][1]	dizi[1][2]		
dizi[2][0]	dizi[2][1]	dizi[2][2]	dizi[2][3]	

- Düzensiz dizilerde boyut çift [][] yapısıyla veriliyor. İlk [] satır, ikinci [] ifade düzensiz yapan sütuna aittir.

```
int [][] dizi= new int[3][];
dizi[0]=new int[5];
dizi[1]=new int[3];
dizi[2]=new int[4];
```

Düzensiz Diziler(Jagged Arrays)

- Düzensiz dizilerde eleman sayısı sabit olmadığı için dizi elemanlarını taramak için Array sınıfının metotları ve ya özellikleri kullanılır. **Length** özelliği bir dizinin eleman sayısını verir.

```
int [][] dizi= new int[3][];
dizi[0]=new int[5];
dizi[1]=new int[3];
dizi[2]=new int[4];
Console.WriteLine(dizi.GetLength(0)); // dizinin satır sayısı
Console.WriteLine(dizi[2].GetLength(0));//2.satırdaki sütun
                                                sayısı
```

Düzensiz Diziler(Jagged Arrays)

```
using System;

class dizi_5
{
    static void Main()
    {
        int[][] dizi = new int[3][];

        dizi[0] = new int[] { 1, 2 };
        dizi[1] = new int[] { 3, 4, 5, 6, 7 };
        dizi[2] = new int[] { 8, 9, 0 };

        for (int i = 0; i < dizi.GetLength(0); i++)
         for (int j = 0; j < dizi[i].GetLength(0); j++)
         Console.WriteLine("dizi[{0}][{1}]={2}",i,j,dizi[i][j]);
    }
}
```

System.Array Sınıfı

- Dizi nesneleri **System.Array** sınıfından türemişlerdir. Dolayısıyla bu sınıfın bazı özellik ve metotlarını bünyelerinde barındırırlar.
- **Özellikler:**

IsFixedSize	**Dizinin eleman sayısının sabit olup olmadığını verir. (Boolean)**
IsReadOnly	**Dizideki elemanların sadece okunabilir olup olmadığını verir. (Boolean)**
Length	**Dizideki eleman sayısını verir.**
Rank	**Dizinin boyutunu verir. (1 boyutlu, 2 boyutlu)**

System.Array Sınıfı

- **Metotlar:**

BinarySearch	Tek boyutlu dizide binary search algoritmasına göre arama yapar.
Clear	Dizinin elemanlarını varsayılan değere çeker.
Clone	Dizinin bit bit kopyasını çıkarır.
Copy	Dizinin bir bölümünü başka bir diziye kopyalar.
CopyTo	Bir dizinin belirlenen bir kısmını başka bir diziye kopyalar.
GetValue	Dizideki ilgili eleman değerini verir.
IndexOf	Dizi içindeki bir değerin ilk görüldüğü indeksi verir.
Reverse	Diziyi tersine çevirir.
SetValue	Bir dizinin bir elemanına değer atar.
Sort	Bir boyutlu dizilerde sıralama yapar.
CreateInstance	Yeni bir dizi nesnesi oluşturur.
GetLength	Dizinin i. boyutundaki değerinin verir.
GetUpperBound	Dizinin i. boyuttaki son indis değerini verir.
Resize	Mevcut dizinin elaman sayısını değiştirme.

System.Array Sınıfı

- **GetLength:** Dizinin i. boyuttaki değerini verir.
  - int[, ,] dizi = new int[5,4,3];
  - int x = dizi.GetLength(0); → x=5
  - int y = dizi.GetLength(1); → y=4
  - int z = dizi.GetLength(2); → z=3
- **GetUpperBound:** Dizinin i. boyuttaki son indis değerini verir.
  - int[, ,] dizi = new int[5,4,3];
  - int X = dizi. GetUpperBound(0); → X=4
  - int Y = dizi. GetUpperBound(1); → Y=3
  - int Z = dizi. GetUpperBound(2); → Z=2
- **Rank:** Dizi boyut sayısını verir.
  - int[, ,] dizi = new int[5,4,3];
  - dizi.Rank→ 3

System.Array Sınıfı

- **GetValue**: Metodu belirtilen indekslerdeki elemanın değerini verir.
- int[ ] dizi = { 1, 22, 34 };

```
Console.WriteLine(dizi.GetValue(2)); // 34 değerini verir
Console.WriteLine(dizi[2]); // 34 değerini verir
```

- **SetValue**: Metodu ilgili dizinin belirtilen indekslerindeki elemanın değerini değiştirmek için kullanılır.

- dizi.SetValue(20,0);//(değer,1.index)
- dizi.SetValue(20,0,0);//(değer,1.index,2.index)

CreateInstance( )

- Çalışma zamanında dizinin tipinin belirlenmesi için kullanılır. CreateInstance metodu ile aşağıdaki gibi dizi nesnesi oluşturulabilir:
  - Array dizi=Array.CreateInstance(typeof(int),5);
  - 5 elemanlı int türünden bir dizi tanımlanır.
- İlk parametre her zaman Type türünden olmalıdır. Yukarıdaki gibi bir kullanımda, typeof() operatörü yardımıyla int türünün type sınıfındaki karşılığı elde edilmiş olmaktadır. Örn:
  - CreateInstance(Type,int,int,int);
  - 3 boyutlu bir dizi oluşturulmakta.
  - CreateInstance(Type,int[]);
    Çok boyutlu diziler oluşturmak için kullanılır. Çok boyutlu dizilerin her birinin kaç elemanlı olduğunu belirlemek için int türden bir dizi metoda parametre olarak gönderilir.
  - Array dizi=Array.CreateInstance(typeof(int),3,5);
  - int tipinde 2 boyutlu bir dizi tanımlanır.

CreateInstance( )

```
using System;

class dizi_6
{
    static void Main()
    {
        Array dizi = Array.CreateInstance(typeof(int), 5, 4, 3);
        Random r = new Random();

        for (int i=0; i< dizi.GetLength(0); i++)
            for (int j=0; j< dizi.GetLength(1); j++)
                for (int k = 0; k < dizi.GetLength(2); k++)
                {
                    dizi.SetValue(r.Next(10, 100), i, j, k);
                    Console.WriteLine("dizi[{0},{1},{2}]={3,3}",
                                      i,j,k,dizi.GetValue(i,j,k));
                }
    }
}
```

Resize( )

- Dizi değişkenlerinin elaman sayısını daha önceki atanan değerler kaybolmadan değiştirmeye yarar.

- string iller=new string[2];
- iller[0]="Elazığ" ;
- İller[1]="Ankara";
- Array.**Resize**(**ref** iller, 3);
- iller[2]= "İstanbul";

CopyTo ( )

- CopyTo metodu ile bir dizinin tamamı, başka bir dizinin istenilen yerine kopyalanabilir. Örn:

- **int[ ] dizi1= {1,2,3,4,*5,6,7*};**
- **int dizi2=new int[10];**
- **dizi1. CopyTo(dizi2,3);**

- dizi1'in elemanlarını dizi2'nin 3. indisinden itibaren kopyalanır.

Copy ()

- Aynı işlem Copy() metodu kullanılarak da gerçekleştirilebilir.
- Copy (Array dizi1,Array dizi2,int uzunluk);

- Bu şekildeki bir kullanım, uzunluk kadar elemanı dizi1'den dizi2'ye kopyalar. Kopyalama işlemi 0. indekten başlar.

  - Copy (Array dizi1, int x, Array dizi2, int y, int uzunluk);
  - dizi1'in x. İndeksinden sonraki uzunluk kadar elemanı dizi2'nin y. İndeksinden sonrasına kopyalar.
  - *Array.Copy(dizi1,2,dizi2,3,1); // dizi1'in 2. indisinden itibaren, 1 elemanı, dizi2'nin 3. indis elemanından itibaren kopyalanır.*

Örnek

```
using System;

class dizi_7
{
    static void Main()
    {
        int[] dizi1 = { 1, 2, 3, 4, 5, 6, 7 };
        int[] dizi2 = new int[10];
        int[] dizi3 = new int[10];

        dizi1.CopyTo(dizi2, 2);
        foreach (int i in dizi2)
            Console.Write(i);

        Console.WriteLine();

        Array.Copy(dizi1, 2, dizi3, 5, 3);

        foreach (int i in dizi3)
            Console.Write(i);
    }
}
```

Array.Sort ( )

- Dizileri küçükten büyüğe sıralamaya yarar. Sort() metodunun birçok kullanım biçimi vardır. Temelde aşağıdaki şekilde kullanımı yaygındır. Örn:
- Array.Sort(Array);

  - Array Dizi=Array.CreateInstance(typeof(string),3);
  - Dizi.SetValue(“Ali”,0);
  - Dizi.SetValue(“Veli”,1);
  - Dizi.SetValue(“Sami”,2);
  - Array.Sort(Dizi);
  - foreach(string s in array) Console.WriteLine(s);

Array.Sort ( )

```
using System;

class dizi_8
{
    static void Main()
    {
        string[] dizi = { "Zeynep", "Fatma", "Ali", "Yılmaz",
   "Gökhan", "Osman", "Feride" };

        Console.WriteLine("Dizinin Elemanları\n==================");
        foreach (string s in dizi)
            Console.WriteLine(s);

        Array.Sort(dizi);
        Console.WriteLine();

        Console.WriteLine("Sıralanmış Dizi\n===============");
        foreach (string s in dizi)
            Console.WriteLine(s);

    }
}
```

Array.BinarySearch

- Sıralı dizilerde arama işlemi yapar. **BinarySearch** metodunun iki önemli kullanım biçimi vardır.

- **BinarySearch(Array dizi, object nesne);**
  - Bu yapı, dizi içerisinde nesneyi arar; eğer aranan nesne bulunursa bulunduğu index ile geri dönülür. Eğer bulunamazsa negatif bir sayıyla geri dönülür.

- **BinarySearch(Array dizi, int baslangic, int uzunluk, object nesne);**
  - Bu yapı ise, nesneyi başlangıç indeksinden itibaren uzunluk kadar eleman içerisinde arar. Eğer bulunamazsa negatif bir sayıyla geri dönülür.

Array.BinarySearch

```
using System;

class dizi_9
{
    static void Main()
    {
        string[] dizi = { "Zeynep", "Fatma", "Ali", "Yılmaz",
  "Gökhan", "Osman", "Feride" };

        Console.Write("Aranacak İsmi Giriniz :");
        string isim = Console.ReadLine();

        Array.Sort(dizi);
        int indeks = Array.BinarySearch(dizi, isim);

        if (indeks < 0)
            Console.WriteLine("Aranan isim dizide bulunamadı!");
        else
            Console.WriteLine("Aranan isim dizinin {0}. elemanında
  bulundu...",indeks);
    }
}
```

Diğer Metotlar

- **Clear()** metodu, belirtilen indis aralığındaki elemanları sıfırlar. Bir dizi elemanını , 0, Boşluk (Null) ya da false yapar.
  - Array.Clear(dizi,1,3);// 1. indisinden itibaren 3 elemanı sıfır yapar.
  - Array.Clear(dizi,0,2);// İlk indisten (0. indis) itibaren 2 elemanı sıfır yapar.
- **Reverse()** metodu, dizinin tümünü yada belirtilen indis aralığındaki elemanları ters çevirir.
  - Array.Reverse(dizi); veya
  - Array.Reverse (dizi,1,3);
    - Alttaki gibi bir kullanımda ise,1. elemandan itibaren ilk 3 elemanı ters çevirir.

Diğer Metotlar

- Diziler tıpkı değişkenler gibi birbirine atanabilirler. Örnek:
- **int**[] a={1,6,7,12};
- **int**[] b=a;
- Burada **b** dizisine **a** dizisi adres gösterme yoluyla aktarılmıştır, dolayısıyla da a dizisinin bir elemanında yapılan bir değişiklik **b** dizisini, **b** dizisinin bir elemanında yapılan bir değişiklik a dizisini etkileyecektir. Çünkü diziler birbirlerine adres gösterme yoluyla atanır. Ayrıca aşağıdaki gibi bir kullanım da doğrudur:
- **int**[] a={1,6,7,12};
- **int**[] b={1,5}; b=a;
- Burada **b** dizisi sahip olduğu elemanları kaybedip **a** dizisinin adresine yönlendi. Bu ve benzer şekilde diziler birbirlerine atanabilir. Yani aynı adrese yönlendirilebilirler.

var ve dynamic değişkenleri ile dizi tanımlamak.

- **var**, gelen verinin değerine göre türü belirlenir ve bellekte ona göre yer ayrılır. Aynı değişkene farklı türden değer atanamaz.
  - `var dizi_x = new[] { 2005, 2010, 2015, 2020 }; //Int32`
  - `var dizi_y = new var[3]; //Hatalı tanımlama Derleme zamanı hatası`
  - `var [] dizi_k = new int[5]; //Derleme zamanı hatası: [] olmayacak`
  - `var dizi_t = new int[5];`
  - `var dizi_z = new[,] { { "x","y" }, { "m","n" } }; //string`
- **dynamic**, derleme zamanında gerçekleştirilen tip kontrollerinin pas geçilerek, bu kontrollerin tamamen çalışma zamanında gerçekleştirilmesidir.
  - `dynamic [] d_x = new  dynamic[3];`
  - `d_x[0] = 1; //Int32`
  - `d_x[1] = "x"; //string`
  - `d_x[2] = true; //bool`

ArrayList Sınıfı ve Özellikleri

- **Arraylist**, dinamik olarak büyüyüp küçülen, farklı değişken türlerini ve nesneleri depolayabilen koleksiyon tabanlı bir sınıftır.
  - İlk başlangıçta eleman sayısı belirtilmediği zaman (ArrayList dizi = new ArrayList()) boyut 2 ile başlar elemanlar eklendikçe ikinin katları şeklinde kendini artırır (2,4,8,16,32… gibi). Başlangıçta boyut belirtilirse (ArrayList dizi = new ArrayList(3)); boyut aşıldığında 2'nin katları şeklinde artar.
- Arraylist, dizilerin eksiklerini gidermek ve kısıtlamalarını ortadan kaldırmak için oluşturulmuş bir sınıftır.
- **Dizi ve Arraylist Arasındaki Farklar;**
  - Diziler sabit uzunlukta tanımlanırlar, oluşturulduktan sonra büyüyüp, küçültülemezler. ArrayList'de ise böyle bir kısıtlama yoktur. ArrayList'e yeni elemanlar eklendikçe boyutu otomatik olarak arttırılır, elemanlar silindikçe ise boyutu azaltılır.
  - Diziler tanımlanırken tutacağı elemanların türünün belirtilmesi gerekir. Bu tür dışındaki elemanların dizi içerisine eklenmesi mümkün olmaz. ArrayList'te ise farklı türden değişkenler ve nesneler aynı koleksiyon içerisinde saklanabilir.

ArrayList Sınıfı ve Özellikleri

- **ArrayList Tanımlama, İçerisine Eleman Ekleme ve Silme**
- Öncelikle uygulamalarımızda ArrayList sınıfını kullanabilmek için **System.Collections** namespace' ini çalışmamıza eklememiz gerekir.

```
using System;
using System.Collections;
using System.Text;
namespace KoleksiyonUygulamalari
{ class Program
  { static void Main(string[] args)
    {   ArrayList liste = new ArrayList();
        liste.Add("Ahmet");          liste.Add(34);
        liste.Add('A');
        liste.Remove('A');
      }
  }
}
```

- **Add()** metoduyla farklı türlerden değişkenler listeye ekler. Add metodu içerisine parametre olarak object türünden oluşturulmuş herhangi bir nesne eklenebilir. Listeden eleman silmek içinse **Remove()** metodu kullanılır.

ArrayList Sınıfı ve Özellikleri

- **ArrayList İçerisindeki Verilere Erişmek**
- Yukarıdaki örnekte farklı veri tiplerini bir ArrayList içerisine eklenebilir. Pratikte böyle bir kullanım hem performans hem de algoritmik bakımdan bir çok problem yaratacağı için, genellikle benzer veri türlerinin ArrayList içerisinde saklanması daha uygundur. ArrayList içersindeki verilere, aynı dizilerdeki gibi indis numarasıyla erişim sağlanabilir. Yada tüm ArrayList bir for-each döngüsü yardımıyla listelenebilir.

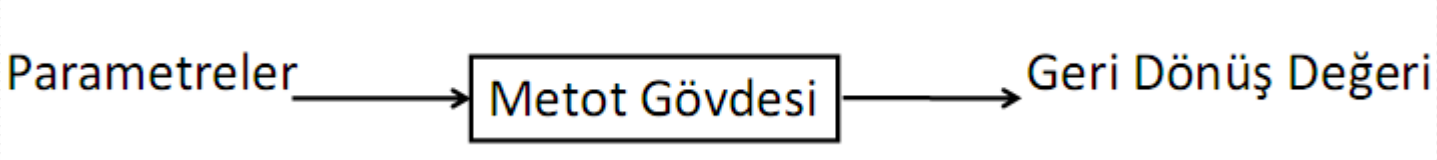
```
class Program
{ static void Main(string[] args)
  {   ArrayList liste = new ArrayList();
      liste.Add("Murat");
      liste.Add("Yeşim");
      liste.Add("Deniz");
      Console.WriteLine(liste[2]);
      foreach (String eleman in liste)
      {   Console.WriteLine(eleman); }
   }
}
```

ArrayList Sınıfı ve Özellikleri

- **ArrayList Sınıfı İçerisinde Kullanılan Diğer Önemli Metodlar**
- **Count:** ArrayList içerisindeki toplam eleman sayısını int türünde döndürür.
- **AddRange:** Diğer koleksiyon tabanlı nesneleri ya da dizileri ArrayList içerisine aktarır.
- **Sort:** Listedeki elemanları sıralar. İstenirse sadece belirli indeks numarasından sonraki elemanlarda sıralanabilir.
- **Reverse:** Tüm listeyi tersine çevirir. Yani listedeki birinci eleman sonuncu, sonuncu eleman birinci olacak şekilde tüm liste tersine döndürülür.
- **Clear:** Tüm ArrayList içerisindeki elemanları siler.
- **BinarySearch:** Liste içerisinde parametre olarak verilen değerin bulunup bulunmadığını arar. Bulunursa indis numarasını döndürür, bulamazsa negatif bir değer döndürür. BinarySearch ile arama yapmak için öncelikle listenin sıralanması gerekir.
- **CopyTo:** Tüm liste ya da listenin bir bölümünün başka bir ArrayList'e ya da diziye kopyalanmasını sağlar.

Metot Kavramı

- Programın herhangi bir yerinde kullanmak için belirli bir işi yerine getirmek amacıyla tasarlanmış alt programlara **metot** denir.
- Metotlar tek başlarına çalıştırılabilen yapılar değildir.
- Metodun iş yapması için kendisini çağıran metottan aldığı bilgilere parametre(argüman), kendisini çağıran fonksiyona döndürdüğü bilgiye ise geri dönüş değeri (return value)



Metot Bildirimi

- C#'ta bildirilen bütün metotlar mutlaka bir sınıfın içinde olmalıdır. Bir sınıfın üyesi olmayan metotlar bildirilemez. Metotları tanımlarken kullanılan bildirim şu şekildedir:

- **[erişim] <dönüş değeri> metot ismi (parametre listesi)**

  {

  metot gövdesi

  }

Örn:

```
 int MetotAdi(int a,int b)
{ return a+b; }
```

- **Erişim belirteçleri:** metoda nerelerden erişilebileceğini ayarlar (public, private…) Belirtilmediği durumlarda "private" olarak kabul edilir. Yani sadece tanımlandığı sınıf içinde kullanılabilen bir metot olur.

Metot Bildirimi

- C#'ta bir metot kullanılacaksa metodun içinde bulunduğu sınıf türünden bir nesne tanımlanır ve '.' operatörü ile metot çağrılır.

```
using System;
class Metotlar
{   int Topla(int a, int b) { return a + b; }

   static void Main()
    {   Metotlar nesne = new Metotlar();
        int a = nesne.Topla(2, 5);
       // refrans değişken ile metoda erişim
        Console.Write(a);
    }
}
```

Metot Bildirimi

- **static** olarak tanımlanan metotlara erişmek için metodun içinde bulunduğu sınıf türünden bir nesne yaratmaya gerek yoktur. static olarak tanımlanan metotlara sadece metodun adını yazarak erişilebilir. Örnek:

```
using System;
class Metotlar
{   static int Topla(int a, int b)   { return a + b; }

    static void Main()
    {       int a = Topla(2, 5);
            // static; metot adı ile metoda erişim
        Console.Write(a);
    }
}
```

Metot Bildirimi

- Eğer metot, içinde bulunduğumuz sınıfta değil de, başka bir sınıf içinde oluşturulmuş ise o metodu kullanabilmek için önce sınıfı yazmamız gerekir. Örnek:

```
using System;
class Metotlar1
{ public static int Topla(int a,int b) { return a+b; }
}
class Metotlar2
{  static void Main()
  {
   int a= Metotlar1.Topla(2,5);
    //farklı  sınıftaki static metoda sınıfın adı ile erişim (nesne oluşturulmadı)
   Console.Write(a);
  }
}
```

Metot Bildirimi

- static olmayan bir metodu başka bir sınıf içinde kullanma
- using System;
- class Metotlar1
- {
- public int Topla(int a,int b) { return a+b; }
- }
- class Metotlar2
- {
- static void Main()
- { Metotlar1 nesne = new Metotlar1();
- //farklı sınıftaki metoda refrans değişken ile erişim(nesne oluşturuldu)
- int a=nesne.Topla(3,9);
- Console.Write(a);
- }
- }

Metot Bildirimi

- **Metotlarla ilgili önemli özellikler**

- **1-** Metotlar isimlendirilirken, değişken isimlendirmede uyulması gereken kurallara uyulmalıdır. Ayrıca Main() isimli ikinci bir metot daha tanımlanamaz.

- **2-** Eğer bir metot geriye değer döndürmeyecekse "**void**" olarak tanımlanır. Eğer giriş parametresi yoksa parantez içi boş bırakılır.

  - ```
    static void Yaz()
    ```
  - ```
    {  Console.Write("deneme");  }
    ```

  - (C ve C++ dillerinde bir fonksiyonun parametre almadığını açıkça bildirmek için parametre parantezine void anahtar sözcüğü yazılır, fakat C#'da böyle bir kullanım yoktur.)

Metot Bildirimi

- **3-** Geri dönüş değeri olmayan metotlarda (void) **return** anahtar sözcüğünün yanında herhangi bir ifade olmadan da kullanılabilir.
- **Örnek:**

```
using System;
class Metotlar1
{
  static void Yaz(object a,int b)
  {
      if(b>10)  return;
    /* değer tutmayan return komutu kullanımı sadece çıkışı sağlar */
       for(;b>0;b--)  Console.Write(a);
  }
 static void Main()
 {    Yaz("Merhaba",10); }
}
```

Metot Özellikleri

```
static void islem(int a, int b)
    {
        return a + b;
    }
static void Main(string[] args)
    {
        int c = islem(5, 4);
    }
```

- Geri dönüş değeri olmayan metotlarda return anahtar sözcüğünü bir ifade ile kullanmak yasaktır

Metot Bildirimi

- **4-**“**return**” anahtar sözcüğü ile geri değer döndürürken de türlerin uyumlu olmasına dikkat edilmelidir.

- **5-** Metotlara parametre gönderirken tür dönüşümü kurallarına uyulmalıdır, uygun türler kullanılmalıdır. Geri dönüş değerinde yapılan gizli tür dönüşümleri ve tür uyumsuzluk kuralı metot parametreleri için de geçerlidir.

```
static void islem(int a, int b)
    {
        Console.WriteLine(a + b);
        return;

    }
    static void Main(string[] args)
    {
        double x, y;
        x = 5.45D;
        y = 6.88D;
        islem(x, y);
        Console.ReadLine();
    }
```

Metot Bildirimi

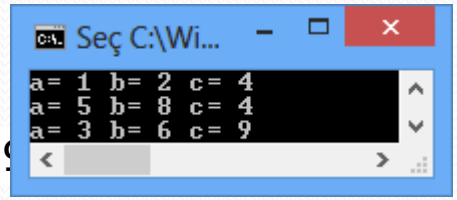
- Geri dönüş tipi object olamaz, var ve dynamic olur.
- object kullanılacaksa işlem yapılması durumunda cast yani unboxing yapılmalı yoksa hata verir.

```
object topla (object a, object b)
{
    object c = (int)a + (int)b;
    Console.WriteLine((int)a + (int)b);
    return c;

}
```

Metot Bildirimi

- **6-** Metot çağırırken parametreler eksiksiz girilmelidir. Eksik parametreli metot çağrıları geçersizdir. Eğer parametreler daha önce tanımlıysa eksik girilebilir. Ama bir parametrenin değeri atandıktan sonra bundan sonraki diğer parametrelerin değeri de atanmalıdır.

```
class Program
{    public static void metotlar(int a, int b = 2, int c=4)
    { Console.WriteLine("a= " + a + " b= " + b + " c= " + c);
    }
    static void Main(string[] args)
    {
        metotlar(1); metotlar(5,8); metotlar(3,6,9
    }
}
public static void metotlar(int a, int b = 2,int c)
```



- Derleme hatası verir. **b** değeri belirlenmişse bundan sonraki tüm değerler belirlenmelidir.

Metot Özellikleri

- **7-**Bir metot içerisinde başka bir metodun yapısı bildirilemez.

```
class Program
  {
    static void islem(double a, double b)
    {
      Console.WriteLine(a + b);
      return;
      int carp(int m,int n)
      { return m*n;}
    }
    static void Main(string[] args)
    {
      int x, y;
      x = 5;
      y = 6;
      islem(x, y);
      Console.ReadLine();
    }

  }
```

Metot Özellikleri

- **8-** Metot parametre parantezindeki değişkenlerin türlerinin tek tek belirtilmesi gerekir. ‘ **,** ’ ile ortak tür bildirimi yapılamaz.

```
static int islem(int a,  b)
    {           return a + b;
    }
static void Main(string[] args)
    {
        int c = islem(5, 4);
```

Metot Özellikleri

- **9-**Metot parametrelerinin metot içerisinde tanımlanması geçersizdir.

```
static int islem(int a,  b)
    {       int b;
            return a + b;
    }
static void Main(string[] args)
    {
        int c = islem(5, 4);
    }
```

Metot Özellikleri

- **10-** Metotlar içinde tanımlanan değişkenler, programın akışı metoda geldiğinde tanımlanır. Metodun işlevi bittiğinde ise bellekten silinirler.

- Programın akışı tekrar metoda geldiğinde değişkenler yeniden tanımlanır ve sonunda yine bellekten silinir. Bu tür değişkenlere otomatik ömürlü nesneler denir.

Metot Parametresi Olarak Diziler

- Diziler C# ‘ta ayrı bir tür olarak ele alındığı için metotlara dizileri aktarmak normal veri türlerinde olduğu gibidir.

```
using System;
 class Metotlar
 {  static void Yaz(int[] dizi)
    {  foreach(int i in dizi)
           Console.WriteLine(i);
    }
     static void Main()
     {  int[] dizi={1,2,4,7,9};
        Yaz(dizi);
     }
}
```

Metot Parametresi Olarak Diziler

- Eğer sadece **int []** türündeki değil, bütün türlerdeki dizileri ekrana yazan genel bir metot yazmak istiyorsak dizi tipi **Array** tanımlanır:

```
using System;
 class Metotlar1
 {
     static void Yaz (Array dizi)
     {  foreach(object i in dizi)
            Console.WriteLine(i);
     }
     static void Main() {
     int [ ] dizi={1,2,4,7,9}; Yaz(dizi); }
}
//object yerine var veya dynamic kullanılabilir
```

Metot Parametresi Olarak Diziler

- Dizilerde değişkenlerdeki gibi bir bilinçsiz tür dönüşümünden bahsetmek imkansızdır. Array yerine **object** veya **var** kullanılırsa hata verir, "**dynamic**" kullanılırsa hata vermez.
  - var sadece tek tür alabildiğinden hata verir;
  - Object te ise unboxing yapılmalı.

```
using System;
 class Metotlar1
 {
    static void Yaz (object [] dizi)
    {  foreach(object i in dizi)
        Console.WriteLine(i);
    }
static void Main() {
 int[] dizi={1,2,4,7,9}; Yaz(dizi); }
}
```

```
using System;
 class Metotlar1
 {
    static void Yaz (dynamic [] dizi)
    // veya  static void Yaz (dynamic dizi)
    {  foreach(dynamic  i in dizi)
        Console.WriteLine(i);
    }
static void Main() {
 int[] dizi={1,2,4,7,9}; Yaz(dizi); }
}
```

Değer ve Referans Parametreleri

- Referans türleri metotlara aktarılırken bit bit kopyalanmaz, yani metoda aktarılan sadece bir referanstır. Metot içinde yapılacak değişiklikler nesnenin(dizinin) değerini değiştirebilir.

- C# dilinde değer tipleri metoda bit bit kopyalanır.

- Değer tipleri parametre olarak verildiğinde fonksiyon içerisinde yeni bir nesne oluşturulur ve değeri buraya kopyalanır. Yani fonksiyon içerisinde değer tipindeki bir veri değiştiğinde çağıran programdaki ana değer değişmez.

Dizi ve değişken parametreler arasındaki fark

- Aşağıdaki iki programı karşılaştırın:

```
using System;
class Metotlar1 {
 static void Degistir(int[] dizi)
 { for(int i=0;i<5;i++)
   dizi[i]=10; }
static void Yaz(Array dizi)
 { foreach(object a in dizi)
Console.WriteLine(a); }
 static void Main() {
 int[] dizi={1,2,4,7,8}; Degistir(dizi);
Yaz(dizi); } }
```

```
using System;
class Metotlar1 {
static void Degistir(int sayi)
 { sayi=10; }
static void Yaz(int sayi)
 { Console.WriteLine(sayi);
}
 static void Main()
 { int sayi=1; Degistir(sayi); Yaz(sayi); }
}
```

Ref ve Out Anahtar sözcükleri

- Değer tiplerinin metoda referans tip olarak aktarılması için "ref" ve "out" anahtar sözcükleri kullanılır.

- **ref** anahtar sözcüğü ile belirtilen parametreler değer tipi de olsa referans olarak aktarılırlar. Genellikle değer tiplerini referans olarak metotlara geçirmeye zorlamak için kullanılırlar.

- Referans tipleri metotlara zaten referans olarak aktarıldıkları için **ref** anahtar sözcüğünü kullanmaya gerek yoktur. Fakat, kullanılması da kısıtlanmamıştır.

Ref ve Out Anahtar sözcükleri

- **Örnek:**
- using System;
- class Metotlar1
- {
- static void Degistir (ref int sayi) { sayi=10; }
- static void Yaz(int sayi) { Console.WriteLine(sayi); }
- static void Main()
- { int sayi=1; Degistir(ref sayi); Yaz(sayi); }
- }

Ref ve Out Anahtar sözcükleri

- **ref** sözcüğü, hem metodu çağırırken , hem de metodu yaratırken değişkenden önce yazılması gerekiyor.

```
using System;

class RefOrnek
{
    static void DegerTipi(ref int x)
    {
        x = 111;
    }

    static void Main()
    {
        int x = 25;

        DegerTipi(x);
        Console.WriteLine(x);
    }
}
```

Ref ve Out Anahtar sözcükleri

- ref sözcüğüyle bir değişkenin metoda adres gösterme yoluyla aktarılabilmesi için esas programda değişkene bir ilk değer verilmelidir. Yoksa program hata verir.

```
using System;
class Metotlar1
{
  static void Degistir(ref int sayi)      {       sayi=10;   }
  static void Yaz(int sayi)   {       Console.WriteLine(sayi);   }
  static void Main()
  {    int sayi;
       Degistir(ref sayi);
       Yaz(sayi);
  }
}
```

Ref ve Out Anahtar sözcükleri

- İlk değer verilmediği durumlarda program hata verecektir. Bu tür hataları önlemek için **out** anahtar sözcüğünü kullanırız.

- out anahtar sözcüğünün ref anahtar sözcüğünden tek farkı ilk değer verme zorunluluğunun olmayışıdır.

- **NOT:** ref sözcüğünün dizilerle kullanımı gereksiz olmasına rağmen C# bunu kısıtlamamıştır. Ancak out sözcüğü dizilerle kullanılamaz.

- **ref ve out farkları ise;**
  - **ref kullanırken varsayılan değer ataması zorunludur, out için bu zorunluluk yoktur.**
  - **Metot'a out anahtar sözcüğü ile geçilen değişkenin aynı metot içinde değer ataması yapılması zorunlu iken, ref için böyle bir zorunluluk yoktur.**

Ref ve Out Anahtar sözcükleri

```
using System;

class OutOrnek
{
    static void DegerTipi(out int x)
    {
        x = 111;
    }

    static void Main()
    {
        int x;

        DegerTipi(out x);
        Console.WriteLine(x);
    }
}
```

Metotların Aşırı Yüklenmesi

- Derleyici hangi metodu çağıracağına karar vermek için, metot bildirimi ile metot çağrımı arasındaki tam uyumu arar. Eğer tam uyumluluk bulunamaz ise parametreler arasında küçük türün büyük türe dönüşmesinin yasal olabileceği metot çağrılır.

```
static void OrnekMetot(double x, double y)
{
  Console.WriteLine("1.Metot");
}
static void OrnekMetot(byte x, byte y)
{
   Console.WriteLine("4.Metot");
}

static void Main()
{
    OrnekMetot(1919,1923);
    OrnekMetot(12, 34);
    Console.ReadLine();
}
```

Değişken Sayıda Parametre Alan Metotlar

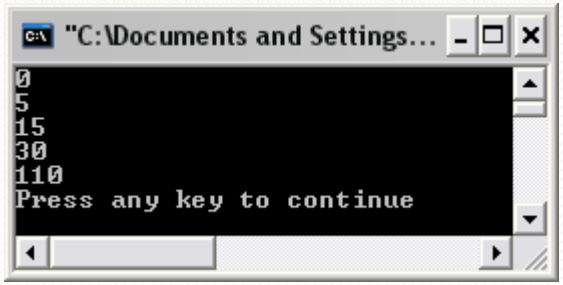
- Metotlara gönderilebilecek parametre sayısı bilinmediği zamanlarda değişik sayıda parametre alan metotlar tanımlanabilir.

- Bu metotlar parametreleri dinamik bir dizi içinde depolayarak kullanılmasını sağlarlar.

- Değişken sayıda parametre alan metot tanımlamak için **params** anahtar sözcüğü kullanılır .

Değişken Sayıda Parametre Alan Metotlar

```
using System;
class paramsa
{
    static int Toplam(params int[] sayılar)
    {
        if (sayılar.Length == 0)
            return 0;
        int toplam = 0;
        foreach (int sayi in sayılar)
            toplam += sayi;
        return toplam;
    }
    static void Main()
    {
        Console.WriteLine(Toplam());
        Console.WriteLine(Toplam(5));
        Console.WriteLine(Toplam(5, 10));
        Console.WriteLine(Toplam(5, 10, 15));
        Console.WriteLine(Toplam(5, 10, 15, 30, 50));
    }
}
```



Değişken Sayıda Parametre Alan Metotlar

- Değişken sayıda parametre içeren metotlar, aşırı yüklenmiş metotlar olduğunda değerlendirmeye alınmazlar (params ifadesi). Bu programdaki gibi metodumuzda değişken parametre yanında bir ya da daha fazla normal sabit parametre de olabilir. Ancak değişken parametre mutlaka en sonda yazılmalıdır.

```
using System;
class Metotlar
{
    static void Metot1(int x, int y)
    { Console.WriteLine("1. metot çağrıldı."); }
    static void Metot1(int x, params int[] y)
    { Console.WriteLine("2. metot çağrıldı."); }
    static void Main()
    { Metot1(3, 6); }
}                                   // 1. metot çağrılır
```

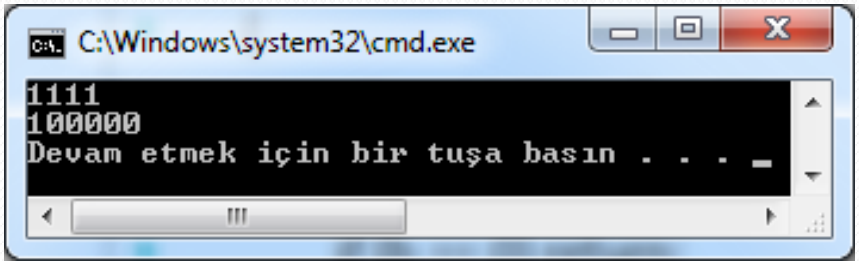
Değişken Sayıda Parametre Alan Metotlar

```
using System;
class Metotlar
{
    static void Metot1(int x, int y)
    { Console.WriteLine("1. metot çağrıldı."); }
    static void Metot1(int x, params int[] y)
    { Console.WriteLine("2. metot çağrıldı."); }
    static void Main()
    { Metot1(3, 6, 8); }
}
                                //Burada 2. metot çağrılır.
```

Rekürsif metotlar

- Kendi kendini çağıran metotlara özyineli(recursive) metot denilir. Bu metotlarda, çağrımı sonlandıran bir kontrol yapısı da olmalıdır.
- **Örnek:** 10 luk sayıyı 2 lik sayıya dönüştüren program

```
using System;
class Program
{
    static void BitYaz(int b)
    {   if (b == 0) return;
            BitYaz(b >> 1);
            Console.Write(b & 1);  // Console.Write(b ); ikiye bölümünden kalan sonucu yazar
     }
    static void Main()
    {   BitYaz(15); Console.WriteLine();   // 1 3 7 15
        BitYaz(32); Console.WriteLine();   // 1 2 4 8 16 32
    }
}
```



Visual Studio.Net -C#

5. HAFTA

Main Metodu

Yapıcı ve Yıkıcı Metotlar

Statik Metotlar

ÖRNEK UYGULAMALAR

Örnek 1: Diziler

```
using System;

class dizi_3
{
    static void Main()
    {
        Random r = new Random();

        int[] dizi = new int[20];
        char[] chr = new char[20];

        for (int i = 0; i < 20; i++)
        {
            dizi[i] = r.Next(1, 51);
            chr[i] = (char)r.Next(20, 126);
        }
        for (int i = 0; i < 20; i++)
        {
            Console.Write("{0,2}. değer {1,2} -> ", i, dizi[i]);
            for (int j = 0; j < dizi[i]; j++)
                Console.Write(chr[i]);
            Console.WriteLine();
        }
    }
}
```

Örnek 2:Matris Dizileri

```
using System;

class dizi_4
{
    static void Main()
    {
        int[,] dizi = { { 1, 2 }, { 3, 4 }, { 5, 6 } };

        for (int i=0; i<3; i++)
            for (int j =0; j < 2; j++)
            {
                Console.WriteLine(dizi[i, j]);
            }
    }
}
```

Örnek 3 : Düzensiz Diziler

```
using System;

class dizi_6
{
    static void Main()
    {
        int[][] dizi = new int [3][];

        dizi[0] = new int[] {1,2};
        dizi[1] = new int[] {3,4,5,6,7};
        dizi[2] = new int[] {8,9,0};

        foreach (int[] boyut in dizi)
        {
            foreach (int eleman in boyut)
            {
                Console.Write("{0,3}", eleman);
            }
            Console.WriteLine();
        }
    }
}
```

Örnek 4: diziler

```
int[,] d = {{2,3},{4,5},{6,7}}; // 3x2 lik int dizi.
d[2,1]=33; //doğru
d[2][1]=33; //hata
Console.WriteLine(d.GetValue(2,1)); //  7 değerini yazacak
int[][] dd = new int[2][];  // şekildeki düzensiz dizi oluşturuluyor.
dd[0] = new int[2];  //ilk satırın 2 sutunlu olacağı
dd[1] = new int[1];  //ikinci satırın 1 sütunlu olacağı
dd[0][0] = 23;
dd[0][1] = 234;
dd[1][0] = 44;
Random rnd = new Random();
int[] dizi =new int[3];
dizi[0] = rnd.Next(2,10); // 2 ile 10 arası rastgele sayı
dizi[1] = rnd.Next(50);   //0 ile 50 arası
dizi[2] = rnd.Next();  //rastgele integer değer
Console.WriteLine(dizi.GetValue(2));
//dizi değişkeninin 2. indis elemanını (3.eleman) yazar.
Console.WriteLine(dd[1][0]);
```

23	234
44	

Örnek 5: foreach döngü

```
string[] isimler ={ "ali", "ahmet", "selda", "canan", "melike" };
Console.WriteLine("aranan isim=");

string aranan=Console.ReadLine();

foreach(ss in isimler)   {

 if (aranan.Equals(isimler[0]))
  Console.WriteLine("aranan isim bulundu...");
   else Console.WriteLine("isim yok");
   }
  Console.ReadKey();
```

Örnek

```
using System;

class metotlar_3
{
    static int BuyukBul(int a, int b)
    {
        if (a > b)
            return a;
        return b;
    }

    static void Main()
    {
        int s1, s2;
        Console.Write("1. Sayıyı Girin:");
        s1 = Convert.ToInt32(Console.ReadLine());
        Console.Write("2. Sayıyı Girin:");
        s2 = Convert.ToInt32(Console.ReadLine());

        int enbuyuk = BuyukBul(s1, s2);

        Console.WriteLine("En büyük: {0}\'dir.", enbuyuk);
    }
}
```

Örnek

- Bir metodun geri dönüş değerini aynı metot için parametre olarak kullanabiliriz.

```
using System;

class metotlar_4
{
    static int BuyukBul(int a, int b)
    {
        if (a > b)
            return a;
        return b;
    }

    static int BuyukBul3(int a, int b, int c)
    {
        return BuyukBul(a, BuyukBul(b, c));
    }

    static void Main()
    {
        int s1, s2, s3;
        Console.Write("1. Sayıyı Girin:");
        s1 = Convert.ToInt32(Console.ReadLine());
        Console.Write("2. Sayıyı Girin:");
        s2 = Convert.ToInt32(Console.ReadLine());
        Console.Write("3. Sayıyı Girin:");
        s3 = Convert.ToInt32(Console.ReadLine());

        int enbuyuk = BuyukBul3(s1, s2, s3);

        Console.WriteLine("En büyük: {0}\'dir.", enbuyuk);
    }
}
```

Metot Parametresi Olarak Diziler

```
using System;
class metotlar_diziparametre
{
    static void DiziYaz(int[] a, int Sekil)
    {
        if (Sekil == 0)
        {
            foreach (Object o in a)
                Console.Write(o.ToString() + " ");
            Console.WriteLine();
        }
        else if (Sekil == 1)
        {
            int x=1;
            foreach (Object o in a)
            {
                Console.Write("{0,5}",o.ToString());
                if (x % 3 == 0) Console.WriteLine();
                x++;
            }
        }
        else
        {
            foreach (Object o in a)
                Console.WriteLine(o.ToString());
        }
        Console.WriteLine();
    }

    static void Main()
    {
        int[] dizi = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
        DiziYaz(dizi, 0);
        DiziYaz(dizi, 1);
        DiziYaz(dizi, 2);
    }
}
```

Örnek

```
using System;

class metotlar_5a
{
    static void DegerTipAktarim(int Sayi)
    {
        Sayi = 30;
    }

    static void Main()
    {
        int x = 100;
        Console.WriteLine(x);

        DegerTipAktarim(x);
        Console.WriteLine(x);
    }
}
```

- x değişkeninin değerini metot içinde değiştirmek değerini değiştirmemiştir. Çünkü x değer olarak metoda aktarılmıştır.

Metotların Aşırı Yüklenmesi Örnek

```
using System;
class Metotlar
{   static void Metot1(float x,float y)
  { Console.WriteLine("1. metot çağrıldı.");
  }
  static void Metot1(double x,double y)
  { Console.WriteLine("2. metot çağrıldı.");
  }
  static void Main()
  {   Metot1(5,6);   }
}
```

- Bu programda iki metodun da parametre sayısı eşit, iki metotta da tam tür uyumu yok ve iki metotta da bilinçsiz tür dönüşümü mümkün. Bu durumda en az kapasiteli türlü metot çağrılır. Yani bu programda birinci metot çağrılır.

```
using System;
class Metotlar
{   static void Metot1(float x,float y)
  { Console.WriteLine("1. metot çağrıldı.");
  }
  static void Metot1(int x,int y)
  { Console.WriteLine("2. metot çağrıldı.");
  }
  static void Main()
  {     Metot1(5,6.4f);   }
}
```

- Metot1 çağrılır.

Metotların Aşırı Yüklenmesi Örnek

- using System;
- class Metotlar
- {
- static void Metot1(float x,float y)
- { Console.WriteLine("1. metot çağrıldı.");
- }
- static void Metot1(int x,int y)
- { Console.WriteLine("2. metot çağrıldı.");
- }
- static void Main()
- { Metot1('f','g'); }
- }
- Bu durumda ikinci metot çağrılır. Çünkü char hem int e hem de float a bilinçsiz olarak dönüşebilir. Ancak int daha az kapasitelidir.

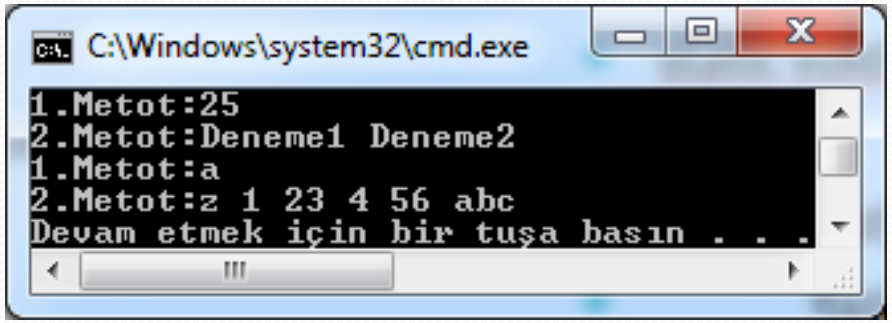
- using System;
- class Metotlar
- { static void Metot1(int x,int y,int z)
- { Console.WriteLine("1. metot çağrıldı."); }
- static void Metot1(int x,int y)
- { Console.WriteLine("2. metot çağrıldı."); }
- static void Metot1(float x,int y)
- { Console.WriteLine("3. metot çağrıldı."); }
- static void Main()
- { Metot1(3,3,6);
- Metot1(3.4f,3);
- Metot1(1,'h'); }
- }
- Burada sırasıyla 1., 3. ve 2. metotlar çağrılacaktır.

Değişken Sayıda Parametre Alan Metotlar Örnek

```
using System;
class Metotlar
{   static int Islem(string a, params int[] sayilar)
    {   if (a == "carp")
        {   if (sayilar.Length == 0) return 1;
            int carpim = 1;
            foreach (int i in sayilar) carpim *= i;
            return carpim;
        }
        else if (a == "topla")
        {   if (sayilar.Length == 0) return 0;
            int toplam = 0;
            foreach (int i in sayilar) toplam += i;
            return toplam;
        }
        else return 0;
    }
    static void Main()
    {   Console.WriteLine(Islem("topla", 3, 4, 7, 8));
        Console.WriteLine(Islem("carp", 5, 23, 6));
    }
}
```

Değişken Sayıda Parametre Alan Metotlar Örnek

```
using System;
namespace params_ornek{
 class Program    {
   static void Yaz(object o)
   {
    Console.WriteLine("1.Metot:" + o);
   }
  static void Yaz(params object[] o)
  {
   if (o.Length == 0) return;
   Console.Write("2.Metot:");
   foreach (object n in o)
    Console.Write(n.ToString() + " ");
   Console.WriteLine();
  }
  static void Main(string[] args)
  {
   Yaz(25);
   Yaz("Deneme1", "Deneme2");
   Yaz('a');
   Yaz('z', 1, 23f, 4, 56, "abc");
  }
 }
}
```



Kendini çağıran (Recursive) metotlar Örnek

```
using System;
class faktoriyel {
        static int Faktoriyel(int a)      {
                if(a==0)
                return 1;
                return a*Faktoriyel(a-1); //fonksiyon kendi içerisinde çağrıldı
            }
        static void Main()    {
        Console.Write("Faktöriyelini alacağınız sayıyı giriniz :");
        int fak_değeri=Convert.ToInt32(Console.ReadLine());
        Console.WriteLine();
        Console.WriteLine("Girmiş olduğunuz sayının faktöriyeli :"+
                                Faktoriyel(fak_değeri));
        }
}
```

Visual Studio.Net -C#

5. HAFTA

Main Metodu

Yapıcı ve Yıkıcı Metotlar

Statik Metotlar

Main Metodu

- Main metodunu diğer metotlardan temel olarak farkı yoktur, tek farkı programın çalışmaya başladığı nokta olmasıdır.

- Main metodunun int türünden bir geri dönüş değeri de olabilir. Bu değer işletim sistemine programın çalışması hakkında bilgi vermek amacıyla kullanılır. Örneğin Main metodu **0** değerini döndürürse işletim sistemi programın düzgün şekilde sonlandırıldığını, **1** değerini döndürürse de hatalı sonlandırıldığını anlayabilecektir.

- Main metodu parametre de alabilir. Komut satırından çalıştırıldığında verilen parametreleri bir **string** dizisi içinde alarak işleyebilir.

- static int Main (**string [] args**)

Main Metodu

- static int Main (**string [] args**)
- Burada komut satırından girilen argümanlar string[] türündeki args dizisine aktarılıyor. Bu diziyi programımızda gönlümüzce kullanabiliriz. Örneğin programımız deneme.exe olsun. Komut satırından;
- ***deneme ayşe bekir rabia***
- girilirse ilk sözcük olan deneme ile programımız çalıştırılır ve ayşe, bekir ve rabia sözcükleri de string[] türündeki args dizisine aktarılır. komut satırında program adından sonra girilen ilk sözcük args dizisinin 0. indeksine, ikinci sözcük 1. indeksine, üçüncü sözcük 2. indeksine vs. aktarılır. Örnek bir program:
- **using** System;
- **class** Metotlar
- { **static void** Main(**string**[] args) {
- Console.WriteLine("Komut satırından şunları girdiniz: ");
- **foreach**(**string** i **in** args) Console.WriteLine(i); }
- }

System.Math Sınıfı()

- .Net sınıf kütüphanesinde belirli matematiksel işlemleri yapan birçok metot ve iki tane de özellik vardır. System.Math sınıfındaki metotlar static oldukları için bu metotları kullanabilmek için içinde bulundukları sınıf türünden bir nesne yaratmaya gerek yoktur. System.Math sınıfındaki iki özellik matematikteki pi ve e sayılarıdır. PI ve E özellikleri double türünden değer tutarlar.

```
using System;
class Metotlar
{
    static void Main()
    {
        double e=Math.E;
        double pi=Math.PI;
        Console.Write("e->"+e+" pi->"+pi);
    }
}
```

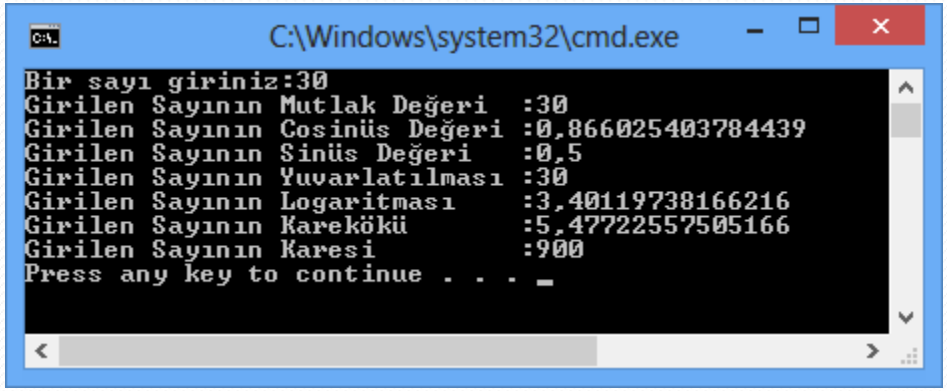
System.Math Sınıfı()

- System.Math sınıfındaki bütün metotlar

Abs(x)	Bir sayının mutlak değerini tutar.
Cos(x)	Sayının kosinüsünü tutar.
Sin(x)	Sayının sinüsünü tutar.
Tan(x)	Sayının tanjantını tutar.
Ceiling(x)	x sayısını x ten büyük en küçük tamsayıya yuvarlar.
Flor(x)	x sayısını x ten küçük en büyük tamsayıya yuvarlar.
Max(x,y)	x ve y sayılarından en büyüğünü bulur.
Min(x,y)	x ve y sayılarından en küçüğünü bulur.
Pow(x,y)	x üzeri y yi hesaplar.
Round(x)	x i ne yakın tam sayıya yuvarlar.
Sqrt(x)	x in karekökünü bulur.
Log(x)	x sayısının e tabanında logaritmasını alır.
Exp(x)	e üzeri x'i hesaplar.
Log10(x)	x sayısının 10 tabanında logaritmasını hesaplar.

System.Math Sınıfı()

```
using System;
class argümanliste
{
  static void Main()
  {
     double a;
     Console.Write("Bir sayı giriniz:");
     a = Convert.ToDouble(Console.ReadLine());
     Console.WriteLine("Girilen Sayının Mutlak Değeri  :" + Math.Abs(a));
     Console.WriteLine("Girilen Sayının Cosinüs Değeri :" + Math.Cos(a*Math.PI /180));
     Console.WriteLine("Girilen Sayının Sinüs Değeri   :" + Math.Sin(a*Math.PI /180));
     Console.WriteLine("Girilen Sayının Yuvarlatılması :" + Math.Round(a));
     Console.WriteLine("Girilen Sayının Logaritması    :" + Math.Log(a));
     Console.WriteLine("Girilen Sayının Kareköku       :" + Math.Sqrt(a));
     Console.WriteLine("Girilen Sayının Karesi         :" + Math.Pow(a, 2));
   }
}
```



Sınıflar(Classes)

- Nesne Yönelimli Programlama(NYP) nedir?
- Çözülmesi istenen problemi çeşitli parçalara ayırıp her bir parça arasındaki ilişkiyi gerçeğine uygun şekilde belirleme tekniğine NYP denir. Nesnelerin kaynak kodundaki karşılığı sınıflar dır.

- NYP de her şeyin nesnelere dayalı olması gerekir. C# ‘ta her şey nesne olduğu için %100 nesne yönelimlidir.

Sınıflar(Classes)

- Sınıflar nesne yönelimli programlamanın en önemli öğesidir. Sınıflar sayesinde programlarımızı parçalara bölüp karmaşıklığını azaltırız.
- Sınıf bildirimleri **class** anahtar sözcüğü kullanılarak yapılır ve sınıf bildirimleri için bellekte yer tahsis edilmez.
- Sınıfların üye elemanları değişkenler(özellik) ve metotlardır.
  - Metotlar bir veya daha fazla komutun bir araya getirilmiş halidir; parametre alabilirler, geriye değer döndürebilirler.
  - Özellikler ise bellek gözeneklerinin programlamadaki karşılıklarıdır. Bu bakımdan özellikler değişkenlere benzerler. Aradaki en temel fark değişkenlerin bir metot içinde tanımlanıp yalnızca tanımlandığı metot içinde etkinlik gösterebilmesine rağmen özelliklerin tıpkı metotlar gibi bir üye eleman olmasıdır. Bu bakımdan özelliklerin tuttuğu değerlere daha fazla yerden erişilebilir.

Sınıflar(Classes)

- Her metot ve özellik için, bildirim yapmadan önce erişim belirleyici türünü bildirilmesi gerekir. Erişim belirleyiciler bir metoda ve özelliğe nereden erişileceğini tanımlar.

```
class sınıf_adı
{
        <erişim belirleyici> <veri türü> özellik1;
        <erişim belirleyici> <veri türü> özellik2;

        <erişim belirleyici> <geri dönüş tipi> metot1(parametreler)
        {
              metot ifadeleri, deyimleri
        }

        <erişim belirleyici> <geri dönüş tipi> metot2(parametreler)
        {
              metot ifadeleri, deyimleri
        }
}
```

Sınıflar(Classes)

- C# dilinde 5 tane erişim belirleyici vardır.
- **public:** erişim belirteci ile tanımlanan sınıfın bir özelliğine ya da metoduna istenilen yerden erişime izin verilmiş olur. public üye elemanlar genelde sınıfın dışa sunduğu ara yüzü temsil eder.
- **private:** erişim belirteci ile tanımlanan üyelere ancak tanımlandıkları sınıfın içindeki diğer üye elemanlar erişebilir, dışarıdan erişmek mümkün değildir. Erişim belirteci verilmeyen tanımlamaların varsayılan değeri **private** olur.
- **protected:** erişim belirteci sınıf türemesi yoksa private ile aynıdır. Fakat bir sınıftan başka bir sınıf türetiliyorsa private üyeler diğer sınıfa aktarılmaz, sadece public ve protected elemanlar aktarılırlar.

Sınıflar(Classes)

- Sadece sınıflar için kullanılan diğer erişim belirleyicileri

- **internal:** internal olarak tanımlanan öğe, bulunduğu assembly'nin (Dll veya Exe dosyası) içinde erişilebilirdir. Dll veya Exe dosyasının içerisinde erişim için kısıtlama yoktur, ama dışarıdan erişilemez. (Her hazırlanan proje farklı assembly dosyası oluşturur)

- **protected internal:** protected internal erişim belirleyicisi, protected ve internal erişim belirleyicilerinin VEYA (OR) işlemiyle birleştirilmiş halidir. Tanımlandığı class'ın içinde ve o class'tan türetilmiş diğer class'ların içinde erişilebilir. Ayrıca, aynı assembly içinde olmasalar dahi, tanımlandığı class'tan türetilmiş diğer class'ların içinde de erişilebilirdir.

Sınıf Bildirimi

- Sınıf bir veri yapısıdır. Gerçek hayattaki herhangi bir nesne bu yapı ile modellenebilir.
- Sınıf tanımlamaları yapılırken amaca uygun isimler seçilmelidir.
- Örneğin bir kredi kartı hesabı için bir çok özellik vardır, hesap no,limit,kart sahibi. Bu bizim için gerçek hayattaki bir modeli temsil eden bir yapıdır. Örnekte sadece özellikleri olan bir sınıf tanımlanmıştır. Elemanlarının hepsi public tanımlandıkları için sınıfın tüm üye değişkenlerine dışarıdan erişilebilir.

```
class KrediHesabi
{
    public int HesapNo;
    public double Limit;
    public string KartSahibi;
}
```

Sınıf Nesneleri Tanımlama

- Oluşturulan sınıflardan nesneler tanımlamak için **new** anahtar sözcüğü kullanılır.

- Nesnelerin bildirilmesi ve tanımlanması şu şekilde olur:
  - **Ogrenci ogr1; (Bildirim)**

- Bildirim ile nesneye erişim için bir değişken tanımlanmış olur. Fakat nesne bellekte oluşturulmamıştır. Yer tahsisatı yapabilmek için **new** kullanılır. Bunun için de;
  - **Ogrenci ogr1 = new Ogrenci();**

  kullanılır.

- ogrenci ogr1; ***// nesneyi gösteren referans tanımlandı***
- ogr1 = new Ogrenci(); ***// Nesne için bellek tahsil edildi***

Sınıf Nesneleri Tanımlama

- Aynı proje dosyası içeresinde birden fazla sınıf tanımlamak mümkündür. Bu işlem karmaşıklığı azaltmak için farklı dosyalarda da yapılabilir. Derleyici bu dosyaların hepsini birden derleyebilir.

- Örnek derlendiğinde bazı uyarı mesajları çıkacaktır. Bu mesajlar ileride anlatılacak özel metotlarla giderilebilir.

- Sınıflar, **new** anahtar sözcüğü ile tanımlandığında sınıfın içerisindeki bütün üyeler varsayılan değere atanır.

- Sınıftan tanımlanan bir nesnenin üyelerine **'.'** operatörü ile ulaşılır. **public** olarak tanımlanmış üye elemanların değerleri değiştirilebilir.

Sınıf Bildirimi

- **Örnek:**
- **using** System;
- class SinifIsmi
- { **public** int ozellik1=55; // başlangıç değerleri verildi.
- **private** string ozellik2; // nesne oluşturulduğu anda varsayılan değerler atanıyor.
- float ozellik3; // private yazılmasa bile private gibi davranır
- **public** int metot1(int a,int b)
- { **return** a+b; }
- **private** void metot2(string a)
- { Console.WriteLine(a); }
- }
- class EsasSinif {
- **static** void Main() {//private erişim olmaz hata verir.
- SinifIsmi **nesne=new** SinifIsmi(); Console.WriteLine(nesne.ozellik1); Console.WriteLine(nesne.ozellik2); Console.WriteLine(nesne.ozellik3);
- Console.WriteLine(nesne.metot1(2,5)); nesne.metot2("deneme"); } }

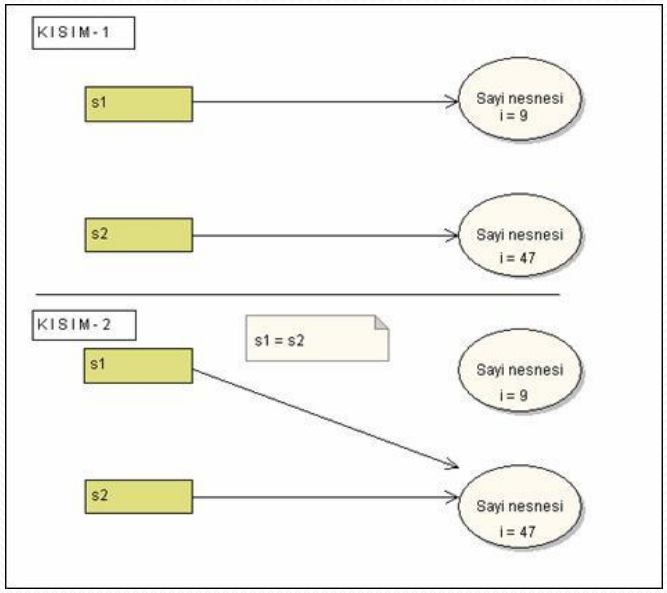
Birden Fazla Sınıf Nesnesi Tanımlama

- **Bir sınıftan birden fazla nesne de tanımlanabilir ve bu nesnelerle sınıfın özellik ve metotlarına erişilebilir.** Fakat bu tanımlanmış nesneler ile işlem yapılırken referans tiplerde anlatılan kurallara dikkat edilmelidir. Değer tipleri birbirine atanırken bitsel olarak kopyalanır. Referans türleri böyle değildir.

- Tanımlanmış iki nesne değeri atama operatörü ile birbirine eşitlenirse her iki nesne de aynı bellek bölgesindeki verileri temsil eder. Birinde yapılan değişikler diğerini de etkiler.

Birden Fazla Sınıf Nesnesi Tanımlama

```
using System;
class Sayi {
        public int i; }

class NesnelerdeAtama {

 public static void Main(string [] args) {
        Sayi s1 = new Sayi();
        Sayi s2 = new Sayi();

  s1.i = 9;    s2.i = 47;
  Console.WriteLine("1: s1.i: " + s1.i +", s2.i: " + s2.i);
  s1= s2; // referanslar kopyalaniyor.. nesneler degil
  Console.WriteLine("2: s1.i: " + s1.i +", s2.i: " + s2.i);
  s1.i = 27;
  Console.WriteLine("3: s1.i: " + s1.i +", s2.i: " + s2.i);
 }
}
```



Sonuçları yorumlayınız
1: s1.i: 9, s2.i: 47
2: s1.i: 47, s2.i: 47
3: s1.i: 27, s2.i: 27

Sınıf içinde başka bir sınıf tanımlama

Sınıflar ayrı ayrı tanımlanabildiği gibi sınıf içinde başka bir sınıfta oluşturulabilir. İçteki sınıfın eleman veya metotlarına erişmek için kurallar aynıdır.

```
using System;
class Sinif
{   int x=10; static int y = 0;
    class Sinif2
    {   public int x = 0;  static public int y=10;
    }
    static void Main()
    {   Sinif a = new Sinif();
        Sinif2 b = new Sinif2();
        //veya Sinif.Sinif2 b = new Sinif2();//
        a.x = 7; Sinif.y = 15;
        b.x = 12; Sinif2.y = 1;
    }
}
```

Lab-Uygulama

- Evinizin üyelerinin (annemizin, babamızın vs.) özelliklerini gireceğiniz, yaşlarını hesaplayacağınız, mesleklerini belirleyeceğiniz Evim sınıfını oluşturunuz.
- Bu sınıfa ait özellikleri ve metotları belirleyiniz.
- Haftaya rasgele biri seçilecek ve ödevini gösterecek

this Anahtar Sözcüğü

- Metotlar içinde nesnelerin özellikleri ve diğer metotlar kullanılabilir. Aslında burada nesnenin referansı da gizli bir şekilde aktarılmaktadır.

- Birçok durumda sorun yaşanmamakla birlikte bazen nesnenin üyeleri ile metot içindeki tanımlamalar aynı olabilir. Bu durumda metot değişkenleri ile nesnenin üyelerini birbirinden ayırmak için **this** anahtar sözcüğü kullanılır.

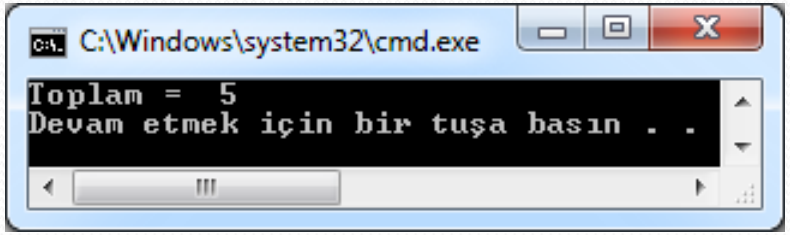
this Anahtar Sözcüğü

- this anahtar sözcüğü ilgili nesnenin referansını belirtir.
- **this** anahtar sözcüğü, içinde bulunulan nesneye ait bir referans döndürür
- Bunun sayesinde nesnelere ait global alanlara erişme fırsatı bulunur;

this Anahtar Sözcüğü

```
using System;
class Program
   {
   int topla; // Global alandaki değişken
   public void sayiTopla(int topla) // yerel alandaki değişken
   {
      this.topla += topla;
   }
     public void sayiyiEkranaYaz() {
          Console.WriteLine("Toplam =  " + topla);
  }
      static void Main(string[] args)
      {  Program th = new Program();
         th.sayiTopla(2);
         th.sayiTopla(3);
         th.sayiyiEkranaYaz();
      }
   }
```

this olmasaydı sonuç 0 olurdu. Neden?



Kapsülleme (Encapsulation)

- Kapsülleme (**Encapsulation**) kavramı bir nesnenin bazı özellik ve işlevlerini başka sınıflardan ve nesnelerden saklamaktır. **Private** erişim belirteci sayesinde bu şekilde tanımlanan bir alanı (field' i) başka sınıflardan gizlemiş oluruz. Ayrıca bu alan başka sınıflarda kullanılamaz.
- Kapsülleme (**Encapsulation**) sayesinde nesneler bilinçsiz kullanımdan korunmuş olur. Fakat bazı durumlarda private alanlara erişmemiz ve özelliklerini kullanmamız gerekebilir. Bu durumda özellik (**property)** kavramı devreye girer. Özellik bir alanın değerini geri döndürmeye (**Get**) ve değerini ayarlamaya (**Set**) olanak sağlar.

set ve get Anahtar Sözcükleri

- Bir nesnenin üye değişkenlerine değer atarken ya da içerisindeki değeri kullanırken belli kontrollerin yapılması gerekiyorsa ya da kod bloklarının çalıştırılması isteniyorsa **set** ve **get** ifadeleri kullanılır.

- **set** sözcüğü nesnenin özelliklerine değer atandığında çalışır.
- **get** sözcüğü ise özellik değeri okunduğunda ya da farklı bir ifadeye aktarılmaya çalışıldığında çalışır.

- set ve get yönteminin daha iyi anlaşılabilmesi için aynı işlemi yapan iki örneği ; 1.örnekte metotlar kullanılarak, 2. örnekte ise set ve get yöntemi kullanarak yapalım

set ve get Anahtar Sözcükleri

1.Örnek

```
using System;
class MetotSinif
{int Sayi;
 public void SayiBelirle(int sayi)
    {
        if (sayi < 0) Sayi = 0;
        else Sayi = sayi;
    }
    public int SayiyiAl()
    {
        if (Sayi > 100)
            return Sayi / 100;
        else
            return Sayi;
    }
}
```

```
class AnaSinif
{
 static void Main()
 {
 MetotSinif nesne =new MetotSinif();
 nesne.SayiBelirle(34);
 Console.WriteLine(nesne.SayiAl());
 }
}
```

set ve get Anahtar Sözcükleri

2.Örnek

```
class Set_Get
{
 int Sayi;
 public int SahteOzellik
 {
  set
  {
   if (value < 0) Sayi = 0;
   else Sayi = value;
  }
  get
  {
  if (Sayi>100) return Sayi/100;
  else return Sayi;
  }
 }
}
```

```
class AnaSinif
{
 static void Main()
 {
 Set_Get nesne = new Set_Get();
 nesne.SahteOzellik = 110;
 Console.Write(nesne.SahteOzellik);
 }
}
```

set ve get Anahtar Sözcükleri

- 2.Örnekte SahteOzellik adlı bir özellik oluşturduk. Bu özellik gerçekten de sahtedir. Özelliğe bir değer atanmaya çalışıldığında set bloğundaki, özellik kullanılmaya çalışıldığında da get bloğundaki komutlar çalıştırılır. Aslında C# kütüphanesindeki özelliklerin çoğu bu yöntemle oluşturulmuştur.
  - Örneğin ileride göreceğimiz Windows programlamada bir buton nesnesinin Text özelliğini değiştirdiğimizde butonun üzerindeki yazı değişir. Halbuki klasik özelliklere değer atama yönteminde sadece özelliğin değeri değişirdi. Bu yöntemde ise bir özelliğe değer atadığımızda çalışacak komutlar yazabiliyoruz.
- Programdaki value sözcüğü özelliğe girilen değeri tutar. Özelliğe girilen değer hangi türdeyse o türde tutar. Ayrıca bu oluşturulan SahteOzellik özelliğinin metotlara oldukça benzerdir. Görüldüğü gibi, değişkenin değerini değiştirdiğimizde çalışmasını istediğimiz kodları set blokları arasına yazarız.

set ve get Anahtar Sözcükleri

- Herhangi bir sahte özelliğin **set** veya **get** bloklarından yalnızca birini yazarak o özelliği salt okunur veya salt yazılır hâle getirebiliriz.
- Örneğin Array sınıfının Length özelliği salt okunur bir özelliktir.

- **NOT:** C# 2.0'da ya da daha üst versiyonlarda bir sahte özelliğin set ve get bloklarını ayrı ayrı private veya public anahtar sözcükleriyle belirtebiliyoruz. Yani bir özelliğin bulunduğu sınıfın dışında salt okunur ya da salt yazılır olmasını sağlayabiliyoruz.

Özelliklerde Erişim Belirleyiciler

```
private int m_uzunluk;
public int Uzunluk
{
   get
   {return m_uzunluk;
   }

   set
   { m_uzunluk = value;
   }
}
```

get ve set blokları public olarak tanımlıdır.

```
private int m_uzunluk;
public int Uzunluk
{
   get
   {return m_uzunluk;
   }

   private set
   { m_uzunluk = value;
   }
}
```

get ve set blokları; C#2.0'dan itibaren farklı erişim belirleyicileri ile belirlenebilmektedir.

- Değişken için bildirilen erişim belirleyicisi; get veya set bloğunda tanımlanan erişim belirleyicisinden daha yüksek erişim yetkisine sahip olmalıdır.

set ve get Anahtar Sözcükleri

- **NOT:** get ve set anahtar sözcükleriyle erişim belirleyiciler kullanırken uymamız gereken bazı özellikler vardır:
  - Daima özellik bildiriminde kullanılan erişim belirleyicisi get veya set satırında kullanılan erişim belirleyicisinden daha yüksek seviyeli olmalıdır. Örneğin özellik bildiriminde kullanılan erişim belirleyici private ise get veya set satırında kullanılan erişim belirleyici public olamaz.
  - get veya set satırında kullanılan erişim belirleyici özellik bildiriminde kullanılan erişim belirleyiciyle aynı olmamalıdır. (Zaten gereksizdir.)
  - set ve get satırlarındaki erişim belirleyicileri yalnızca, özelliği public olarak belirtmiş ancak özelliğin set veya get bloklarının herhangi birisini private yapmak istediğimizde kullanılabilir.
  - get veya set için erişim belirleyicisi kullanacaksak sahte özelliğin bloğu içinde hem get hem de set bloğunun olması gerekir.

Dizililerin metot ve özelliklerde kullanılması

- Aslında tanımladığımız her dizi Array sınıfı türünden bir nesnedir. İşte bu yüzden tanımladığımız dizilerle Array sınıfının metot ve özelliklerine erişebiliriz.

- Örneğin Length, Array sınıfı türünden bir özelliktir ve dizinin eleman sayısını verir. DiziAdi.Length yazılarak bu özelliğe erişilebilir.

- Metot ve özelliklerin geri dönüş tipi bir dizi olabilir. Örnek:

Dizilerin metot ve özelliklerde kullanılması

```
using System;
class YardimciSinif
{   public int[] Dizi = { 7, 4, 3 };
    public int[] Metot()
    {
        int[] a = { 23, 45, 67 };
        return a;
    }
}
class AnaSinif
{   static void Main()
    {   YardimciSinif nesne = new YardimciSinif();
        Console.WriteLine(nesne.Dizi[0]);
        Console.WriteLine( nesne.Metot()[2] );
    }
}
```

a [2]

Dizilerin metot ve özelliklerde kullanılması

- Bir dizinin türünü istenilen türe dönüştürmeye yarayan bir sınıf örneği bulunmamaktadır.

- C#'ta diziler arasında bilinçsiz tür dönüşümü mümkün olmadığı gibi, bu işi yapacak metot da yok.

- Aşağıda verilen örnekteki metotlar bu iş için kullanabilirsiniz.

- **Hatırlatma:** Metotların parametresindeki dizi, Array DiziAdi yöntemiyle oluşturulduğu için bu dizinin elemanlarına klasik indeksleme yöntemiyle erişemeyiz. Bu yüzden bu dizinin elemanlarına ulaşmak için Array sınıfının GetValue() metodu kullanıldı. (Diğer yol foreach-object, dynamic, var kullanma)

Dizililerin metot ve özelliklerde kullanılması

```
using System;
class Donustur
{   public static int[] Inte(Array dizi)
  {
     int[] gecici=new int[dizi.Length];
     for(int i=0;i<dizi.Length;i++)
gecici[i]=Convert.ToInt32(dizi.GetValue(i));
     return gecici;
  }
  public static string[] Stringe(Array dizi)
  {
     string[] gecici=new string[dizi.Length];
     for(int i=0;i<dizi.Length;i++)
       gecici[i]=dizi.GetValue(i).ToString();
     return gecici;
  }
}
```

```
class AnaProgram
{
  static void Main()
  {
    string[] a={"2","5","7","9"};
    int[] b=Donustur.Inte(a);
    Console.WriteLine(b[1]+b[3]);
    int[] c={2,7,9,4};
    string[] d=Donustur.Stringe(c);
    Console.WriteLine(d[0]+d[3]);
  }
}
//farklı türleri kendiniz ekleyebilirsiniz.
```

Dizililerin metot ve özelliklerde kullanılması

```
using System;
class Donustur
{   public static int[] Inte(Array dizi)
  {
     int[] gecici=new int[dizi.Length];
       int j = 0;
         foreach (object i in dizi)
         {
            gecici[j] = Convert.ToInt32(i);
Console.WriteLine(gecici[j]);
            j++;
         }
     return gecici;
  }
.....
}
```

```
class AnaProgram
{
  static void Main()
  {
    string[] a={"2","5","7","9"};
    int[] b=Donustur.Inte(a);
    Console.WriteLine(b[1]+b[3]);
    int[] c={2,7,9,4};
    string[] d=Donustur.Stringe(c);
    Console.WriteLine(d[0]+d[3]);
  }
}
//farklı türleri kendiniz ekleyebilirsiniz.
```

Yapıcı Metotlar(Constructors)

- Bir nesnenin dinamik olarak yaratıldığı anda otomatik olarak çalıştırılan metotlar vardır. Bu metotlar sayesinde bir nesnenin üye elemanlarına ilk değerler verilebilir ya da gerekli ilk düzenlemeler yapılabilir. Bu metotlara yapıcı metotlar (constructors) denir.

- Yapıcı metot tanımlanmasa dahi her sınıfın varsayılan bir yapıcı metodu (default constructor) mutlaka bulunur.

- **Daha önceden bahsettiğimiz gibi bir nesne oluşturulduğunda sayısal değerler için 0, referanslar için null, bool türü için false atanır, bu atama bir yapıcı metot ile gerçekleştirilir.**

Yapıcı Metotlar(Constructors)

```
using System;
class Deneme
{
Deneme() { Console.WriteLine("Bu sınıf türünden bir nesne yaratıldı."); }
}
class AnaProgram
{
   static void Main()
   {
      Deneme a=new Deneme();
   }
}
```

Varsayılan Yapıcı Metot (Default Constructor)

- **Genel Özellikleri:**
- **Yapıcıların dönüş değeri yoktur, isimleri sınıf ile aynı isimdir. Varsayılan yapıcı metodun parametresi yoktur. Nesnenin üye elemanlarına varsayılan değerlerini atar.**
- **Yapıcı metotlar bir değer tutamaz ve normal metotlardan farklı olarak void anahtar sözcüğü de kullanılmaz.**
- **Yapıcı metotlar dışarıdan çağrıldığı için genelde public olarak tanımlanırlar. private olursa dışarıdan erişilemez**
- **Eğer kendi yapıcı metodumuzu tanımlarsak varsayılan yapıcı metotlar çalıştırılmaz.**
- **Yapıcı metotlar da aşırı yüklenebilir. Aşırı yükleme durumunda boş yapıcı metot mutlaka bulunmalıdır.**
  - **Örneğin public Deneme();, public Deneme(int a);, public Deneme(int a, int b);**

Varsayılan Yapıcı Metot (Default Constructor)

```
using System;

class Toplama
{
    public int X;
    public int Y;

    public int Islem()
    {
        return X + Y;
    }

    public void Yaz()
    {
        Console.WriteLine("X = {0}", X);
        Console.WriteLine("Y = {0}", Y);
    }
}

class Program
{
    static void Main()
    {
        Toplama t = new Toplama();
        t.Yaz();
    }
}
```

```
C:\Windows\system32\cmd.exe
X = 0
Y = 0
Devam etmek için bir tuşa basın . . .
```

```
using System;

class Toplama
{
    public int X;
    public int Y;

    public Toplama()
    {
        X = -1;
        Y = -1;
    }
    public int Islem()
    {
        return X + Y;
    }
    public void Yaz()
    {
        Console.WriteLine("X = {0}", X);
        Console.WriteLine("Y = {0}", Y);
    }
}
class Program
{
    static void Main()
    {
        Toplama t = new Toplama();
        t.Yaz();
    }
}
```

```
C:\Windows\system32\cmd.exe
X = -1
Y = -1
Devam etmek için bir tuşa basın . . .
```

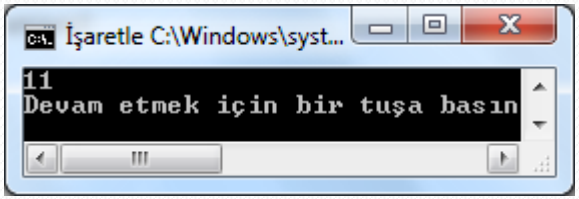
Yapıcı Metot (Default Constructor)

- Yapıcı metotlar aşırı yüklenebilir. Bu şekilde tanımlanan yapıcı metotlardan, imzası uygun olan seçilerek çalıştırılır.

```
using System;
 class Deneme
 {
   Deneme()   {     Console.WriteLine(0);   } // aşırı yüklemede mutlaka olmalı
   Deneme(int a)   {     Console.WriteLine(a);   }
   Deneme(int a,int b)   {     Console.WriteLine(a+b);   }
   Deneme(int a,int b,int c)   {     Console.WriteLine(a+b+c);   }
   static void Main()
   {
     Deneme a=new Deneme(5,6);
   }
 }
```

Yapıcı Metot (Default Constructor)

- Aşırı yükleme **this** anahtar sözcüğü kullanılarak da oluşturulabilir. Bu örnekte this anahtar sözcüğü sayesinde ikinci, üçüncü ve dördüncü yapıcı metotlar içeriğini aynı isimli ve üç parametre alan metottan alıyor. this anahtar sözcüğüyle kullanılan ikinci, üçüncü ve dördüncü yapıcı metotların yaptığı tek iş, birinci yapıcı metoda belirli parametreleri göndermek oluyor.
- **using** System;
- **class** Deneme
- { Deneme(**int** a,**int** b,**int** c) { Console.WriteLine(a+b+c); } Deneme():**this**(0,0,0) { }
- Deneme(**int** a):**this**(a,0,0) { }
- Deneme(**int** a,**int** b):**this**(a,b,0) { }
- **static void** Main() { Deneme a=new Deneme(5,6); }
- }
- **NOT:** this anahtar sözcüğü bu şekilde yalnızca yapıcı metotlarla kullanılabilir.



Yapıcı Metot (Default Constructor)

- Eğer bir sınıfta parametre alan bir veya daha fazla yapıcı metot varsa ve parametre almayan yapıcı metot yoksa -bu durumda varsayılan yapıcı metot oluşturulmayacağı için- Sinif nesne=new Sinif(); gibi bir satırla parametre vermeden nesne oluşturmaya çalışmak hatalıdır. Çünkü Sinif nesne=new Sinif(); satırı Sinif sınıfında parametre almayan bir yapıcı metot arar, bulamaz ve hata verir. Ancak boş yapıcı metot kullanılmayacaksa hata vermez. Örnek:
- **class** A
- { // public A() {} tanımlanması gerekir yoksa hata verir
- **public** A(**int** a){}
- }
- **class** B
- { A a=new A(); }
- Derleme anında hata verir.

Kopyalayıcı Yapıcı Metot (Copy Constructors)

- Yapıcı metot parametre olarak kendi sınıfından bir nesne alıyorsa kopyalayıcı metot adı verilir.

```
using System;

class Toplama
{
    public int X;
    public int Y;

    public Toplama(int x, int y)
    {
        X = x;
        Y = y;
    }

    public Toplama(Toplama T)
    {
        X = T.X;
        Y = T.Y;
    }

    public int Islem()
    {
        return X + Y;
    }

    public void Yaz()
    {
        Console.WriteLine("X = {0}", X);
        Console.WriteLine("Y = {0}", Y);
    }
}
```

```
class Program
{
    static void Main()
    {
        Toplama t = new Toplama(15, 22);
        t.Yaz();

        Toplama t2 = new Toplama(t);
        t2.Yaz();
    }
}
```

Yıkıcı (Destructors) Metotlar

- Nesnelere erişim mümkün olmadığı durumlarda bu nesnelerin heap'ten silinmesi gerekir, çünkü bu nesneler bellek bölgesi işgaline sebep olur. C++' ta bu işlemi kullanıcı kendi yapmak zorundadır.

- C#' ta bu işlem için Gereksiz Nesne Toplayıcısı (Garbage Collector) mekanizması mevcuttur. Bu mekanizma gereksiz nesnelerin tuttuğu referans bölgelerini iade eder, ancak bu nesnelerin faaliyet alanlarının ne zaman iade edileceği kesin olarak bilinmez.

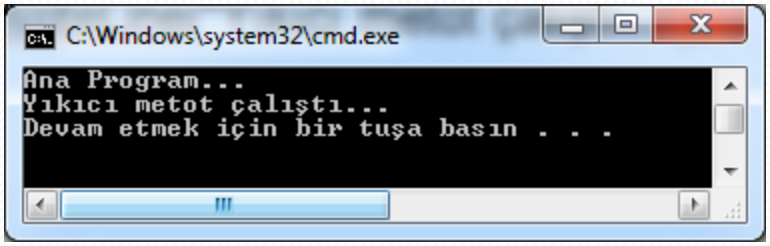
- Bu durumda yıkıcı metotlar (destructors) bellek iade işleminden hemen önce çalışarak bu belirsizliğin çözümünde yardımcı olurlar.

Yıkıcı (Destructors) Metotlar

- Yıkıcı metotlar bildirilirken sınıf isminin önüne “~” (tilda) işareti eklenir.
- Herhangi bir dönüş değeri ve parametresi yoktur.
- Erişim belirteci (public, private) de kullanılmaz.
- Bir sınıfın sadece bir tane yıkıcı metodu olabilir.

Yıkıcı (Destructors) Metotlar

```
using System;
class Yikici
{
    ~Yikici()
    {
        Console.WriteLine("Yıkıcı metot çalıştı...");
    }
}
class Program
{
    static void Main()
    {
        Yikici y = new Yikici();
        Console.WriteLine("Ana Program...");

    }
}
```

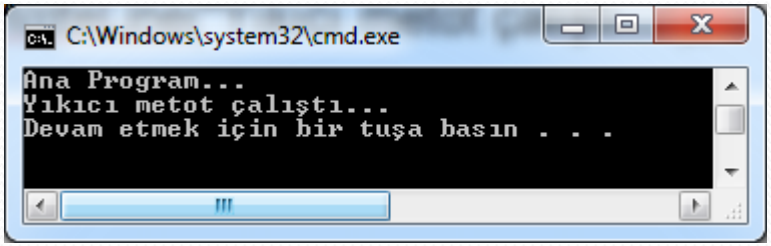


Yıkıcı (Destructors) Metotlar

- Örnek programda **y** nesnesi faaliyet alanını doldurmasına rağmen hemen yok edilmemiştir.
- Bir alt satırda bulunan Console.WriteLine() metodu çalışmış ardından gereksiz nesne toplayıcısı işleyerek nesneyi silmiştir.
- Nesne yok edilirken de yıkıcı metodu çalıştırılmıştır.
- Gereksiz nesne toplayıcı (Garbage Collector) programcının isteği dışında çalışmaktadır.
- System.GC sınıfı kullanılarak istenilen anda bu mekanizma çalıştırılabilir. Fakat bu tavsiye edilen bir kullanım değildir.

Yıkıcı (Destructors) Metotlar

```
using System;
class Yikici
{
    ~Yikici()
    { Console.WriteLine("Yıkıcı metot çalıştı..."); }
}
class Program
{
    static void Main()
    {
        Yikici y = new Yikici();
        GC.Collect();
        Console.WriteLine("Ana Program...");
    }
}
```



Dispose() metodu

- Oluşturulan bir nesnenin kullandığı kaynakların geri alınması için kullanılabilecek yöntemlerden bir diğeri de **Dispose()** metodudur.

- Türetmiş olduğumuz bir nesnenin hiçbir kaynak tarafından kullanılmadığı ana kadar nesneyi bellekten kaldırmaz tabi bu bazen çok uzun zaman alabilir dolayısı ile bellekte şişmeler olur ve ***OutOfMemoryException*** alırız.

- Garbage Collector'a kaynakları boşaltabilmesine yardımcı olmak için Dispose metodu devreye giriyor. Dispose metoduna ulaşmak için yazmış olduğumuz sınıfa IDisposable Interface(Arayüz)sini uygulamamız gerekmektedir.

- IDisposable Interface’si ile Dispose metodu gelir ve bu metod uygulandığı sınıftan türeyen nesnenin işi bittiği anda bellekten kaldırılmasını sağlar.
- Arayüz(Interface) kavramına daha sonra değinilecektir.
- Yıkıcı metot olsa dahi çalışmayacaktır.

Dispose() metodu

- Öğrenci Sınıfımızı aşağıdaki kod ile oluşturalım.

```
namespace Samples
{ class Ogrenci: IDisposable
  {  string _Adi, _SoyAdi, _DogumTarihi;

    public string Adi
    { get { return _Adi; }          set { _Adi = value; }  }

    public string SoyAdi
    { get { return _SoyAdi; }      set { _SoyAdi = value; } }

    public DateTime DogumTarihi
    { get { return _DogumTarihi; }  set { _DogumTarihi = value; } }
```

Dispose() metodu

```
public void Dispose()
{
 GC.SuppressFinalize(this);
/*Dispose Metodu çağırıldığında Garbage Collector'ün SuppressFinalize metoduna this diyerek bu sınıftan türetilmiş olan nesneyi sonlandırmaya, serbest bırakmaya zorluyoruz. Bu sayede bizim bu kod ile bizzat sertbest bıraktığımız kaynaklar ile destructor'umuz tekrar uğraşmak zorunda kalmıyor buda bize performas olarak geri dönüş sağlıyor.*/
 }
```

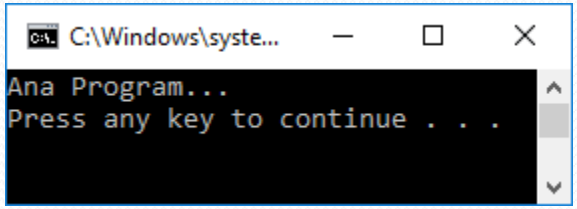
- Şimdi ogrenci sınıfından bir nesne türetelim ve onu dispose metodu ile bellekten kaldıralım.
- C# 4.0 sonra arayüz tanımı `class Ogrenci: IDisposable` olmasa da olur `class Ogrenci { }` ama Dispose () metodu olmalı.

Dispose() metodu

```
  public static void Main()
  {
    Ogrenci insOgrenci = new Ogrenci();
    insOgrenci.Adi = "onur";
    insOgrenci.SoyAdi = "yılmaz";

   /* Bilgilerini aldığımız öğrencimizi veri tabanımıza insert ettiğimizi var sayıyoruz ve ogrenci sınıfından türetmiş olduğumuz nesneyi dispose metodunu çağırarak bellekte yer işgal etmesini engelliyoruz.*/

   insOgrenci.Dispose();
        }
    }
}
```

Dispose() metodu

```
using System;
class Yikici    /* : IDisposable olmasada olur*/

{ ~Yikici()
  { Console.WriteLine("Yıkıcı metot çalıştı..."); }
  public void Dispose() {GC.SuppressFinalize(this); }
  }

  class Program
  {
   static void Main()
   {
    Yikici y = new Yikici();
    y.Dispose();
    /*Yıkıcı metot çalışmayacaktır*/
    Console.WriteLine("Ana Program...");
   }
}
```



Yıkıcı (Destructors) Metotlar

- C#'da yıkıcı metotlar genelde nesnenin kullandığı bellek alanını iade etmek için kullanılmaz.

- Daha çok nesne yok edilirken bir takım işlemlerin yapılması, özellikle sınıfın global üye değişkenlerinin değerlerinin işlenmesi ya da değiştirilmesi için kullanılır.

Statik Üye Elemanlar

- Daha önce de belirtildiği gibi metotlara sınıflar üzerinden erişilir. Bazı durumlarda da ise sınıf türünden nesneler oluşturulur ve bu nesneler üzerinden ilgili metoda erişebiliriz.

- Statik elemanlar bir sınıfın global düzeydeki elemanlarıdır. Yani statik üyeleri kullanmak için herhangi bir nesne tanımlamamıza gerek yoktur.

- Bir üye elemanın statik olduğunu bildirmek için bildirimden önce **static** anahtar sözcüğü eklenir.

- Bir sınıf nesnesi oluşturulduğunda static üye elemanlar için bellekte ayrı bir yer ayrılmaz. (derleme zamanında zaten static bölümünde yerleri ayrılmıştır)

Statik Metotlar

- Nesne ile doğrudan iş yapmayan metotlar statik olarak tanımlanabilir.
- Statik tanımlanan metotlara **SınıfAdı.Metot** şeklinde erişilir.
- Statik metotlara nesne üzerinden erişmek mümkün değildir.

Statik Metotlar

```
using System;

class AritmetikIslem
{
    public static int Topla(params int[] dizi)
    {
        int toplam = 0;

        foreach (int eleman in dizi) toplam += eleman;

        return toplam;
    }
}
class Program
{
    static void Main()
    {
        Console.WriteLine(AritmetikIslem.Topla(1, 5, 9, 15));
    }
}
```

```
using System;

class AritmetikIslem
{
    public static int Topla(params int[] dizi)
    {
        int toplam = 0;

        foreach (int eleman in dizi) toplam += eleman;

        return toplam;
    }
}
class Program
{
    static void Main()
    {
        AritmetikIslem islem = new AritmetikIslem();
        Console.WriteLine(islem.Topla(1, 5, 9, 15));
    }
}
```

Statik Metotlar

- Main() metodu da herhangi bir nesneye ihtiyaç duymadan çalışabilmesi için static olarak tanımlanmaktadır. Eğer static olarak tanımlanmasaydı metodun çalışabilmesi için içinde bulunduğu sınıf türünden bir nesneye ihtiyaç olacaktı. Bu durumda da Main metodu işlevi ile çelişecekti. Çünkü Main metodu programımızın çalışmaya başladığı yerdir.

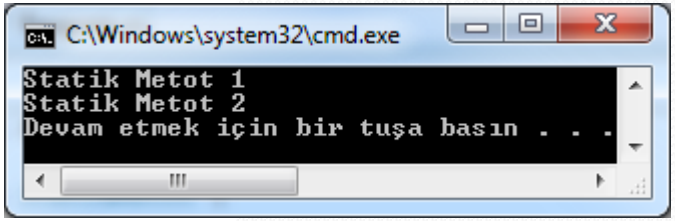
- static anahtar sözcüğü erişim belirleyiciden önce ya da sonra yazılabilir.

- static tanımlanmış metotlar içinden diğer static metotlar çağrılabilir, normal metotlar çağrılamaz. Normal metotlar nesne gerektirdiği için nesne adreslerine ihtiyaç duyarlar (bu adresler gizli şekilde ya da this ile aktarılır).

- static metotlar sınıfın global metotlarıdır ve referansları yoktur. Bu yüzden static bir metot içinden normal bir metot çağırmak geçersizdir.

Statik Metotlar

```
using System;

class StatikMetot
{
    public static void SMetot1()
    {
        Console.WriteLine("Statik Metot 1");
    }

    public static void SMetot2()
    {
        SMetot1();
        Console.WriteLine("Statik Metot 2");
    }
}
class Program
{
    static void Main()
    {
        StatikMetot.SMetot2();
    }
}
```



Statik Metotlar

```
using System;

class StatikMetot
{
    public void SMetot1()
    {
        Console.WriteLine("Statik Metot 1");
    }

    public static void SMetot2()
    {
        SMetot1();
        Console.WriteLine("Statik Metot 2");
    }
}
class Program
{
    static void Main()
    {
        StatikMetot.SMetot2();
    }
}
```

- Örneklerden görüldüğü gibi statik olmayan elemanlara ulaşabilmek için bir nesne olmalıdır. Eğer bir nesne var ise statik olmayan elemanlara erişilebilir.
- StaticMetot x=new StaticMetot()
- x.SMetot1(); şeklinde çağrılabilir.

Statik Değişkenler

- Herhangi bir nesne ile statik değişkenlere ulaşmamız mümkün değildir. Statik olarak tanımlanmış olan değişkenlere ancak sınıfın ismi yardımıyla ulaşılabilir.
- **SınıfAdı.değişken** şeklinde erişilir.
- Statik değişkenler sınıf içeresinde global özellikler tanımlamak için kullanılırlar.
- Bu değişkenler tanımlandıklarında varsayılan değere atanırlar.
- Statik üyelere ancak statik bir metot içerisinden erişilebilir.

Statik Değişkenler

```
using System;
namespace ConsoleApplication2
{
    class Statik
    {
        public static int b = 10;
        public static int deneme()
        {
            int a = 5; b = a;
            return b;
        }
    }
        class Program
        {
            static void Main(string[] args)
            {
                Statik b = new Statik();// gereksizdir ama hata vermez.
                Console.WriteLine(b.deneme());//Hata verir, Statik metotlara nesne üzerinden
                                              //erişmek mümkün değildir.
                Console.WriteLine(Statik.deneme());// deneme metoduna erişim var. 5 değerini yazar
                Console.WriteLine(Statik.b);// b değişkenine erişim var. 5 değerini yazar
                Console.WriteLine(Statik.a);// Hata verir, static veya statik olmayan metod
                                            //içerisindeki değerlere erişim yoktur.
               Console.ReadLine();
            }
        }
    }
```

Statik Yapıcı Metotlar

- Normal metotlar gibi yapıcı metotlar da static olabilirler
- Statik yapıcı metotlar bir sınıfın statik değişkenleri ile ilgili işlemler yapmada kullanılırlar.
- Bir nesne ilk defa yaratıldığında statik üye değişkenini değiştirmek için genellikle statik yapıcı metotlar kullanılır.
- Statik yapıcı metotlar ilk nesne tanımlandığında çalıştırılır.

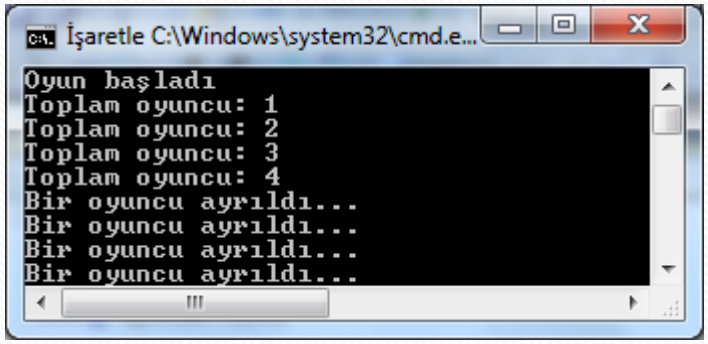
Statik Yapıcı Metotlar

```
using System;
class Deneme
{  static Deneme()
   {      Console.WriteLine("Static metot çağrıldı.");    }
   Deneme()
   {      Console.WriteLine("Static olmayan metot çağrıldı.");    }
   static void Main()
   {  Deneme a=new Deneme();
      Deneme b=new Deneme();
   }
}
```

- Bu program ekrana şunları yazacaktır:
- Static metot çağrıldı.
- Static olmayan metot çağrıldı.
- Static olmayan metot çağrıldı.
- Gördüğünüz gibi bir sınıf türünden bir nesne yaratıldığında önce static metot sonra (varsa) static olmayan metot çalıştırılıyor. Sonraki nesne yaratımlarında ise static metot çalıştırılmıyor.

Statik Yapıcı Metotlar

```
using System;
class Oyuncu
{   static int Toplam;

    Oyuncu()
    { Toplam++; Console.WriteLine("Toplam oyuncu: " + Toplam); }

    static Oyuncu()
    { Console.WriteLine("Oyun başladı"); }
    ~Oyuncu()
    {
        Console.WriteLine("Bir oyuncu ayrıldı..."); Toplam--;
    }
    static void Main()
    {
        Oyuncu ahmet = new Oyuncu();
        Oyuncu osman = new Oyuncu();
        Oyuncu ayse = new Oyuncu();
        Oyuncu mehmet = new Oyuncu();
    }
}
```



Static Sınıflar

- Bir sınıf statik olarak bildirildiğinde o sınıf türünden bir nesne yaratılamaz. Statik metotlar C# 2.0'dan itibaren dile eklenmiştir.
- **Statik sınıflar içerisinde static olmayan metot veya özellik tanımlanamaz.**
- **Static sınıfların yapıcı metotları olmaz. Dolayısıyla yapıcı metot tanımlamaya çalışırsak derleyici hata verir.**
- Daha sonra bahsedilecek olan nesne konusundaki türeme başlığındatekrar bahsedeceğimiz üzere; **static sınıflar türetmeyi desteklemez**.

Static Olmak, Non-Static Olmak

- **Bir sınıfa ait her nesne için özel bir iş söz konusu ise, üye elamanları veya metot non-static tanımlanmalıdır, Eğer yapılan iş genel bir iş ise, üye elamanları veya metot static olarak tanımlanmalıdır.**
- **Örnek:**

```
class Urun
    {
        public int Id;
        public string Ad;
        public double Fiyat;
        public double KdvOran;
    }
```

Static Olmak, Non-Static Olmak

```
class Program
    {
        static void Main(string[] args)
        {
            Urun urun1 = new Urun();
            urun1.Id = 1;
            urun1.Ad = "Monitör";
            urun1.Fiyat = 200;
            urun1.KdvOran = 0.18;
            Urun urun2 = new Urun();
            urun2.Id = 2;
            urun2.Ad = "Klavye";
            urun2.Fiyat = 80;
            urun2.KdvOran = 0.18;
        }
    }
```

KdvOran her iki nesne içinde ortak olduğuna göre yeniden tanımlamaya gerek yoktur. Bu yüzden static tanımlanması hafıza ve hız açısından daha önemlidir.

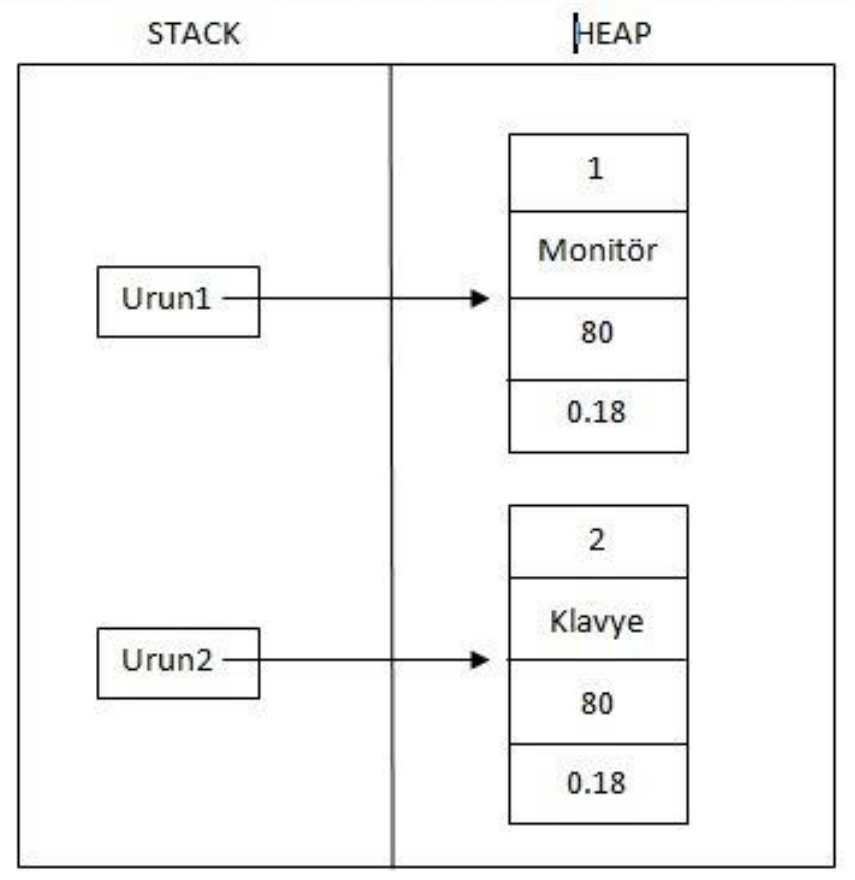
Static Olmak, Non-Static Olmak

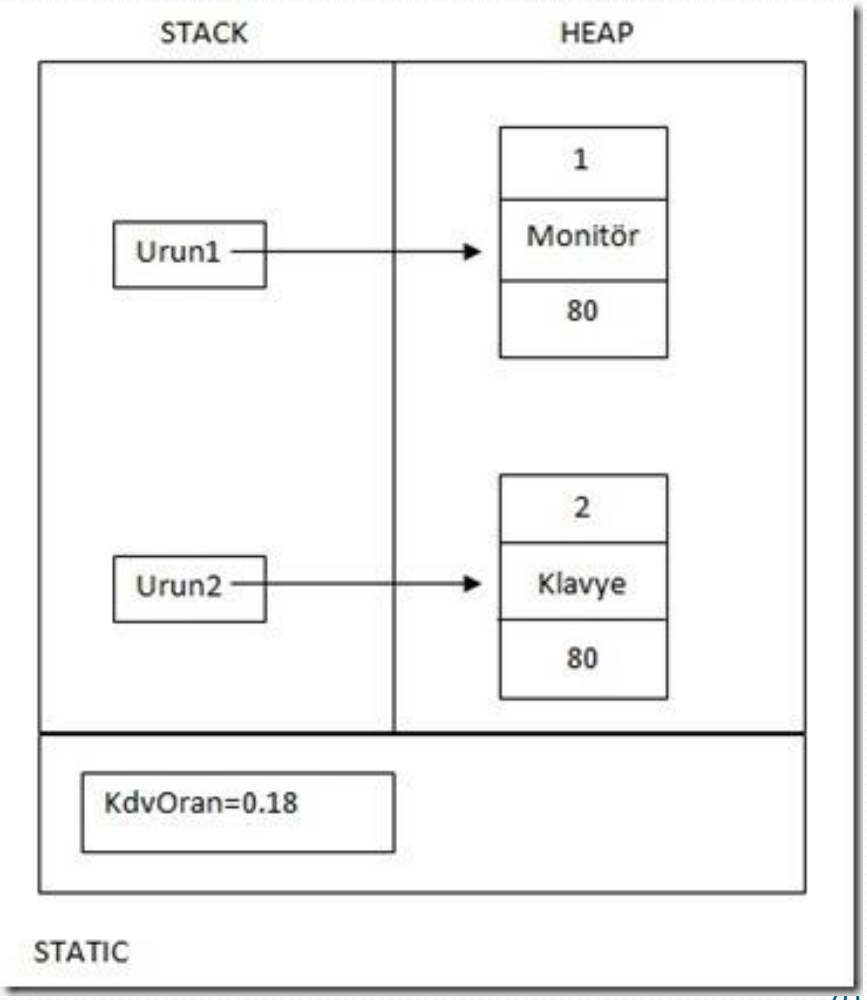
```
class Urun
    {
        public int Id;
        public string Ad;
        public double Fiyat;
        public static double KdvOran;
    }
```

```
class Program
    {
        static void Main(string[] args)
        {
            Urun urun1 = new Urun();
            urun1.Id = 1;
            urun1.Ad = "Monitör";
            urun1.Fiyat = 200;

            Urun urun2 = new Urun();
            urun2.Id = 2;
            urun2.Ad = "Klavye";
            urun2.Fiyat = 80;

            Urun.KdvOran = 0.18;
        }
    }
```

KdvOran static tanımlandığından bir kez kullanıldı. Eğer 1000 ürün olsaydı kullanımın ne kadar önemli olduğu daha net anlaşılır.

Static Olmak, Non-Static Olmak





const ve readonly Elemanlar

- Değişkenler konusunda gördüğümüz **const** anahtar sözcüğü özelliklerle de kullanılabilir.

- const olarak tanımlanan özellikler aynı zamanda static'tir. Dolayısıyla const özellikleri tekrar static anahtar sözcüğüyle belirtmek hatalıdır. const özellikler static özelliklerin taşıdığı tüm özellikleri taşır.

- const anahtar sözcüğü yalnızca object dışındaki özelliklerle (ve değişkenlerle) kullanılabilir. Dizilerle vs. kullanılamaz. const özellikler tanımlanır tanımlanmaz bir ilk değer verilmelidir.

- Sabit tanımlanmış üyeler de tıpkı statik tanımlılar gibi sınıfın genel elemanlarıdır ve nesne üzerinden erişilemez.

- Sabitlere erişmek için SınıfAdı.Sabit şeklinde kullanılırlar.

const ve readonly Elemanlar

```
using System;

class Matematik
{
    public const double PI = 3.14;
}

class Program
{
    static void Main()
    {
        Console.WriteLine(Matematik.PI);

        Matematik m = new Matematik();

        Console.WriteLine(m.PI);
    }
}
```

const ve readonly Elemanlar

- Referans tiplerin (object, diziler, nesneler) değerlerinin çalışma zamanında ayarlanmasından dolayı, referans tipler sabit olamaz.

- string veriler bu kuralın dışındadırlar. Yani const ile tanımlanabilirler.

- Referans tipleri sabit olarak tanımlamak için "readonly" anahtar sözcüğü kullanılır.

const ve readonly Elemanlar

```
using System;
class Sinif
{
   readonly int[] a={1,5,8};
   readonly object b=5;
   static void Main()
  {
   Sinif n=new Sinif();
   Console.WriteLine(n.a[0]);
   Console.WriteLine(n.b);
  }
}
```

const ve readonly Elemanlar

- Statik ifadelere de readonly eklenirse referans tipleri const gibi tanımlanmış olur.
- Static readonly ve const olarak tanımlanmış değişkenlerin tek farkı ilk değer verilme zamanında ortaya çıkar.
- const değişkenlerine derleme zamanında static readonly değişkenlerine ise çalışma zamanında ilk değer verilir.

const ve readonly Elemanlar

- Şimdi bir de salt okunur nesne oluşturalım:
- using System;
- class Sinif
- {
- int a=5;
- static readonly Sinif nesne=new Sinif();
- static void Main()
- { Console.WriteLine(nesne.a); }
- }
- //Şimdilik nesnelerin salt okunur olması saçma gelebilir. Çünkü salt okunur nesnelerle ulaştığımız özellikleri değiştirebiliriz. Salt okunur yaptığımız nesnenin kendisidir.

const ve readonly Elemanlar

- readonly ile ilgili önemli bilgiler:

- **readonly** anahtar sözcüğü bir metot bloğunun içinde kullanılamaz.
- **readonly** anahtar sözcüğü bir nesne için kullanılacaksa static sözcüğü de kullanılmalıdır.
- **readonly** ve static sözcüklerinin sırası önemli değildir.
- **readonly** bir dizi ya da object türü ayrıca static olarak belirtilebilir.
- **readonly** anahtar sözcüğ ü const' un kullanılabildiği int gibi türlerle de kullanılabilir.
- **readonly'nin** constun aksine özellikleri staticleştirme özelliği yoktur.
- **static** sözcüğü de readonly gibi bir metot içinde kullanılamaz.

Visual Studio.Net -C#

6. HAFTA

Operatör Overloading, İndeksleyiciler, Yapılar, Enum sabitleri

Örnek Uygulamalar

Örnek

- Programlamadaki metot kavramı aslında matematikteki fonksiyonlar konusunun aynısıdır. Matematikteki fonksiyonlar konusunda öğrendiğiniz bütün kuralları metotlarda uygulayabilirsiniz.
- Örneğin: Bir fonksiyona parametre olarak o fonksiyonun tersini verirsek sonuç x çıkacaktır. Örneğin f(x)=2x+5 olsun. f(x) fonksiyonunun tersi $f^{-1}(x)=(x-5)/2$'dir. Şimdi f(x) fonksiyonuna parametre olarak (x-5)/2 verirsek sonuç x olacaktır. Yani $f(x) \circ f^{-1}(x)=x$'tir. Şimdi bu kuralı bir metotla doğrulayalım.
- `using System;`
- `class Metotlar {`
- `static float Fonksiyon(float x) { return 2*x+5; }`
- `static float TersFonksiyon(float x) { return (x-5)/2; }`
- `static void Main() { float x=10;`
- `Console.WriteLine(Fonksiyon(x));`
- `Console.WriteLine(TersFonksiyon(x));`
- `Console.WriteLine(Fonksiyon(TersFonksiyon(x))); } }`

- Bu program ekrana sırasıyla 25, 2.5 ve 10 yazacaktır.

Sınıflar
Örnek

```
using System;

class Ogrenci
{
   public ulong OgrenciNo;
   public string Ad;
   public string Soyad;
   public string Bolum;
   public byte Sinif;
}
class Program
{
    static void Main()
    {
        Ogrenci ogr = new Ogrenci();

        Console.WriteLine(ogr.OgrenciNo);
        Console.WriteLine(ogr.Ad);
        Console.WriteLine(ogr.Soyad);
        Console.WriteLine(ogr.Bolum);
        Console.WriteLine(ogr.Sinif);
    }
}
```

Birden Fazla Sınıf Nesnesi Tanımlama Örnek

```
using System;
class KrediHesabi
{
    public ulong HesapNo;
}
class AnaSinif
{
   static void Main()
   {
      KrediHesabi hesap1=new KrediHesabi();
      KrediHesabi hesap2=new KrediHesabi();
      hesap1.HesapNo=3456;
      hesap2.HesapNo=1111;
     Console.WriteLine(hesap1.HesapNo);
     Console.WriteLine(hesap2.HesapNo);
   }
}
```

Sınıf Bildirimi Örnek

- **Örnek:**
- using System;
- class SinifIsmi
- {public int ozellik1=55; // başlangıç değerleri verildi.
- public string ozellik2="deneme";
- public float ozellik3=123.78f;
- public int metot1(int a,int b)
- { return a+b; }
- public void metot2(string a)
- { Console.WriteLine(a); }
- }
- class EsasSinif {
- static void Main() {
- SinifIsmi nesne=new SinifIsmi(); Console.WriteLine(nesne.ozellik1); Console.WriteLine(nesne.ozellik2); Console.WriteLine(nesne.ozellik3); Console.WriteLine(nesne.metot1(2,5)); nesne.metot2("deneme"); } }

Örnek

Bu program bir metoda girilen iki parametreyi ve bunların çarpımını ekrana yazar. Eğer bir dikdörtgen söz konusu olduğunu düşünürsek dikdörtgenin enini, boyunu ve alanını ekrana yazar.

```
using System;
class Dortgen {
public int En;
public int Boy;
public int Alan() {
int Alan=En*Boy; return Alan; }
public void EnBoyBelirle (int en,int boy) {
En=en; Boy=boy; }
public void Yaz() {
Console.WriteLine("****************");
Console.WriteLine("En:{0,5}",En);
Console.WriteLine("Boy:{0,5}",Boy);
Console.WriteLine("Alan:{0,5}",Alan());
Console.WriteLine("****************"); } }
class AnaSinif
{ static void Main()
{ Dortgen d1=new Dortgen();
d1.EnBoyBelirle(20,50);
d1.Yaz();
Dortgen d2=new Dortgen();
d2.EnBoyBelirle(25,12); d2.Yaz();
}
}
```

Örnek

Programdaki hatanın nedenini bulunuz

```
using System;
class Dortgen {
public int En=20; public int Boy=5;
public int Alan()
{ int Alan=En*Boy; return Alan; }
static void Main()
{Console.WriteLine("***************")
; Console.WriteLine("En:{0,5}",En);
Console.WriteLine("Boy:{0,5}",Boy);
Console.WriteLine("Alan:{0,5}",Alan());
Console.WriteLine("***************");
} }
```

Örnek

Programdaki hatanın nedenini bulunuz

```
using System;
class Dortgen {
public int En=20; public int Boy=5;
public int Alan()
{ int Alan=En*Boy; return Alan; }

static void Main()
{Console.WriteLine("***************")
; Console.WriteLine("En:{0,5}",En);
Console.WriteLine("Boy:{0,5}",Boy);
Console.WriteLine("Alan:{0,5}",Alan());
Console.WriteLine("***************");
} }
```

- Cevap :
- Burada En ve Boy özellikleriyle Alan() metodu tek başına kullanılamaz.
- Önce Dortgen sınıfı türünden bir nesne yaratıp bu nesne üzerinden bu özellik ve metotlara erişilmelidir.
- Eğer özellikler ve metod static olsaydı direk kullanılablirdi.

Örnekler

Programdaki hatanın nedenini bulunuz

```
using System;
class Dortgen
{
   public int En = 30; public int Boy = 23;
   public int Alan()
   { int Alan = En * Boy; return Alan; }
   static void Main()
   { yaz d1 = new yaz(); d1.Yaz(); }
}
class yaz
{
   public void Yaz()
   {
      Console.WriteLine("En:{0,5}", Dortgen.En);
      Console.WriteLine("Boy:{0,5}", Dortgen.Boy);
      Console.WriteLine("Alan:{0,5}", Dortgen.Alan());
   }
}
```

Örnekler

Programdaki hatanın nedenini bulunuz

```
using System;
class Dortgen
{
   public int En = 30; public int Boy = 23;
   public int Alan()
   { int Alan = En * Boy; return Alan; }
   static void Main()
   { yaz d1 = new yaz(); d1.Yaz(); }
}
class yaz
{
   public void Yaz()
   {
      Console.WriteLine("En:{0,5}", Dortgen.En);
      Console.WriteLine("Boy:{0,5}", Dortgen.Boy);
      Console.WriteLine("Alan:{0,5}", Dortgen.Alan());
   }
}
```

- Çünkü, static olmayan metotların gövdesinde <u>sadece aynı sınıftaki</u> özellik ve metotlar static olsun olmasın direkt olarak kullanılabilir.
- public static int En=30;
- public static int Boy=23;
- public static int Alan() { … }
- yazılmalıydı

- Static metotların gövdesinde ise <u>aynı sınıftaki</u> özellik ve metotlar static olmadan direk olarak kullanılamaz. Değikenler ya static olacak yada sınıfa erişim için bir nesne oluşturulacak.

Örnekler

Çıktılarını bulunuz

```
using System;
class Dortgen
{
   int En;
   int Boy;
   static void Main()
   {
      int En=50;
      int Boy=100;
      Console.WriteLine(En+"\n"+Boy);
   }
}
```

```
using System;
class Dortgen
{
   static int En=8;
   static int Boy=3;
   static void Main()
   {
      int En=50;
      int Boy=100;
      Console.WriteLine(En+"\n"+Boy);
   }
}
```

Örnekler

Çıktılarını bulunuz

```
using System;
class Dortgen
{
  static int En=8;
  static int Boy=3;
  static void Main()
  {
    int Boy=100;
    Console.WriteLine(En+"\n"+Boy);
  }
}
```

```
using System;
class Dortgen
{   public  int En=8;
     int Boy=3;
    static void Main()
   {
   Console.WriteLine(En+"\n"+Boy);
   }
}
// hata verir. Çözüm yollarını söyleyiniz.
```

Static Üye konusunda cevaplarınızı gözden geçiriniz.

- 1.yol :Main başındaki static ifadesi kaldırılır(Program yine hata verir. Main mutlaka static olacak
- 2.yol:Değişkenler static yapılır
- 3.yol: sınıf için nesne oluşturularak değişkenlere erişilir.

Örnek

```
// 1 ‘den 100 e kadar sayıların toplamı
using System;
        class Topla
        {          public int tpl;

            public Topla(int toplam)
            {    tpl = toplam;      }

            public int Toplama
            {   get { return tpl; } // değeri görüntüleme
                set { tpl= value+tpl; } // yeni değeri depola
             }

        static void Main(string[] args)
    {     Topla ekle = new Topla(0);
          Console.WriteLine("Başlangıç değeri->" + ekle.Toplama);
           for (int i = 1; i < 100; i++)              ekle.Toplama = i;
           Console.WriteLine("Sayıların toplamı" + ekle.Toplama);
            }
       }
```

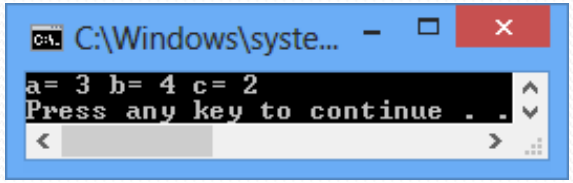
Yapıcı Metot (Default Constructor)-Örnek

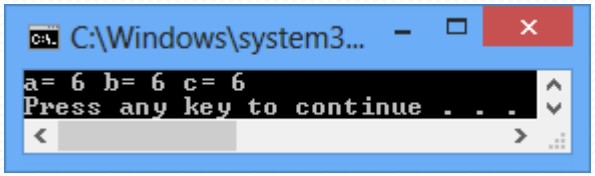
```
using System;

class Toplama
{
    public int X;
    public int Y;

    public Toplama(int x, int y)
    {
        X = x;
        Y = y;
    }

    public Toplama():this(-1,-1)
    {
    }

    public Toplama(int x):this(x, 1)
    {
    }

    public int Islem()
    {
        return X + Y;
    }

    public void Yaz()
    {
        Console.WriteLine("X = {0}", X);
        Console.WriteLine("Y = {0}", Y);
    }
}
```

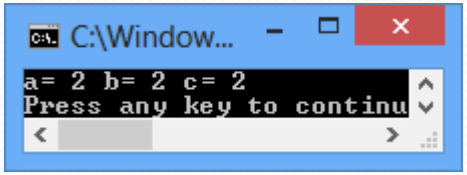
```
class Program
{
    static void Main()
    {
        Toplama t = new Toplama(5, 6);
        t.Yaz();
        Toplama y = new Toplama();
        y.Yaz();
        Toplama u = new Toplama(7);
        u.Yaz();
    }
}
```

Örnek: Random

```
class Program
{  // Random rnd = new Random(); //kullanılsaydı yine farklı değer üretirdi.
    public  int Zar()
    { Random rnd = new Random();     return rnd.Next(1,7); }
    public static void metotlar(int a, int b, int c)
    { Console.WriteLine("a= " + a + " b= " + b + " c= " + c); }
    static void Main(string[] args)
    {  Program a = new Program();
       metotlar(a.Zar(), a.Zar(), a.Zar());
    }
}
```







- Random, arka arkaya varsayılan yapıcı metot çağrısı kullanılarak oluşturulan nesneler aynı varsayılan değerler olacaktır (sistem saati aynı değer olduğundan) ve bu nedenle, aynı rasgele sayılar kümesi oluşturacaktır. Kümeden seçilen değerler aynı değer olacaktır.
- /*static void Main(string[] args)
- { Random rnd = new Random(); Program a = new Program();
- metotlar(rnd.Next(1,7), rnd.Next(1,7), rnd.Next(1,7)); } şeklinde kullansaydık her defasında farklı değer üretirdi.

Visual Studio.Net -C#
6. HAFTA
Operatör Overloading, İndeksleyiciler, Yapılar, Enum sabitleri

Operatör Aşırı Yükleme (Operator OverLoading)

- Nesneye yönelik programlama mantığında yazılan sınıflar için operatörlere yeni anlamlar yüklenebilir. Bu işlem operatör metotları ile gerçekleştirilir.

- **Operatör metotları**, bir nesnenin ifadeler içinde operatörle kullanıldığında davranışını belirler. Operatör metotları klasik metotlar gibidir.

- Temel veri türleri için operatörler zaten belirlenmiştir. Dolayısıyla bunlara söz konusu olması mümkün değildir. Fakat kendi tasarladığımız sınıflar için operatörlere yeni anlamlar yüklenebilir.

Operatör Metotları Tanımlanırken Uyulması Gereken Kurallar

- 1- Operatör metotları **static** olarak tanımlanmalıdır.
- 2- Operatör metotları isimlerinde "**operator**" anahtar sözcüğü kullanılmalıdır. (**operator+**, **operator*** gibi)
- 3- Bütün operatör metotları tek ya da iki parametre almalıdır.
- 4- Klasik metotlar gibi, operatör metotları da aşırı yüklenebilir.
- 5- Operatör metotların da parametrelerden biri mutlaka sınıf türünden olmalıdır.
- 6- Operatör metotlarında ref ve out anahtar sözcükleri kullanılmamalıdır.

Operatör Aşırı Yükleme (Operator OverLoading)

- Tanımlama şekli;
  - public static dönüş-tipi operator op (parametre-tipi parametre)

- Dönüş tipi: Herhangi bir tip olabileceği gibi operatörün aşırı yüklemekte olduğu sınıf ile aynı tipte olabilir. Genellikle tercih edilen sınıf ile aynı tipte olmasıdır.
- Parametre-tipi : ref ve out anahtar kelimeleri hariç dönüş tipi kullanılabilir. Sınıf tipinde de olabilir.
- Op: Aşırı yüklenecek operatör (**+**,**/**,*****,**-**).

Operatör Aşırı Yükleme (Operator OverLoading)

- **Örnek**

- Karmaşık sayılara ait olmak üzere tanımlanan KarmasikSayi sınıfından çeşitli karmaşık sayı nesneleri üretilecektir ve bu nesneler üzerinde aritmetiksel ve mantıksal işlemler yapan çeşitli operatör metotları tanımlanacak ve bunlar aşırı yüklenecektir.

Karmaşık Sayı sınıfının oluşturulması

```
using System;

class KarmasikSayi
{
    private double mGercek;
    private double mSanal;

    public double Gercek
    {
        get { return mGercek; }
        set { mGercek = value; }
    }

    public double Sanal
    {
        get { return mSanal; }
        set { mSanal = value; }
    }

    public KarmasikSayi(double x, double y)
    {
        mGercek = x;
        mSanal = y;
    }

    public KarmasikSayi()
    {
        mGercek = 0;
        mSanal = 0;
    }
```

```
    public KarmasikSayi(KarmasikSayi k)
    {
        mGercek = k.mGercek;
        mSanal = k.mSanal;
    }

    public void Yaz()
    {
        if (mSanal > 0)
            Console.WriteLine("{0}+{1}i",
                                        mGercek,
    mSanal);
        else
            Console.WriteLine("{0}-{1}i",
                                        mGercek,
    -mSanal);
    }
}

class Program
{
    public static void Main()
    {
        KarmasikSayi k = new KarmasikSayi(-5,-6);
        k.Yaz();
    }
}
```

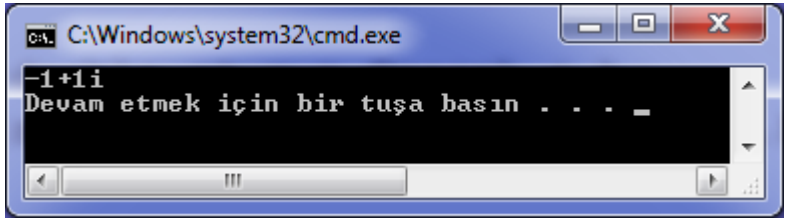
Aritmetiksel Operatörlerin Aşırı Yüklenmesi

- Dört işlem operatörleri (+, -, *, /) herhangi bir koşul olmaksızın aşırı yüklenebilirler.
- Tanımlanacak olan operator+ metodu karmaşık sayılar üzerinde toplamı işlemi yaparak ve sonucu yeni bir kompleks sayı nesnesi olarak döndürmektedir.

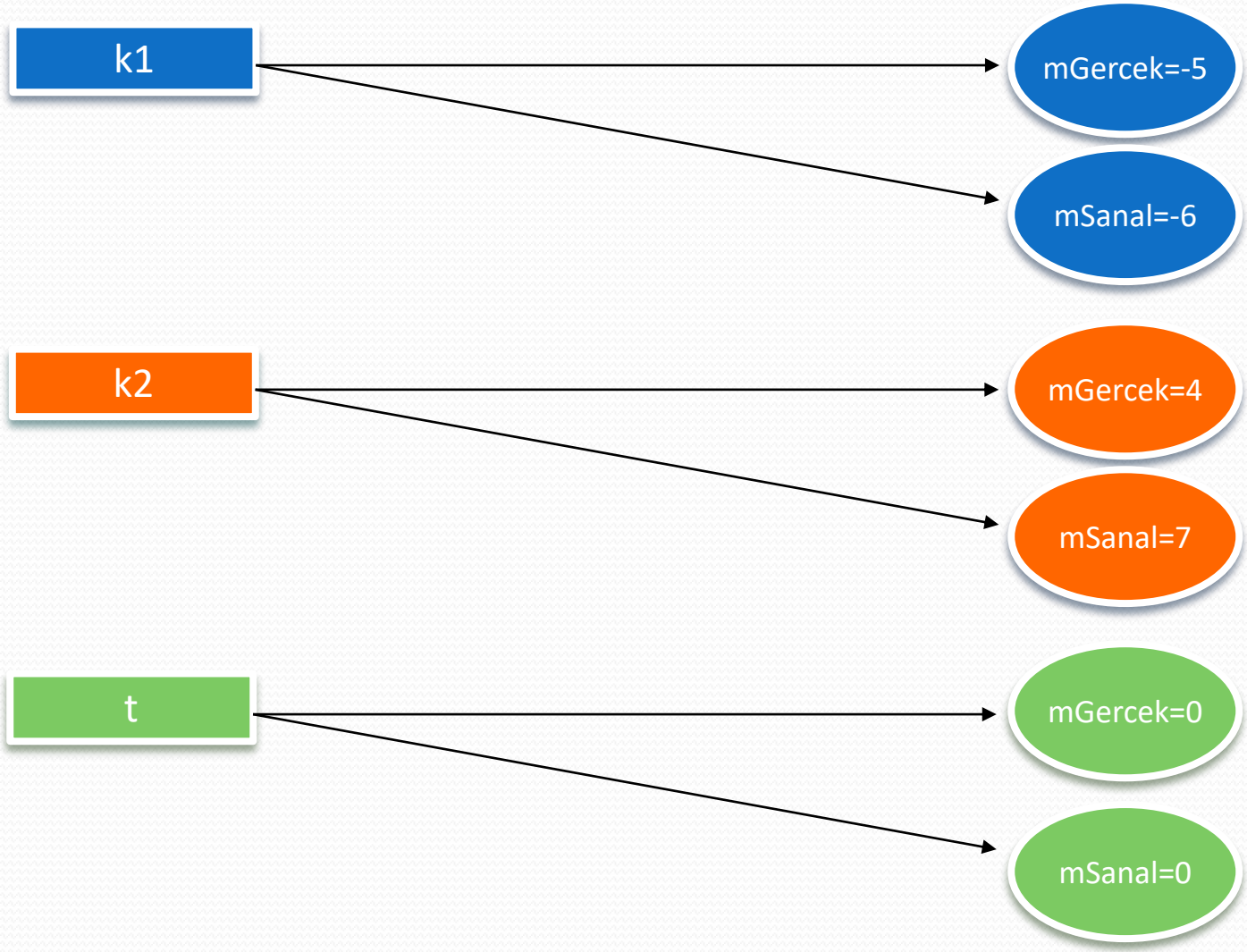
operator +

```
public static KarmasikSayi operator +(KarmasikSayi a,
  KarmasikSayi b)
    {
        double gt = a.Gercek + b.Gercek;
        double st = a.Sanal + b.Sanal;
        return new KarmasikSayi(gt, st);
    }
```

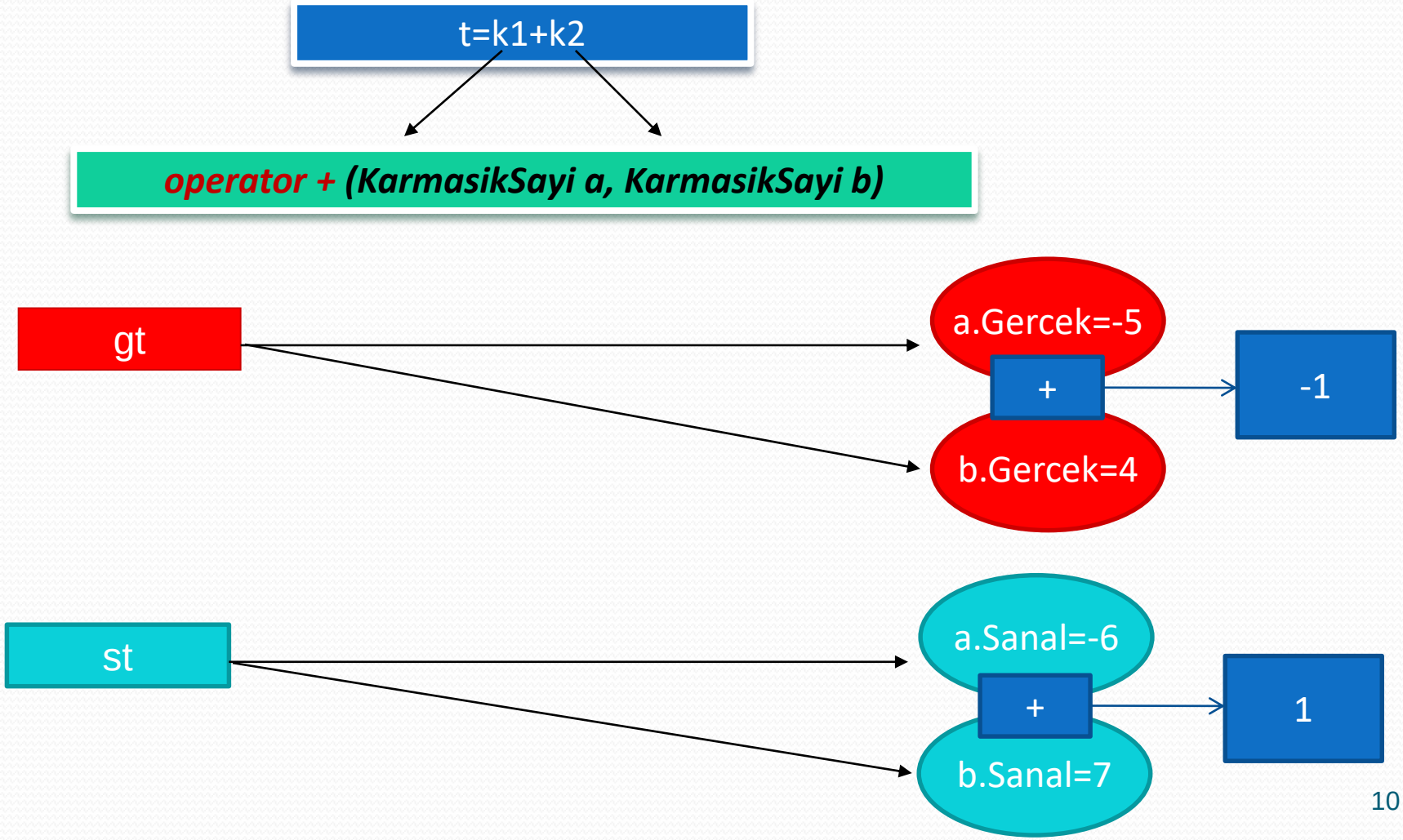
```
class Program
{
    public static void Main()
    {
        KarmasikSayi k1 = new KarmasikSayi(-5,-6);
        KarmasikSayi k2 = new KarmasikSayi(4, 7);
        KarmasikSayi t = k1 + k2;
        t.Yaz();
    }
}
```



Aritmetiksel Operatörlerin Aşırı Yüklenmesi

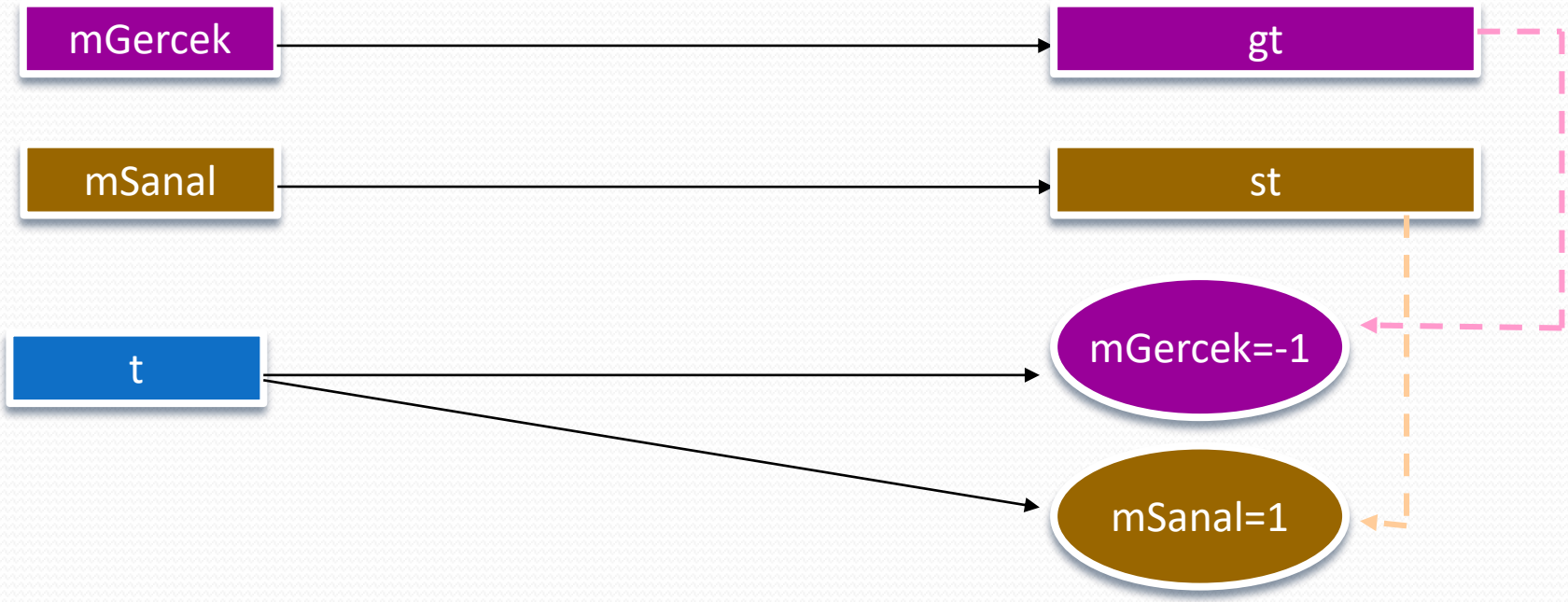


Aritmetiksel Operatörlerin Aşırı Yüklenmesi



Aritmetiksel Operatörlerin Aşırı Yüklenmesi

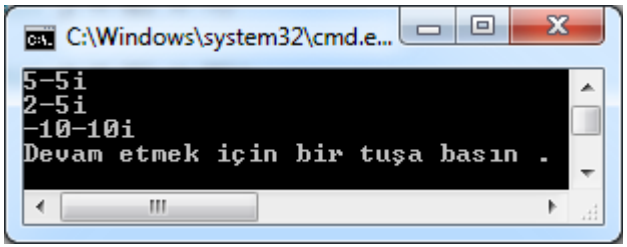
```
return KarmasikSayi ( gt, st );
```



Aritmetiksel Operatörlerin Aşırı Yüklenmesi

```
public static KarmasikSayi operator +(KarmasikSayi a, double b)
    {
        double gt = a.Gercek + b;
        return new KarmasikSayi(gt, a.Sanal);
    }

    public static KarmasikSayi operator +(double b, KarmasikSayi a)
    {
        return a + b;   // tekrar üsteki metot çağrılıyor
    }
```

```
class Program
{
    public static void Main()
    {
        KarmasikSayi k1 = new KarmasikSayi(5,-5);
        KarmasikSayi t = 10 + k1;
        t.Yaz();
        KarmasikSayi y = k1 + 7;
        y.Yaz();
        KarmasikSayi z = k1 + k1;
        z.Yaz();
    }
}
```



Aritmetiksel Operatörlerin Aşırı Yüklenmesi

```
using System;
class program
{   public int x, y;
    public program (int dd, int ff)     {   x = dd;   y=ff; }

    public  void yaz ()    { Console.WriteLine(x + " " + y); }

    public static program operator +(program a, program b )
    {   int c = a.x + a.y; int z = b.x + b.y;
        return new program(c, z);
    }
}
class Sinif
{    static void Main()
    {    program a = new program(5,8);   program b = new program(7,9);
         program m = new program(6,2);
         program c = a + b + m;      c.yaz();
    }
}
```

Aritmetiksel Operatörlerin Aşırı Yüklenmesi

- Toplama operator metoduna benzer şekilde çıkarma operatör metodu da kolaylıkla tanımlanabilir.

```
public static KarmasikSayi operator -(KarmasikSayi a,
  KarmasikSayi b)
    {
        double gf = a.Gercek - b.Gercek;
        double sf = a.Sanal - b.Sanal;
        return new KarmasikSayi(gt, st);
    }
```

```
class Program
{
    public static void Main()
    {
        KarmasikSayi k1 = new KarmasikSayi(-5,-6);
        KarmasikSayi k2 = new KarmasikSayi(4, 7);
        KarmasikSayi t = k1 - k2;
        t.Yaz();
    }
}
```

Aritmetiksel Operatörlerin Aşırı Yüklenmesi

```
public static KarmasikSayi operator -(KarmasikSayi a, double b)
    {
        double gf = a.Gercek - b;
        return new KarmasikSayi(gf, a.Sanal);
    }

    public static KarmasikSayi operator -(double b, KarmasikSayi a)
    {
        double gf = b- a.Gercek ;
        return new KarmasikSayi(gf, a.Sanal);
    }
```

```
class Program
{
    public static void Main()
    {
        KarmasikSayi k1 = new KarmasikSayi(5,-5);
        KarmasikSayi t = 10 - k1;
        t.Yaz();
        KarmasikSayi y = k1 - 7;
        y.Yaz();
        KarmasikSayi z = k1 - k1;
        z.Yaz();
    }
}
```

Aritmetiksel Operatörlerin Aşırı Yüklenmesi

- Çarpma ve bölme işlemleri için de aynı yaklaşımla operatör metodları oluşturalım.

```
public static KarmasikSayi operator *(KarmasikSayi a,
 KarmasikSayi b)
   {
       double sanal1 = a.Gercek * b.Sanal;
       double sanal2 = a.Sanal * b.Gercek;
       double sc = sanal1 + sanal2;

       double gercek1 = a.Gercek * b.Gercek;
       double gercek2 = a.Sanal * b.Sanal;
       double gc = gercek1 - gercek2;

       return new KarmasikSayi(gc, sc);
   }
```

Aritmetiksel Operatörlerin Aşırı Yüklenmesi

```
public static KarmasikSayi operator /(KarmasikSayi a,
 KarmasikSayi b)
   {
 KarmasikSayi beslenik = new KarmasikSayi(b.Gercek, -
 b.Sanal);

 KarmasikSayi pay = a * beslenik
 double payda = b.Gercek * b.Gercek + b. Sanal * b.
 Sanal);

 double bolumGercek = pay.Gercek / payda;
 double bolumSanal = pay.Sanal / payda;

 return new KarmasikSayi(bolumGercek, bolumSanal);
   }
```

Operatör Aşırı Yükleme (Operator OverLoading)

- Aritmetiksel operatörlerin dışında ilişkisel operatörler, mantıksal operatörler ve dönüşüm operatörleri de aşırı yüklenebilir.

- C# dilinde **işlemli atama operatörleri (+=)** , **&&**, **||**, **[]**, **()**, **=**, **?:** , **is** , **sizeof**, **new, typeof** gibi operatörler aşırı yüklenemez.

- Operatörlerin var olan öncelik sırası da değiştirilemez.

İlişkisel Operatörlerin Aşırı Yüklenmesi

- İlişkisel operatör metotları (!= , == , < , > , <= , >=), **true** veya **false** değer ile geri dönerler.

- Bu operatörlerin aşırı yüklenmesinde dikkat edilecek tek husus zıt anlamlı operatörlerin her ikisinin de aynı anda yüklenmiş olması gerektiğidir. Yani operator == bildirilmişse operator != metodu da bildirilmelidir.

İlişkisel Operatörlerin Aşırı Yüklenmesi

```
public static bool operator ==(KarmasikSayi a,
 KarmasikSayi b)
   {
 if (a.Sanal == b.Sanal && a.Gercek == b.Gercek)
 return true;
 else return false;
   }


public static bool operator !=(KarmasikSayi a,
 KarmasikSayi b)
   {
 return !(a==b);
   }
```

true ve false Operatörlerin Aşırı Yüklenmesi

- **true** ve **false** operatörleri de aynı anda olmak koşuluyla aşırı yüklenebilirler. Eğer bu operatörler aşırı yüklenirlerse koşul ifadelerini aşağıdaki gibi kullanmak mümkün olur:
- **if (sınıf nesnesi)**
- **{deyim}**
- **else**
- **{deyim}**
-

true ve false Operatörlerin Aşırı Yüklenmesi

```
public static bool operator true(KarmasikSayi a)
   {
 if (a.Sanal != 0 || a.Gercek != 0)
 return true;
 else return false;
   }

public static bool operator false(KarmasikSayi a)
   {
 if (a.Sanal == 0 || a.Gercek == 0)
 return true;
 else return false;


   }
```

Mantıksal Operatörlerin Aşırı Yüklenmesi

- Mantıksal operatör metotlarının aşırı yüklenmesini 2 kısımda incelemek mümkündür:

- **Birinci grup** : **&, |,** ve **!**
  - & ve |, ! operatörlerinin aşırı yüklenebilmesi için herhangi bir şart yoktur
- **İkinci grup : &&** ve **||** operatörleri
  - && ve || operatörleri direkt olarak aşırı yüklenemezler.
  - Bazı ön şartları vardır.

Mantıksal Operatörlerin Aşırı Yüklenmesi

- **&** ve **|** operatörleri aşırı yüklenmiş olmalıdır
- **true** ve **false** operatörlerinin aşırı yüklenmiş olması gerekir.
- **operator &** ve **operator |** metotlarının parametreleri ve geri dönüş değeri ilgili sınıfın türünden olmalıdır.

- Yukarıdaki şartlar sağlandığı takdirde bizim ayrıca && ve || operatörlerini aşırı yüklememize gerek kalmaz. Yukarıdaki şartlar sağlandığı takdirde sanki && ve || operatörleri aşırı yüklenmiş olur.

Mantıksal Operatörlerin Aşırı Yüklenmesi

```
public static bool operator |(KarmasikSayi
 a,KarmasikSayi b)
   {
 if ((a.Sanal != 0 || a.Gercek != 0) | ( b.Sanal!= 0
 || b.Gercek !=0))
 return true;
 else return false;
   }


public static bool operator &(KarmasikSayi
 a,KarmasikSayi b)
   {
 if ((a.Sanal == 0 || a.Gercek == 0) | ( b.Sanal== 0
 || b.Gercek ==0))
 return false;
 else return true;
   }
```

```
public static bool operator !(KarmasikSayi a)
   {
 if ((a.Gercek != 0) | ( a.Sanal!= 0 ))
 return false;
 else return true;
   }
```

Operatörlerin Aşırı Yüklenmesi ile ilgili örnek

```
class Sinif
{ public int Sayi;
  public Sinif(int sayi)    { Sayi= sayi; }
  public static bool operator true (Sinif a) { return true;  }
  public static bool operator false (Sinif a){ return false; }
  public static Sinif operator & (Sinif a, Sinif b) {return new Sinif(20);}
  public static Sinif operator | (Sinif a, Sinif b) {return new Sinif(30);}
}
 class AnaProgram
{  static void Main()
   {  Sinif a=new Sinif(50);       Sinif b=new Sinif(10);
      Console.WriteLine((a||b).Sayi);
      Console.WriteLine((a&&b).Sayi);
   }
}
```

Operatörlerin Aşırı Yüklenmesi ile ilgili örnek

- Yukarıdaki programda ekrana alt alta 50 ve 20 yazar. Peki neden 50 ve 20 yazdı? İşte mantığı:

true operatörü	**false operatörü**	**a||b**	**a&&b**
true	false	ilk operand	& operatör metodu
false	true	| operatör metodu	ilk operand
true	true	ilk operand	ilk operand
false	false	| operatör metodu	& operatör metodu

- Tablodan da görebileceğiniz gibi örneğimizde true operatörü true, false operatörü false değer ürettiği için a||b ifadesi ilk operand olan a'yı, a&&b ifadesi ise & operatör metodunun geri dönüş değerini tutar.

Dönüşüm Operatörlerin Aşırı Yüklenmesi

- Tür dönüşüm operatörleri ile aşağıdaki bir ifadenin nasıl bir sonuç üretebileceğini bilebiliriz.
  - KarmasikSayi k = new KarmasikSayi();
  - int a = k;
  - int b = (int) k;
- Dönüşüm operatörleri aşırı yüklenmez ise, derleyici yukarıdaki gibi bir durumla karşılaştığında ne yapacağını bilemez.

Dönüşüm Operatörlerin Aşırı Yüklenmesi

```
public static implicit operator HedefTur(DonusturulecekTur)
    {
  return HedefTur
    }
```

veya

```
public static explicit operator HedefTur(DonusturulecekTur)
    {
  return HedefTur
    }
```

Dönüşüm Operatörlerin Aşırı Yüklenmesi

Örn:

```
Eğer k KarmasikSayi türünden bir nesne ise;

int a = k ; /* atamasının geçerli olabilrmesi için */
```

```
public static implicit operator int(KarmasikSayi k)
   {
  return 10; // return k.Gercek ifadesi de olabilirdi.
    }
```

Örn:

```
Eğer k KarmasikSayi türünden bir nesne ise;

int a = (int) k ; /* atamasının geçerli olabilrmesi için */
```

```
public static explicit operator int(KarmasikSayi k)
   {
  return 10; // return k.Gercek ifadesi de olabilirdi.
    }
```

// Her iki örnekte de k.Gercek double ise return (int) k.Gercek ile bilinçli tip dönüşümü yapılmalı. Tip dönüşüm kuralları bunun içinde geçerli

Indeksleyiciler (Indexers)

- Daha önceden de bildiğiniz gibi tanımladığımız diziler System.Array sınıfı türünden bir nesnedir. Bu sayede tanımladığımız diziler üzerinden System.Array sınıfının üye elemanlarına erişebiliriz.

- Örneğin System.Array sınıfına ait olan Length özelliği dizimizin kaç elemandan oluştuğunu tutar. Sort() metodu ise dizimizin elemanlarını sıralar.

- Dizilerin elemanlarına [] operatörüyle erişilir. Bunu sağlayan System.Array sınıfındaki bir indeksleyeciden başka bir şey değildir. Sinif türünden oluşturduğumuz "**a**" nesnesini "**a[5]**" şeklinde yazarak bir anlam kazanması sağlanabilir.

Indeksleyiciler (Indexers)

- İndeksleme, dizi elemanlara erişmek için kullanılan “[ ]” operatöründen gelmektedir.
- İndeksleyiciler, olsaydı operator [] metodu yerine (böyle bir metot yoktur) faaliyet gösterirlerdi.
- İndeksleyiciler genelde sınıfın üye elemanlarından birisi dizi ya da benzer bir türden ise kullanışlı olur. Ancak bu amacın dışında da kullanılması tamamıyla serbesttir.
- Ancak operatörlerin taşıdıkları anlamlar çerçevesinde kullanılmaları daha doğru olur. İndeksleyiciler tıpkı diziler gibi tek boyutlu indeksleyiciler ve çok boyutlu indeksleyiciler olmak üzere ikiye ayrılır.

Tek Boyutlu İndeksleyici

- İndeksleyiciler sahte özelliklere oldukça benzerdir. Ancak indeksleyici tanımlarken parametre normal parantez yerine köşeli parantezler arasına yazılır.
- Esas programda "**[**" ve "**]**" arasına girilen veri indeksleyiciye parametre olarak gönderiliyor.
- İndeksleyicilere özel bir ad verilmiyor, bunun yerine **this** anahtar sözcüğü kullanılıyor.

Indeksleyiciler (Indexers)

- Genel Tanımlaması:
- ElemanTipi **this** [IndeksTipi indeks]
- {
- get { return değer; }
- set { işlemler; …; }
- }

- İndeksleyicilerde indeks değeri dizilerde olduğu gibi tamsayı olmak zorunda değildir. Fakat ideal kullanımda tamsayı olarak seçilmelidirler.

Tek Boyutlu İndeksleyici

```
using System;
class Sinif
{ public int Sayi;
  public int this[int indeks]
  { get { return Sayi; }
    set { Sayi= value; }
  }
}
class AnaProgram
{ static void Main()
  { Sinif a = new Sinif(); a[5]=30;
    Console.WriteLine(a[5]);
  }
}
```

- a[5]=30; satırıyla a[5]'in değil a'nın Sayi özelliği değiştiriliyor.
- 30 yazar

Tek Boyutlu İndeksleyici

```
using System;
class Sinif
{ public int Sayi;
  public int this[int indeks]
  { get { return Sayi; }
    set { if(indeks>0) Sayi+=value;
          else if(indeks<0) Sayi-=value;
          else Sayi=Sayi;
        }
} }
class AnaProgram
{ static void Main()
  { Sinif a=new Sinif();
    a[5]=45; a[-10]=23; a[100]=87; a[-80]=100;
    Console.WriteLine(a[0]);
 }
}
```

Çıktı: 9

Çok Boyutlu indeksleyiciler

- Çok boyutlu indeksleyicilerin tek boyutlu indeksleyicilerden tek farkı indeksleyicinin aldığı parametre sayısıdır. Örnek:

```
class Sinif
{ private int Sayi;
  public int this[int i, int j]
  { get { return i + j + Sayi; }
    set { Sayi=i * j + value; }
  }
}
class AnaProgram
{ static void Main()
  { Sinif a=new Sinif(); a[5,4]=45; Console.WriteLine(a[-6,12]); }
}
```

- Bu programda a[5,4]=45; satırında 5*4+45 işlemi yapılır ve sonuç Sayi özelliğine atanır. Console.WriteLine(a[-6,12]); satırında ise -6+12+65 işlemi yapılır ve sonuç ekrana yazılır. (sonuç: 71)

Indeksleyiciler (Indexers)

- İndeksleyiciler daha çok sınıfın üye dizilerinden birine direkt erişmek için kullanılırlar. Diziler dinamik olarak oluşturulabilir. Nesne ismine indeks verilerek üye dizinin elemanının değeri elde edilir.

- **NOT:**
- İndeksleyiciler de aşırı yüklenebilir. Bunu ise parametre türünü farklı yaparak sağlarız. Fakat bu çok kullanılan bir yöntem değildir.
- İndeksleyicilerin parametre ve geri dönüş tipi herhangi bir tip olabilir. int olması şart değildir.
- İndeksleyicilerin set ve get blokları sahte özelliklerdeki set ve get bloklarının tüm özelliklerini taşırlar. Örneğin set bloğunu koymayarak nesnenin indekslerine değer atanmasını engelleyebiliriz.
- İndeksleyicilerin en yaygın kullanımı sınıfımız içindeki bir dizinin elemanlarına direkt nesne ve [] operatörünü kullanarak erişmektir. Bunu mantığınızı kullanarak yapabilirsiniz.

```
using System;
class indeksleyici
{   private int[] dizi;
    public indeksleyici(int dizi_uzunluk)
    { dizi = new int[dizi_uzunluk]; }

    public int DiziBoyutu
    { get { return dizi.Length; } }

    public int this[int indeks]
    {   get { return dizi[indeks]; }
        set { dizi[indeks] = value; }
    }
}
class AnaProgram
{   static void Main()
    {   indeksleyici i = new indeksleyici(10);
        for (int k = 0; k < i.DiziBoyutu; k++)
            Console.WriteLine("i[{0}]={1}", k, i[k] = k);
    }
}
```

```
i[0]=0
i[1]=1
i[2]=2
i[3]=3
i[4]=4
i[5]=5
i[6]=6
i[7]=7
i[8]=8
i[9]=9
```

Yapılar (Structs)

- C# dilinde yapılar değer tipindedir. Sınıf bildirimine çok benzer şekilde tanımlanırlar. **struct** anahtar sözcüğü kullanılır.

- Bazı durumlarda belli bir grup verinin bir arada tanımlanması için sınıfların kullanılması verimsiz olur. Sınıf kullanıldığında stack alanında referans tipte değişken oluşturulur ve üyeler içinde heap alanında ayrıca bellek ayrılır. Verilere ulaşmak için referans kullanmak istenmeyebilir.

- Bu gibi durumlarda az sayıda ve veri tipindeki değişkenleri yapı şeklinde tanımlamak hız ve verimlilik sağlayabilir.

Yapılar (Structs)

- **Yapı Tanımlaması:**
- `struct YapıAdı`
- `{`
- `Elemanlar;`
- `}`

```
struct Ogrenci
{   public int Numara;
    public string Ad;
    public string Soyad;
}
```

- **Yapılar daha çok birbiri ile ilişkili değerleri bir araya toplamak için kullanılır.**
- **Yapılar değer tipidir ve yapı türündeki nesneler stack alanında saklanır.**
- **Yapılar da diğer tüm nesneler gibi object sınıfından türetilmiştir.**
- **Yapılar kalıtımı desteklemez, türetme yapılamaz.**

Yapılar (Structs)

- Yapılar tanımlandıktan sonra bir yapı nesnesi oluşturmak için yine **new** operatörü kullanılır.
- Varsayılan yapıcı metot ya da bizim belirlediğimiz yapıcı metot çalışarak ilk değer ataması yapar.
- Sınıflardan farklı olarak **new** kullanılmadan da yapılar tanımlanabilir.
- Bu şekilde tanımlanan yapı nesnelerinin üye elemanlarına ilk değer elle verilmelidir.
- Yapı nesnesinin elemanlarına sınıflarda olduğu gibi "." ile erişilir.

Yapılar (Structs)

```
using System;

class Yapilar
{
    public struct Ogrenci
    {
        public int Numara;
        public string Ad;
        public string Soyad;
    }

    public static void Metot(Ogrenci o)
    {
        o.Numara = 999;
    }

    public static void Main()
    {
        Ogrenci ogr=new Ogrenci();

        ogr.Numara = 123;
        ogr.Ad = "Ali";
        ogr.Soyad = "Türk";

        Metot(ogr);

        Console.WriteLine("{0} {1} {2}", ogr.Numara, ogr.Ad, ogr.Soyad);
    }
}
```

Yapılar (Structs)

- Yapıların birden fazla yapıcı metotları olabilir. Ancak varsayılan yapıcı metodu biz bildiremeyiz.
- Ör: struct yapi
- { yapi() { …} // Varsayılan yapıcı metot tanımlanamaz hata verir }
- Bütün yapıcıların parametre alma zorunluluğu vardır.
- Yapıcı metot bildirildiği zaman bütün elemanlara ilk değer verme zorunluluğu vardır. Bu değer ataması mutlaka parametre ile olmak zorunda değildir.
- Yapı nesneleri faaliyet alanı bitince otomatik olarak **stack** bölgesinden silinirler. Programcının nesnenin kaynaklarını geri almak için yapması gereken bir şey yoktur. Bu yüzden yapılarda **yıkıcı metotlar bildirilemez**.
- Sınıflarda olduğu gibi yapılar içinde değişkenler için set-get metotları ile indeksleyiciler kullanılabilir.

Yapılar (Structs)

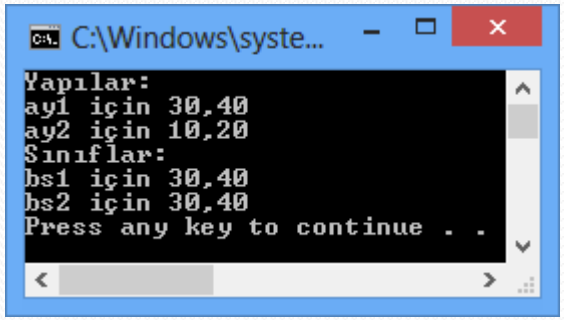
```
using System;

class Yapilar
{
    public struct Ogrenci
    {
        public int Numara;
        public string Ad;
        public string Soyad;

        public Ogrenci(int no, string ad, string soyad)
        {
            Numara = no;
            Ad = ad;
            Soyad = soyad;
        }
    }

    public static void Main()
    {
        Ogrenci ogr = new Ogrenci(123,"Ali","Türk");

        Console.WriteLine("{0} {1} {2}", ogr.Numara, ogr.Ad,
                                                    ogr.Soyad);
    }
}
```

Avantajları

- Yapıların bazı avantajları göz önünde bulundurularak belli problemlerin çözümünde kullanılabilirler:
  - Stack bellek alanında yer ayırma, kopyalama gibi işlemler heap alanından daha hızlıdır.
  - Sınıf nesneleri Garbage Collection mekanizması ile heap alanından silindikleri için yıkıcı metotların ne zaman çağrılacağı kesin bilinemez. Yapılarda yıkıcı metot olmamasına rağmen faaliyet alanı bitiminde bellekten otomatik silinirler.
  - Yapıları, atama oparatörü ile kopyalayabiliriz. Yapılarda, atama işlemi sonucunda değerler kopyalanır, fakat sınıf nesnelerinde sınıfın içeriği değil referanslar kopyalanır.

Yapılar (Structs)

```
using System;
struct ayapi { public int cx; public int cy;}
class bsinif { public int cx; public int cy;}
class anasinif
{
    static public void Main()
    {   ayapi ay1, ay2;
        ay1.cx = 10; ay1.cy = 20;
        ay2 = ay1;
        ay1.cx = 30; ay1.cy = 40;
        Console.WriteLine("Yapılar:");
        Console.WriteLine("ay1 için {0},{1}", ay1.cx, ay1.cy);
        Console.WriteLine("ay2 için {0},{1}", ay2.cx, ay2.cy);

        bsinif bs1, bs2;
        bs1 = new bsinif();
        bs1.cx = 10; bs1.cy = 20;
        bs2 = bs1;
        bs1.cx = 30; bs1.cy = 40;
        Console.WriteLine("Sınıflar:");
        Console.WriteLine("bs1 için {0},{1}", bs1.cx, bs1.cy);
        Console.WriteLine("bs2 için {0},{1}", bs2.cx, bs2.cy);
    }
}
```



- **Lab uygulama ödevi**
- Belirli bir dersi alan öğrencilerin bilgilerinin bellekte kaydedildiği ve üzerinde bazı işlemlerin yapılabildiği aşağıdaki ekran çıktısı verilen programı yazınız. Örnek Yapı:

```
public struct Ogrenci
{    public int[] Numara;
     public string[] Ad;
     public string[] Soyad;
     public int[] Not;
}
```

Numaralandırmalar(Enumaration)

- Enum sabitleri sayesinde bazı özel sözcüklerin bazı tam sayıları temsil etmesini sağlayabiliriz. Numaralandırma (enumeration) belli bir sayısal değer alan toplu sabitler tanımlamak için kullanılan bir yapıdır.

- **Tanımlama şekli;**
- **enum** anahtar sözcüğü ile tanımlanır.
- ***enum isim { sozcuk1, sozcuk2 };***
- ***enum isim : tur{ sozcuk1, sozcuk2}***

- Burada isim numaralandırmaya verilen tip ismini, parantez içindeki "," ayrılan ifadelerde tanımlayıcıların listesini belirler.

Numaralandırmalar(Enumaration)

- enum sabitleri çeşitli sembolleri sayılarla ifade etmek için geliştirilmiş bir veri yapısıdır.

- enum sabitlerinin varsayılan değeri int türündedir. Fakat diğer tamsayı türleri ile de belirtilebilir. Tanımlanması:
  - `enum isim {sabit1,sabit2,…} //varsayılan değer tipi int`
  - `enum isim : byte {sabit1,sabit2,…} // tür değeri byte`

- Numaralandırma sabitlere sıfırdan başlayarak artan birer tamsayı değeri verilmesi esasına göre gerçekleştirilir.

- Enum sabitlerinin türünü byte, sbyte, short, ushort, int, uint, long ve ulong türlerinden biri yapabiliriz.

Numaralandırmalar(Enumaration)

- Örneğimizde bir metot kendisine ilk parametre olarak gönderilen dizinin elemanlarını ekrana yazsın. Metodun ikinci parametresinde de dizinin elemanlarının ekrana nasıl yazılacağı ayarlansın. Eğer ikinci parametre YANINA olarak girilirse dizinin elemanları ekrana yan yana yazılsın. Eğer ikinci parametre ALTINA olarak girilirse dizinin elemanları ekrana alt alta yazılsın:

```
using System;
enum BICIM : byte{  YANINA, ALTINA }

class Program {
 static void DiziYaz ( Array dizi, BICIM b )
   { if(b==BICIM.YANINA)
     { foreach( int s in dizi) Console.Write( s.ToString()+" "); Console.WriteLine(); }
    else foreach( int s in dizi) Console.WriteLine(s);
   }
  static void Main() {
    int[] a=new int[10];
    DiziYaz(a, BICIM.YANINA);
    DiziYaz(a, BICIM.ALTINA); } }
```

Numaralandırmalar(Enumaration)

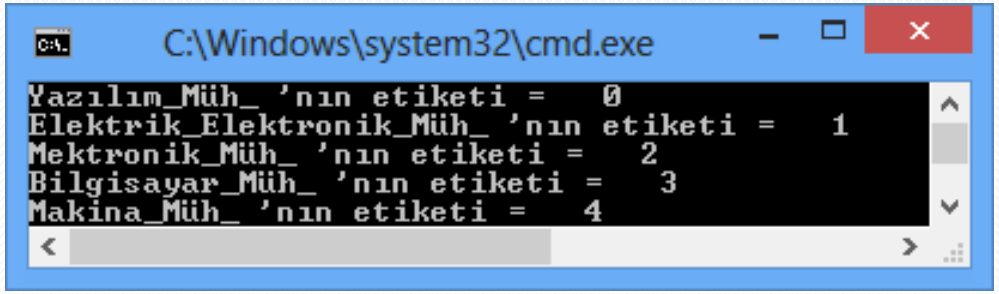
```
using System;
class Program
{   enum notu : byte { basarisiz, gecmez, gecer, orta, iyi, pekiyi }
    static void Main()
    {   Console.Write("Lütfen notunuzu giriniz: ");
        notu a = (notu)Convert.ToByte(Console.ReadLine());
        Console.WriteLine(a);
    }
}
```

Sözcük	Temsil ettiği sayı
basarisiz	0
gecmez	1
gecer	2
orta	3
iyi	4
pekiyi	5

programımızda byte türünü notu türüne (enum sabitine) bilinçli olarak dönüştürdük. Bilinçsiz olarak dönüştüremezdik. Ayrıca direkt olarak stringten nota dönüşüm yapamazdık. Yalnızca enum sınıfı ile string ve object dışındaki temel veri türleri arasında dönüşüm yapabiliriz.

Numaralandırmalar(Enumaration)

```
using System;
class üniversite
{     enum bolumler
      {  Yazılım_Müh_,
         Elektrik_Elektronik_Müh_,
         Mektronik_Müh_,
         Bilgisayar_Müh_,
         Makina_Müh_,
      }
      public static void Main()
      {   bolumler b;
          for (b = bolumler.Yazılım_Müh_;b<= bolumler.Makina_Müh_; b++)
             Console.WriteLine("{0} 'nın etiketi ={1,4:d}", b, (int)b);
      }
}
```



Enum sıra numaralarını değiştirme

- Tanımlama içinde farklı sayı değerleri vermek için sabitler “=“ ile farklı değerlere atanabilir.
- **enum** not:**byte**{basarisiz=6,basarili=10}
- Bu örnekte basarisiz 6'yı, basarili 10'u temsil eder.
- **enum** not:**byte**{basarisiz=6,basarili}
- Bu örnekte basarisiz 6'yı, basarili 7'yi temsil eder.
- **enum** not:**byte**{basarisiz=-21, basarili}
- Bu örnekte basarisiz -21'i basarili -20'yi temsil eder.
- **enum** not:**byte**{basarisiz, gecmez=5,gecer,orta,iyi=2,pekiyi}
- Bu örnekte ise şu temsiller söz konusudur
- **Atamalar tek ve ilk değerler mantıklı verilmelidir.**
- **Bir birini örten değerler veya aynı sabit verilmez.**

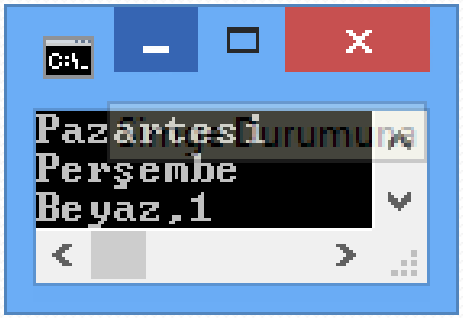
Sözcük	Temsil ettiği sayı
basarisiz	0
gecmez	5
gecer	6
orta	7
iyi	2
pekiyi	3

System.Enum sınıfı

- enum sabitlerine **enumAdı.Sabit** şeklinde ulaşılır.
- enum sabitleri de birer veri tipi oldukları için fonksiyona parametre olarak gönderilebilir.
- enum static olduğu için başka sınıftan erişildiğinde nesne oluşturulmaz. Ör: sınıfAdı.enumdeğişkeni şeklinde erişim yapılır. (başka sınıftan erişebilmek için erişim kuralları bunun içinde geçerli (public, private…)
- System.Enum sınıfından türetildikleri için bazı metotlar ile sabit sembollerin isimleri alınabilir.
- enum yapısı bir sınıfı ifade ettiği için bir sınıf içinde kullanılma zorunluluğu yoktur. Sınıf içinde olmadığı zaman erişim belirleyici public durumundadır ve sadece enum adı ile erişilir.

System.Enum sınıfı

```
class Program
{
 public enum Renkler : byte { Siyah, Beyaz, Kırmızı, Mavi, Sarı}
}
class Sinif
{
 enum Gunler : byte { Pazartesi, Salı, Çarşamba, Perşembe, Cuma,
Cumartesi, Pazar }

    static void Main()
    {
        string[] a = Gunler.GetNames(typeof(Gunler));
        Console.WriteLine(a[0]); //veya
        Console.WriteLine(Gunler.GetNames(typeof(Gunler))[3]);

        Program.Renkler b = (Program.Renkler)1;
        Console.WriteLine(b +","+ (byte)b);
    }
}
```



System.Enum sınıfı

```
using System;
enum Renkler : byte { Siyah, Beyaz, Kırmızı, Mavi, Sarı}
class Sinif
{
     static void Main()
    {
        Renkler b = (Renkler)1;
        Console.WriteLine(b + ","+ (byte)b);
    }
}
```

- Sınıf dışında kullanımında enum varsayılan erişim belirleyicisi public tir.

Visual Studio.Net -C#
7. HAFTA
İsim alanları,
Kütüphane dosyası (.dll)
Harici Takma İsimler(Extern Alias)
System İsim Alanı

Operatörlerin Aşırı Yüklenmesi ile ilgili örnekler

```
using System;
class Zaman {
    public int Saat;     public int Dakika;    public int Saniye;
    public Zaman(int saat, int dakika, int saniye)
    {  Dakika=dakika+saniye/60;
       Saniye=saniye%60;
       Saat=saat+Dakika/60;
       Dakika=Dakika%60;
    }
 public static Zaman operator+(Zaman a, Zaman b)
 {      int ToplamSaniye=a.Saniye+b.Saniye;
        int ToplamDakika=a.Dakika+b.Dakika;
        int ToplamSaat=a.Saat+b.Saat;
        return new Zaman(ToplamSaat,ToplamDakika,ToplamSaniye);
    }
}
```

Operatörlerin Aşırı Yüklenmesi ile ilgili örnek

```
class AnaProgram
 {
    static void Main()
    {
        Zaman zaman1=new Zaman(5,59,60);
        Zaman zaman2=new Zaman(1,0,120);
        Zaman zaman3=zaman1+zaman2;

Console.WriteLine("{0}.{1}.{2}",zaman3.Saat,zaman3.Dakika,
zaman3.Saniye);
    }
 }
//ekrana 7.2.0 yazacaktır
```

Operatörlerin Aşırı Yüklenmesi ile ilgili örnek

- Eğer Zaman türünden bir nesneyle int türünden bir nesne toplanmak istenirse int türünden nesneyi saniye olarak hesaba katabiliriz. Bunun içinse Zaman sınıfına şu operatör metodunu ekleyebiliriz:

```
public static Zaman operator+(Zaman a, int b)
 { int ToplamSaniye=a.Saniye+b;
return new Zaman (a.Saat,a.Dakika,ToplamSaniye);
 }
```

Operatörlerin Aşırı Yüklenmesi ile ilgili örnek

- Operatör metotlarının geri dönüş tipinin illaki söz konusu sınıf tipinden olmasına gerek yoktur. Örneğin Zaman sınıfındaki ilk toplama operatör metodunu şöyle değiştirebiliriz.

```
public static string operator+(Zaman a, Zaman b)
{ int ToplamSaniye=a.Saniye+b.Saniye;
 int ToplamDakika=a.Dakika+b.Dakika;
 int ToplamSaat=a.Saat+b.Saat;
Zaman nesne=new Zaman(ToplamSaat,ToplamDakika,ToplamSaniye);
return nesne.Saat+"."+nesne.Dakika+"."+nesne.Saniye;
}
```

Operatörlerin Aşırı Yüklenmesi ile ilgili örnek

- İlişkisel operatörün aşırı yüklenmesi

```
public static bool operator==(Zaman a, Zaman b)
{
if(a.Saniye==b.Saniye&&a.Dakika==b.Dakika&&a.Saat==b.Saat)
   return true;
else
   return false;
 }
 public static bool operator!=(Zaman a, Zaman b)
 {
     return !(a==b);
 }
```

Operatörlerin Aşırı Yüklenmesi ile ilgili örnek

- Bu metotları asıl programımız içinde şöyle kullanabiliriz:

```
Zaman zaman1=new Zaman(5,59,60);
Zaman zaman2=new Zaman(2,35,40);
 if(zaman1)
    Console.WriteLine("Öğleden sonra");
 else
    Console.WriteLine("Öğleden önce");
 if(zaman2)
    Console.WriteLine("Öğleden sonra");
 else
    Console.WriteLine("Öğleden önce");
```

Operatörlerin Aşırı Yüklenmesi ile ilgili örnek

- Mantıksal operatörlerin aşırı yüklenmesi
- Örnek (sınıfımıza eklenecek):
-
- `public static bool operator|(Zaman a,Zaman b)`
- `{`
- `if(a.Saat>12 || b.Saat>12)`
- `return true;`
- `else`
- `return false;`
- `}`

Operatörlerin Aşırı Yüklenmesi ile ilgili örnek

- Tür dönüşüm operatörünün aşırı yüklenmesi
- Bilinçsiz tür dönüşümü için;
- public static implicit operator HedefTur(KaynakTurdekiNesne)
- { return HedefTurdenVeri; }
- using System;
- class Sinif
- { public int Sayi;
- public Sinif(int sayi) { Sayi = sayi; }
- public static implicit operator int(Sinif a) { return a.Sayi; }
- }
- class AnaProgram
- { static void Main()
- { Sinif a = new Sinif(50);
- int b = a; Console.WriteLine(b);
- }
- }

Operatörlerin Aşırı Yüklenmesi ile ilgili örnek

- Bilinçli tür dönüşümü için;
- **public static explicit operator** HedefTur(KaynakTurdekiNesne)
- { **return** HedefTurdenVeri; }

```
using System;
class Sinif
{   public int Sayi;
    public Sinif(int sayi) { Sayi = sayi; }
    public static explicit operator int(Sinif a)
    { return a.Sayi; }
}
class AnaProgram
{   static void Main()
    {   Sinif a = new Sinif(50);
        int b = (int)a; Console.WriteLine(b);
    }
}
```

Operatörlerin Aşırı Yüklenmesi ile ilgili örnek

- **BİLGİLENDİRME:** Şimdi konunun başında gördüğümüz Zaman sınıfını düşünün. Artık implicit metot sayesinde Zaman türünden bir nesneye a="12.45.34"; şeklinde değer atayabilirsiniz. Bunun için Zaman sınıfının implicit metodu içinde gerekli parçalamaları yapar ve sonucu bir Zaman nesnesinde tutarsınız. Gördüğünüz gibi adeta kendi sabitlerimizi yazabiliyoruz. Bunun gibi daha birçok şey hayal gücünüze kalmış.
- **NOT:** Artık bu öğrendiğimiz implicit metot sayesinde Sinif türünden bir nesneyi new anahtar sözcüğünü kullanmadan direkt Sinif a=23; gibi bir şey yazarak oluşturabiliriz. Ancak bunun için tabii ki sınıfımız içinde int türünden Sinif türüne implicit dönüşüm metodunu oluşturmamız gerekir.
- **NOT:** Bildiğiniz gibi C#'ta bazı türler arasında bilinçsiz tür dönüşümü olur. Örneğin C#'ta byte türünden inte bilinçsiz tür dönüşümü mümkündür. Ancak biz programımızı yazarken bunları kafamıza takmamıza gerek yok. Yani Sinif türünden int'e dönüşüm yapan metot oluşturmuşsak Sinif türünden byte'a dönüşüm yapan metot oluşturamayız diye bir kural yoktur. İstediğimiz gibi tür dönüşüm metotlarını yazabiliriz. Yeter ki bir türden bir türe dönüşüm yapan birden fazla metot oluşturmayalım ve dönüşümün taraflarından bir tanesi Sinif türünden olsun.

Operatörlerin Aşırı Yüklenmesi ile ilgili örnek

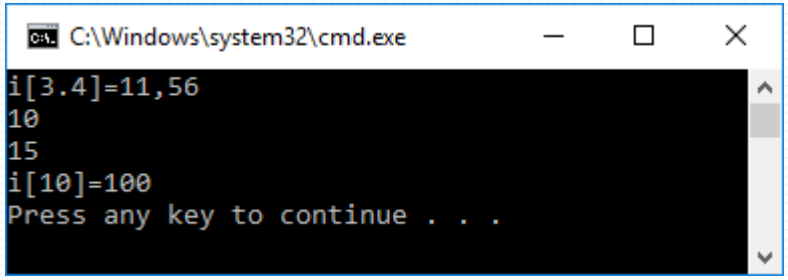
- Atama operatörünü (=) aşırı yükleyemeyiz. Çünkü zaten gerek yoktur. Biz atama operatörünü aşırı yükleyelim veya yüklemeyelim zaten kullanabiliriz.

- İşlemli atama operatörlerinde ise şöyle bir kural vardır: Örneğin + operatörünü aşırı yüklemişsek += operatörünü de kullanabiliriz. Bu durum bütün işlemli atama operatörlerinde geçerlidir.

- Hiçbir operatörün öncelik sırasını değiştiremeyeceğimiz gibi temel veri türleri (string, int, vb.) arasındaki operatörlere de müdahale edemeyiz. Bu konuda işlenenler dışında hiçbir operatörü aşırı yükleyemeyiz. Örneğin dizilerin elemanlarına erişmek için kullanılan [] operatörünü, yeni bir nesneye bellekte yer açmak için kullanılan new operatörünü, ?: operatörünü, vb. aşırı yükleyemeyiz.

Tek Boyutlu İndeksleyici Örneği

```
using System;
class indexleyici
{   public double rakam;
    public double this[double index]
    {   get  {     return Math.Pow(index, 2);   //index*index   }
        set { rakam = value; }
    }
}
class indexleyicistart
{    static void Main()
    {   indexleyici i = new indexleyici();
        Console.WriteLine("i[3.4]={0}", i[3.4]);
        i[10] = 10;
        Console.WriteLine(i.rakam);
        i[10] = 15;
        Console.WriteLine(i.rakam);
        Console.WriteLine("i[10]={0}", i[10]);
    }
}
```



Yapılar (Structs)

```
using System;

class Yapilar
{
    struct Ogrenci
    {
        public int Numara;
        public string Ad;
        public string Soyad;
    }

    public static void Main()
    {
        Ogrenci ogr1=new Ogrenci();

        ogr1.Numara = 123;
        ogr1.Ad = "Ali";
        ogr1.Soyad = "Türk";

        Ogrenci ogr2 = ogr1;

        ogr2.Numara = 456;

        Console.WriteLine("{0} {1} {2}", ogr1.Numara, ogr1.Ad, ogr1.Soyad);
        Console.WriteLine("{0} {1} {2}", ogr2.Numara, ogr2.Ad, ogr2.Soyad);
    }
}
```

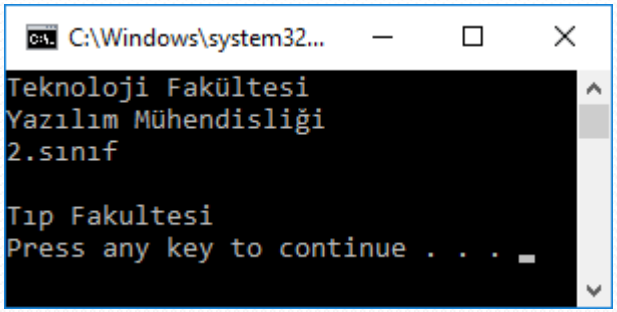
Numaralandırmalar(Enumaration)

```
using System;
enum notu : byte { basarisiz, basarili }
class Program
{
    static void Main()
    {
        notu a = notu.basarili;
        if (a == (notu)1)
            Console.Write("Başarılısınız");
        else
            Console.Write("Başarısızsınız");
    }
}
```

- Burada da byte türü notu türüne bilinçli olarak dönüştürüldü.

Yapılar (Structs) Örneği

```
using System;
struct üniversite
{   public string fakulte;     public string bolum;    public string Ogr_gor;
    public üniversite(string a, string b, string c)
    {   fakulte = a; bolum = b;    Ogr_gor = c;   }
}
class structornek
{
  public static void Main()
  {
   üniversite k1 = new üniversite("Teknoloji Fakültesi", "Yazılım Mühendisliği", "2.sınıf");
   üniversite k2;
   Console.WriteLine(k1.fakulte);
   Console.WriteLine(k1.bolum);
   Console.WriteLine(k1.Ogr_gor);
   k2.fakulte = "Tıp Fakultesi";
   Console.WriteLine();
   Console.WriteLine(k2.fakulte);
    }
}
```

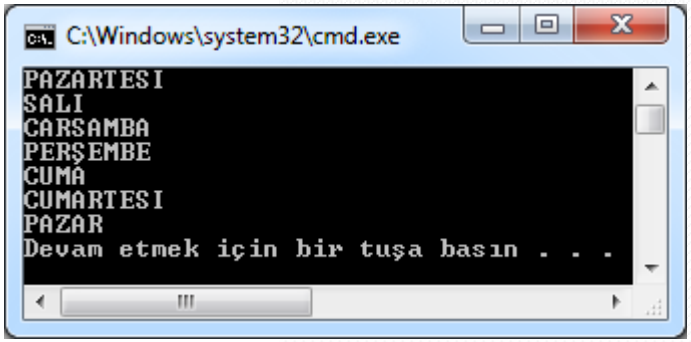


Numaralandırmalar(Enumaration)

```
using System;

enum GUNLER : byte
{
    PAZARTESI,
    SALI,
    CARSAMBA,
    PERSEMBE,
    CUMA,
    CUMARTESI,
    PAZAR
}

class Numaralandirma
{
    public static void Main()
    {
        Console.WriteLine((int)GUNLER.PAZARTESI);
        Console.WriteLine((int)GUNLER.PAZAR);
    }
}
```

System.Enum sınıfı

```
using System;

enum GUNLER : byte
{
    PAZARTESI,
    SALI,
    CARSAMBA,
    PERSEMBE,
    CUMA,
    CUMARTESI,
    PAZAR
}

class Numaralandirma
{
    public static void Main()
    {
        string[] sabitler = Enum.GetNames(typeof(GUNLER));
        foreach (string s in sabitler)
            Console.WriteLine(s);
    }
}
```



Visual Studio.Net -C#
7. HAFTA
İsim alanları,
Kütüphane dosyası (.dll)
Harici Takma İsimler(Extern Alias)
System İsim Alanı

İsim Alanları ve System İsim Alanı

- İsim alanları en temel anlamıyla sınıfları organize etmeye (gruplandırmaya) yarayan bir yapıdır.
- Eski programlama dillerinde, çok kişi tarafından yazılan projelerde, isim çakışmaları meydana gelebilmekteydi. Bu çakışmalar fonksiyon, alt program ya da sınıf isimlerinin değiştirilmesi ile çözülebiliyordu.
- Bu çözüm sonucunda farklı kişiler ya da firmalar tarafından yazılan aynı isimli metotların firma1Menu(), firma2Menu() vb. şeklinde isimlendirilmesi neden olur.
- Bu çözüm çok kullanıldıkça fonksiyon isimlerinin önüne anlamsız bir çok ekler gelecektir.

İsim Alanları ve System İsim Alanı

- C++ ve C# dillerindeki isim alanları (namespace) ya da Java dilindeki paketler (package) buna birer örnektir. Bu yapılar sayesinde aynı isimde sınıflar, yapılar kolaylıkla tanımlanabilir ve daha önceki tanımlananlarla herhangi bir soruna neden olmaz.

- Programlara isim alanları using anahtar sözcüğü ile bildirilir. İsim alanlarının bildirilmesi zorunlu olmamakla birlikte sıkıcı tekrarları ortadan kaldırır. İsim alanı bildirilmediğinde sınıfın, değişkenin ya da metodun tam ismiyle kullanılması gerekmektedir.

İsim Alanları ve System İsim Alanı

```
class Program
{
    static void Main()
    {
        System.Console.WriteLine("Tam isim");
    }
}
```

Ya da →

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("using ile");
    }
}
```

İsim Alanları ve System İsim Alanı

- Temel veri türlerini kullanmak için **System** isim alanını eklemek gerekmez. C# derleyicisi bunları otomatik olarak System isim alanını kullanarak işletir.
- `class Sinif`
- `{  static void Main()`
- `    {  int a = 5; //geçerli tanımlama`
- `        Int32 a = 5; //geçersiz tanımlama`
- `    }`
- `}`
- C# ikisinde de aynı yapıyı (System.Int32) kullanmamıza rağmen ikincisine izin vermedi. Programın başına using System; satırını ekleseydik ikincisine de izin verecekti. Yani C#, değişkenleri C#'a özgü şekilde oluşturmuşsak System isim alanını eklememiz zorunlu değildir. Yapıların gerçek hâlleriyle değişken oluşturmak ise System isim alanının eklenmesini gerektirir.

İsim Alanı Bildirimi

- İsim alanı namespace anahtar sözcüğü ile bildirilir. Namespace için yine sınıflarda olduğu gibi bir parantez blok açılır. Bu bloğa ise isim alanı içinde yer alacak bildirimler tanımlanır.
- Örnek bir isim alanı bildirimi
  - `namespace IsimAlani`
  - `{  class Deneme { … }  }`
  - `class AnaSinif`
  - `{`
  - `static void Main()`
  - `{ IsimAlani.Deneme d = new IsimAlani.Deneme(); }`
  - `}`
- İsim alanları içinde değişken tanımlaması ve ya metot bildirimi yapılamaz. Bir isim alanının içinde yalnızca sınıflar, temsilciler, enum sabitleri, arayüzler ve yapılar direkt olarak bulunabilir. Bundan sonra tüm bunlara kısaca "tür" denir.

İsim Alanı Bildirimi

- Farklı konumlarda ya da dosyalarda aynı adlı isim alanları tanımlanabilir. Bu hataya neden olmaz. Çünkü isim alanları mantıksal yapılardır.
- İsim alanları genellikle kod organizasyonu ve isim çakışmalarını engellemek için kullanılır.
- Eğer farklı dosyalardaki farklı türleri aynı isim alanına koyarsak bunlar aynı isim alanında sayılır.

İsim Alanı Bildirimi

- Örneğin Deneme1.cs dosyası şöyle olsun:

```
using System;
namespace Deneme
{   class Sinif1
    {   public Sinif1()
        {
            Console.WriteLine("Sinif1 nesnesi");
        }
    }
}
```

İsim Alanı Bildirimi

- Deneme2.cs dosyası şöyle olsun:

```
using System;
namespace Deneme
{
    class Sinif2
    {
        public Sinif2()
        {
            Console.WriteLine("Sinif2 nesnesi");
        }
    }
}
```

İsim Alanı Bildirimi

- Deneme3.cs dosyamız şöyle olsun:

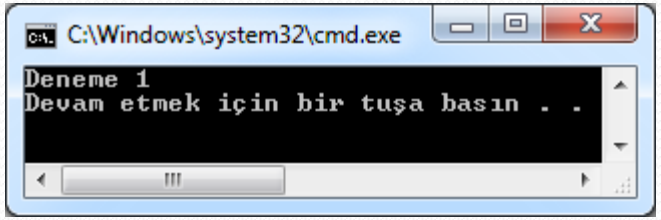
```
using System;
class Sinif3
{
    static void Main()
    {
        Console.WriteLine("Burası ana program");
        Deneme.Sinif1 a = new Deneme.Sinif1();
        Deneme.Sinif2 b = new Deneme.Sinif2();
    }
}
//Bu dosyaların hepsi aynı klasörde bulunacak
```

```
Çıktı: Burası ana program.
Sinif1 nesnesi
Sinif2 nesnesi.
```

- csc Deneme1.cs Deneme2.cs Deneme3.cs komutunu verelim. Bu komutla C# derleyicisinin bu üç dosyayı tek dosya gibi düşünmesini sağladık. C# bu tür bir durumda Sinif1 ve Sinif2 sınıflarını aynı isim alanında sayar.

İsim Alanı Bildirimi

```
using System;
using isimalan1; //using bildirimi ile kullanma
namespace isimalan1
{
    class Deneme
    {    public Deneme()
        { Console.WriteLine("Deneme 1"); }
    }
}
class Program
{    static void Main()
    { Deneme d1 = new Deneme(); }
}
```



using kullanımı

- **using** anahtar sözcüğünün bir başka kullanımı da istediğimiz bir bloğun sonunda nesnelerin Dispose() metodunu çağırmaktır. Örnek:

```
using System;
class Deneme : IDisposable // arayüz tanımı kullanılmasada olur.
{    public void Dispose()
    { Console.WriteLine("Dispose() metodu çağrıldı.");
      // GC.SuppressFinalize(this) kullanımına gerek yoktur
    }
}
class AnaSinif
{ static void Main()
  {
  using (Deneme d = new Deneme())
  { Console.WriteLine("using bloğu"); } //d.Dispose() metodu burada çağrılır. Yazmaya gerek yoktur.
    Console.WriteLine("using bloğu dışı");
    }
}
```

Bu program ekrana sırayla şunları yazar.

```
using bloğu
Dispose() metodu çağrıldı.
using bloğu dışı
```

using kullanımı

- using bloğunu aşağıdaki şekilde de yazabilirdik:
  - **using** (**Deneme** d1=new **Deneme**(), d2=new **Deneme**())
  - { Console.WriteLine("using bloğu"); } *//d1.Dispose() ve d2.Dispose() metodu burada çağrılır.*
- Bu using bloğunu aşağıdaki örnekteki gibi yazamazdık:
  - **using** (**Deneme** d1=new **Deneme**, **Deneme** d2=new **Deneme**())
  - { Console.WriteLine("using bloğu"); } //Hata verir
- Yani nesneler aynı türden olmalıdır ve ortak tanımlama yapılmalıdır. İkinci kez tanımlama yapılamaz.
- Aynı using bloğunda farklı sınıf tanımlar için nesne tanımlamaz .
  - **using** (Deneme d1=new Deneme, Deneme2 d2=new Deneme2())
  - { Console.WriteLine("using bloğu"); } // Hata verir
- Aşağıdaki şekilde kullanılabilir
  - **using** (Deneme d1=new Deneme ())
  - **using** (Deneme2 d2=new Deneme2())
  - { Console.WriteLine("using bloğu"); } *//d1.Dispose() ve d2.Dispose() metodu burada çağrılır.*

using ile takma isim (alias) verme

- using anahtar sözcüğü ile takma isim (alias) vermek de mümkündür.

- Bazı durumlarda using ile tanımlanmış iki isim alanı içerisinde aynı isimli sınıflar olabilir. Bu durumda doğrudan sınıf isminin kullanılması hataya neden olur. Çünkü derleyici, iki isim alanı içerisinde yer alan aynı isimli sınıflardan hangisinin kastedildiğini anlayamaz.
- Bunu aşmak için takma isimler tanımlanabilir.
- `using System;`
- `using Sinif1 = IsimAlani1.Sinif;`
- `using Sinif2 = IsimAlani2.Sinif;`

using ile takma isim (alias) verme

```
using System;
using IsimAlani1;// yazılmasa da olur
using IsimAlani2;// yazılmasa da olur
//Eğer aşağıdaki iki satırı yazmasaydık programımız hata verirdi.
using Sinif1=IsimAlani1.Sinif;
using Sinif2=IsimAlani2.Sinif;
//Bu iki satırla derleyicinin IsimAlani1.Sinif sınıfını Sinif1,
IsimAlani2.Sinif sınıfını Sinif2 olarak tanımasını sağladık.
 namespace IsimAlani1
 {   class Sinif
    {    public Sinif() { Console.WriteLine("Burası IsimAlani1");}}
 }
 namespace IsimAlani2
 {    class Sinif
    { public Sinif(){Console.WriteLine("Burası IsimAlani2");}}
 }
 class AnaSinif
 {    static void Main()
    {    Sinif1 a=new Sinif1();       Sinif2 b=new Sinif2();
      //Takma isimler kullanıldı
    }   }
```

using ile takma isim (alias) verme

- **NOT:** Takma ad kullanımı .Net Framework kütüphanesindeki sınıflarda da geçerlidir.
- Örnek:
- `using System;`
- `using yaz = System.Console;`
- `class AnaSinif`
- `{`
- `    static void Main()`
- `    {`
- `        yaz.WriteLine("Deneme");`
- `    }`
- `}`

- // yaz=Console; ifadesi tek başına yazılsaydı hata verirdi mutlaka isim alanı ile birlikte yazılmalı.

İç İçe Geçmiş İsim Alanları Tanımlamak (Nested NameSpaces)

- İsim alanları içerisinde başka isim alanları da tanımlanabilir. Bu şekilde hiyerarşik düzenlemeler yapılır.

- Bu şekilde tanımlanmış isim alanlarına "." operatörü ile erişilir. using ile tanımlanırken de aynı yöntem kullanılır.

İç İçe Geçmiş İsim Alanları Tanımlamak (Nested NameSpaces)

```
using System;

namespace Alan
{
    class Sinif1
    {
        public Sinif1() { }
    }

    namespace AltAlan
    {
        class Sinif2
        {
            public Sinif2() { }
        }
    }
}

class Program
{
    static void Main()
    {
        Alan.Sinif1 s1 = new Alan.Sinif1();
        Alan.AltAlan.Sinif2 s2 = new Alan.AltAlan.Sinif2();
    }
}
```

Kütüphane dosyası oluşturma (dll)

- C# ‘ ta çoklu dosya kullanımı ile birden fazla kaynak kod referans edilerek veya dll dosyası halinde diğer kaynak kodlar tarafından kullanılabilir.
- Örneğin: Elimizde çeşitli sayı tiplerinin dönüşümünü yapan ‘Program.cs’ ve ‘topla’ isimli fonksiyon içeren ‘Program1.cs’ adlı kod dosyalarımız olsun.

Kütüphane dosyası oluşturma (dll)

- program.cs

```
using toplama;

namespace kutuphane
{
    class program
    {
        static void Main(string[] args)
        {
            float a,b;
            string c;
            Console.WriteLine("a sayısı gir");
            a = Convert.ToSingle(Console.ReadLine()); //float a çevrim
            Console.WriteLine("b sayısı gir");
            b =float.Parse(Console.ReadLine()); // float a çevrim
            Console.WriteLine("toplam={0}",toplama.program.topla(a, b));
            Console.ReadKey();      }
    }
```

Kütüphane dosyası oluşturma (dll)

- program1.cs

```
namespace toplama
{
class program
    {
        public static float topla(float a, float b)
        {
            return (a + b);
        }
    }
}
```

Kütüphane dosyası oluşturma (dll)

- Program.cs kaynak kod dosyasının içindeki

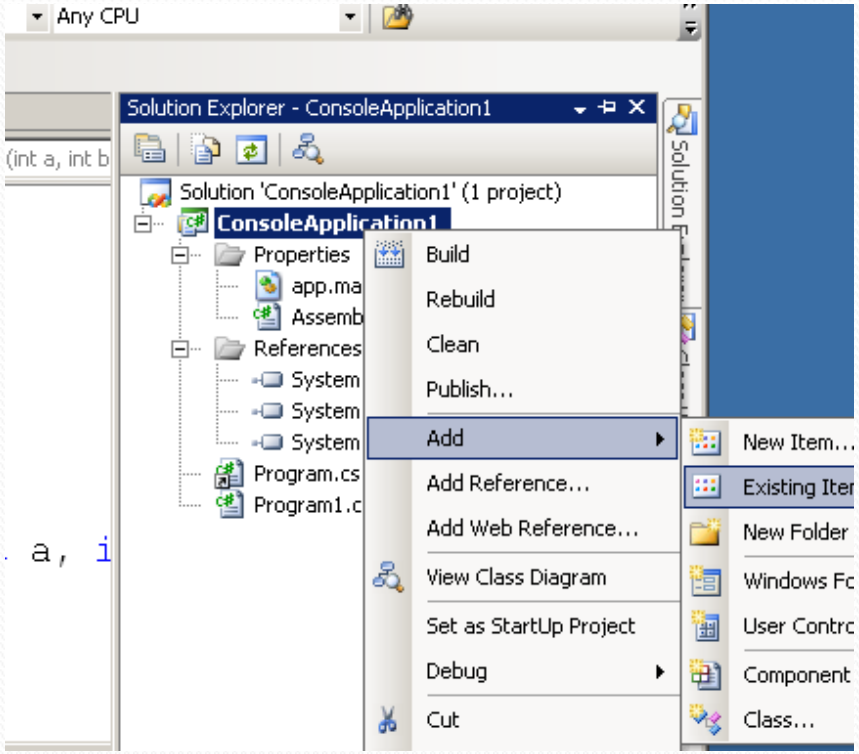
```
Console.WriteLine("toplam={0}",toplama.program.topla(a, b));
```

- Yukarıdaki toplama.program.topla(a,a) kısmında bu kaynak kod içinden başka bir alanadı(namespace) içindeki ‘Program’ sınıfı içindeki ‘topla’ isimli metot çağrılmaktadır.

- Bu durumda toplama alan adını içeren Program1.cs dosyasının bir şekilde Program.cs’ye dahil edilmesi gerekir. Bunun için VStudio’yu kullanalım.

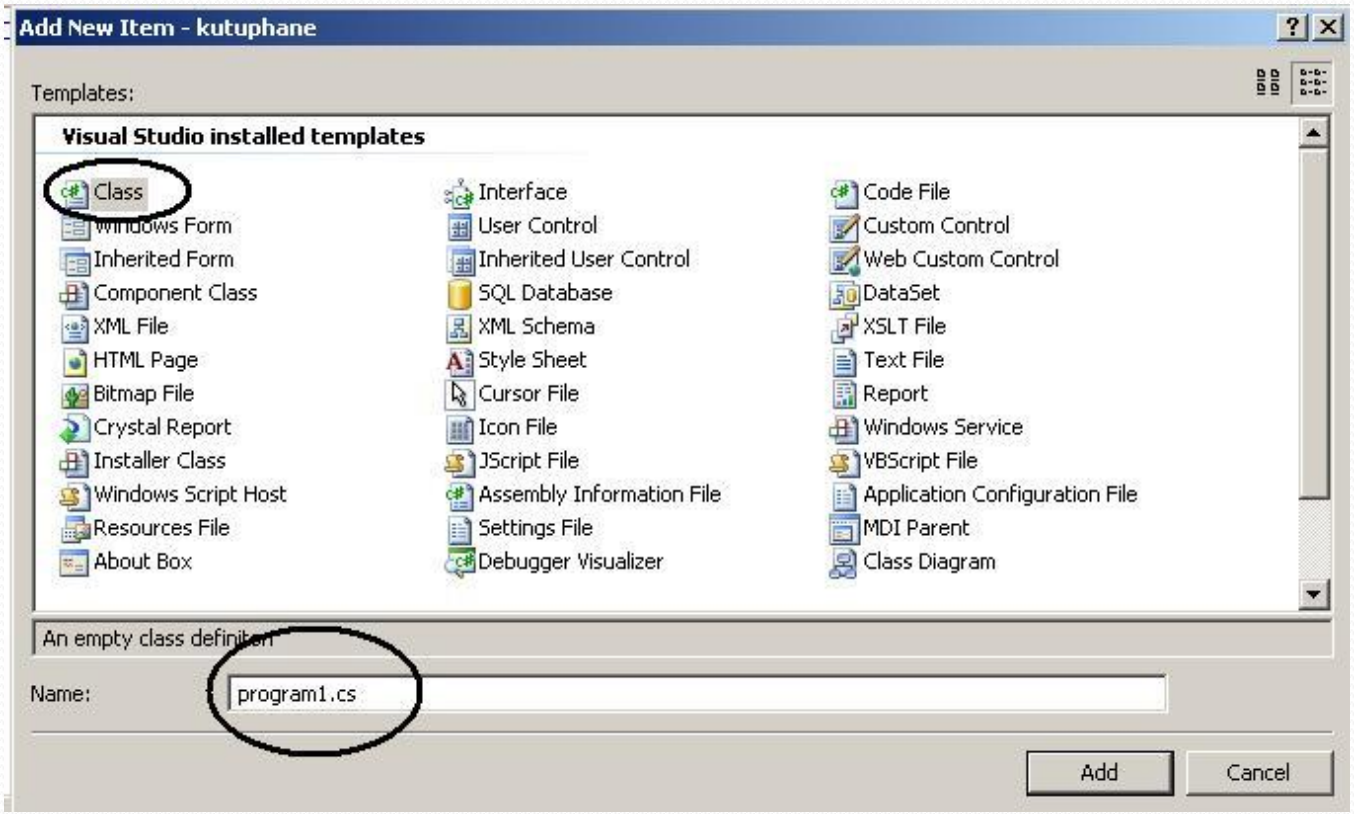
Kütüphane dosyası oluşturma (dll)

- İlk durumda Visual Studio.NET'i kullanarak. Sağ taraftaki 'Solution Explorer' üzerinde sağ buton yaparak 'Add New Item' ile yeni class dosyası eklenir dosyası projeye dahil edilir.



Kütüphane dosyası oluşturma (dll)

- Projeye yeni bir sınıf dosyası ekliyoruz. Metodun saklı olacağı sınıf dosyasını



Kütüphane dosyası oluşturma (dll)

- Solution explorer'a dahil ettik projeyi.F6 derle,F5 çalıştır.

kutup - Microsoft Visual Studio

File Edit View Refactor Project Build Debug Data Tools Window Community Help

Debug | Any CPU

program1.cs | Start Page | Program.cs

toplama.program | topla(float a, float b)

```
using System;
using System.Collections.Generic;
using System.Text;

namespace toplama
{
    class program
    {
        public static float topla(float a, float b)
        {
            return (a + b);
        }
    }
}
```

Solution Explorer - Solution 'kutup' (1 project)

- Solution 'kutup' (1 project)
  - kutuphane
    - Properties
    - References
      - System
      - System.Data
      - System.Xml
    - Program.cs
    - program1.cs

Error List

1 Error | 0 Warnings | 0 Messages

	Description	File
1	The type or namespace name 'toplama' could not be found (are you missing a using directive or an assembly reference?)	Program.cs

Error List | Output

Ready | Ln 1 | Col 1 | Ch 1 | INS

Kütüphane dosyası oluşturma (dll)

- Projeye dahil edilen Program1.cs içindeki toplama alanadını using deyimini kullanarak artık kendi kaynak kodumuz içinde kullanabiliriz.

- Eğer Program1'i dll olarak dahil etmek istersek. Yapmamız gereken komut satırından C# derleyicisini (csc.exe) kullanmak olacaktır. (Csc /? Yardım bilgisini verir)

Visual Studio 2005 Command Prompt

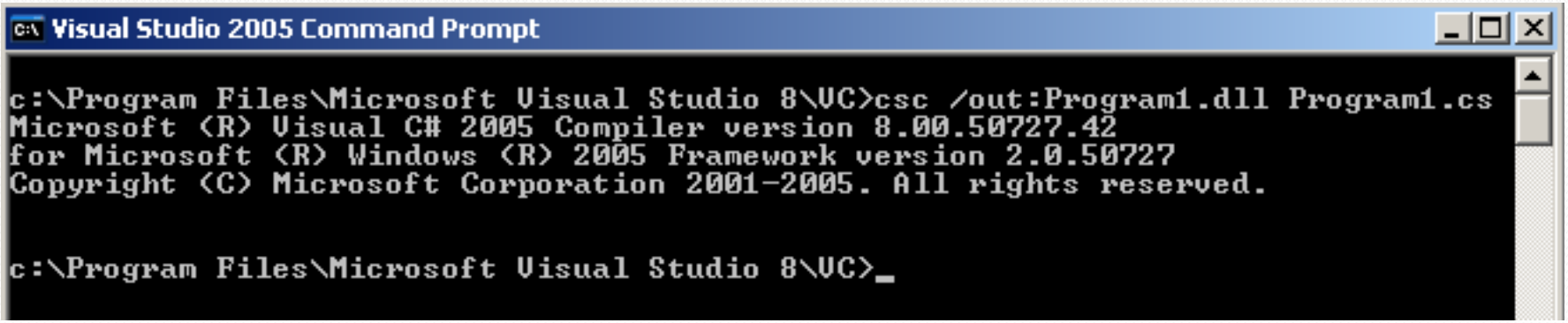
```
Setting environment for using Microsoft Visual Studio 2005 x86 tools.

c:\Program Files\Microsoft Visual Studio 8\VC>csc
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

fatal error CS2008: No inputs specified

c:\Program Files\Microsoft Visual Studio 8\VC>
```

Kütüphane dosyası oluşturma (dll)

- İlk önce dahil edilecek olan Program1.cs dosyasını dll olarak derleyeceğiz.
- Program1.dll dosyasını elde ettikten sonra bu dosyayı Program.cs dosyasını derlerken referans olarak dahil edeceğiz.

> csc /out:Program1.dll Program1.cs veya csc/t:library Program1.cs

Artık DLL dosyamız elimizde.



Kütüphane dosyası oluşturma (dll)

- dll dosyamızı şimdi Program.cs dosyasına referans olarak göstereceğiz. Sonuçta referans ettiğimiz yerdeki alanadları veya metotlara ulaşabileceğiz.Buradaki ‘ /r ‘ parametresi referans edilen dosyayı göstermektedir.

.. > csc /r:Program1.dll Program.cs

Artık çalıştırılabilir Program.exe elde edilmiş oldu.

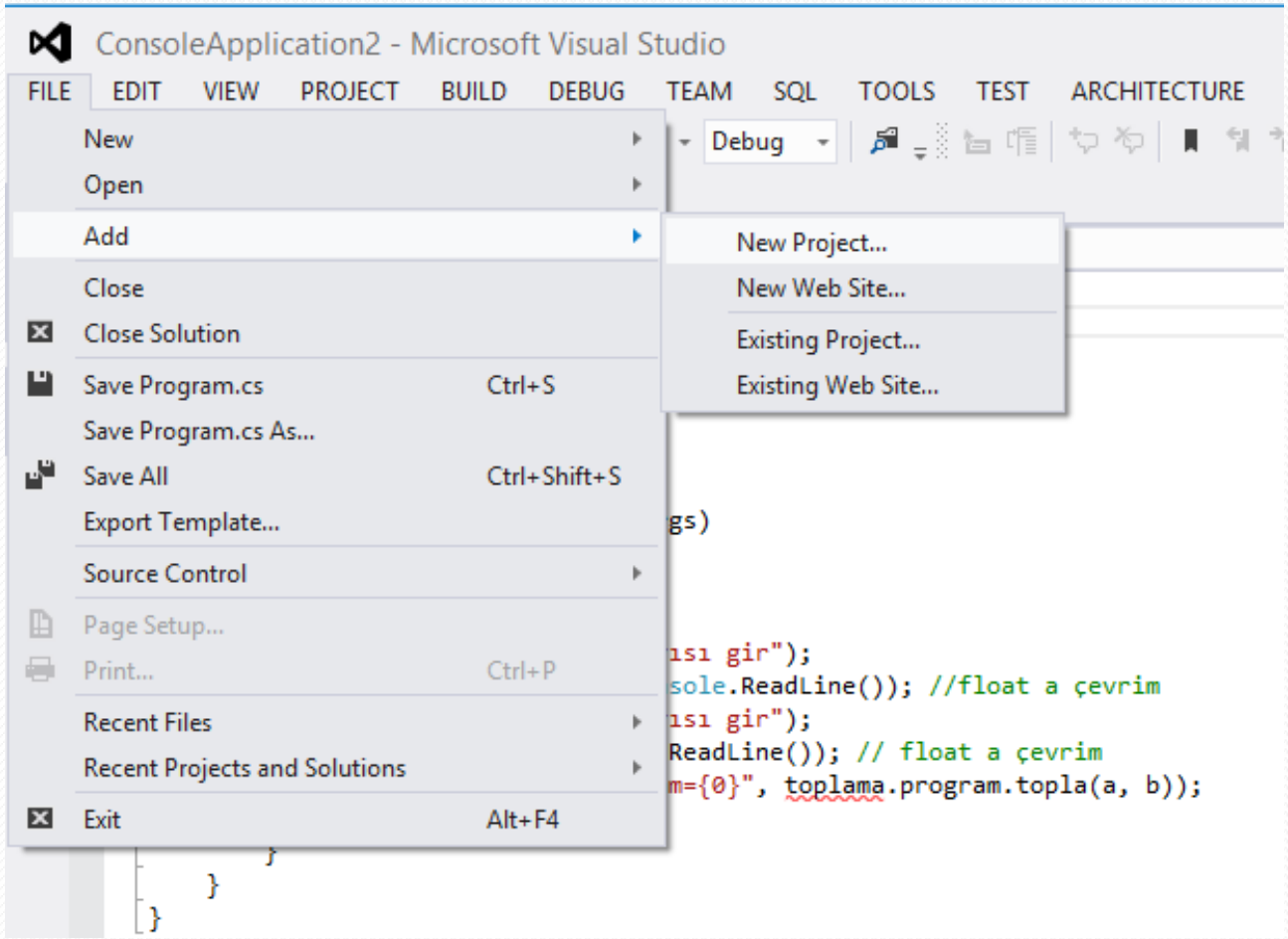
```
c:\Program Files\Microsoft Visual Studio 8\VC>csc /r:Program1.dll Program1.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.


c:\Program Files\Microsoft Visual Studio 8\VC>dir Program.exe
 C sürücüsü birimi: Yavuz
 Birim Seri Numarası: 94BE-CDDC

 c:\Program Files\Microsoft Visual Studio 8\VC dizini

03/08/2006  06:06             3,584 Program.exe
               1 Dosya            3,584 bayt
               0 Dizin   18,564,861,952 bayt boş

c:\Program Files\Microsoft Visual Studio 8\VC>
```

Birden fazla dll dosyası eklenecek ise

csc /r:a.dll /r:b.dll /r:c.dll Program.cs

Kütüphane dosyası oluşturma (dll)

- Visual Studio ortamında Kütüphane dosyalarını oluşturabiliriz.



New Project

- Recent
- Installed
  - Templates
    - Visual C#
      - Windows Store
      - Windows
      - Web
      - Office
      - Cloud
      - Reporting
      - SharePoint
      - Silverlight
      - Test
      - WCF
      - Windows Phone
      - Workflow
    - LightSwitch
    - Other Languages
    - Other Project Types
    - Modeling Projects
  - Samples
- Online

.NET Framework 4 | Sort by: Default | Search Installed Templates (Ctrl+E)

Windows Forms Application	Visual C#
WPF Application	Visual C#
Console Application	Visual C#
ASP.NET Web Forms Application	Visual C#
Class Library	Visual C#
Portable Class Library	Visual C#
Blank App (XAML)	Visual C#
ASP.NET MVC 3 Web Application	Visual C#
ASP.NET MVC 4 Web Application	Visual C#
Grid App (XAML)	Visual C#
Silverlight Application	Visual C#

Type: Visual C#

A project for creating a C# class library (.dll)

Name: toplama

Location: C:\Users\etanyildizi\Documents\Visual Studio 2012\Projects

Browse...

Solution: Create new solution

Solution name: toplama

☑ Create directory for solution

☐ Add to source control

OK | Cancel

Kütüphane dosyası oluşturma (dll)

ConsoleApplication2 - Microsoft Visual Studio

Quick Launch (Ctrl+Q)

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP

Start Debug

Class1.cs | Object Browser | Program.cs

toplama.program | topla(float a, float b)

```
namespace toplama
{
    class program
    {
        public static float topla(float a, float b)
        {
            return (a + b);
        }
    }
}
```

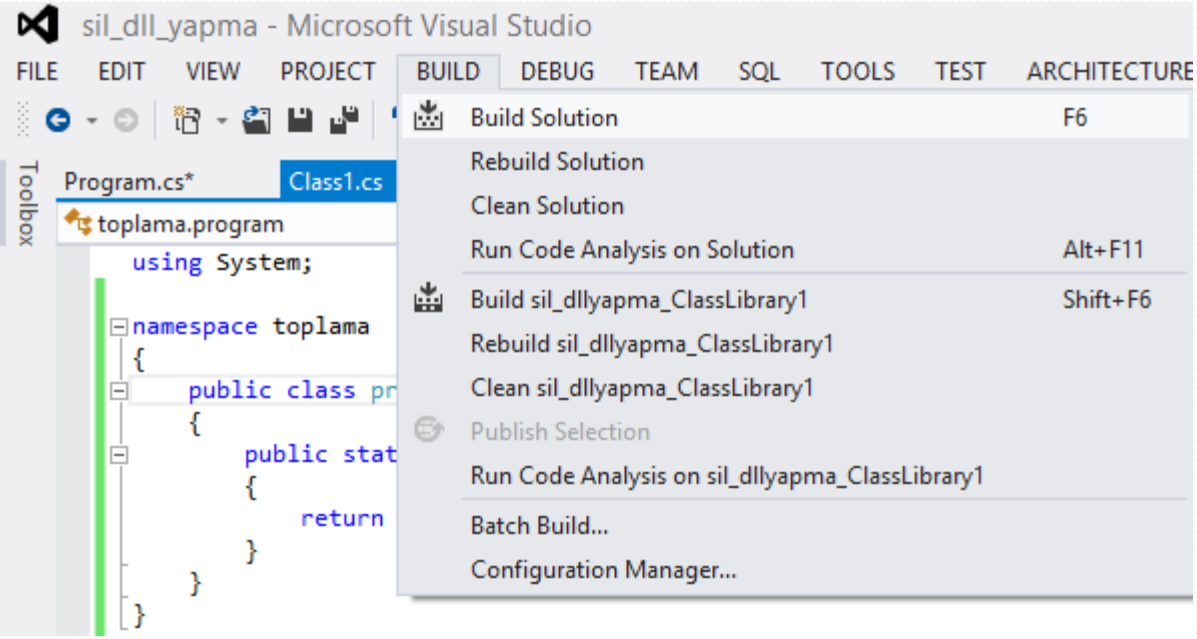
Solution Explorer

Search Solution Explorer (Ctrl+ş

- Solution 'ConsoleApplicati
  - ConsoleApplication2
    - Properties
      - AssemblyInfo.cs
    - References
    - bin
    - obj
    - App.config
    - Program.cs
  - toplama
    - Properties
    - References
    - Class1.cs

Kütüphane dosyası oluşturma (dll)

- F6 ile projeyi build ediniz ve kütüphane dosyasını ana projeye ekleyiniz. Farklı dosya içerisinde olduğundan sınıf public olmalı.



Kütüphane dosyası oluşturma (dll)

- Add Reference diyerek oluşturulan dll ana dosyaya eklenir.

Program.cs

kutuphane.program | Main(string[] args)

```
using System;
using toplama;

namespace kutuphane
{
    class program
    {
        static void Main(string[] args)
        {
            float a, b;

            Console.WriteLine("a sayısı gir");
            a = Convert.ToSingle(Console.ReadLine()); //float a çevri
            Console.WriteLine("b sayısı gir");
            b = float.Parse(Console.ReadLine()); // float a çevrim
            Console.WriteLine("toplam={0}", toplama.program.topla(a, b));
            Console.ReadKey();
        }
    }
}
```

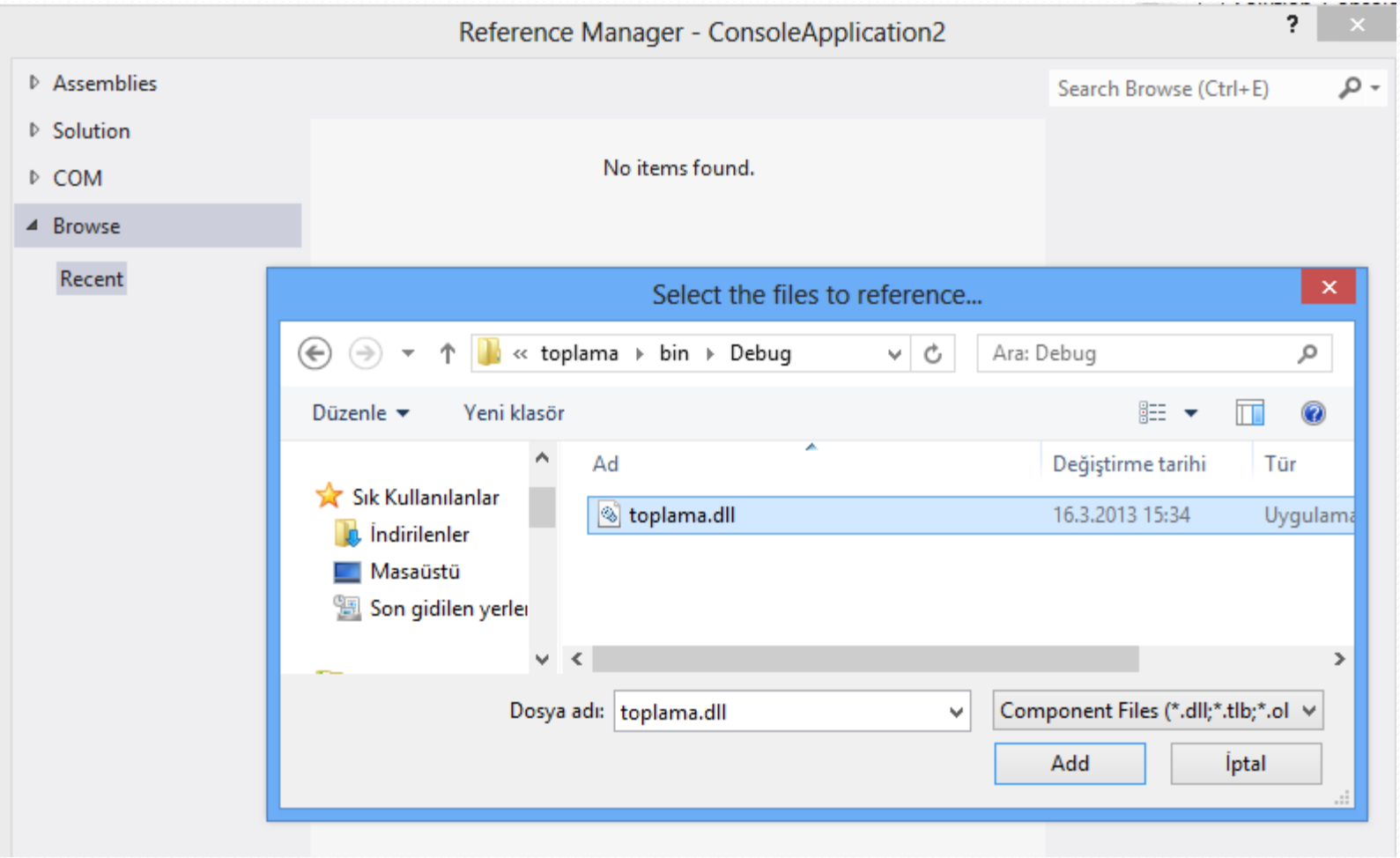
Solution Explorer

Search Solution Explorer (Ctrl-

- Solution 'ConsoleApplic
  - ConsoleApplication2
    - Properties
      - AssemblyInfo.
    - References
    - bin
    - obj
    - Debug
      - Temporary
      - Temporary
      - Temporary
    - App.config
    - Program.cs

Add Reference...
Add Service Reference...
Manage NuGet Packages...
Scope to This
New Solution Explorer View

Kütüphane dosyası oluşturma (dll)



ConsoleApplication2 - Microsoft Visual Studio

Quick Launch (Ctrl+Q)

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP

Start Debug

Class1.cs | Program.cs*

kutuphane.program | Main(string[] args)

```
using System;
using toplama;

namespace kutuphane
{
    class program
    {
        static void Main(string[] args)
        {
            float a, b;

            Console.WriteLine("a sayısı gir");
            a = Convert.ToSingle(Console.ReadLine()); //float a çevrim
            Console.WriteLine("b sayısı gir");
            b = float.Parse(Console.ReadLine()); // float a çevrim
            Console.WriteLine("toplam={0}", toplama.program.topla(a, b));
            Console.ReadKey();
        }
    }
}
```

100 %

Error List

0 Errors | 0 Warnings | 0 Messages | Search Error List

Description	File	Line	Colu...	Project

Solution Explorer

Search Solution Explorer (Ctrl+ş

- Solution 'ConsoleApplication
- ConsoleApplication2
  - Properties
    - AssemblyInfo.cs
  - References
    - Microsoft.CSharp
    - System
    - System.Core
    - System.Data
    - System.Data.DataSe
    - System.Xml
    - System.Xml.Linq
    - toplama
  - bin

Solution... | Team Exp... | Class View

Properties

Server Explorer | Toolbox

Harici Takma İsimler (External Alias)

- Daha önce bahsedildiği gibi, farklı isim alanları altında aynı isimli sınıflar tanımlandığında ve her iki isim alanı da using deyimi ile programa eklendiğinde kod bloğu içersinden sınıflara erişmek istediğimizde takma isimler tanımlamak gerekiyordu.
- Böylelikle referans edilmiş her bir sınıfa takma ismi yardımıyla erişebilmekteydik.
- Bu tarz bir çözüm büyük program parçalarında anlam bütünlüğünün bozulmasına neden olabilir.
- Bunun için C# 2.0'dan itibaren C# dilinde oluşturulmuş olan sınıf kütüphanelerine (dll'ler) harici takma isimler verebilmekteyiz.

Harici Takma İsimler (External Alias)

- İsim alanları ve sınıflar aynı olduğundan herhangi bir sınıfı kullanmaya kalktığımızda çakışma olacaktır.

- Şimdi dosya1.cs ve dosya2.cs dosyalarını oluşturun ve içine şu kodları yazın:
- dosya1.cs

```
namespace IsimAlani
 {
public class bir { }
public class iki { }
}
```

dosya2.cs

```
namespace IsimAlani
 {
public class bir { }
public class iki { }
}
```

Harici Takma İsimler (External Alias)

- Bu iki dosyanın içeriği tamamen aynı. Ancak siz sınıfların içeriği farklıymış gibi düşünün. Şimdi her iki dosyayı da kütüphane dosyası (DLL) hâline getiriniz. Konsol veya proje ekleyerek bunu gerçekleştiriniz.

- csc /r:Dosya1=dosya1.dll /r:Dosya2=dosya2.dll program.cs
  Artık elimizde dosya1.dll ve dosya2.dll adlı iki sınıf kütüphanesi var. Bunlar harici kütüphanelerdir. Ana programı yazarken öncelikle alias lar belirtilecek.
-
- extern alias Dosya1;
- extern alias Dosya2;

Harici Takma İsimler (External Alias)

- **extern alias Dosya1;**
- **extern alias Dosya2;**
- **using System;**
- **class Ana {**
- **static void Main()**
- **{Dosya1::IsimAlani.Sinif a=new Dosya1::IsimAlani.Sinif();**
- **Dosya2 :: IsimAlani.Sinif b=new Dosya2 :: IsimAlani.Sinif(); } }**
- Aynı şekilde bu tür uzun bir nesne yaratımı yapmak istemiyorsak programın başına
- **using** Dosya1::IsimAlani; veya **using** Dosya1.IsimAlani;
- **using** Dosya2::IsimAlani;
- **:: operatörü,** (iki tane iki nokta yan yana) operatörü ile takma isim verilmiş DLL kütüphanlerindeki tür ve isim alanlarına erişiriz.

"::" yerine, "." kullanımı da hata vermez

Add New Project

- Recent
- Installed
  - Visual C#
    - Windows Store
    - Windows
    - Web
    - Office
    - Cloud
    - Reporting
    - SharePoint
    - Silverlight
    - Test
    - WCF
    - Windows Phone
    - Workflow
  - LightSwitch
  - Other Languages
  - Other Project Types
  - Modeling Projects
- Online

.NET Framework 4 | Sort by: Default | Search Installed Templates (Ctrl+E)

Windows Forms Application	Visual C#
WPF Application	Visual C#
Console Application	Visual C#
ASP.NET Web Forms Application	Visual C#
Class Library	Visual C#
Portable Class Library	Visual C#
Blank App (XAML)	Visual C#
ASP.NET MVC 3 Web Application	Visual C#
ASP.NET MVC 4 Web Application	Visual C#
Grid App (XAML)	Visual C#
Silverlight Application	Visual C#
Split App (XAML)	Visual C#
Silverlight Class Library	Visual C#

Type: Visual C#

A project for creating a C# class library (.dll)

Name: ClassLibrary1

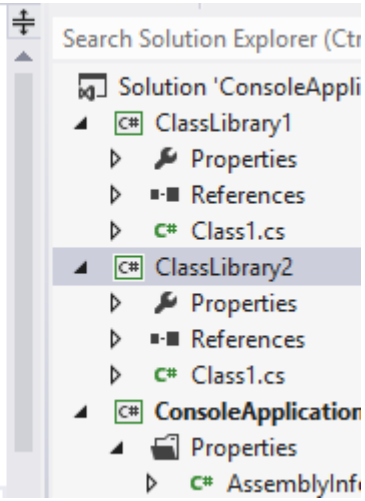
Location: C:\Users\etanyildizi\Documents\Visual Studio 2012\Projects\ConsoleApplication2

Browse...

OK | Cancel

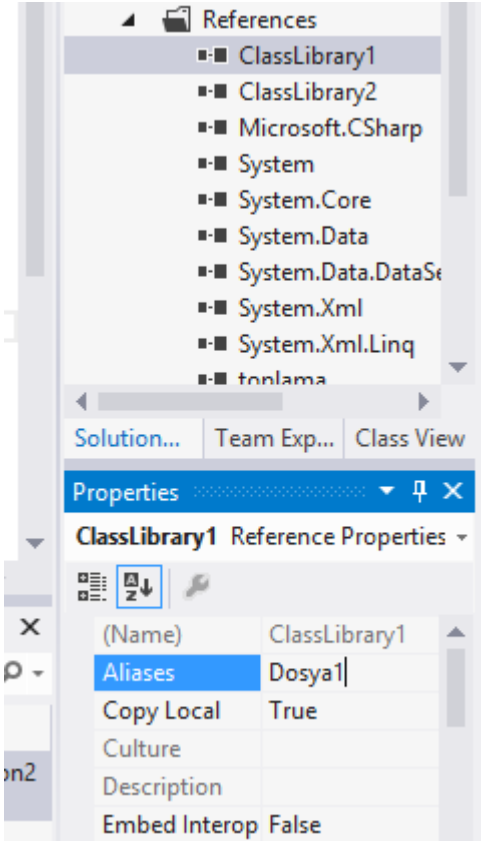
Harici Takma İsimler (External Alias)

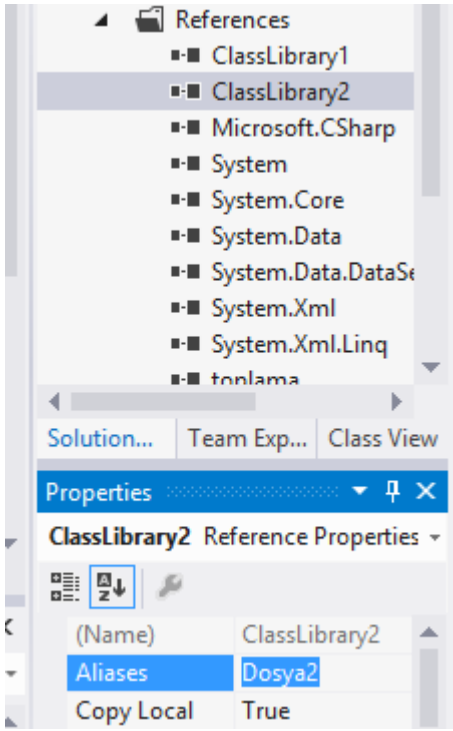
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace isimalani
{
    public class Class1
    {
    }

    public class Class2
    {
    }
}
```



Aynı şekilde ClassLibrary2 oluşturunuz. Ama aynı namespace kullanınız . Daha sonra kütüphane dosyalarını ekleyip Properties özelliğinden Aliases değerini tanımlayınız

Harici Takma İsimler (External Alias)





Harici Takma İsimler (External Alias)

```
extern alias Dosya1;
extern alias Dosya2;// extern ifadeleri using ten önce olmalı
using System;

class Ana {
static void Main()

 {Dosya1::isimalani.Class1 aa=new Dosya1::isimalani.Class1();
  Dosya2.isimalani.Class1 bb=new Dosya2.isimalani.Class1();

}

}
```

Harici Takma İsimler (External Alias)

```
extern alias Dosya1;
extern alias Dosya2; // extern ifadeleri using ten önce olmalı
using System;

using D1 = Dosya1.isimalani;
using D2 = Dosya2::isimalani;

class Ana {
static void Main()

 { D1.Class2 cc = new D1.Class2();
   D2.Class2 dd = new D2.Class2();
  }
}
```

global harici takma ismi

- .Net Framework kütüphanesi, kendi oluşturduğumuz kütüphaneler ve takma ad verilmemiş DLL kütüphanelerimiz otomatik olarak global takma adını alır. Yani programlar aşağıdaki gibi de yazılabilir.
- using System;
- namespace ia
- { class Yardimci
- { public Yardimci()
- { Console.WriteLine("Yardımcı"); } }}

- namespace ana
- { class Sinif
- { static void Main()
- { global::ia.Yardimci a=new global::ia.Yardimci(); } }
- }

global harici takma ismi

- Başka bir örnek:
- `class prg`
- `{`
- `static void Main()`
- `{ global::System.Console.WriteLine("Deneme"); }`
- `}`

- **NOT:** extern ifadeleri using ifadelerinden önce yazılmalıdır.
  NOT: DLL dosyaları direkt türleri içerebileceği gibi isim alanlarını ve isim alanının altında da türleri içerebilir.
  NOT: Eğer türler farklı bir DLL dosyasındaysa türleri public olarak belirtmek gerekir. Ancak aynı dosyada farklı bir isim alanındaysa, ya da dosyaları birlikte derleme söz konusuysa public anahtar sözcüğünü kullanmaya gerek yoktur. Şimdiye kadar öğrendiğimiz bütün türler public anahtar sözcüğüyle belirtilebilir.

System İsim Alanı

- .NET sınıf kütüphanesinde yer alan System isim alanı içersinde oldukça kullanışlı bazı sınıflar bulunmaktadır.
- Bunlardan System.Array, System.Random, System.Convert, System.Math ve System.GC ’den daha önce bahsedilmişti.
- System isim alanı içerisinde yer alan temel veri tiplerinin sahip olduğu bazı metotlar da zaman zaman kullanılmaktadır.

System İsim Alanı

- Tip.Parse(); : string biçimindeki verileri tip türüne çevirir.
- Nesne.CompareTo(object o); : metodu çağrılan nesne ile o nesnesini karşılaştırır. Değerler eşit ise sıfır (**0**), nesne değeri küçük ise negatif (**-1**), büyük ise pozitif (**1**) değer döndürür.
- Nesne.Equals(object o); : metodu çağrılan nesne ile o nesnesini karşılaştırır. Eşit ise **true** aksi halde **false** döndürür.
- Nesne.ToString(); : nesne değerini string şeklinde geri döndürür.

- int a=2; int b=32;
- b.CompareTo(a)
- b<a ise -1, b>a ise 1, b==a ise 0

- b. Equals(a)
- b==a ise true, b!=a ise false

System İsim Alanı

- **double** ve **float** değerler için bir önceki anlatılan metotlara ek olarak aşağıdaki metotlar da bulunur. Bu metotlar **true/false** yani boolean geri dönüş değerine sahiptir:
  - IsInfinity: Sayının sonsuzu temsil edip etmediğini kontrol eder.
  - IsNaN: Anlamlı bir sayı olup olmadığını kontrol eder.
  - IsPositiveInfinitiy, IsNegativeInfinity: Sayının + ya da - sonsuz olup olmadığını kontrol eder.
  - double.IsInfinity(double d);

- **char** veriler içinde bazı metotlar bulunur:
  - char.GetNumericValue(char x) : Eğer değer sayısal karakter içeriyorsa sayısal değeri aksi halde -1 döndürür.
  - Bunun dışında kullanılan metotlar tabloda veriliyor.

System İsim Alanı

- Diğer metotlar,

Metot	Koşul
IsControl	Karakter kontrol karakteri ise
IsDigit	Karakter bir rakam ise
IsLetter	Karakter bir harf ise
IsLetterOrDigit	Karakter bir harf ya da rakam ise
IsLower	Karakter küçük harf ise
IsNumber	Karakter rakam ise
IsPunctuation	Karakter noktalama işareti ise
IsSeperator	Karakter boşluk gibi ayırıcı ise
IsSurroGate	Karakter Unicode yedek karakteri ise
IsSymbol	Karakter sembol ise
IsUpper	Karakter büyük harf ise
IsWhiteSpace	Karakter tab ya da boşluk karakteri ise

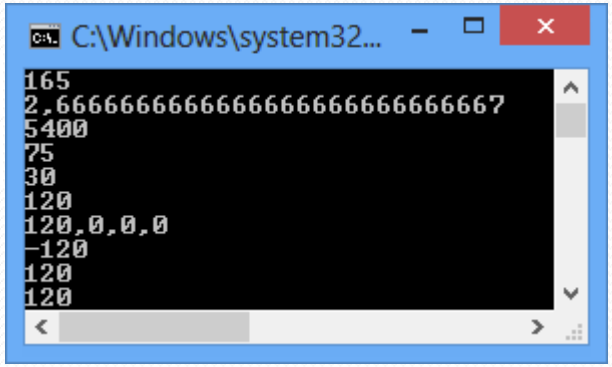
```
char.IsPunctuation("23.",2)→ True
char.IsPunctuation('2')→ False
```

System İsim Alanı

- Decimal değerler içinde bazı metotlar mevcuttur:
- `Decimal.Add(d1, d2); //d1 ve d2'nin toplamını decimal türünden tutar. (+)`
- `Decimal.Divide(d1, d2); //d1'in d2'ye bölümünü decimal türünden tutar. (/)`
- `Decimal.Multiply(d1, d2); //d1 ile d2'nin çarpımını decimal türünden tutar. (*)`
- `Decimal.Subtract(d1, d2); //d1-d2'nin sonucunu decimal türünden tutar. (çıkarma işlemi) (-)`
- `Decimal.Remainder(d1, d2); //d1'in d2'ye bölümünden kalanı decimal türünden tutar. (mod alma işlemi)`
- `Decimal.Floor(d1); //d1'den büyük olmayan en büyük tam sayıyı decimal türünden tutar. (aşağı yuvarlama)`
- `Decimal.GetBits(d1); /*d1 için decimal sayı tanımlarken kullandığımız yapıcı işlevdeki beş parametreyi int türündeki bir dizi olarak tutar.*/`
- `Decimal.Negate(d1); //d1'in negatifini tutar.`
- `Decimal.Round(d1, sayi); /*sayi int türünde olmalıdır. Bu metot ile d1'in ondalık kısmındaki hane sayısı sayi kadar kalır. Yani d1 12.53666 ve sayi 3 ise 12.537 tutulur. Son kaybedilen hane 5 ya da 5'ten büyükse son kalan hane 1 artılır. sayi 0 olursa d1 tam sayıya yuvarlanmış olur.*/`
- `Decimal.Truncate(d1); //d1'in tam sayı kısmını tutar. Herhangi bir yuvarlama yapılmaz.`

System İsim Alanı

```
int sayi = 5;  decimal d1 = 120, d2 = 45;
int[] dizi2 = Decimal.GetBits(d1);
Console.WriteLine(Decimal.Add(d1, d2) + "\n" +
Decimal.Divide(d1, d2) + "\n" +
Decimal.Multiply(d1, d2) + "\n" +
Decimal.Subtract(d1, d2) + "\n" +
Decimal.Remainder(d1, d2) + "\n" +
Decimal.Floor(d1) + "\n" +
dizi2[0] + "," + dizi2[1] + "," +
dizi2[2] + "," + dizi2[3]+ "\n" +
Decimal.Negate(d1) + "\n" +
Decimal.Round(d1, sayi) + "\n" +
Decimal.Truncate(d1));
```



C# Temel Tür Yapıları

- **Tamsayı Tipindeki Yapılar**

Byte	sByte	Int16	UInt16
Int32	UInt32	Int64	UInt64

```
using System;
class class1
{
    public static void Main()
    {
        string a = "25";
        int b = Int32.Parse(a);
        Console.WriteLine(UInt64.MaxValue);
        Console.WriteLine(UInt64.MinValue);
        Console.WriteLine(b);
    }
}
```

C# Temel Tür Yapıları

- **Char Tipinde Yapılar**

ToLower(char str)	str büyük harf ise küçük harfe çevirir.
ToUpper(char str)	str küçük harf ise büyük harfe çevirir.

- `char ch='a'; Console.WriteLine(char.ToUpper(ch) + "\n");`
- Çıktı: A

- **Boolean Tipinde Yapılar**
- Boolean veri yapısında iki tane özellik vardır. Bunlar FalseString ve TrueString'dir. Bu özelliklerde String türündendir, false ve true yazılarını içerirler. CompereTo(), Equals(), Parse() ve ToString() boolean yapısında bulunan diğer metotlardır. Kendisine has bir metodu yoktur.

C# Temel Tür Yapıları

- **DateTime ve TimeSpan**

- C# dilinde tarih ve saat işlemleri System isim alanında bulunan **DateTime** ve **TimeSpan** yapıları ile gerçekleştirilir.

- **DateTime** yıl, ay, gün, saat, dakika, saniye gibi bilgileri tutan bir yapıdır.

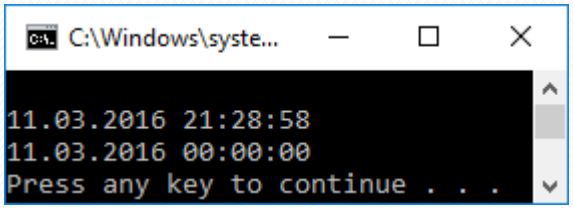
- **TimeSpan** ise iki zaman bilgisi arasındaki farkı temsil etmek için kullanılır.

C# Temel Tür Yapıları

- **DateTime Özellikleri**
- **DateTime.Now:** Sistem saatinin hem tarih hem de zaman bilgisini öğrenmek
  - **DateTime.Now.Today:** Sistem tarihini öğrenmek
  - **DateTime.Now.DayOfWeek:** Haftanın gününü öğrenmek
  - **DateTime.Now.DayOfYear:** Yılın kaçıncı günü olduğunu öğrenmek
  - **DateTime.Now.Day, Month ve Year:** Sistem tarihinin gün ay, yıl bilgisini öğrenmek
  - **DateTime.Now.Hour, Minute, Second ve MilliSecond :** Sistem zamanının saat, dakika, saniye ve milisaniye bilgisini öğrenmek
  - **DateTime.Now.Ticks:** DateTime yapısının bu özelliğinden yararlanılarak bir işlemin ne kadar sürdüğünü öğrenebilirsiniz.
    - **Ticks** özelliği 1 Ocak 0001'den içinde bulunulan ana kadar geçen süreyi tick sayısı cinsinden içermektedir. Bu özelliğin içeriği her 100 nano saniyede 1 artmaktadır.
    - *Saniye = Tick_sayi / 10.000.000* değerini verir
    - İşlemin başında ve sonunda Ticks özelliği kullanılarak ikisi arasındaki farkı alarak programın çalışma süresini bulabilirsiniz.

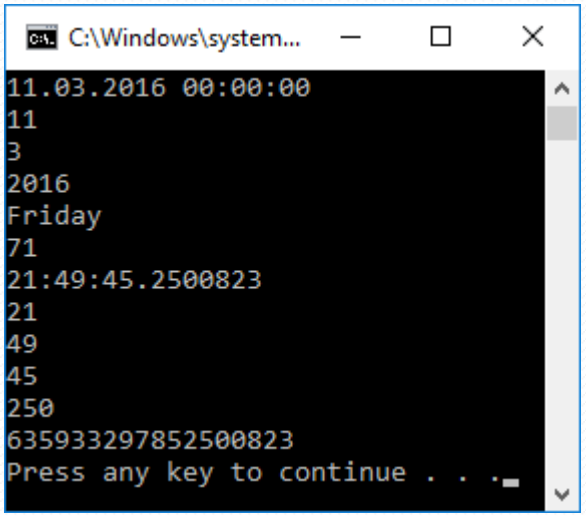
DateTime

```
using System;
class zaman
{
    public static void Main()
    {
        DateTime tarihsaat = new DateTime();
        tarihsaat = DateTime.Now;
        Console.WriteLine(tarihsaat);
        DateTime tarih = new DateTime();
        tarih = DateTime.Today;
        Console.WriteLine(tarih);
    }
}
```



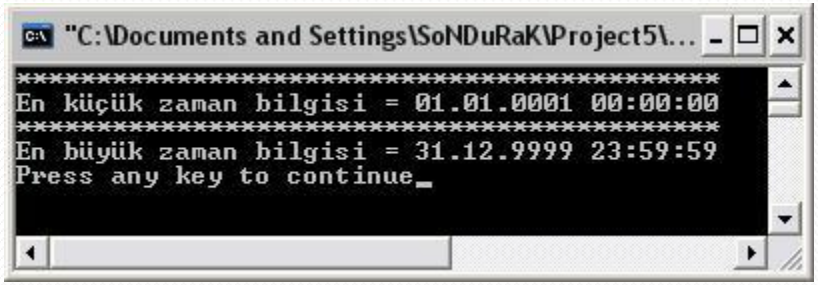
DateTime

```
using System;
class zaman
{ public static void Main()
  { DateTime Bugun = new DateTime();
    Bugun = DateTime.Now;
    Console.WriteLine(Bugun.Date);
    Console.WriteLine(Bugun.Day);
    Console.WriteLine(Bugun.Month);
    Console.WriteLine(Bugun.Year);
    Console.WriteLine(Bugun.DayOfWeek);
    Console.WriteLine(Bugun.DayOfYear);
    Console.WriteLine(Bugun.TimeOfDay);
    Console.WriteLine(Bugun.Hour);
    Console.WriteLine(Bugun.Minute);
    Console.WriteLine(Bugun.Second);
    Console.WriteLine(Bugun.Millisecond);
    Console.WriteLine(Bugun.Ticks);}
}
```



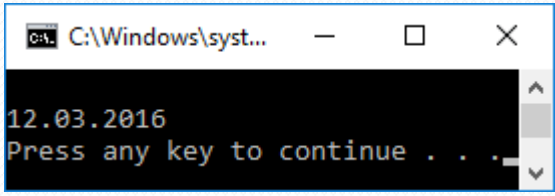
DateTime

```
using System;
class zaman
{
    public static void Main()
    {
     Console.WriteLine("****************************");
     Console.WriteLine("En küçük zaman bilgisi = " + DateTime.MinValue);
     Console.WriteLine("****************************");
     Console.WriteLine("En büyük zaman bilgisi = " + DateTime.MaxValue);
    }
}
```



C# Temel Tür Yapıları

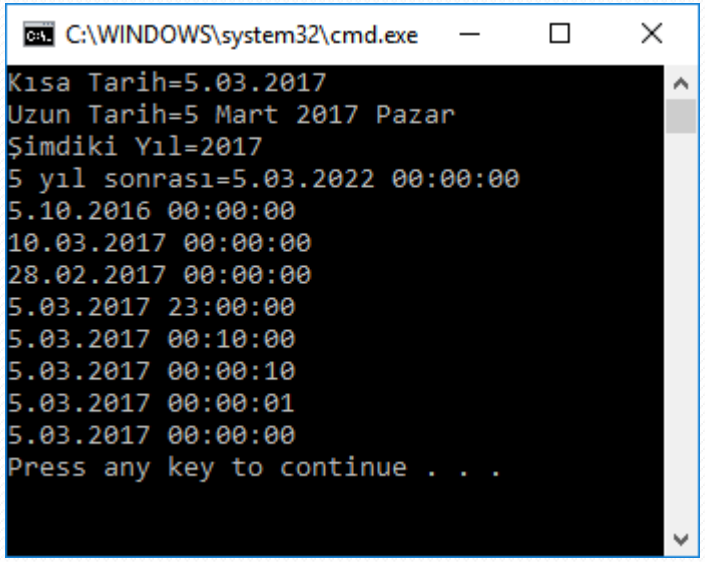
- **DateTime Metotları**
- **ToShortDateString**(): Sistemin sadece tarih kısmını verir.
  - `DateTime Tarih = DateTime.Today;`
    - `Console.WriteLine(Tarih.ToShortDateString());`



- **ToLongDateString():** Sistemin sadece tarih bilgisinin uzun halini verir.
  - `Console.WriteLine(Tarih.ToLongDateString());`
- **Add(TimeSpan value):** Geçerli tarihe belli sayıda gün, tarih veya yıl eklenmek
- **AddDays(), AddMonths(), AddYears(), AddHours(), AddMinutes() AddSeconds() Metotları:** Değişik şekilde elde edilmiş geçerli tarih bilgisine gün, ay, yıl, zaman bilgisine saat, dakika, saniye gibi değerleri ekleyip veya çıkarıp başka bir tarih ve zaman bilgisi elde etme.
  - `Console.WriteLine(Tarih.AddDays(5));` Mevcut tarihin gününden 5 gün sonrasını verir.
  - `Console.WriteLine(Tarih.AddDays(-5));` Mevcut tarihin gününden 5 gün öncesini verir.

C# Temel Tür Yapıları

```
using System;
class zaman
{ public static void Main()
  { DateTime Tarih = DateTime.Today;
    Console.WriteLine("Kısa Tarih="+Tarih.ToShortDateString());
    Console.WriteLine("Uzun Tarih="+Tarih.ToLongDateString());
    Console.WriteLine("Şimdiki Yıl="+Tarih.Year);
    Console.WriteLine("5 yıl sonrası=" + Tarih.AddYears(5));
    Console.WriteLine(Tarih.AddMonths(-5));
    Console.WriteLine(Tarih.AddDays(5));
    Console.WriteLine(Tarih.AddDays(-5));
    Console.WriteLine(Tarih.AddHours(23));
    Console.WriteLine(Tarih.AddMinutes(10));
    Console.WriteLine(Tarih.AddSeconds(10));
    Console.WriteLine(Tarih.AddMilliseconds(1000));
    Console.WriteLine(Tarih.AddTicks(10));
  }
}
```



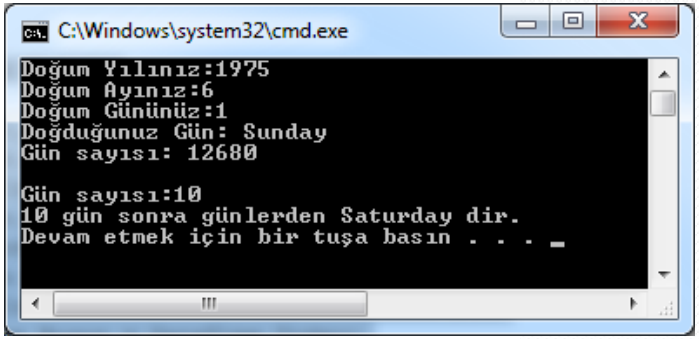
DateTime ve TimeSpan

- **TimeSpan** ve **DateTime** yapıları ile tanımlanmış bazı operatörler bulunur.

- İki tarih arasındaki farkı bulmak için çıkarma (**-**), ileriki bir tarihi hesaplamak için toplama (**+**), iki tarih arasında büyüklük küçüklük karşılaştırması yapmak için de (**<**) ve (**>**) operatörleri aşırı yüklenmiştir.

DateTime ve TimeSpan

```
using System;

class Program
{
    static void Main()
    {
        int yil, ay, gun;
        Console.Write("Doğum Yılınız:");
        yil = Convert.ToInt32(Console.ReadLine());
        Console.Write("Doğum Ayınız:");
        ay = Convert.ToInt32(Console.ReadLine());
        Console.Write("Doğum Gününüz:");
        gun = Convert.ToInt32(Console.ReadLine());

        DateTime Bugun = DateTime.Today;
        DateTime DogumGunu = new DateTime(yil, ay, gun);

        TimeSpan fark = Bugun - DogumGunu;

        Console.WriteLine("Doğduğunuz Gün: {0}", DogumGunu.DayOfWeek);
        Console.WriteLine("Gün sayısı: {0}", fark.Days);

        Console.WriteLine();
        Console.Write("Gün sayısı:");
        gun = Convert.ToInt32(Console.ReadLine());

        TimeSpan GunSayisi = new TimeSpan(gun, 0, 0, 0);
        DateTime Gelecek = DateTime.Today + GunSayisi;
        Console.WriteLine("{0} gün sonra günlerden {1} dir.", gun, Gelecek.DayOfWeek);
    }
}
```



Visual Studio.Net -C#
8. HAFTA
System İsim Alanı

İsim Alanı Bildirimi

```
using System;

namespace PrgDil3
{
    class Deneme
    {
        public int a;
        public int b;

        public Deneme(int a, int b)
        {
            this.a = a;
            this.b = b;
        }

        public static int Topla(int x, int y)
        {
            return x + y;
        }
    }
}

class Program
{
    static void Main()
    {
        PrgDil3.Deneme d = new PrgDil3.Deneme(1,2);
        Console.WriteLine(d.a);
        Console.WriteLine(PrgDil3.Deneme.Topla(2, 3));
    }
}
```

İsim Alanı Bildirimi

```
using System;

namespace isimalan1
{
    class Deneme
    {
        public Deneme()
        {
            Console.WriteLine("Deneme 1");
        }
    }
}

namespace isimalan2
{
    class Deneme
    {
        public Deneme()
        {
            Console.WriteLine("Deneme 2");
        }
    }
}

class Program
{
    static void Main()
    {
        isimalan1.Deneme d1 = new isimalan1.Deneme();
        isimalan2.Deneme d2 = new isimalan2.Deneme();
    }
}
```

Visual Studio.Net -C#
8. HAFTA
System İsim Alanı

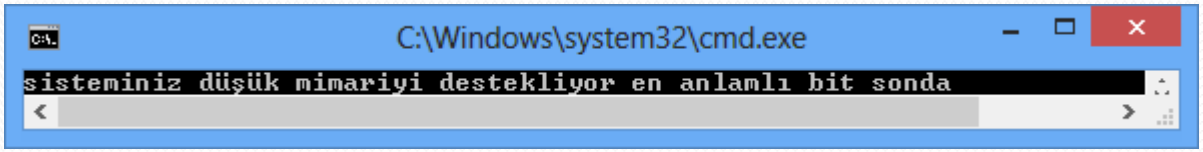
System.BitConverter

- Alt seviye programlama da veriler byte dizisi şeklinde yani bitsel olarak işlenir.
- .NET içersinde de bite çevirmek amacıyla BitConverter sınıfı bulunur.
- Bu sınıfın IsLittleEndian isimli bir özelliği bulunur. Bu özellik işlemci mimarisine göre verileri bellekte depolarken hangi sıralamada yerleştirildiğini kontrol eder.
- En önemli metodu da GetBytes'tır. Amacı da farklı sayı türlerini byte dizisine çevirmektir.

System.BitConverter

- Mimarinizi öğrenmek için;

```
using System;
class bitconverter
{
    public static void Main()
    {
        if (BitConverter.IsLittleEndian)
            Console.WriteLine("sisteminiz düşük mimariyi destekliyor en anlamlı bit sonda");
        else
            Console.WriteLine("sisteminiz yüksek mimariyi destekliyor en anlamlı bit başta");
    }
}
```



System.BitConverter

- `int a=258; // 258/256=1 => 258-256¹ *1 = 2`
- `//00000000 00000000 00000001 00000010`
- 256^3 256^2 256^1 256^0
- `byte[] dizi=BitConverter.GetBytes(a);`
- `foreach(byte b in dizi)`
- `{ Console.WriteLine(b); }`
- 2
- 1
- 0
- 0
- `byte[] dizi={2,1,0,0};` → en anlamlı bit en sonda
  `Console.WriteLine(BitConverter.ToInt32(dizi,0));`
- 258
- Bilinçli tür dönüşümünde kayıplar en anlamlı bitlerde olur.

System.Buffer

- **Buffer** sınıfı ile tür bilgisinden bağımsız bir biçimde **byte** düzeyinde veri işleme yapılır.
- Dizilerin belirli alanları başka bir diziye tür bilgisine bakılmaksızın aktarılabilir. Tüm veriler byte dizisi şeklinde düşünülür.
- BlockCopy(), GetByte(), SetByte() en önemli metotlarıdır.

System.Buffer

- `byte[] kaynak={1,2,3,1};`
- *00000001 , 00000010 , 00000011 , 00000001*
- `short[] hedef=new short[4];`
- *0000000000000000, 0000000000000000 , 0000000000000000 , 0000000000000000*
- `Buffer.BlockCopy(kaynak,0,hedef,0,4);`
- *hedef dizisinin yeni hâli:*
- *0000001000000001 , 0000000100000011 , 0000000000000000 , 0000000000000000*
- `foreach(short a in hedef)  Console.Write(" "+a);`
- Bu program, mimarimiz Little Endian ise ekrana: 513 259 0 0 yazar.
  - Burada derleyici her elemanı bellekte 1 bayt yer kaplayan kaynak dizisinin elemanlarını her elemanı bellekte 2 bayt kaplayan hedef dizisine olduğu gibi kopyaladı. Derleyici burada karşılaştığı ilk baytı düşük anlamlı bayta, ikinci baytı da yüksek anlamlı bayta kopyaladı.

- Bilinçsiz tür dönüşümü Buffer.BlockCopy mantığı ile gerçekleşir.

System.Buffer

- **ByteLength() metodu**
- Kendisine parametre olarak verilen bir dizideki toplam bayt sayısını bulur. Örneğin kendisine parametre olarak gönderilen 3 elemanlı short türünden bir dizi sonucu 6 sayısı gönderir. Geri dönüş tipi int tir. Örnek:
  - `short[] dizi=new short[4];`
  - `Console.WriteLine(Buffer.ByteLength(dizi)); // 8 yazar`

- **GetByte() metodu**
  - `static byte GetByte(Array dizi, int a)`
  - dizinin a. baytını verir.

- **SetByte() metodu**
  - `static void SetByte(Array dizi,int a, byte deger)`
  - a. baytı deger olarak değiştirir.

System.Buffer

- Örnek:
  - byte[] dizi={0,3,2,1,4}; Console.WriteLine(Buffer.GetByte(dizi,3)); // Ekrana 1 yazar
    - Bu kod 1 değerini döndürür. Eğer dizinin tipi byte değil de short olsaydı işler değişirdi. Çünkü o zaman hem sıfırla beslenen baytlar da hesaba katılırdı ve hem de bilgisayarımızın mimarisi sonucu etkilerdi. Bu durumu örneklendirelim:
  - short[] dizi={0,3,2,1,4};
  - 0-> 00000000 00000000 , 2->00000000 00000011, 4->00000000 00000010 ,
  - 1 0 3 2 5 4
  - 6-> 00000000 00000001, 8-> 00000000 00000100
  - 7 6 9 8
  - Console.WriteLine(Buffer.GetByte(dizi, 4)); // Ekrana 2 yazar
  - Console.WriteLine(Buffer.GetByte(dizi,8)); // Ekrana 4 yazar
- Örnek:
  - byte[] dizi={2,1,4}; Buffer.SetByte(dizi,1,5);
  - Console.WriteLine(dizi[1]); *//Ekrana 5 yazılır.*
  - Yine eğer dizinin türü byte değil de int olsaydı işler değişirdi.

C# Temel Tür Yapıları

- **Convert Sınıfı**

```
using System;
class ortalamaHesapla
{   public static void Main()
    {   Console.Write("Vize1 = ");
        int vize1 = Convert.ToInt32(Console.ReadLine());
        Console.Write("Vize2 = ");
        int vize2 = Convert.ToInt32(Console.ReadLine());
        Console.Write("Final = ");
        int final = Convert.ToInt32(Console.ReadLine());
        double a = ((double)vize1 * 0.2 + (double)vize2 * 0.3 +
(double)final * 0.5);
        Console.WriteLine("Ortalama = {0}", a);
    }
}
```

- C# 6.0 dan sonra {deger} şeklinde yazımı

```
Console.WriteLine($"Ortalama = {a}");
```

String İşlemleri

- Programlarda kullanılan verilerin büyük bir çoğunluğu string türündendir.
- C# da string işlemleri System.String sınıfı içeresinde yer alan üye metot ve özelliklerle yapılır. String veriler için indeksleyici de tanımlanmıştır. Yani bir karakter dizisi gibi işlem görebilir. Fakat karakterler salt okunurdur.
- Stringler değişik şekillerde tanımlanabilirler:

- `string a = “C# Programlama Dili”;`
- `char[] dizi = {’1’,’2’,’3’,’4’,’5’};`
- `String s = new String(dizi);//12345`
- `String s = new String(dizi,1,2); //dizi[1]'den itibaren 2 eleman stringe atandı. → 23`
- `String s = new String(‘x’,10);//xxxxxxxxxx`

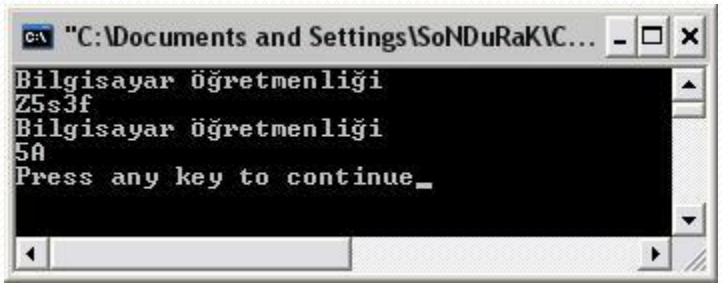
String İşlemleri

- String sınıfının metotları oldukça fazladır. Önemli metotlarından bazıları şunlardır:
- **string.Concat()**
  - `static string Concat(params Array stringler)`
  - String verilerin ardarda eklenmesini sağlar. + operatörü ile eşdeğerdir.
- **string.Compare()**
  - İki string değeri karşılaştırır. **==** ve **!=** operatörleri ile benzer işlem gerçekleştirir. Fakat Compare metodunun bazı aşırı yüklemeleri ile daha gelişmiş (büyük küçük harf duyarlılığı gibi.) karşılaştırmalar yapılabilir.
  - **static int** Compare(**string** a, **string** b)
  - **static int** Compare(**string** a, **string** b, **bool** c)
  - **static int** Compare(**string** a, **int** indeks1, **string** b, **int** indeks2)
  - **static int** Compare(**string** a, **int** indeks1, **string** b, **int** indeks2, **bool** c)
  - kıyaslamada ilk elemanların a[indeks1] ve b[indeks2] sayılmasıdır. Yani stringlerdeki bu elemanlardan önceki elemanlar yok sayılır.

String İşlemleri

- String.Concat()

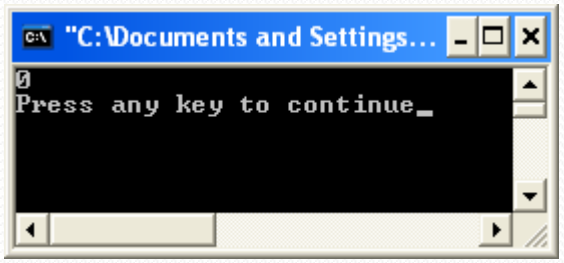
```
using System;
class Concat
{    public static void Main()
    {   String str1 = String.Concat("Bilgisayar", " Öğretmenliği");
        Console.WriteLine(str1);
        String str2 = String.Concat("Z", "5", "s", "3", "f");
        Console.WriteLine(str2);
        String str3 = "Bilgisayar" + " Öğretmenliği";
        Console.WriteLine(str3);
        String str4 = String.Concat(5, "A");
        Console.WriteLine(str4);
    }
}
```



String İşlemleri

- String.Compare()

```
using System;
class class1
{  public static void Main()
   { string a="Ali";
     bool b=true;//false büyük küçük duyarsız
     string s="aLi";
     int c=String.Compare(a,s,b);
    Console.WriteLine(c);
   }
}
```



String İşlemleri

- stringnesne.CompareTo(string str) : İki stringi karşılaştırır. Değerler eşit ise 0, nesne değeri küçük ise negatif, büyük ise pozitif değer döndürür.
- stringnesne.IndexOf()
  - string içersinde alt stringlerin aranmasını sağlar. Geriye aranan alt stringin bulunduğu konumu ya da bulunamadı (-1) bilgisini döndürür.
  - int IndexOf(string a)
  - int IndexOf(char b)
- stringnesne.LastIndexOf()
  - IndexOf ile aynı çalışır Farkı ise aranan karakterin en son nerede görüldüğünün indeksini geri döndürür.
  - int LastIndexOf(string a)
  - int LastIndexOf(char b)

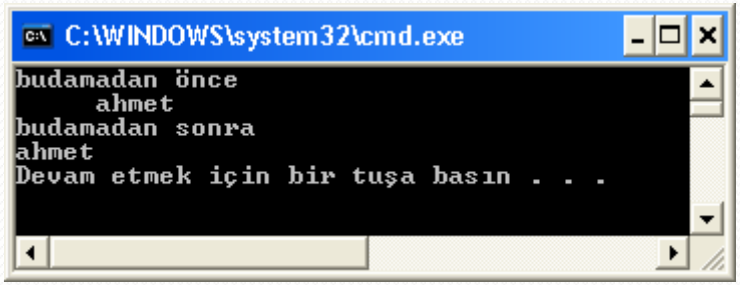
String İşlemleri

- IndexOf
- using System;
- class arama
- { public static void Main()
- { string yazı="fırat üniversitesi";
- Console.WriteLine(yazı.IndexOf("ver"));
- Console.WriteLine(yazı.IndexOf('t'));
- Console.WriteLine(yazı.IndexOf('c'));
- }
- }

String İşlemleri

- stringnesne.Trim()
  - Bir string ifadenin başındaki ve sonundaki boşlukları ya da belirlenmiş karakterleri atar.

```
using System;
class tarih  {
    public static void Main()
    {   string a = "     ahmet    ";
        Console.WriteLine("budamadan önce");
        Console.WriteLine(""+a);
        a = a.Trim ();
        Console.WriteLine("budamadan sonra");
        Console.Write("" + a);
    }
}
```

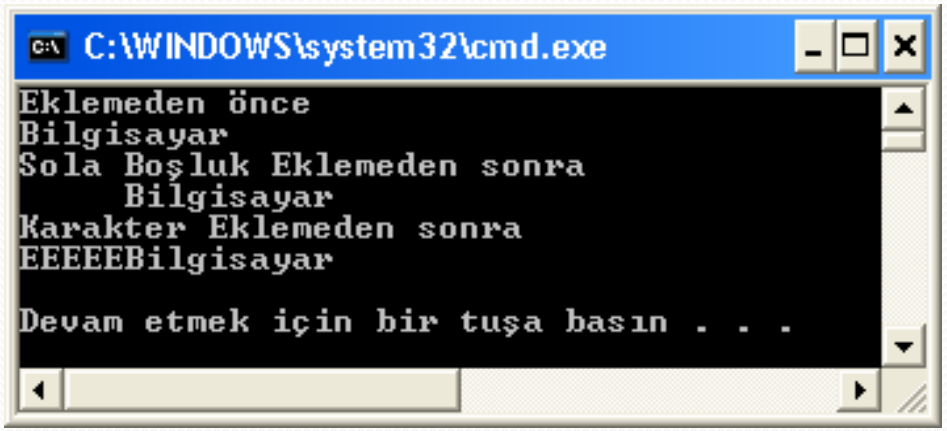


String İşlemleri

- PadRight, PadLeft
  - Bir string'in sağına ya da soluna yeni karakterler ilave etmek için kullanılır.
  - string PadRight (int boyut);
    - stringin uzunluğu boyuta eşit olana kadar stringin sağına boşluk ekler
  - string PadRight (int boyut, char a);
    - stringin uzunluğu boyuta eşit olana kadar stringin sağına 'a' karakterini ekler
  - string PadLeft (int boyut);
    - stringin uzunluğu boyuta eşit olana kadar stringin soluna boşluk ekler
  - string PadLeft (int boyut, char a);
    - stringin uzunluğu boyuta eşit olana kadar stringin soluna 'a' karakterini ekler

String İşlemleri

```
using System;
class tarih
{
    public static void Main()
    {
        string a = "Bilgisayar";
        Console.WriteLine("Eklemeden önce");
        Console.WriteLine(""+a);
        a = a.PadLeft (15);
        Console.WriteLine("Sola Boşluk Eklemeden sonra");
        Console.WriteLine("" + a);
        Console.WriteLine("Karakter Eklemeden sonra");
        a = "Bilgisayar";
        a = a.PadLeft (15,'E');
        Console.WriteLine("" + a);
        Console.WriteLine("");
    }
}
```

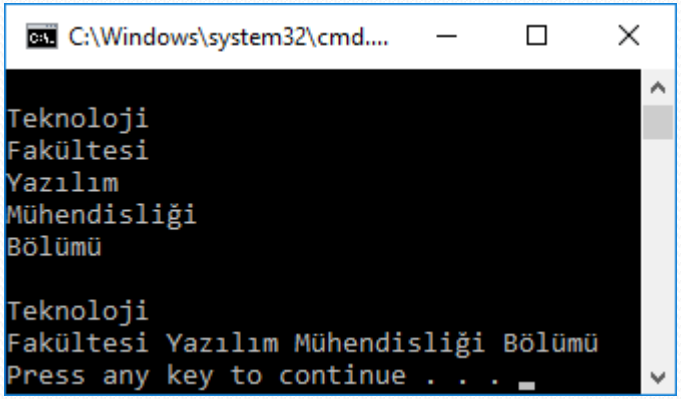


String İşlemleri

- **bool string.StartsWith(string a) :** Metodu çağıran "**string a**" ile başlıyorsa **true**, diğer durumlarda **false** değeri üretir.
- **bool string.EndsWith(string b) :** Metodu çağıran "**string b**" ile bitiyorsa **true**, diğer durumlarda **false** değeri üretir.
- `stringnesne.Split()`
  - Belli bir biçime sahip olan toplu string verileri, belirtilen ayırıcı karakterlerden ayırıp ayrı bir string dizisi üretir.
    - `string [] Split(params char[] ayırıcı)`
    - `string [] Split(params char[] ayırıcı, int toplam)`
    - İkinci metotta ise bu parçalama işlemi en fazla toplam sayısı kadar yapılır.
- `string.Join()`
  - Split metodunun tersi gibi çalışır. Ayrı stringleri belli bir ayırıcı karakter ile birleştirip tek bir string üretir.
    - `static string join(string ayırıcı,string[] yazılar)`
    - `static string join(string ayırıcı,string[] yazı,int başla,int toplam)`
    - İkinci metotta ise [basla] ‘dan itibaren toplam kadar yazı elemanı birleştirilir.

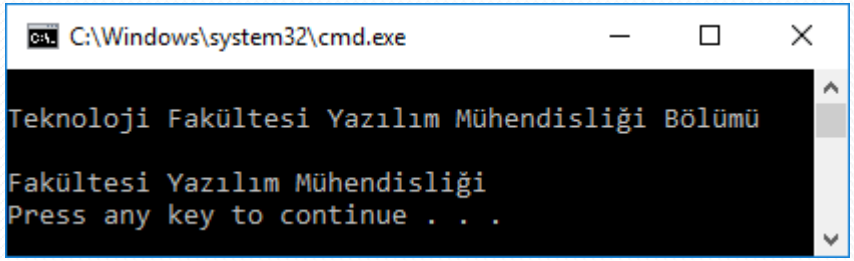
String İşlemleri

```
using System;
class Ayırmaişlemi
{ public static void Main()
 {
 string str="Teknoloji Fakültesi Yazılım Mühendisliği Bölümü";
 char[] ayırıcı={' '};
 string[] ayır=str.Split(ayırıcı);
 foreach(string i in ayır)
   Console.WriteLine(i);
 Console.WriteLine();
 ayır = str.Split(ayırıcı,2);
 foreach (string i in ayır)
   Console.WriteLine(i);
 }
}
```



String İşlemleri

```
using System;
class bireştirmeişlemi
{ public static void Main()
  {
 string [] str={"Teknoloji", "Fakültesi", "Yazılım",
"Mühendisliği", "Bölümü"};
 string birles=String.Join(" ",str);
 Console.WriteLine(birles);
 Console.WriteLine();
 birles = String.Join(" ", str,1,3);
 Console.WriteLine(birles);
 }
}
```



String İşlemleri

- stringnesne.ToUpper()
  - Stringin karakterlerinin hepsini büyük harfe çevirip geri döndürür.
- stringnesne.ToLower()
  - Stringin karakterlerinin hepsini küçük harfe çevirip geri döndürür.
- stringnesne.Remove(**int** indeks, **int** adet)
  - String içinden belli sayıda karakterin atılmasını sağlar.
- stringnesne.Insert(**int** indeks, **string** str)
  - String içine belirli indeksten itibaren yeni bir string eklemek için kullanılır.
- stringnesne.Replace(**char |string** c1,**char|string** c2)
  - String içindeki belirlenen karakter ya da karakterleri başkalarıyla değiştirir.
- stringnesne.SubString(**int** indeks |**int** indeks,**int** toplam)
  - String ifadenin belli bir kısmının elde edilmesini sağlar.

String İşlemleri

```
C:\Windows\system32\cmd.exe
TEKNOLOJİ FAKÜLTESİ YAZILIM MÜHENDİSLİĞİ BÖLÜMÜ
teknoloji fakültesi yazılım mühendisliği bölümü
teknoloji yazılım mühendisliği bölümü
teknoloji Fakültesi yazılım mühendisliği bölümü
teknoloji Fakültesi Sazılım mühendisliği bölümü
Fakültesi Sazılım mühendisliği bölümü
mühendisliği
Press any key to continue . . .
```

```
using System;
class digermetotlar
{
   public static void Main()
  {
  string str="Teknoloji Fakültesi Yazılım Mühendisliği Bölümü";
  str=str.ToUpper();                Console.WriteLine(str+"\n");
  str=str.ToLower();                Console.WriteLine(str+"\n");
  str=str.Remove(9,10);             Console.WriteLine(str+"\n");
  str=str.Insert(10,"Fakültesi "); Console.WriteLine(str+"\n");
  str=str.Replace('y','S');         Console.WriteLine(str+"\n");
  str=str.Substring(10);            Console.WriteLine(str+"\n");
                                    //10. karakterden sonrasını gösterir.
  str = str.Substring(18,12);       Console.WriteLine(str + "\n");
                                    //18. karakterden sonra 12 karakter gösterir.
 }
}
```

Biçimlendirme

- Program çıktılarının belli bir düzende olması oldukça önemlidir. Bazı zamanlar standart çıktıların anlaşılması zor olabilir.

- Bu problemi çözmek için biçimlendirme teknikleri kullanılır. Bu teknikler yalnızca biçimlendirmeyi destekleyen komutlar tarafından kullanılabilir.

- **Console.WriteLine(), String.Format()** ve **ToString()** metotları biçimlendirmeyi destekleyen metotlardır.

- Bu metotlarda kullanılan **{ }** parantezleri arasındaki ifadeler belli değişkenlerin belli bir düzende biçim metnine aktarılmasını sağlıyor:
  - Console.WriteLine("Merhaba **{**0**}**!",isim)

Biçimlendirme

- En genel kullanımı
  - **{ değişken_no, genişlik : format}**
  - **int** a=54;
- Sadece değişken verildiğinde değişkenin türüne göre varsayılan ayarlar kullanılır.
- **Genişlik,** yazılacak olan verinin diğer verilerle olan mesafesini ve hangi yöne hizalanması gerektiğini belirler.
  - Console.WriteLine("{0,10} numara",a);
  - Console.WriteLine("{0,-10} numara",a);
  - 54 numara
  - 54 numara
- **Format,** ise verinin türüne göre değişik biçimlendirilmesini sağlar.

Biçimlendirme

Belirleyici	Açıklama	Duyarlılık Anlamı
C/c	Para birimi	Ondalık basamakların sayısını verir.
D/d	Tam sayı verisi	En az basamak sayısını belirtir, gereğinde bos olan basamaklar sıfır ile beslenir.
E/e	Bilimsel notasyon	Ondalık basamak sayısını verir.
F/f	Gerçek sayılar (float)	Ondalık basamak sayısını verir.
G/g	E ve F biçimlerinden hangisi kısa ise o kullanılır	Ondalık basamak sayısını verir.
N/n	Virgül kullanarak gerçek sayıları yazar	Ondalık basamak sayısını verir.
P/p	Yüzde	Ondalık basamak sayısını verir.
X/x	Onaltılık sayı sisteminde yazar.	En az basamak sayısını belirtir, gereğinde boş olan basamaklar sıfırla beslenir.

Biçimlendirme

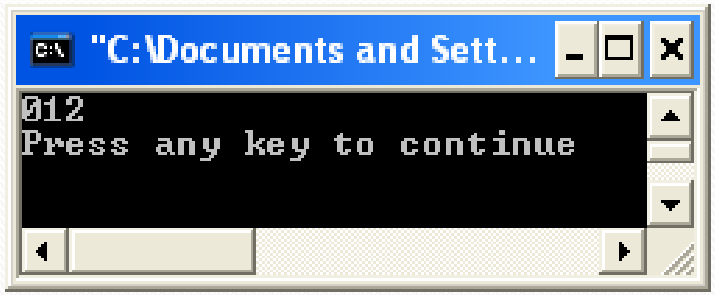
```
using System;
class Formatlama {
   static void Main() {
      float f=568.87f;
      int a=105;
      Console.WriteLine("{0:C3}",a);
      Console.WriteLine("{0:D5}",a);
      Console.WriteLine("{0:E3}",f);
      Console.WriteLine("{0:F4}",f);
      Console.WriteLine("{0:G5}",f);
      Console.WriteLine("{0:N1}",f);
      Console.WriteLine("{0:P}",a);
      Console.WriteLine("{0:P}",1);
      Console.WriteLine("{0:X5}",a);
      Console.WriteLine("{0:C3}",f); } }
```

```
105,000 TL
00105
5,689E+002
568,8700
568,87
568,9
%10.500,00
%100,00
00069
568,870 TL
```

Biçimlendirme

- **String.Format() ve ToString() Metotları İle Biçimlendirme**

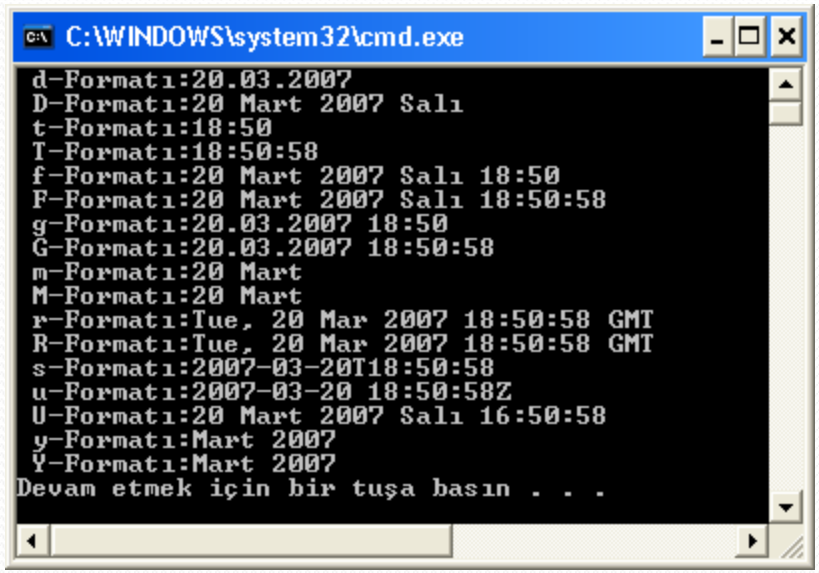
```
class StringFormat
{ public static void Main()
 { int a=12;
   string str=String.Format("{0:d3}",a);
   Console.WriteLine(str);
 }
}
```




```
class Tostring
{ public static void Main()
 { int a=12;
   string str=a.ToString("C3");
   Console.WriteLine(str);
 }
}
```

Tarih ve Saat Biçimlendirme

```
using System;
class TarihveSaat
{
 public static void Main()
 { DateTime d=DateTime.Now;
   Console.WriteLine(" d-Formatı:{0:d}",d);
   Console.WriteLine(" D-Formatı:{0:D}",d);
   Console.WriteLine(" t-Formatı:{0:t}",d);
   Console.WriteLine(" T-Formatı:{0:T}",d);
   Console.WriteLine(" f-Formatı:{0:f}",d);
   Console.WriteLine(" F-Formatı:{0:F}",d);
   Console.WriteLine(" g-Formatı:{0:g}",d);
   Console.WriteLine(" G-Formatı:{0:G}",d);
   Console.WriteLine(" m-Formatı:{0:m}",d);
   Console.WriteLine(" M-Formatı:{0:M}",d);
   Console.WriteLine(" r-Formatı:{0:r}",d);
   Console.WriteLine(" R-Formatı:{0:R}",d);
   Console.WriteLine(" s-Formatı:{0:s}",d);
   Console.WriteLine(" u-Formatı:{0:u}",d);
   Console.WriteLine(" U-Formatı:{0:U}",d);
   Console.WriteLine(" y-Formatı:{0:y}",d);
   Console.WriteLine(" Y-Formatı:{0:Y}",d);
 }
}
```



Biçimlendirme

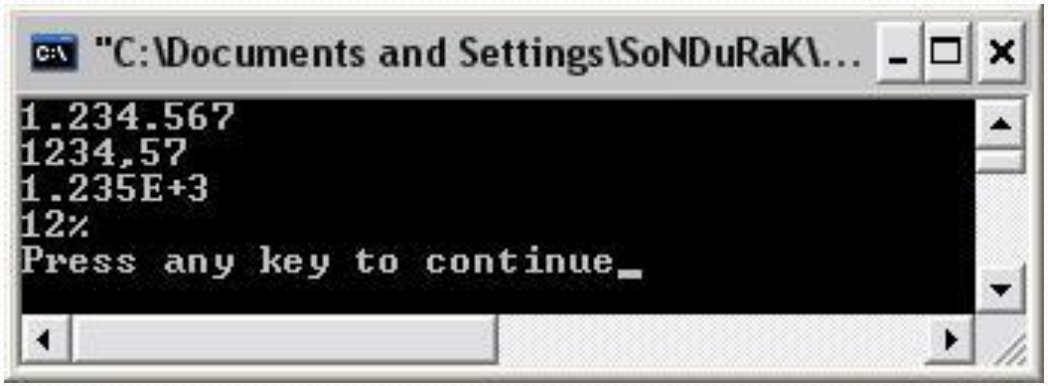
- Standart biçimlendirmelerin dışında özel biçimlendirmeler de tasarlanabilir.
- **“#” “,” “.” “0” “E”** ve **“%”** karakterleri ile özel biçimlendirmeler oluşturulabilir.

  - **#** → Rakam değerleri için kullanılır.
  - **,** → Büyük sayılarda binlikleri ayırmak için kullanılır.
  - **.** → Gerçek sayılarda ondalıklı kısımları ayırmak için kullanılır.
  - **0** → Yazılacak karakterin başına ya da sonuna 0 ekler.
  - **%** → Yüzde ifadelerini belirtmek için kullanılır.
  - **E0**, **e0**, **E+0**, **e+0**, **E-0**, **e-0** → Sayıları bilimsel notasyonda yazmak için kullanılır.

  - **{0:#,###.##}**
  - **{0:#%}**
  - **{0:#,###E+0}**

Biçimlendirme

```
using System;
class özelbiçimlendirme
{
  public static void Main()
  { Console.WriteLine("{0:#,###}",1234567);
    Console.WriteLine("{0:#.##}",1234.5678);
    Console.WriteLine("{0:#,###E+0}",1234567);
    Console.WriteLine("{0:#%}",0.123);
  }
}
```



Biçimlendirme

- **Düzenli İfadeler**
- Düzenli ifadeler değişken sayıda karakterden oluşabilecek ancak belirli koşulları sağlayan ifadelerdir.

- Örneğin e-posta adreslerini düşünebiliriz. Dünyada milyonlarca e-posta adresi olmasına ve farklı sayıda karakterden oluşabilmesine rağmen hepsi kullaniciadi@domainismi.domain tipi düzenindedir.

- Örneğin iletisim@microsoft.com bu düzenli ifadeye uymaktadır.
- C#'taki düzenli ifade işlemleri temel olarak System.Text.RegularExpressions isim alanındaki Regex sınıfı ile yapılmaktadır.

- Bir karakter dizisinin oluşturulan düzenli ifadeye uyup uymadığını yine bu isim alanındaki Match sınıfıyla anlarız.

- Düzenli ifadeler başlı başına bir kitap olabilecek bir konudur. Burada sadece ana hatları üzerinde durulacaktır.

Biçimlendirme

- **Düzenli İfadelerin Oluşturulması**
- Bir ifadenin mutlaka istediğimiz karakterle başlamasını istiyorsak ^ karakterini kullanırız. Örneğin ^9 düzenli ifadesinin anlamı yazının mutlaka 9 karakteri ile başlaması demektir. "9Abc" yazısı bu düzene uyarken "f9345" bu düzene uymaz.
- Belirli karakter gruplarını içermesi istenen düzenli ifadeler için \ karakteri kullanılır:
  - **\D** ifadesi ile yazının ilgili yerinde rakam olmayan tek bir karakterin bulunması gerektiği belirtilir.
  - **\d** ifadesi ile yazının ilgili yerinde 0-9 arası tek bir karakterin bulunması gerektiği belirtilir.
  - **\W** ifadesi ile yazının ilgili yerinde alfanumerik olmayan karakterin bulunması gerektiği belirtiliyor. Alfanumerik karakterler a-z, A-Z ve 0-9 aralıklarındaki karakterlerdir.
  - **\w** ifadesi ile yazının ilgili yerinde bir alfanumerik karakterin bulunması gerektiği belirtiliyor.
  - **\S** ifadesi ile yazının ilgili yerinde boşluk veya tab karakterleri haricinde bir karakterin olması gerektiği belirtiliyor.
  - **\s** ifadesi ile yazının ilgili yerinde yalnızca boşluk veya tab karakterlerinden biri bulunacağı belirtiliyor.

Biçimlendirme

- Bu öğrendiğimiz bilgiler ışığında 5 ile başlayan, ikinci karakteri rakam olmayan, üçüncü karakteri ise boşluk olan bir düzenli ifade şöyle gösterilebilir:
  - **^5\D\s**
  - Tahmin edersiniz ki aynı zamanda burada düzenli ifademizin yalnızca 3 harfli olabileceği belirttik. Yukarıdaki ^5\D\s ifadesine filtre denilmektedir.
- Belirtilen gruptaki karakterlerden bir ya da daha fazlasının olmasını istiyorsak + işaretini kullanırız.
  - **\w+**
  - filtresi ilgili yerde bir ya da daha fazla alfanumerik karakterin olabileceğini belirtiyor. "123a" bu filtreye uyarken "@asc" bu filtreye uymaz. + yerine * kullansaydık çarpıdan sonraki karakterlerin olup olmayacağı serbest bırakılırdı.
- Birden fazla karakter grubundan bir ya da birkaçının ilgili yerde olacağını belirtmek için | (mantıksal veya) karakteri kullanılır. Örnek:
  - **m|n|s**
  - ifadesinde ilgili yerde m, n ya da s karakterlerinden biri olmalıdır. Bu ifadeyi parantez içine alıp sonuna + koyarsak bu karakterlerden biri ya da birkaçının ilgili yerde olacağını belirtmiş oluruz:
  - **(m|n|s)+**

Biçimlendirme

- Sabit sayıda karakterin olmasını istiyorsak {adet} şeklinde belirtiriz. Örnek:
  - **\d{3}-\d{5}**
  - filtresine "215-69857" uyarken "54-34567" uymaz.
- **?** karakteri, hangi karakterin sonuna gelmişse o karakterden en az sıfır en fazla bir tane olacağı anlamına gelir. Örnek:
  - **\d{3}B?A**
  - Bu filtreye "548A" veya "875BA" uyarken "875BBA" uymaz.
- . (nokta) işareti ilgili yerde "\n" dışında bir karakterin bulunabileceğini belirtir. Örneğin
  - **\d{3}.A**
  - filtresine "123sA" ve "8766A" uyar. "236\nA"; uymaz
- **\b** bir kelimenin belirtilen yazıyla sonlanması gerektiğini belirtir. Örnek:
  - **\d{3}dır\b**
  - filtresine "123dır" uyarken "123dırb" uymaz.

Biçimlendirme

- **\B** karakteri sınırlandırma olmayan durumu belirtir.. Örnek:
- \d{3}dır\B
  - filtresine "123dır" veya "dır123" uymazken "123dır8" uyar.
- Köşeli parantezler kullanarak bir karakter aralığı belirtebiliriz. Örneğin ilgili yerde sadece büyük harflerin olmasını istiyorsak [A-Z] şeklinde, ilgili yerde sadece küçük harfler olmasını istiyorsak [a-z] şeklinde, ilgili yerde sadece rakamlar olmasını istiyorsak [0-9] şeklinde belirtebiliriz. Ayrıca sınırları istediğimiz şekilde değiştirebiliriz. Örneğin [R-Y] ile ilgili yerde yalnızca R ve Y arası büyük harfler olabileceğini belirtiriz.

- **Regex sınıfı**
- Regex sınıfı bir düzenli ifadeyi tutar. Bir Regex nesnesi şöyle oluşturulur:
- Regex nesne=new Regex(**string** filtre);
- Yani bu yapıcı metoda yukarıda oluşturduğumuz filtreleri parametre olarak veririz. Regex sınıfının Match metodu ise kendisine gönderilen bir yazının düzenli ifadeye uyup uymadığını kontrol eder ve uyan sonuçları Match sınıfı türünden bir nesne olarak tutar.

Biçimlendirme

- **Match sınıfı**
- Match sınıfının NextMatch() metodu bir Match nesnesindeki bir sonraki düzenli ifadeyi döndürür. Yazının düzenli ifadeye uyup uymadığının kontrolü ise Match sınıfının Success özelliği ile yapılır.
- Eğer düzenli ifadeye uygun bir yapı varsa Success özelliğinin tuttuğu değer true olur.
- Örnek:
- A\d{3}(a|o)+
- Bu filtreyle düzenli ifademizin şöyle olduğunu söyleyebiliriz:
- İlk karakter A olacak.
- A'dan sonra üç tane rakam gelecek.
- Üç rakamdan sonra "a" ya da "o" karakterlerinden biri ya da birkaçı gelecek.

Biçimlendirme

```
using System;
using System.Text.RegularExpressions;
class duzenli
{  static void Main()
   {  string filtre=@"A\d{3}(a|o)+";
      Console.Write("Yazı girin: ");
      string yazi=Console.ReadLine();
      Regex nesne=new Regex(filtre);
      Match a=nesne.Match(yazi);
      Console.WriteLine(a.Success);
      Console.WriteLine(a.ToString());
      Console.WriteLine(a.Index);
      Console.WriteLine(a.NextMatch());
     Console.WriteLine(a.Index);
   }
}
```

Bu programda kullanıcının

A123aA657oA456oao

girdiğini varsayarsak ekran çıktısı şöyle olur.

```
True
A123a
0
A657o
10
```

Visual Studio.Net -C#

8. HAFTA

Kalıtım

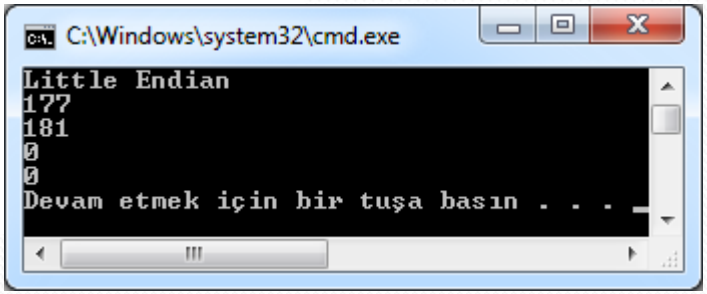
Sanal Metotlar

Özet Metotlar

Arayüzler

System.BitConverter

```
using System;

class Program
{
    static void Main()
    {
        if (BitConverter.IsLittleEndian)
            Console.WriteLine("Little Endian");
        else
            Console.WriteLine("Big Endian");

        int a = 46513;   // 256^0*177+256^1*181+256^2*0+256^3*0=46513

        byte[] b = BitConverter.GetBytes(a);

        foreach (byte x in b)
            Console.WriteLine(x);
    }
}
```



System.Buffer Örnek

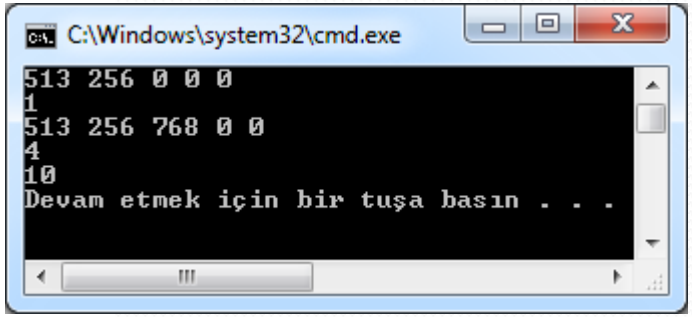
- Short (2 byte)
- 0.byte 1.byte 2.byte 3.byte 4.byte
- 1 2*256 0 1*256 0

- 5.byte // (SetByte ile 5. byte ekleme)
- 3*256

```
using System;

class Program
{
    static void Main()
    {
        byte[] kaynak = { 1, 2, 0, 1 };
        short[] hedef = new short[5];

        Buffer.BlockCopy(kaynak, 0, hedef, 0, 4);

        foreach (short s in hedef)
            Console.Write(s + " ");

        Console.WriteLine("\n" + Buffer.GetByte(hedef, 0));
        Buffer.SetByte(hedef, 5, 3);

        foreach (short s in hedef)
            Console.Write(s + " ");

        Console.WriteLine();
        Console.WriteLine(Buffer.ByteLength(kaynak));
        Console.WriteLine(Buffer.ByteLength(hedef));
    }
}
```

Byte 8 bit
Short 16 bit

$2*256^1+1*256^0=513$
$1*256^1+0*256^0=256$
$3*256^1+0*256^0=768$



Visual Studio.Net -C#

9. HAFTA

Kalıtım

Sanal Metotlar

Özet Metotlar

Arayüzler

Nesne Yönelimli Programlama

- Modern dillerin bir çoğunda nesneye yönelimli programlama tekniği desteklenmektedir. Bu teknik yazılım geliştirmeyi kısaltan ve sistematik hale getiren bir yapıdır.

- C# dili de bu tekniği tamamıyla desteklemektedir. Nesne yönelim tekniği, gerçek hayatı programlar için simule edecek yöntemlerin birleşimidir.

- Bu teknikte geliştirilmek istenen sistem parçalara ayrılır ve bu parçalar arasında ilişkiler kurulur. Parçalar hiyerarşik ya da bağımsız olabilir.

- Bağımsız bileşenler birbirleriyle haberleşerek etkileşimde bulunurlar.

Nesne Yönelimli Programlama

- Nesne yönelimli programlama kavramlarından bazıları şunlardır:

- Nesne yönelimli programlama tekniğinin en temel bileşeni nesnelerdir. Nesneler içeriklerinde veriler barındırırlar. Veriler arası ilişkiler sağlayan fonksiyonlara da sahiptirler. **Nesnelerin veri ve fonksiyon gibi bileşenleri içermesine sarmalama (encapsulation) denilir.**

- Nesne içindeki veriler ve fonksiyonlar nesnenin dışarıya nasıl hizmet verdiğini belirler. Fakat bu hizmeti nasıl verdiği belli değildir. **Nesnenin hizmetlerinden faydalanmak için nesnenin dış dünyadan erişilen ara yüzünün bilinmesi yeterlidir. Buna bilgi saklama (information hiding) adı verilir.**

- Nesnelerin birbirlerinden bağımsız olmasına rağmen aralarında haberleşebilirler. **Hangi nesnenin hangi nesneye mesaj göndereceği, hangi nesnelerin fonksiyonlarının kullanılacağı derleme aşamasında belli olmayabilir. Bu durumda geç bağlama (late binding) mekanizmasından faydalanılır.**

Nesne Yönelimli Programlama

- Tüm nesneler birer sınıf örneğidir. Sınıflar nesnelerin özelliklerini belirlerler. Nesneler derleme ya da çalışma anında oluşturulabilir.

- Kalıtım ile nesneler birbirinden türetilebilir. Türeyen sınıf diğer sınıfın tüm özelliklerini ve kendine has özellikleri içerebilir. Kalıtım yolu ile türetilmiş sınıflar ile hiyerarşik sınıf organizasyonu gerçekleştirilebilir.

- Nesneye yönelimli programlama tekniğinde nesneler çok biçimli olabilir. Çok biçimlilik (polymorphism) kavramı türeme ile alakalıdır ve anlamı bir nesnenin farklı şekillerde davranabilmesidir.

- Temel sınıfa ait nesne referansı üzerinden bir çok sınıfa (türemiş sınıfa) ait farklı versiyonlardaki metotların çağrılabilmesi çok biçimlilik (polymorphism) olarak adlandırılır. Bu konuya daha sonra değinilecektir.

Nesne Yönelimli Programlama

- **Nesne Kavramı**

- Gerçek dünyadaki varlığını bildiğimiz bir çok şey birer nesnedir. Nesne yönelimli programlama tekniğinde de sınıflar nesnelerin biçimlerini belirlerler. Oluşturulan nesneler sınıf türünden nesne olarak adlandırılır. Her nesne kendi içinde tutarlı bir yapıya sahiptir yani veriler arasında sıkı bir bağ bulunur ki bu nesne mantığının temelidir.

- Sınıflardan nesneler oluşturmak için **new** anahtar sözcüğü kullanılıyordu.
  - `Sinif nesne = new Sinif();`

- Sınıflar nesnelerin şeklini belirlerler. Yani nesnenin türünü tanımlarlar. Kısaca sınıflar bir tür bilgisidir.

Kalıtım (Inheritance)

- **Kalıtım,** nesne yönelimli programlama tekniğinin en önemli özelliğidir. Kalıtım yolu ile sınıflar birbirinden türetilir.

- Türeyen sınıflar türedikleri sınıfın özelliklerini kalıtım yoluyla devralırlar ve kendisi de yeni özellikler tanımlayabilir. Türetme ile sınıflar arasında hiyerarşik bir yapı kurulabilir.

- Örnek vermek gerekirse dünya üzerinde yaşayan canlıları sınıflandırmak mümkün. Bu sınıflardan bir tanesi de hayvanlar olabilir. Kedi, Köpek, Kuş, Balık gibi bir çok hayvan türünden bahsedilebilir. Her türün kendine ait değişik özellikleri olabilir. Dolayısıyla her biri için değişik sınıfların tasarlanması gerekebilir.

Kalıtım (Inheritance)

- Fakat ortak bir takım özelliklerinin olması da kaçınılmazdır ve her biri için bağımsız sınıflar tasarlandığında bu benzerlikler her birisi için tekrarlanmak durumunda kalacaktır . Bu yüzden önce tüm hayvanlar için bir sınıf oluşturulup diğer kedi, köpek gibi sınıflar bu sınıfın devamı gibi tasarlanabilir. İşte temelde bir sınıf tanımlanıp diğer sınıfları bu sınıftan türeterek özelleştirmeye kalıtım yoluyla türetme adı verilir.

- Türetme yapmak için sınıf tanımlaması şu şekilde yapılmalıdır:

```
class TüretilenSınıf : TemelSınıf
```

- Türetme işleminden sonra türetilen sınıf temel sınıfın bütün özelliklerine sahip olur.

Kalıtım (Inheritence)

- Tip güvenliği olan dillerde farklı türdeki nesnelerin birbirine atanması istisna durumlar dışında yasaktır. Bu istisna durumlardan biri de türemiş sınıfın referansının temel sınıfa ilişkin bir referansa atanabilmesidir. Bu durumda temel sınıf türeyen sınıfın tüm özelliklerine erişemeyecek olmasına rağmen atama işlemi yapılabilmektedir.

Örneğin:

Bir **Hayvan sınıfı** oluşturulsun; her hayvanın boy, ağırlık gibi fiziksel özellikleri olsun. Ardından bu sınıftan bir **Kedi sınıfı** tanımlayalım. Hayvan sınıfında boy ve ağırlığı gösteren **OzellikGoster()** isimli bir metot olsun. Kedi sınıfında da hayvanın kedi olduğunu gösteren string tipinde özel bir değişken tanımlı olsun.

Kalıtım (Inheritence)

```
class Hayvan //temel sınıf
{
public double boy;
public double kilo;
public void OzellikGoster()
{
Console.WriteLine("Boy="+boy);
Console.WriteLine("Agırlık="+kilo);
}
}
```

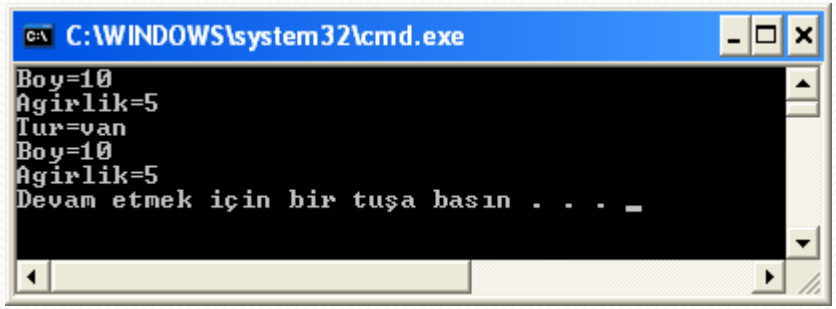
- Ayrıca Kedi sınıfında kedinin türünü yazacak bir de metot olsun. Türetme aşağıdaki şekilde gerçekleştirilir;

```
class Kedi : Hayvan//türetilmiş sınıf
{
    public string Turu;
    public void TurGoster()
    {Console.WriteLine("Tur="+Turu);}
}
```

- Burada Kedi türetilmiş (derived) sınıf, Hayvan da temel (based) sınıftır.

Kalıtım (Inheritence)

```
class MainMetot
{ static void Main()
 {    Kedi k1= new Kedi();
      k1.kilo=5;
      k1.boy=10;
      k1.Turu="van";
      k1.OzellikGoster();
      k1.TurGoster();
      Hayvan h1 = new Hayvan();
      h1.kilo = 5;
      h1.boy = 10;
      //h1.Turu=“van”; //HATA!!
      h1.OzellikGoster();
      //h1.TurGoster();//HATA!!
 }
}
```



Kalıtım (Inheritence)

- Kalıtım yolu ile public ve protected elemanlar aktarılır. Diğer sınıfların kullanımına kapalı ancak türetme ile türemiş sınıfa geçebilen özellikler protected anahtar sözcüğü kullanılır.

- Eğer türetme söz konusu değilse protected olarak bilinen elemanlarla private olanlar arasında bir fark olmayacaktır.

- private özelliklere türetilen sınıflardan erişilemez.
- protected özellikler ise türeyen sınıfa private olarak geçer.

Kalıtım (Inheritence)

- C#'ta yapıcı metotların türetimiyle ilgili şu kurallar geçerlidir:
  - C#'ta yapıcı metotlar fiziksel olarak türetilmez.
  - Türemiş sınıf türünden bir nesne yaratıldığında önce ana sınıfın parametre almayan yapıcı metodu, ardından türemiş sınıftaki imzaya uyan yapıcı metot çalıştırılır.
  - Türemiş sınıf türünden nesne yaratımında daima türemiş sınıfın imzaya uyan bir yapıcı metodu olması gerekir.
  - Türemiş sınıf türünden nesne yaratımlarında, ana sınıfın parametre almayan yapıcı metodu yavru sınıfın üye elemanlarıyla işlem yapar.
  - Türemiş sınıf türünden nesne yaratımında, Türemiş sınıftaki ilgili yapıcı metoda **base** takısı eklenmişse ana sınıfın parametre almayan yapıcı metodu çalıştırılmaz.

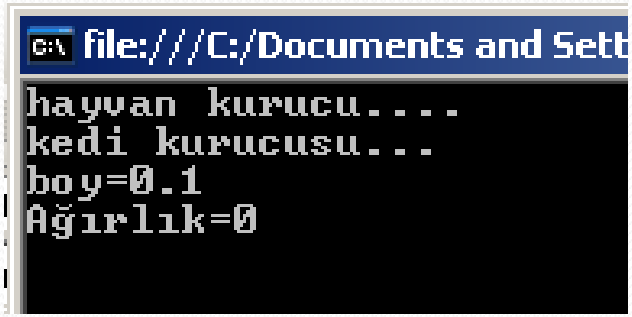
Kalıtım (Inheritence)

```
public Hayvan()
{
Console.WriteLine("hayvan kurucu....");
 boy = 0.1; kilo = 0.2;
    } // temel sınıf yapıcısı
public Kedi()
{  kilo = 0;
   Console.WriteLine("kedi kurucusu...");
} // Türmeiş sınıf yapısıcısı
```

- Main içinde tanımlı satırlar;

```
Kedi k1 = new Kedi();
k1.OzellikGoster();
```

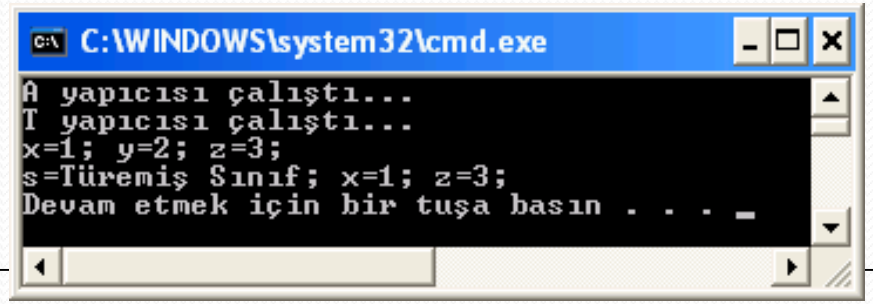
Ekran çıktısı;



İlk önce temel kurucu çalıştı, boy ve ağırlık değerlerini 0.1 ve 0.2 değerlerine eşitledi, ardından türemişin kurucusu çalıştı ve ağırlık değerini 0 yaptı, boy değerini ise türemiş kurucuda herhangi bir işlem yapılmadığı için temel sınıfın kurucu değerini kullanmış oldu.

Kalıtım (Inheritence)

```
using System;
class A
{  public int x;  private int y;
   protected int z;
   public A()
   {x = 1; y = 2;z = 3;
   Console.WriteLine ("A yapıcısı çalıştı...");
   }
public void Listele()
{
Console.WriteLine("x={0};y={1};z={2};",x,y,z);
}
}
class T : A
{ public string s;
  public T()
 { s = "Türemiş Sınıf";
   Console.WriteLine("T yapıcısı çalıştı...");
 }
public void Yaz()
{
Console.WriteLine("s={0};x={1};z={2};",s,x,z);
 }
}
class Program
{
    static void Main()
    {
        T t = new T();
        t.Listele();
        t.Yaz();
    }
}
```



Kalıtım (Inheritence)

Programdaki hatayı bulunuz.

```
using System;
class ana
{
public ana(int a){….}
}

class yavru:ana {…. }

class esas
{
static void Main() { yavru y=new yavru(); }
}
```

Bu program hata verir. Ana sınıfta parametre almayan bir yapıcı metot yoktur. Ana sınıfta bir yapıcı metot tanımlandığı için varsayılan yapıcı metot oluşturulmamıştır.

Kalıtım (Inheritence)

- base anahtar sözcüğü
- Yapıcı metotlar aşırı yüklenmişse türemiş sınıfın yapıcı metotları çağrılırken belli değerlerle temel sınıfında yapıcı metodunun çağrılması mümkündür ve bu işlem base anahtar sözcüğü ile yapılır.
- base anahtar sözcüğü ile parametre gönderimi yalnızca yapıcı metotlarla kullanılabilir. Yani base anahtar sözcüğünü yalnızca türemiş sınıftaki yapıcı metoda ekleyebiliriz ve base anahtar sözcüğünün ana sınıfta var olan bir yapıcı metodu belirtmesi gerekir.

  - public T(string s, int x, int z): base (int x, int z)

Örnekler

```
using System;
class A {
public int Ozellik1;
public int Ozellik2;
public A() { Console.WriteLine("Deneme"); }
public A(int ozellik1,int ozellik2) {
Ozellik1=ozellik1; Ozellik2=ozellik2; } }
 class B:A {
public int Ozellik3;
public int Ozellik4;
public B( int ozellik3,int ozellik4, int
ozellik1,int ozellik2): base(ozellik1,ozellik2)
{ Ozellik3=ozellik3; Ozellik4=ozellik4; } }
```

```
class esas {
static void Main() {
B b=new B(3,4,1,2);
Console.WriteLine (b.Ozellik1+"
"+b.Ozellik2+" "+b.Ozellik3+"
"+b.Ozellik4);
} }
```

- base sayesinde ana sınıfın parametre almayan yapıcı metodu çalışmaz.

Örnekler

- Bu program ekrana 1 2 3 4 yazar. Bu örnekte base anahtar sözcüğü ana sınıftaki yapıcı metodu temsil etmektedir. Türemiş sınıfta base sadece ana sınıftaki değer alan yapıcıya gittiğinden ana sınıfın boş yapıcı metodu çalışmaz, kullanımına gerek yok. Ama B b=new B() olsaydı veya base kullanılmasaydı mecburen ana sınıfın boş yapıcı metodu olacaktı.

- **base**, ana sınıfta hangi yapıcı metodun devreye gireceğini de belirtiyor.

- Not: base olmasaydı, türemiş sınıfın yapıcı metodu ne olursa olsun ana sınıftaki boş yapıcı metod mutlaka çalışır. Diğer yapıcı metotlar çalışmaz, base ile ana metoda ait herhangi bir yapıcı metot çalıştırılabilir.

Örnekler

```
using System;
class A {
public int X;
public int Y;
public A(int x, int y) { X=x; Y=y;
Console.WriteLine("Türemişte uyan metot");
}
public A(int x, int y, int z, int t) {
Console.WriteLine("base çağırdığı metot"); } }
 }
 class B:A {
public int Z;
public int T;
public B( int x, int y) : base(x, y, x, y)
{ Z=x; T=y;
Console.WriteLine("Türemiş metot");
} }
```

```
class esas {
static void Main() {
B b=new B(3,4);
} }
```

Çıktı:

Türemişte uyan metot
base çağırdığı metot
Türemiş metot

Base kullanarak anasınıfın boş yapıcı metodunun kullanım zorunluluğunu ortadan kaldırıyoruz.

Çoklu kalıtım (türetme)

- Sınıflar, ard arda türetilebilir. Yani örneğin **B** sınıfı **A** sınıfında türetilip **C** sınıfı da **B** sınıfından türetilebilir.

- Bu durumda **C** sınıfı türünden bir nesne yarattığımızda eğer **C** sınıfının ilgili yapıcı metoduna base takısını eklememişsek önce **A**, sonra **B**, sonra da **C** sınıfının yapıcı metotları çalıştırılır. Yani gidişat temel sınıftan türemiş sınıfa doğrudur.

- **C** sınıfı hem **A**'nın hem de **B**'nin bütün üye elemanlarına sahip olur. Örnek:

Çoklu kalıtım (türetme)

```
using System;
class A  { public A( )  { Console.WriteLine("A sınıfı"); } }

class B : A { public B( ) { Console.WriteLine("B sınıfı"); } }

class C : B
{   public C( ) { Console.WriteLine("C sınıfı"); }
    static void Main()
    {  C nesne=new C( ); }
}
```

```
A sınıfı
B sınıfı
C sınıfı
```

Çoklu kalıtım (türetme)

- Örnek

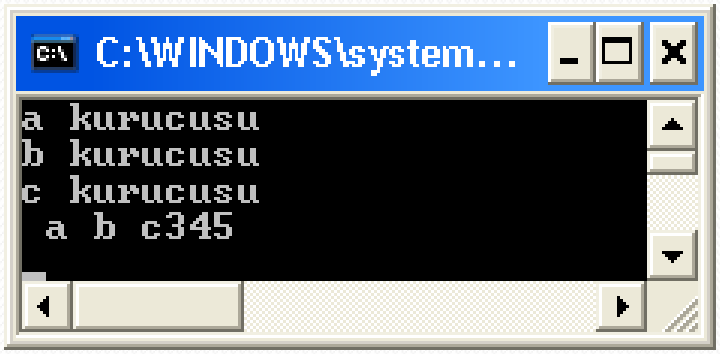
```
class A
{
   public int a;
   public A(int a)
   {
      Console.WriteLine("a kurucusu");
      this.a = a;
   }
}
```

```
class B : A
{
   public int b;
   public B(int a, int b) : base(a)
   {
      Console.WriteLine("b kurucusu");
      this.b = b;
   }
}
```

Çoklu kalıtım (türetme)

```
class C : B
{
    public int c;
    public void yaz()
    {
        Console.WriteLine(" a b c" + a + b + c);
    }
    public C(int a, int b, int c) : base(a, b)
    {
        Console.WriteLine("c kurucusu");
        this.c = c;
    }
}
```

```
class deneme
{
    static void Main(string[] args)
    {
        C cc = new C(3, 4, 5);
        cc.yaz();
        Console.ReadKey();
    }
}
```



Çoklu kalıtım (türetme) Örnek

```
using System;
class A
{   public int OzellikA;
    public A (int a) { OzellikA = a; }
}
class B : A
{   public int OzellikB;
    public B (int b) { OzellikB = b; }
}
class C : B
{   public int OzellikC;
    public C(int c, int b) : base(b)
    { OzellikC = c; }
    static void Main()
    {    C nesne = new C (12, 56);
         Console.WriteLine(nesne.OzellikA + " " + nesne.OzellikB + " " + nesne.OzellikC);
    }
}   // Hata verir
```

- C nesnesi oluşturulurken base ile değeri B sınıfa gönderdi. B sınıfının yapıcı metodu çalıştırıldı fakat B sınıfının yapıcı metodunda base takısı olmadığı için B sınıfına göre ana sınıfın (A sınıfı) parametre almayan yapıcı metodu çalıştırılmaya çalışılmıştır. A sınıfının parametre almayan yapıcı metodu olmadığı için program hata vermiştir.
- Çözüm: public B (int b):base (b);
- B sınıfı değer göndermeyecek ise A ve B sınıflarında varsayılan yapıcılar da tanımlanacak.
- public A() {}
- public B() {}

İsim Saklama (Name Hiding)

- Türemiş sınıfta bazen temel sınıftaki üye elemanları aynı isimli bir eleman veya metot tanımlanmış olabilir. Bu durumda temel sınıftaki elemana veya metoda normal yollarda erişmek mümkün değildir çünkü türeyen sınıftaki eleman veya metot temel sınıftaki elemanı veya metodu gizlemiştir.

- Temel sınıftaki elemana ve metoda erişmek için yine base anahtar sözcüğünden faydalanılır. base ile hem özelliklere hem de metotlara erişilebilir.

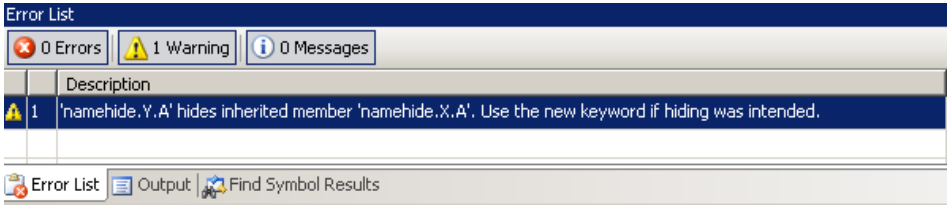
- base anahtar sözcüğünün temel elamana ulaşma şeklindeki kullanımı this referansına benzemektedir.

- this referansı kendisini çağıran sınıfı temsil ederken base anahtar sözcüğü türetmenin yapıldığı temel sınıfı temsil eder.

İsim Saklama (Name Hiding)

```
class X
{    protected int a;
     //public X() { }
      public X (int a)  {   this.a=a;  }
      public int A //dikkat
      {  get  {  Console.WriteLine("X");  return a;  }  }
 }
class Y: X
{
    protected int b;
    public Y (int b) :base(b) { this.b=base.a;  }
   public int A // dikkat- public new int A
   {
     get   { Console.WriteLine("Y sınıfı");  return b; }
    }
}
class Program
{ static void Main()
{    Y y=new Y(5);     int deneme=y.A;
} }
```

- Program uyarısı;

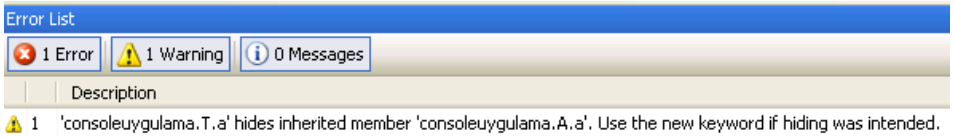


- (Uyarı) Y.A elemanı X.A elemanını gizlemiştir.
- Bu yüzden burada **new** anahtarı kullanılabilir. Her zaman isim gizlenmesi yaparken isim gizlemeyi açıkça belirtmek için temel sınıftaki bir elemanı gizlerken türemiş sınıftaki elemanın bildirimine **new** anahtarı eklemeliyiz. Bu sayede uyarıyı almayı da engellemiş oluruz.
- base(b) parametresi olmasaydı // public X() {} yapıcı metodunu tanımlamak zorundaydık. X Parametreli yapıcısı olduğundan varsayılan yapıcı metodu Y sınıfı çağıramazdı.

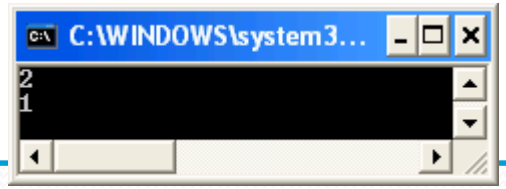
İsim Saklama (Name Hiding)

Örnek

```
using System;
class A
{   public int a;
    public A()   {  a = 1; }
}
class T : A
{   public int a;
  public T()  { a = 2;  }
}
 class Program
 {
   static void Main()
  {
   T t = new T();
  Console.WriteLine(t.a);
   }
}
```




```
using System;
class A
{ public int a;
  public A()  {  a = 1;    }
  public void yaz () {…}
}
class T : A
{
  public new int a;
  public int b ;
  public T()   { a= 2;    b = base.a;    }
 public void yaz () { base.yaz(); …}
}
class Program
{  static void Main()
  { T t = new T();
  Console.WriteLine(t.a);
  Console.WriteLine(t.b);
  Console.ReadLine();
  }
}
```



Sanal Metotlar

- Temel sınıf türünden bir nesneye türemiş sınıf referansı aktarılabilir. Bu aktarım sonrasında bazı metotların nesnelere göre seçilmesi istenebilir. Bu durumda sanal metotlar tanımlanır.

- Sanal metotlar temel sınıflar içinde bildirilmiş ve türeyen sınıflar içinde de tekrar bildirilen metotlardır. İsim saklamaya benzemesine rağmen kullanımda farklıdır.

- Sanal metotlar sayesinde temel sınıf türünden bir referansa türeyen sınıf referansı aktarıldığında, temel sınıf referansı üzerinden kendisine aktarılan türeyen sınıfın sanal metodunu çağırabilir.

- Eğer türeyen sınıf sanal metodu devre dışı bırakmamışsa temel sınıftaki sanal metot çağrılır. Çağrılan metodun hangi türe ait olduğu ise çalışma zamanında belirlenir. Hangi metodun çağrılacağının çalışma zamanında belirlenmesine geç bağlama (late binding) olarak isimlendirilir.

- Bu şekilde aynı nesne referansı üzerinden bir çok sınıfa ait farklı versiyonlardaki metotların çağrılabilmesi çok biçimlilik (polymorphism) olarak adlandırılır.

Sanal Metotlar

- Sanal metot bildirmek için **virtual** anahtar sözcüğü kullanılır.
- Türeyen sınıfta, temel sınıftaki sanal metodu devre dışı bırakmak için **override** anahtar sözcüğü kullanılır.
- Türeyen sınıfta devre dışı bırakılan metotların temel sınıftaki sanal metotların ismi ve parametrik yapısı temel sınıftaki metodun parametrik yapısı ile aynı olmalıdır.
- Statik metotlar sanal olarak bildirilemez.
- Türeyen sınıflar, temel sınıftaki sanal metotları devre dışı bırakmak zorunda değildir. Bu durumda temel sınıf referansları üzerinden temel sınıfa ait metot çağrılır.

Sanal Metotlar

```
using System;
  class A
{
   public void Metot()
 {
  Console.WriteLine("A sınıfı"); }
 }
 class B:A
{
 public void Metot()
{
Console.WriteLine("B sınıfı");
}
 static void Main()
{
  A nesneA=new A();
  B nesneB=new B();
  nesneA=nesneB;
  nesneA.Metot();
 }
}
```

Bu program ekrana **A sınıfı** yazar.

Çünkü nesneA nesnesi A sınıfı türündendir ve bu nesne üzerinden Metot() metoduna erişilirse A sınıfına ait Metot() metodu çalıştırılır.

Sanal Metotlar

```
using System;
  class A
{
   virtual public void Metot()
 {
  Console.WriteLine("A sınıfı"); }
 }
 class B:A
{
 override public void Metot()
{
Console.WriteLine("B sınıfı");
}
 static void Main()
{
  A nesneA=new A();
  B nesneB=new B();
  nesneA=nesneB; // A nesneA =new B();
  nesneA.Metot();
 }
}
```

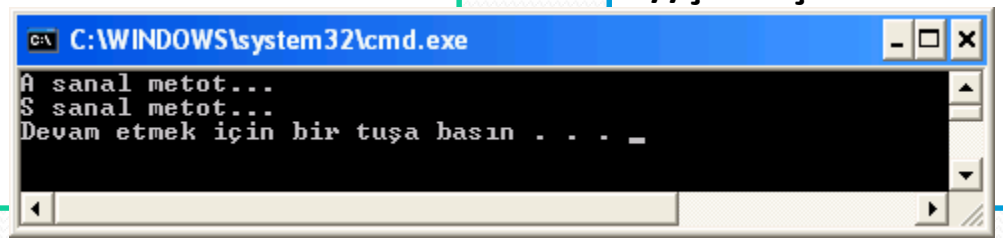
- Bu program ekrana **B sınıfı** yazacaktır.
- Bu programın öncekinden tek farkı A sınıfına ait metodun başına virtual anahtar sözcüğünün getirilmesi ve B sınıfına ait metodun başına da override anahtar sözcüğünün getirilmesi.
- Ana sınıftaki virtual anahtar sözcüğüyle ana sınıfa ait metodun bir sanal metot olmasını, türemiş sınıftaki override anahtar sözcüğüyle de ana sınıfa ait aynı adlı sanal metodun görmezden gelinmesini sağlandı. Bu sayede NesneA nesnesinin gizli türüne ait metot çalıştırıldı..

Sanal Metotlar

```
class A   {
    public virtual void Metot()
   { Console.WriteLine("A sanal metot..."); }
}
class T : A {
 public void Metot()
   { Console.WriteLine("T sanal metot..."); }

}
class S : A {
  public override void Metot() {
  Console.WriteLine("S sanal metot...");
  }
  }
```

```
class Program
{
static void Main()
{
T t = new T();
S s = new S();
A a = new A();
a = t;
a.Metot();
a = s;
a.Metot();
}
}
```

//çok biçimlilik



Sanal Metotlar

```
using System;
class A
{  virtual public void Metot()
   {     Console.WriteLine("A sınıfı");   }
}
class B:A
{   override public void Metot()
   {     Console.WriteLine("B sınıfı");   }
}
class C:B
{
   public void Metot()
   {     Console.WriteLine("C sınıfı");   }
   static void Main()
   {
      A nesneA=new A();
      C nesneC=new C();
      nesneA=nesneC;
      nesneA.Metot();
   }
}
```

B sınıfı yazar.

Çünkü C sınıfına ait metot override edilmediği için C'den önce gelen türeme zincirindeki son override edilen metot çalıştırılacaktır.

Sanal Metotlar

```
using System;
class A
{  virtual public int ozellik
   { set{}
     get{return 12;}
   }
}
class B:A
{  override public int ozellik
   {  get{return 100;}
     set{Console.WriteLine("Bu bir denemedir");}
   }
   static void Main()
   {
     B nesne=new B();
     A nesne2=new A();
     nesne2=nesne;
     Console.WriteLine(nesne2.ozellik);
     nesne2.ozellik=200;
   }
}
```

Ekran Çıktısı

100

Bu bir denemedir

Özet (Abstract) Sınıflar

- Nesne yönelimli programlamada sınıf hiyerarşisi oluşturulurken bazen hiyerarşinin en tepesinde bulunan sınıf türünden nesnelerin tek başına bir anlamı olmayabilir.

- Hiyerarşinin en tepesinde bulunan sınıfın kendisinden türetilecek diğer sınıflar için ortak özellikleri bir arada toplayan bir ara yüz gibi davranması istenebilir.

- Bu tür sınıflara özet sınıflar adı verilir. Metotlar ve özellikler de özet olarak tanımlanabilir.

- Özet sınıflar abstract anahtar sözcüğü ile bildirilirler. Özet sınıf türünden nesneler tanımlanamaz. Özet sınıflar tek başlarına anlamlı bir nesne ifade etmezler. Kullanabilmeleri için mutlaka o sınıftan başka bir sınıf türetilmesi gerekmektedir.
- abstract classSinif { }

Özet (Abstract) Sınıflar

```
using System;
abstract class A
{
    public string Metot()
    { return "Deneme"; }
}
class B : A
{
    static void Main()
    {
        B nesne1 = new B();
        Console.WriteLine(nesne1.Metot());
        //A nesne2=new A();
        /*Yukarıdaki satır olsaydı program hata verirdi. Özet sınıflar için nesne oluşturulmaz*/
        A nesne2 = new B(); // tanımlana bilir, ama gerek yoktur.
    }
}
```

Özet (Abstract) Metotlar

- Özet metotlar da yine abstract anahtar sözcüğüyle tanımlanır. Bu tür metotların gövdesi yoktur.
- Türeyen sınıfta mutlaka devre dışı bırakılmalıdır. Özet sınıflarda yapı itibariyle sanal oldukları için ayrıca virtual kullanılmaz.
- abstract public void Metot();
- Özet sınıflar içinde özet olmayan metotlar bildirilebilir. Fakat tersi özet olmayan sınıflarda özet metotların tanımlanması söz konusu değildir.
- Özet metotlar türeyen sınıfta devre dışı bırakılabilmeleri için private olarak tanımlanamazlar. public ya da protected olabilirler.

Özet (Abstract) Özellikler

- Özellikler de özet olarak bildirilebilir.

- abstract public int X

```
{
        get;
        set;
}
```

- Özet özellik bildiriminde kullanılan set veya get ifadelerinden hangileri kullanılmışsa türeyen sınıfta birebir aynı olmalı ve bunları override ile mutlaka uygulamalıdır.

Özet (Abstract) Özellikler

```
using System;
abstract class A
{   abstract public int ozellik    {    set;     get;     } }
class B:A
{
   override public int ozellik
{  get{return 100;}
   set{Console.WriteLine("Bu bir denemedir");}
}
   static void Main()
{ B nesne=new B();
Console.WriteLine(nesne.ozellik); nesne.ozellik=200; }
 }
```

Özet (Abstract) Özellikler

- using System;
- abstract class A
- { abstract public int ozellik { set; } } // get yok
- class B:A
- {
- override public int ozellik
- { get{return 100;} // get var
- set{Console.WriteLine("Bu bir denemedir");}
- }
- static void Main()
- { B nesne=new B();
- Console.WriteLine(nesne.ozellik); nesne.ozellik=200; }
- }
- // hata verir çünkü temel sınıftaki özellik bloğunda get yoktur. Aynı durum set içinde geçerli olurdu.

Arayüzler (Interfaces)

- Özet sınıfların benzeri olan bir yapı da arayüzlerdir (interface). Bütün metotları ve özellikleri özet (abstract) olarak bildirilmiş sınıflardan çok fazla bir farkı yoktur. Dolayısıyla arayüzlerdeki metot ve özelliklerin gövdesi yazılamaz.

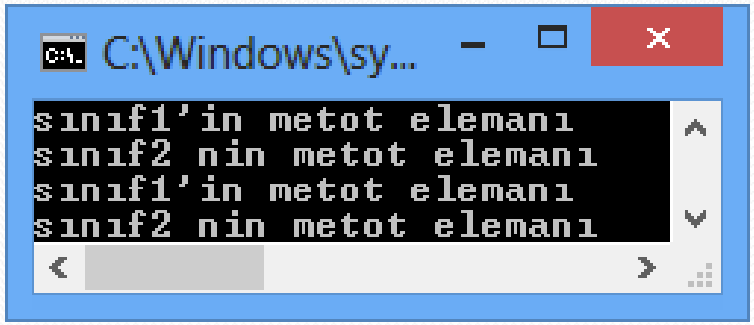
- Arayüzler kısaca kendisini uygulayan sınıfların kesin olarak içereceği özellikleri ve metotları belirler. **Arayüzde tanımlanan metot ve özellikler mutlaka türeyen sınıfta kullanılmalıdır.** Programlamada rehberlik eder ve atlanılmaması gereken metotları hatırlatır.

- **Arayüzler ile referanslar (nesneler) oluşturulabilir, ama new anahtar sözcüğü kullanılmaz .** Arayüz referansları tek başına bir anlam ifade etmez fakat kendisini uygulayan bir sınıf nesnesinin referansı atanabilir. Bu durumda arayüz referansı ile arayüzde bulunan metot ya da özellikler hangi sınıf referansı tutuluyorsa oradan çağrılabilir.

- Arayüzler, interface anahtar sözcüğü ile bildirilirler. Bir arayüzde özellik, metot, indeksleyici (indexer), temsilci (delegate) ve olay (event) bildirimi yapılabilir. Arayüz tanımlamalarının zorunlu olmasa da başına “I” harfinin eklenmesi tanımlamanın arayüz olduğunun kolayca anlaşılmasını sağlar.

Arayüzler (Interfaces)

```
using System;
interface arayüz   { void metot1(); }

class sınıf1 : arayüz
{   public void metot1()
    { Console.WriteLine("sınıf1'in metot elemanı"); }
}
class sınıf2 : arayüz
{   public void metot1()
    { Console.WriteLine("sınıf2 nin metot elemanı"); }
}
public class start
{
    static void Main()
    {   arayüz a;
        sınıf1 s1 = new sınıf1();
        sınıf2 s2 = new sınıf2();
        a = s1;        a.metot1();
        a = s2;        a.metot1();
        s1.metot1(); s2.metot1();
    }
}
```



Arayüzler (Interfaces)

- Arayüz tanımlamalarında dikkat edilecek bazı kısıtlamalar vardır:
  - **Arayüz elemanları statik olamaz.**
  - **Arayüz elemanları public yapıdadır. Ayrıca erişim belirteci ile bildirilemez.**
  - **Üye değişken içeremezler.**
  - **Yapıcı ve yıkıcı metotlar tanımlanamaz ya da bildirilemez.**

```
interface IArayuz
{
 int Metot1();
 int Metot2();
 int sahteozellik { set; get; }
 int this[int indeks] { get; }
}
```

Arayüzlerin Uygulanması

- Arayüzlerin uygulanması sınıf türetmeyle aynı şekilde yapılır. Örnek:
- **class** A:IArayuz { *//IArayuz arayüzündeki bütün elemanları içermeli.* }
- Sınıflar arasında çoklu türetme olmamasına rağmen arayüzler çoklu olarak uygulanabilir. Uygulanacak arayüzler virgül ile ayrılır:
- **class** A: Arayuz1, Arayuz2 { *//Hem Arayuz1 hem de Arayuz2 arayüzündeki bütün elemanları içermeli.* }
- Arayüzler de sınıf türetmeyle aynı şekilde birbirlerinden türetilebilir. Bu durumda türemiş arayüz, ana arayüzün taşıdığı bütün elemanları taşır.
- **Sınıflardan farklı olarak arayüzleri birden fazla arayüzden türetebiliriz.** Örnek:
- **interface** Arayuz1 { **int** Metot1(); }
- **interface** Arayuz2 { **string** Metot2(); }
- **interface** Arayuz3: Arayuz1, Arayuz2 { **double** Metot4(); }
- Burada Arayuz3'ü kullanan bir sınıf her üç metodu da içermelidir.

Arayüzler (Interfaces)

- C# dilinde var olan ara yüzleri uygulamanın bir yolu daha vardır. Buna açık ara yüz uygulama denir. Bunun avantajları şunlardır:
  - Açık ara yüz uygulama yöntemiyle istenirse sınıfa ait bazı üye elemanlara erişimi sınıf nesnelerine kapatırken(private), aynı üye elemanlara ara yüz nesnesiyle erişimi mümkün kılabiliriz.
  - Bir sınıfa birden fazla ara yüz uygulandığında eğer ara yüzlerde aynı isimli üye elemanlar varsa isim çakışmasının önüne geçebiliriz.

Arayüzlerin Uygulanması

- Arayüzler türetilirken new anahtar sözcüğü ile temel arayüzdeki elemanlar gizlenebilir. Bu şekilde aynı isimli yeni elemanlar tanımlanabilir.
- interface Arayuz1 { int Metot1(); }
- interface Arayuz2:Arayuz1 { new int Metot1(); }
- Arayuz2'yi kullanan bir sınıfta Metot1()'in bildirimi yapıldığında geçerli bir bildirim olur mu? Bunun cevabı hayırdır. Arayuz2'nin Arayuz1'e ait metodu gizlemesi sonucu değiştirmez. Bu sorunu halletmek için sınıfımızı şöyle yazabiliriz.
- class deneme:Arayuz2
- { int Arayuz1.Metot1() { ... }
- int Arayuz2.Metot1() { ... } }
- Burada hem Arayuz1'in hem de Arayuz2'nin isteklerini yerine getirdik. Eğer sınıfımız Arayuz2 arayüzünü kullanmasaydı programımız hata verirdi.
- Arayüzü uygulayan sınıf, arayüz dışında da elemanlara sahip olabilir. Yani istenildiği kadar üye eleman eklenebilir

Arayüzler (Interfaces)

```
using System;
interface arayuz { void Metot(); }
class sinif: arayuz
{ void arayuz.Metot() { Console.WriteLine("Deneme"); }
}
class mainMetodu
{
    static void Main()
   { sinif nesne=new sinif();
     ((arayuz)nesne).Metot();
    }
 }
```

- Burada sinif türünden olan nesne arayuz türüne dönüştürüldü ve arayuz nesnesiyle Metot() metoduna erişildi. Private olduğundan doğrudan nesne üzerinden erişilemezdi.

Arayüzler (Interfaces)

- Aynı programı şöyle de yazabilirdik:

```
using System;
interface arayuz
{ void Metot(); }
class sinif : arayuz
{
    void arayuz.Metot() { Console.WriteLine("Deneme"); }
}
class mainMetodu
{
    static void Main()
    {   arayuz nesne = new sinif();
        nesne.Metot();
    }
}
```

sealed Anahtar Sözcüğü

- Bazı durumlarda sınıflardan türetme yapılması istenmeyebilir. Bunu sağlamak için sınıf tanımlamasının başına sealed anahtar sözcüğü eklenir.

```
sealed class Sinif
{
}
```

- Sınıflardan türetme yapılmaması türeyen sınıfın anlamsız olması ya da bazı üyelerin güvenliğini sağlamak olabilir. Tüm üyeleri statik olan sınıflar için de kullanılabilir.
- abstract (özet) sınıflar sealed olarak işaretlenemez.
- **sealed** sınıflara ek olarak **static sınıflar** ve **yapılar** da türetilmeyi desteklemez.

Partial (Kısmi) Tipler

- Şu ana kadar tanımlanan tip bildirimleri tek bir proje dosyası içerisinde bulunmaktaydı. Oysaki bir projede birden çok dosya bulunabilir.

- Sınıf, arayüz ve yapı tanımlamaları partial anahtar sözcüğü ile birden fazla dosyaya dağıtılabilir. Bütün dosyadaki tanımlamalar tek bir bildirimi göstermektedir. (isim alanları tanımlarken de benzer bir durum söz konusuydu.)

Partial (Kısmi) Tipler

- Ozellikler.cs

- ```
  partial class Sinif
  {
      public int x;
      public int y;
  }
  ```

- Metotlar.cs

- ```
  partial class Sinif
  {
      public int Topla()
      {
      }

      public void EkranaYaz()
      {
      }
  }
  ```

Partial (Kısmi) Tipler

- Birleştirilmesini istediğimiz bütün aynı isimli türleri partial olarak bildirmeliyiz. Yalnızca birisini partial olarak bildirmek yeterli değildir.
- Sınıflar, yapılar ve arayüzler partial olarak bildirilebilir.
- Kısmi türlerden biri sealed ya da abstract anahtar sözcüğüyle belirtilmişse diğerinin de belirtilmesine gerek yoktur.
- Partial türler minimum üye eleman düzeyinde iş görürler. Yani aynı metodun gövdesinin bir kısmını bir dosyada, başka bir kısmını başka bir dosyada yazmak partial türler ile de mümkün değildir.

Visual Studio.Net -C#
10. HAFTA
İstisnai Durumlar, Temsilciler ve Olaylar, Şablon Tipler

Örnekler

```
class Program
    {
        static void Main(string[] args)
        {
            model1 oto1 = new model1();
            model2 oto2 = new model2();
            oto1.tur = "Sedan";
            oto1.silindir_sayisi = 4;
            oto1.subap_sayisi = 8;
            oto1.guc = 75;
            oto1.tork = 100;
            oto1.ozellikyaz();
            Console.WriteLine("********");
            oto2.model2_boy = 6;
            oto2.model2_agirlik = 900;
            oto2.model2_renk = "Kırmızı";
            oto2.ozellikyaz();
            Console.WriteLine("********");
            oto2.goster();
            Console.WriteLine("********");
            oto1.goster();
            Console.ReadLine();

        }
    }
}
```

Örnekler:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication9
{
    class oto
    {
        protected double boy=5;
        protected double agirlik=800;
        protected string renk="Sarı";
        public void goster()
        {
            Console.WriteLine("Boy=" + boy);
            Console.WriteLine("Ağırlık=" + agirlik);
            Console.WriteLine("Renk=" + renk);
        }
    }
    class model1:oto
    {
        public string tur;
        public int silindir_sayisi;
        public int subap_sayisi;
        public int tork  ;
        public int guc ;
        public void ozellikyaz()
        {
  Console.WriteLine("Tür=" + tur);
  Console.WriteLine("Boy="+boy);
  Console.WriteLine("Ağırlık=" + agirlik);
  Console.WriteLine("Renk=" + renk);
  Console.WriteLine("Silindir Sayisi=" + silindir_sayisi);
  Console.WriteLine("Subap Sayısı=" + subap_sayisi);
  Console.WriteLine("Tork=" + tork);
  Console.WriteLine("Güç=" + guc);
        }
    }
```

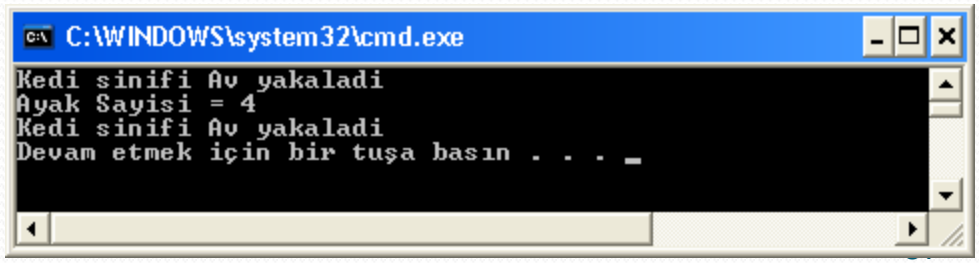
```
class model2 : oto
    {
        public double model2_boy
        {
            get { return boy; }
            set { boy = value; }
        }
        public double model2_agirlik
        {
            get { return agirlik; }
            set { agirlik = value; }
        }
        public string model2_renk
        {
            get { return renk; }
            set { renk = value; }
        }
        public string tur="Hatchback";
        public int silindir_sayisi=8;
        public int subap_sayisi=16;
        public int tork=300;
        public int guc=210;
        public void ozellikyaz()
        {
          Console.WriteLine("Tür=" + tur);
          Console.WriteLine("Boy=" + model2_boy);
          Console.WriteLine("Ağırlık=" + model2_agirlik);
          Console.WriteLine("Renk=" + model2_renk);
          Console.WriteLine("Silindir Sayisi=" + silindir_sayisi);
          Console.WriteLine("Subap Sayısı=" + subap_sayisi);
          Console.WriteLine("Tork=" + tork);
          Console.WriteLine("Güç=" + guc);
        }
    }
```

Örnek:

```
using System;

class A
{
    public int x;
    private int y;
    protected int z;

    public A()
    {
        x = 1;
        y = 2;
        z = 3;
        Console.WriteLine("A yapıcısı çalıştı...");
    }

    public void Listele()
    {
        Console.WriteLine("x={0}; y={1}; z={2};",x,y,z);
    }
}

class T : A
{
    public string s;

    public T()
    {
        s = "Türemiş Sınıf";
        Console.WriteLine("T yapıcısı çalıştı...");
    }

    public void Yaz()
    {
        Console.WriteLine("s={0}; x={1}; z={2};", s, x, z);
    }
}

class Program
{
    static void Main()
    {
        T t = new T();
        t.Listele();
        t.Yaz();
    }
}
```

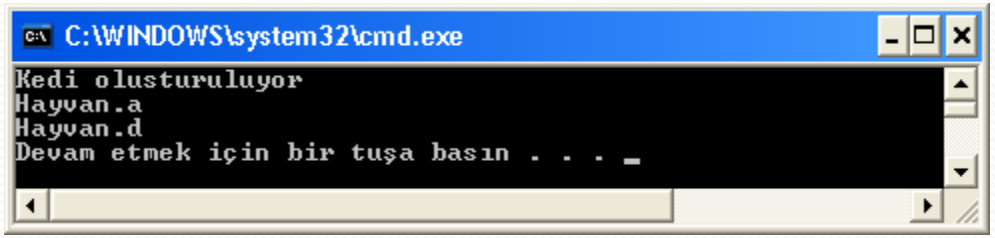
Örnek

```
using System;
class Kedi {
   protected int ayakSayisi = 4 ;
   public void yakalaAv()    {
      Console.WriteLine("Kedi sinifi Av yakaladi");   }
 }
class Kaplan : Kedi {
   public Kaplan()   {
      Console.WriteLine("Ayak Sayisi = " + ayakSayisi);   }
 }
class deneme {
public static void Main(string[] args)   {
      Kedi kd= new Kedi() ;
      kd.yakalaAv();
            Kaplan kp = new Kaplan();
      kp.yakalaAv();   }
}
```



Örnek

```
using System;
public class Hayvan {
        protected String a = "Hayvan.a";
        String b = "Hayvan.b"; //friendly
        private String c = "Hayvan.c";
        public String d = "Hayvan.d"; }
public class Kedi:Hayvan { // Türeyen
  public Kedi() {
        Console.WriteLine("Kedi olusturuluyor");
        Console.WriteLine(a);
        //Console.WriteLine(b); // ! Hata ! erisemez ?
        //Console.WriteLine(c); // ! Hata ! erisemez ?
        Console.WriteLine(d);
  } }
class deneme {
public static void Main(string[] args)    {
        Kedi k = new Kedi(); }
}
```



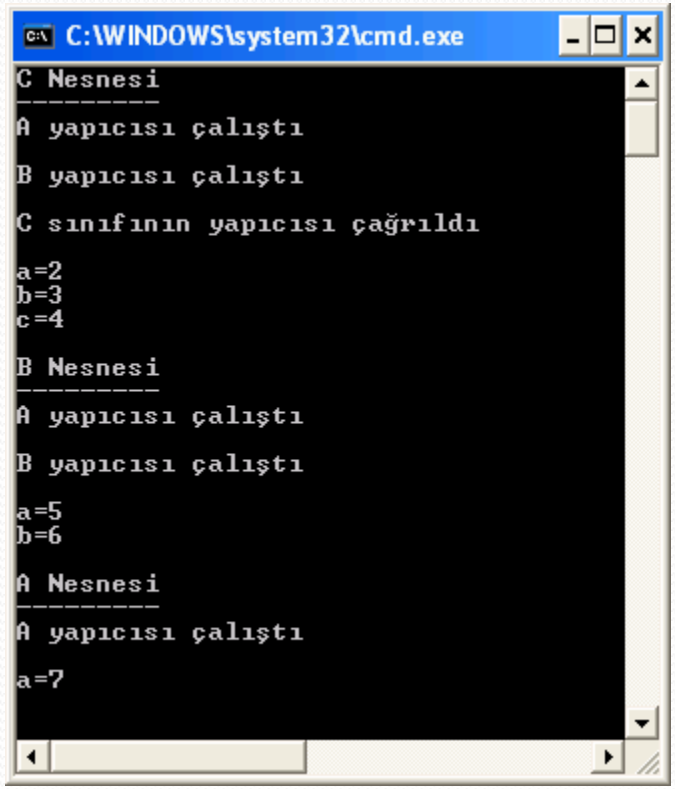
Örnek

```
using System;
class A
{
    public int a;
    public A(int a)
    {
this.a = a;
Console.WriteLine("A yapıcısı çalıştı\n");
    }

}
class B : A
{
    public int b;
    public B(int a,int b):base(a)
    {
this.b=b;
Console.WriteLine("B yapıcısı çalıştı\n");
    }

}
class C:B
{   public int c;
    public C(int a,int b, int c):base(a, b)
    {
this.c = c;
Console.WriteLine("C sınıfının yapıcısı çağrıldı\n");
    }
}
```

```
class Program
{
    static void Main()
    {

        Console.WriteLine("C Nesnesi");
        Console.WriteLine("---------");
        C c = new C(2, 3, 4);
        Console.WriteLine("a="+c.a);
        Console.WriteLine("b=" + c.b);
        Console.WriteLine("c=" + c.c+"\n");

        Console.WriteLine("B Nesnesi");
        Console.WriteLine("---------");
        B b = new B(5, 6);
        Console.WriteLine("a=" + b.a);
        Console.WriteLine("b=" + b.b+"\n");

        Console.WriteLine("A Nesnesi");
        Console.WriteLine("---------");
        A a = new A(7);
        Console.WriteLine("a=" + a.a+"\n");
        Console.ReadLine();
    }
}
```

Örnek

- Yukarıdaki programda, base anahtar sözcüğü sınıf hiyerarşisinin en tepesindeki sınıfı temsil etmektedir. C sınıfında base anahtar sözcüğü B sınıfı anlamına gelirken, B sınıfında base anahtar sözcüğü A sınıfı anlamına gelmektedir.



Örnek

```
class Program
    {
        public static void Goster(oto Oto)
        {
            Console.WriteLine(Oto.Tur); //Hata ulaşamaz
            Console.WriteLine(Oto.MotorGucu);
            Console.WriteLine(Oto.Tork);
            Console.WriteLine(Oto.Renk);
        }

        static void Main(string[] args)
        {
            oto oto1=new oto
    (75,100,"Kırmızı");
            Goster(oto1);
            Console.WriteLine("----------");
            model1 oto2=new
    model1("Fiat",100,110,"Beyaz");
            Goster(oto2);
            Console.WriteLine("----------");
            model2 oto3=new model2
    ("Renault",100,120,"Siyah");
            Goster(oto3);
            Console.ReadLine();

        }
    }
}
```

Örnek

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication9
{
    class oto
    {
        protected double motorgucu=5;
        protected double tork=800;
        protected string renk="Sarı";
        public oto(double guc, double tork, string
    renk)
        {
            this.motorgucu = guc;
            this.tork = tork;
            this.renk = renk;
        }
        public void ozellikgoster()
        {
            Console.WriteLine("Motor Gücü=" +
    motorgucu);
            Console.WriteLine("Tork=" + tork);
            Console.WriteLine("Renk=" + renk);
        }
        public double MotorGucu
        {
            get { return motorgucu; }
            set { motorgucu = value; }
        }
        public double Tork
        {
            get { return tork; }
            set { tork = value; }
        }
        public string Renk
        {
            get { return renk; }
            set { renk = value; }
        }
    }
```

```
class model1:oto
    {
        public string Tur;
        public model1(string tur,double guc,
    double tork,string renk):base
    (guc,tork,renk)
        {
            this.Tur=tur;
        }
        public void TurGoster()
        {
            Console.WriteLine("Türü"+Tur);
        }
    }

    class model2 : oto
    {
        public string Tur;
public model2(string tur,double guc, double
    tork,string renk):base (guc,tork,renk)
        {
            this.Tur=tur;
        }
        public void TurGoster()
        {
            Console.WriteLine("Türü"+Tur);
        }
    }
```

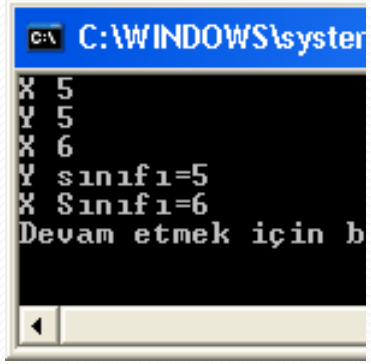
İsim Saklama (Name Hiding)

```
using System;
  class X { protected int a;
          public X(int a)   {
          Console.WriteLine("X " + a);
         this.a=a;   }
       public int A {
         get {
         Console.WriteLine("X Sınıfı="+a);
         return a;  }
         }  }
  class Y: X  {
     protected int b;
     public Y(int a):base(a)  {
         Console.WriteLine("Y " + a);
         this.b=a;  }
     new public int A {
         get {
         Console.WriteLine("Y sınıfı="+b);
         return b;  }
     }    }
```

```
class tarih
{
   static void Main()
   {
      Y y=new Y(5);
      X yy = new X(6);

      int deneme=y.A;
      int deneme2=yy.A;
   }
}


/// işleminin sonucunu söyleyiniz
```

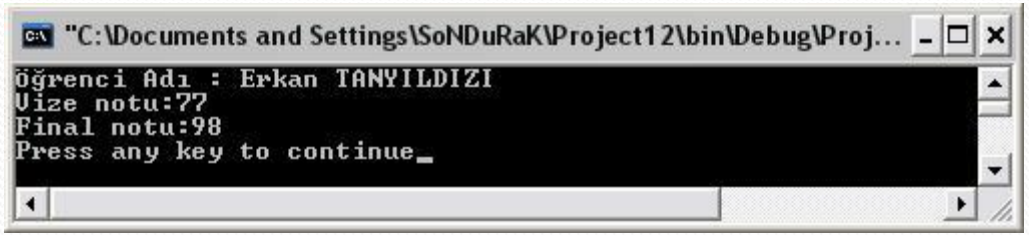
İsim Saklama (Name Hiding)

- Y y=new Y(5);
- Öncelikle Ana sınıfın yapılandırıcısı çalıştı
- Daha sonra türemiş sınıfın yapılandırıcısı çalıştı
- X yy = new X(6);
- Sadece Ana sınıfın yapılandırıcısı çalıştı
- int deneme=y.A;
- Türemiş sınıfıa ait get bloğu çalıştı
- int deneme2=yy.A;
- Ana sınıfa ait get bloğu çalıştı



Özet (Abstract) Sınıflar-Örnek

```
using System;
 abstract class Ogr_not  {
            public int vize;
            public int final;
            public Ogr_not(int v,int f) {
               vize=v;    final=f;      }
}

class ögr:Ogr_not {
            public string str;
            public ögr(string ad,int vize,int final):base(vize,final)
            {   str=ad; }
            public void ad_göster()
            {               Console.WriteLine("Öğrenci Adı : "+str);
                            Console.WriteLine("Vize notu:"+vize+"\n"+"Final notu:" +final);
            }
}
class Ana_sınıf
{           static void Main()
            {               ögr d=new ögr("Erkan TANYILDIZI",77,98);
                            d.ad_göster();
            }
}
```



Özet (Abstract) Sınıflar-Örnek

```
using System;

abstract class A
{
    public int x;
    abstract public int y
    {
        set;
        get;
    }

    public A(int x)
    {
        this.x = x;
    }

    abstract public void Metot();
}

class S : A
{
    int z;

    public S(int x):base(x)
    {
    }
```

```
 public override int y
    {
        get
        {
            return z;
        }
        set
        {
            z = value;
        }
    }

    public override void Metot()
    {
        Console.WriteLine(x);
        Console.WriteLine(y);
    }
}

class Program
{
    static void Main()
    {
        S s = new S(5);
        s.y = 2 * s.x;
        s.Metot();
    }
}
```

Örnekler

```
using System;
class A
{
  public void Metot1()
  {     Metot2();   }
  public void Metot2()
  {     Console.WriteLine("A sınıfı");   }
}
class B:A
{
  public new void Metot2()
  {     Console.WriteLine("B sınıfı");   }
}
class Ana
{
  static void Main()
  {
    B b=new B();
    b.Metot1();
  }
}
```

Bu programda ekrana A sınıfı yazılır.

```
using System;
class A
{   public void Metot1()
  {     Metot2();   }
  public void Metot2()
  {     Console.WriteLine("A sınıfı");   }
}
class B:A
{   public new void Metot1()
  {     Metot2();   }
  public new void Metot2()
  {     Console.WriteLine("B sınıfı");   }
}
class Ana
{   static void Main()
  {     B b=new B();
         b.Metot1();
  }
}
```

Bu programda ekrana B sınıfı yazılır.

Örnekler

```
using System;
class A
{
  public void Metot1()
  {     Metot2();   }
  virtual public void Metot2()
  {     Console.WriteLine("A sınıfı");   }
}
class B:A
{
  override public  void Metot2()
  {     Console.WriteLine("B sınıfı");   }
}
class Ana
{
  static void Main()
  {
    B b=new B();
    b.Metot1();
  }
}
```

Bu programda ekrana B sınıfı yazılır.

```
using System;
 class A
 {  public int OzellikA;
   public A(int a)
   {      OzellikA=a;    }
 }
 class B:A
 {  public int OzellikB;
   public B(int b,int a):base(a)
   {      OzellikB=b;    }
}
 class C:B
 {  public int OzellikC;
   public C(int c,int b,int a):base(b,a)
   {      OzellikC=c;    }
   static void Main()
   {      C nesne=new C(12,56,23);
     Console.WriteLine(nesne.OzellikA+"
"+nesne.OzellikB+" "+nesne.OzellikC);
   } }
```

Arayüzler (Interfaces)

```
using System;

interface IArayuz
{
    int Metot1();
    void Metot2();

    int Sayi
    {
        get;
        set;
    }

    int this[int indeks]
    {
        get;
    }
}
class Program
{
    static void Main()
    {
        Sinif s = new Sinif();
    }
}
```

```
class Sinif : IArayuz
{
    private int sayi;

    public int Metot1()
    {
        return 0;
    }

    public void Metot2()
    {
    }

    public int Sayi
    {
        get { return sayi; }
        set { sayi = value; }
    }

    public int this[int indeks]
    {
        get { return indeks; }
        set { }
    }
}
```

Arayüzler (Interfaces)

```
using System;

interface IArayuz
{
    int Metot1();
    void Metot2();
}

class Sinif : IArayuz
{
    int IArayuz.Metot1()
    {
        Console.WriteLine("Metot1");
        return 0;
    }

    public void Metot2()
    {
        Console.WriteLine("Metot2");
    }
}

class Program
{
    static void Main()
    {
        Sinif s = new Sinif();
        IArayuz a;

        a = s;
        a.Metot1();
    }
}
```

- //veya public int Metot1() olacak
- int IArayuz.Metot2() gibi kullanımda mutlaka sınıf nesnesi ara yüz nesnesine atanmalıdır. Yoksa metoda sınıf nesnesi üzerinden erişilemez.

Örnekler

```
using System;
 using System.Collections;
 class Koleksiyon:IEnumerable
 {   int[] Dizi;
   public Koleksiyon(int[] dizi)
   {      this.Dizi=dizi;    }
   IEnumerator IEnumerable.GetEnumerator()
   {      return new ENumaralandırma(this);    }
   class ENumaralandırma:IEnumerator
   {   int indeks;
     Koleksiyon koleksiyon;
     public ENumaralandırma(Koleksiyon
koleksiyon)
     {  this.koleksiyon=koleksiyon;
       indeks=-1;
     }
     public void Reset()
     {        indeks=-1;      }
```

```
public bool MoveNext()
     {  indeks++;
       if(indeks<koleksiyon.Dizi.Length)
         return true;
       else        return false;
     }
     object IEnumerator.Current
     {    get    { return(koleksiyon.Dizi[indeks]);  }
     }
   }
}
class MainMetodu
{   static void Main()
   { int[] dizi={1,2,3,8,6,9,7};
     Koleksiyon k=new Koleksiyon(dizi);
     foreach(int i in k)
       Console.Write(i+" ");
   }
}
```

Örnekler

Bu program kendi tanımladığımız herhangi bir sınıfın foreach döngüsüyle kullanılabilmesini sağlıyor. Yani foreach(tur1 nesne1 in nesne2){} deyiminde hem nesne1'in türünü (tur1) hem de nesne2'nin türünü kendi oluşturduğumuz türlerden (sınıf, yapı, vb.) biri yapabiliriz. Bu programda nesne2'nin kendi türümüzde olmasını sağlayacağız. Bu örnekten hemen sonraki örnekte de nesne1'in kendi türümüzden olmasını sağlayacağız. Programımıza dönecek olursak foreach döngüsünde nesne2'yi kendi türümüzden yapabilmemiz için bazı metotlar oluşturmamız gerekiyor. System.Collections isim alanındaki IEnumerable ve IEnumerator arayüzleri de bize bu metotları oluşturmaya zorluyor. Ayrıca programımızın ana hattına dikkat ederseniz iki sınıf iç içe geçmiş. C#'ta bu tür bir kullanım mümkündür.
Bu programımızı satır satır incelemeye başlayalım.

```
using System;
using System.Collections;
//Bu satırlarla System ve System.Collections isim alanlarındaki türlere direkt erişim hakkı elde ettik.

class Koleksiyon:IEnumerable {
// Bu satırlarla System.Collections isim alanındaki IEnumerable arayüzünü kullanan bir sınıf başlattık.

int[] Dizi;
// Bu satırla sınıfımıza (Koleksiyon) ait bir özellik bildirdik.
public Koleksiyon(int[] dizi)
{ this.Dizi=dizi; }

// Bu satırlarla sınıfımızın (Koleksiyon) yapıcı metodunu bildirdik. Yapıcı metodumuz int[] türünden bir parametre alıyor ve bu aldığı veriyi sınıfın bir özelliği olan Dizi'ye aktarıyor. Burada this anahtar sözcüğünün kullanılması zorunlu değildir ancak okunurluğu artırır.
```

Örnekler

```
IEnumerator IEnumerable.GetEnumerator()
 { return new ENumaralandırma(this); }
```

// Bu satırlarla sınıfımıza, IEnumerable arayüzündeki GetEnumerator() metodunu açık arayüz uygulama yöntemiyle geçiriyoruz. Metodun geri dönüş tipi IEnumerator arayüzü. Metodun gövdesinde ise ENumaralandırma sınıfının yapıcı metodu kullanılarak ENumaralandırma türünden bir nesne döndürülüyor. Buradan ENumaralandırma sınıfının yapıcı metodunun Koleksiyon sınıfı türünden bir nesne aldığını anlayabiliyoruz. Çünkü buradaki this anahtar sözcüğü bu GetEnumerator metoduna hangi nesne üzerinden erişildiğini temsil ediyor. Bu metot Koleksiyon sınıfında olduğuna göre bu metoda da bir Koleksiyon nesnesi üzerinden erişilmelidir. Burada bir terslik varmış gibi gözüküyor. O da metodun geri dönüş tipiyle geri döndürülülen verinin tipinin birbirine uymaması. Ancak programımızın sonraki kodlarına bakacak olursanız ENumaralandırma sınıfının IEnumerator arayüzünü kullandığını göreceksiniz. Dolayısıyla da bir IEnumerator tipinden nesneye bir ENumaralandırma nesnesi atanabilecek.

```
class ENumaralandırma:IEnumerator {
```

// Burada Koleksiyon sınıfının içinde ENumaralandırma sınıfını oluşturuyoruz ve bu sınıf da System.Collections isim alanındaki IEnumerator arayüzünü kullanıyor.

```
int indeks; Koleksiyon koleksiyon;
```

// Burada sınıfımıza (ENumaralandırma) iki tane özellik ekledik.

```
public ENumaralandırma(Koleksiyon koleksiyon)
 { this.koleksiyon=koleksiyon; indeks=-1; }
```

// Burada ENumaralandırma sınıfının yapıcı metodunu hazırladık. Yeri gelmişken belirtmek istiyorum. İç içe sınıflar iç içe gözükselerde aslında birbirinden bağımsızdır. İç içe sınıfların normal sınıflardan tek farkı içteki sınıfa dıştaki sınıfın dışından erişilmek istendiğinde görülür. Bu durumda içteki sınıfa DisSinif.IcSinif yazarak erişilebilir. Ancak tabii ki bu erişimin mümkün olabilmesi için iç sınıfın public olarak belirtilmesi gerekir.

Örnekler

```
public void Reset() { indeks=-1; }
// Burada sınıfımıza bir metot ekledik. Bu metodu sınıfımızın kullandığı IEnumerator arayüzü gerektiriyordu.
public bool MoveNext()
 { indeks++; if(indeks<koleksiyon.Dizi.Length)
 return true;
 else return false; }
// Burada sınıfımıza bir metot daha ekledik. Yine bu metodu da IEnumerator arayüzü gerektiriyordu.
object IEnumerator.Current
{
 get { return(koleksiyon.Dizi[indeks]); }
}
// Burada IEnumerator arayüzünün gerektirdiği bir sahte özelliği açık arayüz uygulama yöntemiyle hazırladık.
class MainMetodu {
 static void Main() {
// Artık programımızın çalışmaya başlayacağı kısmı yazmaya başlıyoruz.
int[] dizi={1,2,3,8,6,9,7};
Koleksiyon k=new Koleksiyon(dizi);
// Dizimizi ve yeni bir Koleksiyon nesnesi oluşturduk.
```

```
foreach(int i in k) Console.Write(i+" ");
//Ve başardık. nesne2'yi kendi türümüz yaptık.
```

Şimdi nesne1'i kendi türümüz yapacak programı yazalım:

```
using System;
class A
{   public int Ozellik;
    public A(int a) { Ozellik=a; }
}
class Ana {
   static void Main()
  {   A[] dizi=new A[3]; dizi[0]=new A(10);
      dizi[1]=new A(20); dizi[2]=new A(50);
      foreach(A i in dizi)
       Console.WriteLine(i.Ozellik);
   }
  }
// Gördüğünüz gibi kendi oluşturduğumuz sınıf türünden nesnelerle diziler oluşturabiliyoruz. Nasıl ki int[] ile int türündeki veriler bir araya geliyorsa bizim örneğimizde de A türünden nesneler bir araya geldi.
```

Visual Studio.Net -C#
10. HAFTA
İstisnai Durumlar, Temsilciler ve Olaylar, Şablon Tipler

İstisnai Durum Yönetimi (Exception Handling)

- Program ne kadar iyi yazılırsa yazılsın hataların oluşma olasılığı bir çok durumda mümkündür.
- Özellikle çalışma anında meydana gelebilecek hatalar kod yazılırken tespit edilemeyebilir.
- Tüm hatalara karşı tek tek tedbir almak çok zor bir iştir.
- Bu yüzden modern dillerde çalışma anında meydana gelebilecek hataları yakalamak için bir hata yakalama mekanizması mevcuttur.
- Çalışma zamanında beklenmeyen bir hata sonucunda oluşturulan nesnelere **istisnai durum sınıf nesneleri** denir. .NET sınıf kütüphanesinde çok sık oluşabilecek hatalara karşı istisnai durum sınıfları tasarlanmıştır. Bu sınıflar hata ile ilgili çeşitli bilgileri tutmaktadırlar.

İstisnai Durum Yönetimi (Exception Handling)

- .NET sınıf kitaplığında istisnai durum sınıfı hiyerarşisindeki en temel sınıf System isim alanı içersinde yeralan **System.Exception**'dır. **Exception**(istisna) sınıfı oluşan hatalar için çok genel bilgiler verdiğinden türemiş diğer bilgi sınıfları kullanılır:
  - SystemException, ArgumentException, StackOverFlowException, ArithmeticException, IOException, IndexOutOfRangeException…
- İstisnai durumları yakalamak için dört tane anahtar sözcükten faydalanılır:
- **try**, **catch**, **finally** ve **throw**
- try, catch ve finally sözcükleri birer kod bloğunu temsil ederken, throw ise bir hatanın nesnesinin oluşturulmasını sağlar.

İstisnai Durum Yönetimi (Exception Handling)

- **try :** Hatanın kontrol edileceği kod bloğunun yazılacağı kısımdır. Yani bu blokta bir hata meydana gelirse ilgili hata sınıf nesnesi oluşturulacaktır.

- **catch :** try bloğunda yakalanan hataya göre işlemlerin yapılmasını sağlayan bloktur.

- **finally :** try ve catch bloklarında açılan kaynakların kapatılması burada gerçekleşir. Kodlar hata oluştursa dahi çalıştırılır. Bu bloğu belirtme zorunluluğu yoktur.

- **throw :** try bloğu içersinde belirli bir hata olduğunda ilgili hata sınıfı nesnesini oluşturmak için kullanılır. Eğer try içindeki nesneler zaten hata oluşturuyorsa throw'un kullanılmasına gerek olmayabilir.

İstisnai Durum Yönetimi (Exception Handling)

```
using System;
class Program
{
    static void Main()
    {
        int[] x = new int[5];
        try
        {  x[6] = 25; //hatanın fırlatıldığı bölüm
        }
        catch
        { Console.WriteLine("Hata Oluştu...");
        //kural dışı tip için yönetici
        }
        finally
        { Console.WriteLine("Finally Bloğu..:.");
         //kaynakların temizlenmesi
        }
    }
}
```

İstisnai Durum Yönetimi (Exception Handling)

- İlgili hata sınıfı nesnesi kullanıldığında hata hakkında daha ayrıntılı bilgilere sahip olunabilir.
- Eğer catch tanımlamasında bir nesne bulunursa sadece o hata meydana geldiğinde catch bloğu çalıştırılacaktır.
- try bloğu içersinde metotlar da çağrılabilir. Bu sayede metot içinde oluşan çalışma hataları da tespit edilebilir.
- Bir try bloğu içersinde birden fazla hata oluşabilme ihtimali vardır. Bu durumda her bir hata durumu için birer catch bloğu tanımlanabilir.

İstisnai Durum Yönetimi (Exception Handling)

- try bloğundan sonra mutlaka catch ve ya finally bloğu gelmelidir. Bu bloklar arasında başka ifadeler yer almamalıdır.

- İç içe geçmiş try blokları da kullanmak mümkündür.
- **try**
- { *//A*
- **try**
- { *//B* }
- **catch** { *//C //İçteki catch bloğu* }
- **finally** { *//D //İçteki finally bloğu* }
- }
- **catch** { *//Dıştaki catch bloğu* }
- **finally** { *//Dıştaki finally bloğu* }
- }

İstisnai Durum Sınıfları

- System isim uzayında tanımlı, sıkça kullanılan kural dışı durumlar aşağıda verilmiştir
  - **ArrayTypeMismatchException:** Depolanmakta olan değerin tipi dizi tipi ile uyumsuz.
  - **DivideByZeroException:** Sıfıra bölünme hatasında fırlatılır.
  - **IndexOutOfRangeException:** Dizi indexi sınıf dışına taştığında fırlatılır.
  - **InvalidCatsException:** Programın çalışması sırasında geçersiz tür dönüşümü yapıldığında fırlatılır.
  - **OverflowExcepiton:** Aritmetik taşma meydana geldiğinde fırlatılır.
  - **NullReferanceExcepiton:** Null referans üzerinde işlem yapıldığında fırlatılır.
  - **StackOverflowException:** Yığın taşmıştır.
  - **FormatException:** Metotlarda yanlış formatta parametre gönderildiğinde fırlatılır.
  - **ArithmeticExcepiton:** Bu sınıf ile DivideByZeroException ve OverflowException hataları yakalanabilir. Aritmetik işlemler sonucunda oluşan istisnai durumlarda fırlatılır.
  - **OutOfMemoryExcepiton:** Programın çalışması içi yeterli bellek miktarı olmadığında fırlatılır.

İstisnai Durum Yönetimi (Exception Handling)

```
using System;
class deneme
{
  static void Main()
  {
    try
    { int[] a=new int[2];
      Console.WriteLine(a[3]);
    }
    catch(IndexOutOfRangeException)
    { Console.WriteLine("Dizi sınırları aşıldı");  }
  }
}
```

```
using System;
class deneme
{  static void Main()
  { for(;;)
   {  try
      { Console.Write("Lütfen çıkmak için 0 ya da 1 girin: ");
        int a=Int32.Parse(Console.ReadLine());
        int[] dizi=new int[2];
       Console.WriteLine(dizi[a]);
        break;     }
     catch { continue; }    } }  }
```

```
Lütfen çıkmak için 0 ya da 1 girin: 2
Lütfen çıkmak için 0 ya da 1 girin: 5
Lütfen çıkmak için 0 ya da 1 girin: 1
0
```

İstisnai Durum Yönetimi (Exception Handling)

```
using System;
class hatayakalama1
{ public static void Main()
  {  try
    {  int a=50, b=0;   Console.WriteLine(a/b); }
    catch(DivideByZeroException e)
   {  Console.WriteLine("Sıfıra bölünme hatası yakalandı");   }
     finally
    {  Console.WriteLine("\n program sonlandırılıyor");  }
  }
}
```

Çıktı: Sıfıra bölünme hatası yakalandı
program sonlandırılıyor

```
using System;
class deneme
{ static void Main()
{ try { Metot(); }
catch(IndexOutOfRangeException nesne)
{ Console.WriteLine("Metodu kullananda hata yakalandı"); }
}
 static void Metot()
 { try { int[] a=new int[2]; Console.WriteLine(a[3]); }
catch(IndexOutOfRangeException nesne)
 { Console.WriteLine("Metodun kendisinde hata yakalandı."); }
 } }
```

Çıktı: Metodun kendisinde hata yakalandı.

throw anahtar sözcüğü

- Hatalar otomatik olarak üretilmelerine rağmen bazı özel durumlarda **throw** ile kendi istisnai durum nesnelerimizi oluşturabiliriz.

- **throw** anahtar sözcüğü istisnai durum sınıf nesnesi türünden bir nesne gönderir.

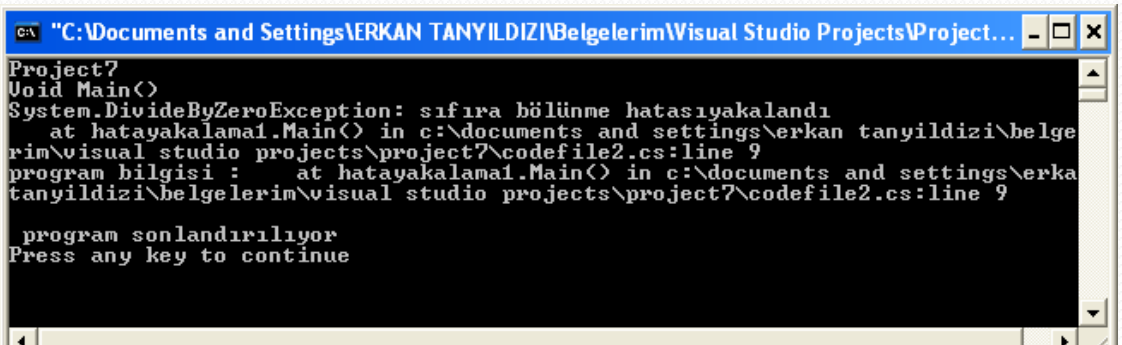
- Bir hata nesnesi throw anahtar sözcüğü yardımıyla şöyle fırlatılabilir:
  - **throw** new IndexOutOfRangeException("Dizinin sınırları aşıldı");
- veya
  - IndexOutOfRangeException nesne=new IndexOutOfRangeException("Dizinin sınırları aşıldı");
  - **throw** nesne;

İstisnai Durum Yönetimi (Exception Handling)

- **"exception" Sınıfının Önemli Üye Elemanları** :
  - **Message: Hatanın özünü tarif eden bir karakter katarı içerir.**
  - **Source: catch bloğunda yakalanan istisnai durum nesnesinin gönderildiği uygulamanın ya da sınıfın adıdır.**
  - **HelpLink: Fırlatın hata ile ilgili yardım dosyasının yol bilgisini saklar.**
  - **StackTrace: İstisnai durumun fırlatıldığı metot ve programla ilgili bilgi içerir.**
  - **TargetSite: İstisnai durumu fırlatan metot ile ilgili bilgi verir**
  - **InnerException: Eğer catch bloğu içerisinde bir hata fırlatılırsa catch bloğuna gelinmesini sağlayan istisnai durumun exception nesnesidir.**

İstisnai Durum Yönetimi (Exception Handling)

```
using System;
class hatayakalama1{
public static void Main(){
 try
  { int a=50,b=0;
     throw new DivideByZeroException("sıfıra bölünme hatası yakalandı");
     Console.WriteLine(a/b);
  }
  catch(DivideByZeroException e)
  { Console.WriteLine(e.Source);          Console.WriteLine(e.TargetSite);
     Console.WriteLine(e.ToString());     Console.WriteLine("program bilgisi : "+e.StackTrace);
  }
  finally
 {  Console.WriteLine("\n program sonlandırılıyor");
 }
}
}
```



İstisnai Durum Sınıfları Oluşturmak

- Kural dışı durumlara yönelik C#'ın sahip olduğu gücün bir parçası, programcının oluşturduğu kural dışı durumları kontrol altına alma becerisinden kaynaklanmaktadır.
- Programcı kendi oluşturduğu hataları kontrol etmek için yine kendi oluşturduğu kural dışı durumları kullanabilir. Kural dışı durum oluşturmak için yapılması gereken yalnızca **Exception** sınıfından türetilmiş bir sınıf (**ApplicationException**) tanımlamaktır.
- Oluşturulan kural dışı sınıfları Exception tarafından tanımlanan ve programcının oluşturduğu sınıfların kullanımına sunulan özelliklere ve metotlara otomatik olarak sahip olacaktır.

Örnek

```
using System;
class Notlar
{   private int mNot;
    public int Not
    {   get{return mNot;}
        set
        {    if(value>100)
             throw new FazlaNotHatasi();
         else if(value<0)
             throw new DusukNotHatasi();
         else
             mNot=value;
        }
    }
    public class FazlaNotHatasi:ApplicationException
    {
      override public string Message
      {
        get{ return "Not 100'den büyük olamaz.";}
      }
    }
```

```
public class DusukNotHatasi:ApplicationException
  {     override public string Message
      {   get{return "Not 0'dan küçük olamaz.";}
      }
  }
}
class Ana
{   static void Main()
  {  try
     {  Notlar a=new Notlar();
        Console.Write("Not girin: ");
        int b=Int32.Parse(Console.ReadLine());
        a.Not=b;
        Console.WriteLine("Notu başarıyla girdiniz.");
     }
     catch ( Exception nesne)
     {        Console.WriteLine(nesne.Message);      }
  }
}
```

Temsilci (Delegate)

- Delegate'ler metotları temsil eder. Temsilcilerin nesneleri oluşturulabilir ve bu nesneler metotları temsil ederler. Delegate'ler metodların bellekteki adreslerini aklında tutar. Temsilci bildirimi delegate anahtar sözcüğü ile yapılır. Bir temsilci bildirim taslağı şu şekildedir:

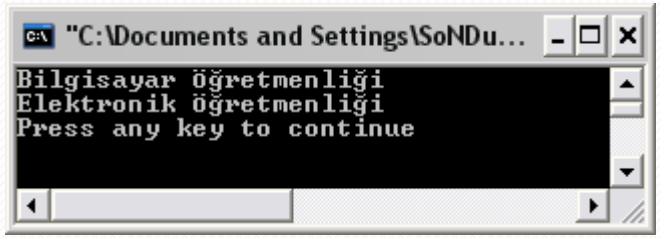
  - [Erişim belirleyicisi] **delegate** dönüş değeri temsilci (parametreler)

    delegate int temsilci (**int** a, **string** b);

- Temsilcilerin kullanılmasının amacı derleme zamanında belli olmayan metotların çalışma zamanını belirlemektir.
- Temsilciler kendi imzalarına uygun herhangi bir metodu temsil edebilirler. Bu metotlar statik de olabilir.
- delegate bir veri tipi olduğundan erişim belirteci ile tanımlanabilir. Return ile değer gönderilir ise en son gönderilen belirli değişkene atanır.

Temsilciler

```
using System;
class delegeler
{   public delegate void temsilci();

    public static void Bilgisayar()
    {  Console.WriteLine("Bilgisayar Öğretmenliği");    }
    public static void Elektronik()
    {
        Console.WriteLine("Elektronik Öğretmenliği");
    }
    public static void Main()
    {
        temsilci nesne = new temsilci(Bilgisayar);
        nesne();
        nesne = new temsilci(Elektronik);
        nesne();
    }
}
```



Çoklu temsilciler

- Bir temsilci birden fazla metodu temsil edebilir. **+** ve **–** operatörleri ile temsilciye metot ekleme ve çıkarma yapılabilir.

- **+** ve **-** metotları yerine daha pratik olması için **+=** ve **-=** de kullanılabilir.

- Çoklu temsilci çağrıldığında metotlar temsilciye eklenme sırasına göre çalıştırılır.

Çoklu temsilciler

```
delegate void temsilci(int a);
class deneme
{
    public static void Metot1(int a)
    { Console.WriteLine("Metot1 çağrıldı." + a); }
    public static void Metot2(int a)
    { Console.WriteLine("Metot2 çağrıldı." + a); }
    public static void Metot3(int a)
    { Console.WriteLine("Metot3 çağrıldı." + a); }
}
class Program
{    static void Main()
    {
        temsilci nesne = null;
        nesne += new temsilci(deneme.Metot2); nesne += new temsilci(deneme.Metot1);
        nesne += new temsilci(deneme.Metot3); nesne(1);
        Console.WriteLine("***"); nesne -= new temsilci(deneme.Metot1);
        nesne(2);
    }
}
```

```
Ekran Çıktısı:
Metot2 Çağrıldı.1
Metot1 Çağrıldı.1
Metot3 Çağrıldı.1
***
Metot2 Çağrıldı.2
Metot3 Çağrıldı.2
```

Temsilciler

```
using System;

class Program
{
    public delegate void Temsilci(int a, int b);

    public static void Topla(int a, int b)
    {
        Console.WriteLine("Toplam:{0}", a + b);
    }

    public static void Cikar(int a, int b)
    {
        Console.WriteLine("Fark  :{0}", a - b);
    }

    static void Main()
    {
        Temsilci t = new Temsilci(Topla);
        t = t + new Temsilci(Cikar);

        t(10, 8);
    }
}
```

Delegate sınıfı

- Kendi yarattığımız temsilciler gizlice System isim alanındaki **Delegate** sınıfından türer. Dolayısıyla kendi yarattığımız temsilcilerin nesneleri üzerinden bu sınıfın static olmayan üye elemanlarına erişebiliriz.

```
using System;
class Ana
{   delegate void temsilci();
    static void Metot1()
    { Console.WriteLine("Metot1"); }
    static void Metot2()
    { Console.WriteLine("Metot2"); }
    static void Main()
    {   temsilci t = new temsilci(Metot1);
        t += new temsilci(Metot2);
        Delegate d = t; temsilci c = (temsilci)d; // tür dönüşümüne dikkat
        c(); // metotların sırasıyla çağrılması
    }
}
```

```
Ekran Çıktısı:
Metot1
Metot2
```

Delegate sınıfı

- Ayrıca Delegate sınıfı özet bir sınıf olduğu için **new** anahtar sözcüğüyle bu sınıf türünden nesne yaratamayız. Yalnızca kendi yarattığımız temsilci nesneleri üzerinden bu sınıfın üye elemanlarına erişebiliriz. Delegate sınıfının bazı üye elemanları:

- **GetInvocationList() :** Çoklu temsilci yapısında bulunan metotları bir delegate dizisi olarak geri döndürür.
- **DynamicInvoke(object[] parametreler) :** Bir temsilcinin temsil ettiği metotları tek tek çağırmak için kullanılır.
- **Combine(Delegate[] temsilciler):** Çoklu bir temsilciye temsilciler ile belirtilen Delegate dizisindeki metotları ekler.

Delegate sınıfı

```
using System;
class deneme
 { delegate void temsilci();
 static void Metot1() { Console.WriteLine("Burası Metot1()"); }
 static void Metot2() { Console.WriteLine("Burası Metot2()"); }
static void Main()
{ temsilci nesne=null;
 nesne+=new temsilci(Metot1);
 nesne+=new temsilci(Metot2);
Delegate[] dizi=nesne.GetInvocationList();
dizi[0].DynamicInvoke();
dizi[1].DynamicInvoke(); }
 }
```

- Bu program alt alta Burası Metot1() ve Burası Metot2() yazacaktır.

Delegate sınıfı

- Eğer Delegate dizisindeki temsilci nesnelerinin temsil ettiği metotların parametreleri varsa bu parametreler DynamicInvoke() metoduna object dizisi olarak verilmelidir.
- **Combine(Delegate[] temsilciNesneleri)**
  Combine(Delegate temsilciNesnesi1, Delegate temsilciNesnesi2)
- Birinci metot temsilciNesneleri adlı Delegate dizisindeki tüm metotları bir çoklu temsilci nesnesi olarak tutar. İkinci metot ise temsilciNesnesi1 ve temsilciNesnesi2 temsilci nesnelerindeki tüm metotları arka arkaya ekleyip bir temsilci nesnesi olarak tutar. Her iki metot da statictir. Örnek:

- *//Diğer kısımlar yukarıdaki programın aynısı. (Burası Main() metodu)*
- temsilci nesne=**null**;
- nesne+=new temsilci(Metot1);
- nesne+=new temsilci(Metot2);
- **Delegate**[] dizi=nesne.GetInvocationList();
- **Delegate** nesne2=**Delegate**.Combine(dizi); nesne2.DynamicInvoke();

Delegate sınıfı

- Şimdi ikinci metodu örnekleyelim:
- *//Diğer kısımlar yukarıdaki programın aynısı. (Burası Main() metodu)*
- **temsilci** nesne1=**null**;
- nesne1+=new temsilci(Metot1);
- nesne1+=new temsilci(Metot2);
- **temsilci** nesne2=new temsilci(Metot1);
- **Delegate** nesne3=**Delegate**.Combine(nesne1,nesne2);
- nesne3.DynamicInvoke();
- Bu program ekran çıktısı şöyle olmalıdır:
- Burası Metot1()
- Burası Metot2()
- Burası Metot1()

İsimsiz metotlar

- Metotlar olmadan temsilci oluşturulmasına isimsiz metotlar adı verilir.
- Örnek:

```
using System;
class isimsiz
{    delegate double temsilci(double a, double b);
    static void Main()
    {   //temsilci nesnesine bir kod bloğu bağlanıyor.
        temsilci t = delegate(double a, double b)
        {
            return a + b;
        };
        //temsilci nesnesi ile kodlar çalıştırılıyor.
        double toplam = t(1d, 9d);
        Console.WriteLine(toplam);
    }
} // Temsilci nesnesine bir metot yerine direkt kodlar atandı.
```

Olaylar (Events)

- Olaylar, temsilcilerin özel bir formudur. Olaylar ile yapılan işlemlerin tamamı temsilciler ile de yapılabilir.

- İşlemleri sadeleştirmek amacıyla event anahtar sözcüğü tanımlanmıştır.

- [Erişim belirleyici] **event** [temsilci türü] [**olay adı**];

Olaylar (Events)

```
using System;
delegate void OlayYoneticisi(); //Olay yöneticisi bildirimi
class Buton //Olayın içinde bulunacağı sınıf bildirimi
{ public event OlayYoneticisi Click; //Olay bildirimi
public void Tiklandi() //Olayı meydana getirecek metot
    { if(Click!=null) Click(); }  // Olaya yüklü metotların çağrılması
}
class AnaProgram
 { static void Main()
 { Buton buton1=new Buton();
 buton1. Click +=new OlayYoneticisi(Click); //Olay sonrası işletilecek metotların eklenmesi
buton1.Tiklandi(); //Olayın meydana getirilmesi.
}
 //Olay sonrası işletilecek metot
 static void Click()  // Olaydan sonra çalıştırılacak metot ya da metotların static olma zorunluluğu yoktur.
{ Console.WriteLine("Butona tıklandı."); }
 } // Bu program ekrana Butona tıklandı. yazacaktır.
```

Olaylar (Events)

- Şimdi programımızı derinlemesine inceleyelim. Programımızdaki en önemli satırlar:
- **delegate void** OlayYoneticisi(); ve Buton sınıfındaki **public event** OlayYoneticisi Click;
- satırıdır. Birinci satır bir temsilci, ikinci satır da bir olay bildirimidir. Bu iki satırdan anlamamız gereken Click olayı gerçekleştiğinde parametresiz ve geri dönüş tipi void olan metotların çalıştırılabileceğidir.
- Ana metottaki
- buton1.Tiklandi();
- satırı ile olay gerçekleştiriliyor. Tiklandi metoduna bakacak olursak;
- **public void** Tiklandi() { **if**(Click!=**null**) Click(); }
- Burada bir kontrol gerçekleştiriliyor. Burada yapılmak istenen tam olarak şu: Eğer Click olayı gerçekleştiğinde çalışacak bir metot yoksa hiçbir şey yapma. Varsa olayı gerçekleştir.
- buton1.Click+=new OlayYoneticisi(Click);
- Main() metodundaki bu satırla da buton1'in Click olayına bir metot ekliyoruz. Yani buton1 nesnesinin Click olayı gerçekleştiğinde Click metodu çalışacak.

Olaylar (Events)

```
using System;
delegate void OlayYoneticisi(); //Olay yöneticisi bildirimi
class Buton
{ public event OlayYoneticisi Olay; //Olay bildirimi
public void Tiklandi() //Olayı meydana getirecek metot
    { if (Olay != null) Olay(); //buton1 içindeki metotlar sırasıyla tetiklensin
    }
}
class Ana
{   static void Metot1()  { Console.WriteLine("Metot1"); }
    static void Metot2()  { Console.WriteLine("Metot2"); }
    static void Click()   { Console.WriteLine("Butona tıklandı."); }
    static void Main()
    {   Buton buton1 = new Buton();
        //Olay sonrası işletilecek metotların eklenmesi
        buton1.Olay += new OlayYoneticisi(Click);
        buton1.Olay += new OlayYoneticisi(Metot2);
        buton1.Olay += new OlayYoneticisi(Metot1);
        buton1.Tiklandi();
    }
 }
```

Çıktı:
Butona tıklandı.
Metot2
Metot1

Olaylar (Events)-Örnek

```
using System;
delegate void OlayYoneticisi(); //Olay yöneticisi bildirimi
class Buton //Olayın içinde bulunacağı sınıf bildirimi
 {  public event OlayYoneticisi Click; //Olay bildirimi
    public void Tiklandi()  { if(Click!=null) Click(); }
 }
 class Pencere
 {  int a;
   public Pencere(int a) { this.a=a; }
   public void Click()  { Console.WriteLine("Pencere sınıfındaki metot çağrıldı - "+a); }
 }
 class AnaProgram
 {  static void Click() { Console.WriteLine("Butona tıklandı."); } //Olay sonrası işletilecek metot
   static void Main()
   {  Buton buton1=new Buton(); Pencere nesne1=new Pencere(1), nesne2=new Pencere(2);
      buton1.Click+=new OlayYoneticisi(Click); //Olay sonrası işletilecek metotların eklenmesi
      buton1.Click+=new OlayYoneticisi(nesne1.Click);
      buton1.Click+=new OlayYoneticisi(nesne2.Click);
     buton1.Tiklandi(); //Olayın meydana getirilmesi.
   }
 }
```

Çıktı:
Butona tıklandı.
Pencere sınıfındaki metot çağrıldı - 1
Pencere sınıfındaki metot çağrıldı - 2

Olaylar (Events)

- Olay yöneticisine yeni metotlar ekleme ve çıkarma anında kontroller yapmak için set- get yapısının benzeri bir yapı olan add ve remove blokları kullanılabilir.
- += add bloğunu, -= ise remove bloğunu çalıştırır.
- **add** ve **remove** bloklarının her ikisi de mutlaka aynı anda tanımlanmalıdır.
- Delege olay sayısı kadar dizi tanımlanmalıdır.
- Olaylar da bir tür olduğu için arayüzlerde bildirilebilir. Ayrıca sanal (virtual) ve özet (abstract) olarak da bildirilebilirler.
- Yalnız add ve remove içeren olaylar özet olarak bildirilemez.
- **Olayların en büyük kullanım alanı nesneler arası minimum bağlantıyla mesajlaşmadır.**

Olaylar (Events)

```
using System;
delegate void OlayYoneticisi();
class Buton
{  public OlayYoneticisi[] olay = new OlayYoneticisi[3];
   public int i = 0;    public int j = 0;
   public event OlayYoneticisi Olay
   {   add  { Console.WriteLine("Olaya metot eklendi ="+(i + 1) + ". Olay");
              if (olay[i] == null)   { olay[i] = value; }
              i++;
              if (i == 3)  { --i; Console.WriteLine("Olay listesi Dolu"); }
            }
       remove { Console.WriteLine("Olaydan metot çıkarıldı.");
               if (olay[i] == value)  {   olay[i] = null;   }
               i--;
               if (i ==-1)   Console.WriteLine("Olay bulunamadı");
             }
    }
    public void Tiklandi()
    {  for (int j = 0; j <= i; j++)  if (olay[j] != null) olay[j]();  }
}
class AnaProgram
{   static void Metot1() { Console.WriteLine("Metot1"); }
    static void Metot2() { Console.WriteLine("Metot2"); }
    static void Click() { Console.WriteLine("Butona tıklandı."); }
    static void Main(){ Buton buton1 = new Buton();
       buton1.Olay += new OlayYoneticisi(Click); buton1.Olay += new OlayYoneticisi(Metot1);
       buton1.Olay += new OlayYoneticisi(Metot2); buton1.Tiklandi();
       buton1.Olay -= new OlayYoneticisi(Metot2);  buton1.Tiklandi();
    }
}
```

```
Olaya metot eklendi =1. Olay
Olaya metot eklendi =2. Olay
Olaya metot eklendi =3. Olay
Olay listesi Dolu
Butona tıklandı.
Metot1
Metot2
Olaydan metot çıkarıldı.
Butona tıklandı.
Metot1
```

Şablon tipler

- Bir sınıftaki tür bilgileri sınıf bildirimi esnasında belirli iken şablon tipler ile sınıftaki türlerin nesne yaratımı sırasında belli olması sağlanır. Bu sayede her nesne için söz konusu türler değişebilir. Şablon tipler sayesinde türden bağımsız işlemler yapılabilir.

```
using System;
class Sinif<SablonTip>   { public SablonTip Ozellik; }

class AnaProgram {
 static void Main() {
Sinif<int> s1=new Sinif<int>();            s1.Ozellik=4;
Sinif<string> s2=new Sinif<string>();    s2.Ozellik="deneme";
 Console.WriteLine(s1.Ozellik+" "+s2.Ozellik); }
 }
```

- Bu programda s1 nesnesinin Ozellik özelliğinin geri dönüş tipi int, ancak s2 nesnesinin Ozellik özelliğinin geri dönüş tipi stringdir.
- **Sınıflar, arayüzler, yapılar, temsilciler ve metotlar şablon tip olarak bildirilebilir.**

Şablon tipler-Sınıf aşırı yükleme

- Bir sınıf (veya yapı, arayüz vs.) birden fazla şablon tip alabilir.
- Örnek:
- class Sinif<SablonTip1, SablonTip2>

- Bu durumda bu sınıf türünden nesne:

- Sinif<int,string> s=new Sinif<int,string>();
- şeklinde oluşturulur.

- Aynı isim alanında isimleri aynı olsa bile farklı sayıda şablon tipi olan sınıflar bildirilebilir. Buna **sınıf aşırı** yükleme denir.

Şablon tipler arasında türeme

- Şablon tipler arasında türeme ile ilgili en çok kullanılan iki kombinasyon aşağıdaki gibi karşımıza çıkar:

- **class** AnaSinif<T>
- { ... }

- **class** YavruSinif<T, Y>:AnaSinif<T>
- { *//Gördüğünüz gibi yavru sınıf en az ana sınıfın şablon tipini içerdi.* }

- **class** YavruSinif2<T>:AnaSinif<**int**>
- { */*Gördüğünüz gibi yavru sınıf ana sınıfın şablon tipine belirli bir tür koydu. Böylelikle yavru sınıftaki ana sınıftan gelen T harfleri int olarak değiştirilecektir.*/* }

Şablon tipler ve arayüzler

- Arayüzlerin de şablon tipli versiyonları yazılabilir.
- Örneğin System.Collections.Generic isim alanında birçok şablon tipli arayüz bulunmaktadır. Bunlardan birisi de IEnumerable arayüzünün şablon tipli versiyonudur. IEnumerable arayüzünün şablon tipli versiyonu aşağıdaki gibidir.
- interface IEnumerable<T>:IEnumerable
- { IEnumerator<T> GetEnumerator(); }

- Buradan anlıyoruz ki bu arayüzü uygulayan sınıf geri dönüş tipi IEnumerator<T> olan GetEnumerator() metodunu ve IEnumerable arayüzünün içerdiği tüm üyeleri içermeli. Bu arayüzü bir sınıfta şöyle uygulayabiliriz:
- class Sinif:IEnumerable<int>
- { IEnumerator IEnumerable.GetEnumerator() { //... }
- IEnumerator<int> IEnumerable<int>.GetEnumerator() { //... }
- }
- Burada hem IEnumerable hem de IEnumerable<T> arayüzlerinin metotları uygulandı. Çünkü IEnumerable<T> arayüzü, IEnumerable arayüzünden türemişti.

Şablon tiplerin metotlara etkisi

- Sınıf düzeyinde bir şablon tip belirlediğimizde bu şablon tip metotların geri dönüş tipine, parametre tiplerine koyulabilir. Ancak bunun sonucunda bazen bir çakışma oluşabilir. Örneğin:

```
class Sinif<T>
{ public int Metot(T a) { return 0; }
public int Metot(int a) { return 1; } }

Sinif<int> a=new Sinif<int>();
Console.WriteLine(a.Metot(10));
```

- Bu gibi bir durumda **Metot(int a)** metodu çalışır. **Normal tipli versiyon şablon tipli versiyonu gizler**.

- Şablon tipler metotlarla kullanıldığı gibi özellik, indeksleyici ve olaylarla da kullanılabilir

Şablon tipler-default operatörü

- Şablon tipler herhangi bir tipi temsil ederler. Bu yüzden C#, yalnızca bazı türlere özgü olan operatörler (+, -, *, ...) kullanılmasına izin vermediği gibi şablon tip türünden bir nesne yaratılıp bu nesneye bir değer verilmesine engel olur.
  - Örnekler:
  - `class Sinif<T> { public int Metot(T a) { return a+2; } }`
  - **Bu sınıf derlenmez.** Çünkü T tipinin hangi tip olduğunu bilmiyoruz. + operatörü bu tip için aşırı yüklenmiş olmayabilir. Bu yüzden C# bu gibi tehlikeli durumları önlemek için bu tür bir kullanımı engeller. Başka bir kullanım:
  - `class Sinif<T> { public int Metot(T a) { T nesne=0; } }`
  - **Yine burada da bir hata söz konusudur.** Çünkü her tipten nesneye 0 atanmayabilir. Benzer şekilde yapı nesnelerine de null değer atanamaz.

- **default** operatörü bir şablon tipin varsayılan değerini elde etmek için kullanılır. Örnek:
- `class Sinif<T> { public void Metot() { T nesne=default(T); } }`
- Varsayılan değer bazı türler için 0 (int, short, float vs.) bazı türler için null (tüm sınıflar) bool türü için de false'tur.

Şablon tipler-KISITLAR

- Şablon tür nesneleriyle herhangi bir üye de (metot, özellik vs.) çalıştıramayız. Çünkü şablon tipin ne olduğunu bilmediğimiz için olmayan bir metodu çalıştırmaya çalışıyor olabiliriz. Tüm bunların sebebi aslında şablon sınıf türünden nesne yaratırken şablonu herhangi bir tür yapabilmemizdi. Eğer şablona koyulabilecek türleri kısıtlarsak bunları yapabiliriz.
- C#'ta şu kısıtlar bulunmaktadır:
- **struct** şablon tip yalnızca yapılar olabilir.
- **class** şablon tip yalnızca sınıflar olabilir.
- **new()** şablon tip yalnızca nesnesi yaratılabilen tiplerden olabilir. (tür abstract, static, vb. olamaz)
- **inheritance (türetme)** şablon tip mutlaka belirtilen bir türden türemiş olmalıdır.
- **interface** şablon tip mutlaka belirtilen bir arayüzden türemiş bir sınıf olmalıdır.

- Kısıtlar where anahtar sözcüğüyle yapılmaktadır.

Şablon tipler-KISITLAR

- where T : struct T tipi değer (value) tipleri olmalıdır.
- where T : class T tipi referans (referance) tipleri olmalıdır.
- where T : new() T tipi yükleyicisi (constructor) parametresiz olan bir tip olmalıdır.
- where T : class_name T tipi oluşturduğunuz bir sınıf yada bu sınıfı ile genişletilmiş alt sınıflar olmalıdır.
- where T : interface_name T tipi belirtilen interface ile genişletilmiş bir obje olmalıdır.

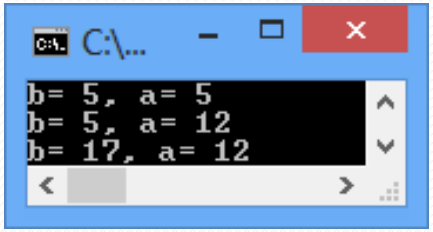
- Referans tipleri : Dynamic, Delegate, Interface, Strings, Object, Class.
- Değer tipleri ise : int , float , double,char vb. tipleridir

Şablon tipler-KISITLAR

- class A{}
- class B:A{}
- class C:A{}
- class D<T> where T:A{} *//A sınıfı veya A dan türeyen sınıf nesnesi olabilir.*
- class AnaProgram {
- static void Main() {
- D<B> nesne1=new D<B>(); D<C> nesne2=new D<C>(); D<A> nesne3=new D<A>();
- *//Aşağıdaki olmaz. D<int> nesne3=new D<int>();* }
- }
- Türeme kısıtı sayesinde şablon tip nesnesiyle bir metodu çağırabiliriz. Bunun içinde ana sınıfa o metodu koyarız, bu sınıftan türeyen yavru sınıflarda o metodu override ederiz (override olmazsa sadece ana sınıftaki metot ve üye özellikler kullanılabilir, türemiş sınıfı kısıt olarak verirsek türemiş ve anasınıf kullanılablir). Son olarak şablon tipe ana sınıftan türeme zorunluluğu getiririz. Ana sınıftan türeyen bütün sınıflar söz konusu metodu içereceği için artık C#, bu metodu şablon tip nesneleri üzerinden çağırmamıza izin verecektir.

Şablon tipler-KISITLAR

```
using System;
class A
{   public int a;
    public A() {a=5;}
    public virtual void yaz() { Console.WriteLine(a= "+ a); }
}
class B:A
{   int b;
    public B()  {b = a;}
    public int y { get{return b; } set{b=value;} }
    public override void yaz() { Console.WriteLine("b= "+b+", a= "+a ); }
}
class D<T> where T:B
{   public void metot(T deger)
    {   deger.yaz();        deger.a = 12;        deger.yaz();
        deger.y = 17;       deger.yaz();
    }
}
class AnaProgram
{   static void Main()
    {   B x = new B();        D<B> nesne1 = new D<B>();     nesne1.metot(x); }
}
```



Şablon tipler-KISITLAR

- Ancak halen new operatörüyle şablon tip türünden nesne oluşturamayız. Bunun için şablon tipe new() kısıtını eklemeliyiz. Yani şablon tipe yalnızca nesnesi oluşturulabilen türler koyulabilecek.
- Örnek:
- class Sinif<T> where T:new()
- {… }

- Sinif<int> x = new Sinif<int>();

- Artık bu sınıfın içinde T türünden nesneleri new operatörüyle oluşturabiliriz. (Temel türler, diziler)Tabii ki C# şablon tipe nesnesi oluşturulamayan bir tür koyulmasını engelleyecektir.

- Başka bir önemli kısıt ise arayüz kısıtıdır. Arayüz kısıtı sayesinde şablon tipe koyulan tipin mutlaka belirli bir arayüzü kullanmış olmasını sağlarız.

Şablon tipler-KISITLAR

```
class A
{   public int a;
    public A() {a=5;} public A(int c) {a=c*5;}

    public virtual void yaz() { Console.WriteLine(a); }
}
class D<T> where T:new()//parametresiz yapıcı olacak
{   private T info;
    public T Bilgi //new ile nesne oluşturuldu.
    {   get   { return info;  }
        set  {  info = value;}
    }
 }
```

Şablon tipler-KISITLAR

```
class AnaProgram
{    static void Main()
    {   A x = new A();
        D<A> nesne1 = new D<A>();
        nesne1.Bilgi = x;
        Console.Write(nesne1.Bilgi.a);// Çıktı→ 5
        A x = new A(9);
        // D<A> nesne1 = new D<A>(); Hata verir new()
        parametresiz yapıcı olacak
        D<int> nesne1 = new D<int>();
        nesne1.Bilgi=6;

    }
}
```

Şablon tipler-KISITLAR

```
public interface IPerson
{    string Ad { get; set; }   }
class H : IPerson
{   string aa;
    public string Ad
    {   get  {  return aa; }
        set  {  aa = value; }
    }
}
class E<T> where T : IPerson
{   public T deger;  }
class AnaProgram
{   static void Main()
    {   H h = new H();
        E<IPerson> nesne2 = new E<IPerson>();
       //veya  E<H> nesne2 = new E<H>();
        nesne2.deger = h;         nesne2.deger.Ad = "Ahmet";
        Console.WriteLine(nesne2.deger.Ad);
  }
}
```

Şablon tipler-KISITLAR

- Örnek program:

```
using System;
class A <T> where T : IComparable<T>
{   public static int Karsilastir(T a, T b) { return a.CompareTo(b); }
}
class Program
{   static void Main()
    {   int s1 = A<int>.Karsilastir(4, 5);
        int s2 = A<float>.Karsilastir(2.3f, 2.3f);
        int s3 = A<string>.Karsilastir("Ali", "Veli");
        int s4 = A<DateTime>.Karsilastir( DateTime.Now, DateTime.Now.AddDays(1));
         Console.WriteLine("{0} {1} {2} {3}", s1, s2, s3, s4);
    }
}
```

- Bu program ekrana -1 0 -1 -1 çıktısını verecektir. .Net Framework kütüphanesindeki **IComparable** arayüzünde **CompareTo**() metodu bulunmaktadır. Dolayısıyla bu arayüzü uygulayan her sınıfta da bu metot bulunur. Bu sayede de C# şablon tip nesnesinden ilgili metodun çağrılmasına izin vermiştir.

Şablon tipler-KISITLAR

- Diğer önemli iki kısıt ise şablon tipe yalnızca bir sınıf koyulabilmesini sağlayan class ve aynı şeyi yapılar için yapan struct kısıtlarıdır. Örnekler:
- **class** Sinif1<T> **where** T:**class** { }
- **class** Sinif2<T> **where** T:**struct** { }
- Tabii ki bir şablon tipe birden fazla kısıt eklenebilir. Bu durumda kısıtlar virgülle ayrılır. Örnek:
- **class** Sinif<T> **where** T: **class**, **IComparable**, **new()** { }
- Kısıtlarda diğerlerinin sırası önemli değildir. Ancak -varsa- **new()** kısıtı en sonda olmalıdır. Bir sınıfa eklenen birden fazla şablon tip varsa her biri için ayrı ayrı kısıtlar koyulabilir. Örnek:

- **class** Sinif<T,S> **where** T:IComparable,IEnumerable **where** S:AnaSinif

- **NOT:** Main() metodu mutlaka şablon tipli olmayan bir sınıfın içinde olmalıdır. Ayrıca Main() metodu ya bir sınıfın içinde ya da bir yapının içinde olmalıdır. Diğer bir deyişle çalışabilir her program en az bir sınıf ya da yapı içermelidir.

Şablon tipli metotlar

- **Şablon tipler metot ve temsilci düzeyinde de tanımlanabilir.**

- Örnek:
- `using System;`
- `class karsilastirma {`
- `static void Main() {`
- `Console.WriteLine(EnBuyuk<int>(4,5));`
  `Console.WriteLine(EnBuyuk<string>("Ali","Veli"));`
- `}`
- `static T EnBuyuk<T>(T p1,T p2) where T:IComparable`
- `{  T geridonus=p2;`
- `  if(p2.CompareTo(p1)<0) geridonus=p1;`
- `  return geridonus; } }`

- Bu program alt alta 5 ve Veli yazacaktır.

Şablon tipi çıkarsama

- Az önceki örneğin sadece Main() metodunu alalım:
- **static void** Main()
- { Console.WriteLine(EnBuyuk<int>(4,5));
  Console.WriteLine(EnBuyuk<string>("Ali","Veli"));
- }
- Burada parametrelerin türleri belli olduğu hâlde ayrıca int ve string türlerini de belirttik. :
- **static void** Main() İstersek bunu şöyle de yazabilirdik
- { Console.WriteLine(EnBuyuk(4,5));
- Console.WriteLine(EnBuyuk("Ali","Veli")); }

- Bu programda metot şöyle düşünecektir: "Benim parametrelerim T türünden, o hâlde yalnızca parametreme bakarak T'nin hangi tür olduğunu bulabilirim." Gördüğünüz gibi metotların aşırı yükleyerek saatlerce uğraşarak yazabileceğimiz programları şablon tipli metotlar kullanarak birkaç dakikada yazabiliyoruz.

Şablon tipli temsilciler

- Temsilciler de şablon tip alabilirler. Bu sayede temsilcinin temsil edebileceği metot miktarını artırabiliriz. Örnek:
- using System;
- delegate T Temsilci<T>(T s1,T s2);
- class deneme {
- static int Metot1(int a,int b) {return 0;}
- static string Metot2(string a,string b){return null;}
- static void Main() {
- Temsilci<int> nesne1=new Temsilci<int>(Metot1);
- Temsilci<string> nesne2=new Temsilci<string>(Metot2); Console.WriteLine(nesne1(1,2)); Console.WriteLine(nesne2("w","q")); }
- }
- Temsilci şablon tipleri de kısıt alabilirler. Örnek:
- delegate T Temsilci<T>(T s1,T s2) where T:struct
- Burada T yalnızca bir yapı olabilir. Yani bu temsilcinin temsil edeceği metodun parametreleri ve geri dönüş tipi yalnızca bir yapı olabilir.

null değer alabilen yapı nesneleri

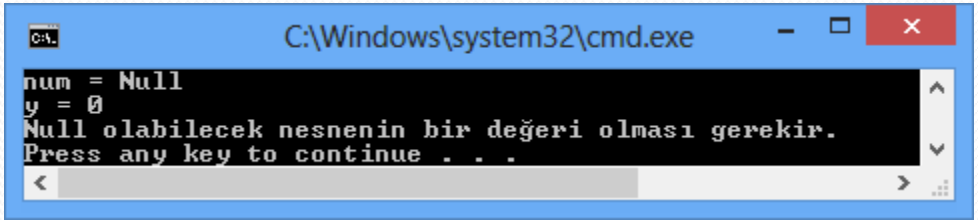
- Bildiğiniz gibi yapı nesneleri null değer alamaz. Örneğin şu kod hatalıdır:
- int a=null;
- Ancak System isim alanındaki Nullable<T> yapısı sayesinde yapı nesnelerinin de null değer alabilmesini sağlayabiliriz. System.Nullable<T> yapısı şu gibidir:
- public struct Nullable<T> where T:struct
- { private T value;
- private bool hasValue;
- public T Value{get {...} }
- public bool HasValue{get {...} }
- public T GetValueOrDefault(){...}
- }
- Bu yapıya göre null değer alabilen yapı nesneleri şöyle oluşturulur:
- Nullable<int> a=new Nullable<int>();
- a=5; a=null;
- Nullable<double> b=new Nullable<double>(2.3);
- Console.WriteLine("{0}{1}",a, b);

? İşareti

- **?** takısı kısa yoldan nullable tipte nesne oluşturmak için kullanılır. Örneğin:
- **Nullable<double>** d=10; ile **double?** d=5;
- satırları birbirine denktir.
- Nullable nesneleri normal nesnelere tür dönüştürme operatörünü kullanarak dönüştürebiliriz. Ancak nullable nesnenin değeri null ise çalışma zamanı hatası alırız. Örnek: **int?** a=5; **int** b=(**int**)a;
- Benzer şekilde tür dönüşüm kurallarına uymak şartıyla farklı dönüşüm kombinasyonları da mümkündür:
- **int?** a=5; **double** b=(**double**)a;
- Normal ve nullable nesneler arasında ters dönüşüm de mümkündür. Normal nesneler nullable nesnelere bilinçsiz olarak dönüşebilir. Örnek:
- **int** a=5; **int?** b=a;
- Nullable nesneler operatörler ile kullanılabilir. Örnek:
- **int?** a=5; **int?** b=10; **int** c=(**int**)(a+b);
- Burada a+b ifadesinin ürettiği değer yine int, **?** türünden olduğu için tür dönüşüm operatörü kullanıldı.

? İşareti

```
class NullableExample
  {   static void Main()
      {   int? num = null;
          if (num.HasValue == true)
          { Console.WriteLine("num = " + num.Value);}
          else
          { Console.WriteLine("num = Null");}
          //y değerine 0 atanıyor
          int y = num.GetValueOrDefault();
        // num.Value throws an InvalidOperationException if num.HasValue is false
          Console.WriteLine("y = " + y);
          try
          { y = num.Value;  }
          catch (InvalidOperationException e)
          { Console.WriteLine(e.Message);  }
      }
  }
```



?? Operatörü

- ?? operatörü **Nullable<T>** yapısındaki GetValueOrDefault() metoduna benzer şekilde çalışır. Örnek:

- int? a=null; int b=a??10;

- Burada eğer a null ise ?? operatörü 10 değerini döndürür.
- **??** operatörünün döndürdüğü değer normal (nullable olmayan) tiptedir.
- Eğer ilgili nullable nesne null değilse olduğu değeri döndürür. Başka bir örnek:

- int? a=5; int b=a??50;

- Burada ise 5 döndürülür.

- **??** operatöründe GetValueOrDefault() metodundan farklı olarak ilgili nesne null olduğunda döndürülecek değeri belirleyebiliyoruz.

Visual Studio.Net -C#

11. HAFTA

Göstericiler, Ön işlemciler, Temel Giriş Çıkış İşlemleri (I/O)

Try-catch-throw Örnekler

```
using System;
class notlar    {
        private int final;
        private int vize;
        public notlar(int f, int v)
        {               if(v>100)
                        {  throw new hatalınot(v,"yapıcı metot"); }
                        final=f;
                        vize=v;
        }
        public void degerver(int f,int v)
        {           if(v>100)
                        throw new hatalınot (v,"degerver metodu");
                    final=f;
                    vize=v;
        }
```

İstisnai Durum Yönetimi (Exception Handling)

```
public int Final
        {           get
                    { return final;}
                    set
                    { final=value;}
        }
        public int Vize
        {           get
                    { return vize;}
                    set
                    {  if (value>100)
                               throw new hatalınot (value,"Vize Giriş Hatası");
                      vize=value;
                    }
        }
```

İstisnai Durum Yönetimi (Exception Handling)

```
public class hatalınot:ApplicationException
        {       private int  hatalıvize;
                private string hatakaynagı;

                public hatalınot(int hatalıvize, string hatakaynagı)
                {       this.hatakaynagı=hatakaynagı;
                        this.hatalıvize=hatalıvize;
                }
                public int Hatalıvize
                {       get
                        {   return hatalıvize;}
                }
                public string Hatakaynagı
                {       get
                        {  return hatakaynagı; }
                }
```

İstisnai Durum Yönetimi (Exception Handling)

```
  public override string ToString()
  {       string str1="Hata Kaynağı: " + hatakaynagı+"\n";
          string str2="Hata Değeri: " + hatalıvize;
          return str1+str2;
  }
}
}
class hatatesti    {
          public static void Main()
          {               notlar a=new notlar(50,0);
                          try
                          {   a.Vize=120;            }

                          catch(notlar.hatalınot e)
                          {  Console.WriteLine(e.ToString());  }
          }
}
```

Örnekler

```
using System;
class deneme
  { static void Main()
  { for(;;)
  { try
  { Console.Write("Lütfen çıkmak için 0 ya da 1 girin: ");
  string a=Console.ReadLine();
if(a=="0"||a=="1") break;
  else
 throw new IndexOutOfRangeException("Devam ediliyor");
}
catch(IndexOutOfRangeException nesne)
{ Console.WriteLine(nesne.Message); continue; }
} } }
```

```
Lütfen çıkmak için 0 ya da 1 girin: 5
Devam ediliyor
Lütfen çıkmak için 0 ya da 1 girin: 1
```

Başka bir yol ile

```
using System;
class deneme
{ static void Main()
 { for(;;) {
try
{ Console.Write("Lütfen çıkmak için 0 ya da 1 girin: ");
string a=Console.ReadLine();
if(a=="0"||a=="1") break;
else
{ IndexOutOfRangeException nesne=new IndexOutOfRangeException("Başa dönüldü");
nesne.HelpLink="http://tr.wikibooks.org"; //Gördüğünüz gibi bu yöntemle nesnenin özelliklerini değiştirebiliyoruz.
throw nesne; }
}
catch(IndexOutOfRangeException nesne) { Console.WriteLine(nesne.Message); continue; } }
} }
```

Temsilciler-Örnek

- using System;
- class Temsilciler
- { public delegate void KomutMetodu(); *//Geri dönüş tipi void olan ve parametre almayan bir temsilci bildirdik.*
- public struct KomutYapisi
- { *//KomutYapisi isminde bir yapı başlattık.*
- public KomutMetodu KomutMetot; *//Yapımıza geri dönüş tipi KomutMetodu temsilcisi olan bir özellik ekledik.*
- public string Komut; *//Yapımıza bir özellik daha ekledik.*
- }
- public static void Komut1() { *//Sınıfımıza bir metot ekledik.*
- Console.WriteLine("Komut1 çalıştı."); }
- public static void Komut2() { *//Sınıfımıza bir metot daha ekledik.*
- Console.WriteLine("Komut2 çalıştı."); }
- public static void Komut3() *//Sınıfımıza bir metot ekledik.* { Console.WriteLine("Komut3 çalıştı."); }
- public static void Komut4() *//Sınıfımıza bir metot ekledik.*
- { Console.WriteLine("Komut4 çalıştı."); }
- static void Main()
- { KomutYapisi[] komutlar=new KomutYapisi[4]; *//KomutYapisi nesnelerinden oluşan bir dizi oluşturduk. (4 elemanlı)*
- komutlar[0].Komut="komut1"; *//komutlar[0] nesnesinin Komut özelliğine değer atadık.* komutlar[0].KomutMetot=new KomutMetodu(Komut1); *//Artık komutlar[0] nesnesinin KomutMetot özelliği Komut1'i temsil ediyor. Aynı durumlar diğer komutlar için de geçerli.*

Temsilciler-Örnek

- komutlar[1].Komut="komut2"; komutlar[1].KomutMetot=new KomutMetodu(Komut2); komutlar[2].Komut="komut3"; komutlar[2].KomutMetot=new KomutMetodu(Komut3); komutlar[3].Komut="komut4"; komutlar[3].KomutMetot=new KomutMetodu(Komut4); Console.Write("Komut girin: "); **string** GirilenKomut=Console.ReadLine(); **for**(**int** i=0;i<komutlar.Length;i++)*//komutlar dizisi içinde dolaşmaya çıkıyoruz.* **if**(GirilenKomut==komutlar[i].Komut) komutlar[i].KomutMetot(); } }

- Ekran Çıktısı:

- Komut girin: komut3
- Komut3 çalıştı.

Temsilcilerde kalıtım durumları

```
using System;
delegate Ana temsilciAna();          delegate Yavru temsilciYavru();
    class Ana { }
    class Yavru : Ana { }
    class program
    {static Ana MetotAna() {return new Ana(); }
    static Yavru MetotYavru(){return new Yavru(); }
        static void Main()
        {  temsilciAna nesneAna = new temsilciAna(MetotYavru);
            nesneAna += new temsilciAna(MetotAna);  nesneAna();
            temsilciYavru nesneYavru = new temsilciYavru(MetotYavru);
            nesneYavru();
              //ancak aşağıdaki kod hatalı.
            //nesneYavru = new temsilciYavru(MetotAna);
        }
    }

// Yavru sınıfın referansı ana sınıfa atanamaz
```

Olaylar (Events)-Örnek

- using System;
- delegate void OlayYoneticisi(); //Olay yöneticisi bildirimi
- class AnaProgram {
- static void Main()
- {
- AnaProgram nesne=new AnaProgram();
- nesne.Olay+=new OlayYoneticisi(Metot); //Olay sonrası işletilecek metotların eklenmesi nesne.Olay(); //Olayın gerçekleştirilmesi }
- //Olay sonrası işletilecek metot
- static void Metot() { Console.WriteLine("Butona tıklandı."); }

- event OlayYoneticisi Olay; //Olay bildirimi
- }

- Burada bütün üye elemanlar aynı sınıfta bildirildi. Bu örneğimizde Olay olayını herhangi bir metot üzerinden değil, direkt çağırıldı.

Olaylar (Events)-Örnek

```
using System;
public delegate int OlayYoneticisi();
interface IArayuz
{   int Metot1();    int Metot2();
    int sahteozellik    {   set;       get;    }
    int this[int indeks]    {       get;    }
    event OlayYoneticisi Olay;
}
class deneme:IArayuz
{    public int Metot1()    {       return 1;    }
    public int Metot2()    {       return 2;    }
    public int sahteozellik
     {       set{}
             get{return 3;}
    }
public int this[int indeks]
  {       get{return indeks;}    }
  public event OlayYoneticisi Olay;
  static void Main()    {
    deneme nesne=new deneme();
    nesne.Olay+=new OlayYoneticisi(nesne.Metot1);
    Console.WriteLine(nesne.Olay());
  }
}
```

- Çıktı: 1

Örnekler

```
using System;
delegate void OlayYoneticisi();
class Buton
{   OlayYoneticisi[] olay=new OlayYoneticisi[2];
//Olay yöneticimiz türünden iki elemanlı bir dizi oluşturduk.
   public event OlayYoneticisi ButonKlik
   {     add
         { int i;
        for(i=0;i<2;++i)
          if(olay[i]==null)
          //Buradaki value olaya eklenen metottur.
          { olay[i]=value; break;          }
          if(i==2)   Console.WriteLine("Olaya en fazla iki metot eklenebilir.");
      }
 remove
 {     int i;
```

```
for(i=0;i<2;++i)
            if(olay[i]==value)
            { olay[i]=null;       break;
            }
         if(i==2)
            Console.WriteLine("Metot bulunamadı");
        }
    }
    public void Kliklendi()
    {
       for(int i=0;i<2;++i)
          if(olay[i]!=null)
            olay[i]();
    }
 }
class Pencere
 {  int PencereNo;
    public Pencere(int no)
    {     PencereNo=no;   }
```

Örnekler

```
public void ButonKlik()
   {     Console.WriteLine("{0} nolu pencere olayı algıladı.",PencereNo);
   }
}
public class OlayTest
{   static void Main()
   {
      Buton buton=new Buton();
      Pencere p1=new Pencere(1);
      Pencere p2=new Pencere(2);
       //Geçerli ekleme:
      buton.ButonKlik+=new OlayYoneticisi(ButonKlik);
      buton.Kliklendi();
      Console.WriteLine();
//Geçerli ekleme:
      buton.ButonKlik+=new OlayYoneticisi(p1.ButonKlik);
      buton.Kliklendi();
      Console.WriteLine();
```

```
//Geçersiz ekleme (Olay dolu):
      buton.ButonKlik+=new OlayYoneticisi(p2.ButonKlik);
      buton.Kliklendi();
      Console.WriteLine();
       buton.ButonKlik-=new OlayYoneticisi(p1.ButonKlik);
      buton.Kliklendi();
      Console.WriteLine();
      buton.ButonKlik-=new OlayYoneticisi(ButonKlik);
      buton.Kliklendi();
      Console.WriteLine();
       //Geçersiz çıkarma (metot yok):
      buton.ButonKlik-=new OlayYoneticisi(ButonKlik);
      buton.Kliklendi();   }
   public static void ButonKlik()
   {     Console.WriteLine("Buton kliklendi");   }
}
```

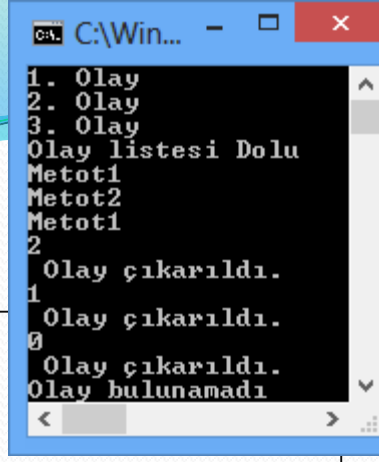
Bu programın ekran çıktısı şöyle olmalıdır:

```
Buton kliklendi
Buton kliklendi 1 nolu pencere olayı algıladı.
Olaya en fazla iki metot eklenebilir.
Buton kliklendi 1 nolu pencere olayı algıladı.
Buton kliklendi Metot bulunamadı
```

Örnek

```
using System;
delegate void OlayYoneticisi();
class deneme
{ OlayYoneticisi [ ] evnt = new OlayYoneticisi[3];
public int i = 0;   public int j = 0;
  public event OlayYoneticisi Olay
  {  add
     {  if (evnt[i] == null)
          {  evnt[i] = value;
             Console.WriteLine((i + 1) + ". Olay"); }
        i++;
          if (i == 3)
          { --i; Console.WriteLine("Olay listesi Dolu"); }
     }
     remove
     {      Console.WriteLine(i);
        {  if (evnt[i] == value)
          {  evnt[i] = null;
             Console.WriteLine(" Olay çıkarıldı.");            }
        i--;
          if (i ==-1)
          Console.WriteLine("Olay bulunamadı");
        }
     }
  }
```

```
// olayları ateşlemek için
    public void Olayim()
    {    for (int j = 0; j <=i; j++)
       {       if (evnt[j] != null) evnt[j]();     }
    }  }
  class Program
  {     static void Metot1() {
Console.WriteLine("Metot1"); } static void Metot2()
{ Console.WriteLine("Metot2"); }
    static void Main()
    {    deneme d = new deneme();
       d.Olay += new OlayYoneticisi(Metot1);
       d.Olay += new OlayYoneticisi(Metot2);
       d.Olay += new OlayYoneticisi(Metot1);
       d.Olayim();
       d.Olay -= new OlayYoneticisi(Metot1);
       d.Olay -= new OlayYoneticisi(Metot2);
       d.Olay -= new OlayYoneticisi(Metot1);
       d.Olayim();
    }
  }
```



Generic (Şablon) tipler

```
public class UrunTemel
{   private int urunKodu;
    private double urunFiyati;
    private string urunBilgisi;
    public UrunTemel(int kod, double fiyat, string bilgi)
    {
        urunKodu = kod;
        urunFiyati = fiyat;
        urunBilgisi = bilgi;
    }
    public int UrunKodu
    {  get { return urunKodu; }
    }
    public double BirimFiyat
    {  get { return urunFiyati; }
    }

    public string UrunTanimi
    {  get { return urunBilgisi; }
    }
}
```

```
public class Lastik : UrunTemel
{   private int capi;
    private int genislik;
    private string tipi;
    public Lastik(int kodu, double fiyat, string bilgi): base(kodu, fiyat, bilgi)   {   }
    public int Cap
    {   get { return capi; }
        set { capi = value; }
    }
    public int Genislik
    {   get { return genislik; }
        set { genislik = value; }
    }
    public string Tip
    {   get { return tipi; }
        set { tipi = value; }
    }
    public override string ToString()
    {  return UrunKodu.ToString() + " " + BirimFiyat.ToString() + " " + UrunTanimi + " " + Cap.ToString() + " " + Genislik.ToString() + Tip;
    }
}
```

Generic (Şablon) tipler

```
public class Silecek : UrunTemel
{
    private int uzunluk;

    public Silecek(int kodu, double fiyat, string bilgi) : base(kodu, fiyat, bilgi)
    {
    }
    public int Uzunluk
    {
        get { return uzunluk; }
        set { uzunluk = value; }
    }

    public override string ToString()
    {
        return UrunKodu.ToString() + " " + BirimFiyat.ToString() + " " + UrunTanimi + " " + Uzunluk.ToString();
    }
}
```

```
public class Urunler<T>  where T : UrunTemel
    {
        private List<T> urunListe;
        public Urunler()
        {
            urunListe = new List<T>();
        }

        public void Ekle(T uye)
        {    urunListe.Add(uye);
        }
        public void Sil(T uye)
        {
            urunListe.Remove(uye);
        }
        public void Listele()
        {
            foreach (T uye in urunListe)
            {
Console.WriteLine(uye.ToString());
            }
        }
    }
```

Generic (Şablon) tipler

```
class Ana
{
    static void Main()
    { Urunler<UrunTemel> urunler = new Urunler<UrunTemel>();

        Lastik lst = new Lastik(1000, 10, "Otomobil Lastiği");
        lst.Tip = "Kış Lastiği";
        lst.Cap = 185;
        lst.Genislik = 75;

        Silecek slc = new Silecek(1001, 5, "On silecek takimi");
        slc.Uzunluk = 60;

        urunler.Ekle(lst);
        urunler.Ekle(slc);

        urunler.Listele();
    }

}
```

Visual Studio.Net -C#

11. HAFTA

Göstericilier, Ön işlemciler, Temel Giriş Çıkış İşlemleri (I/O)

C#'ta Gösterici(Pointer) Kullanımı ve Emniyetsiz(unsafe) Kod

- Her ne kadar .NET platformunda bellek yönetimi gereksiz bilgi toplayıcısı(Garbage collector) dediğimiz mekanizma ile otomatik olarak sağlansa da bazen direk belleğe erişmemiz gerekebilir.

- Bu durumda işaretçiler(pointer) kullanılır. .NET'in alt yapısında pointerlar sıklıkla kullanılırken, pointer kullanımının programdaki görünmeyen hatalara sıkça yol açtığından dolayı programcılar için gizlenmiştir.

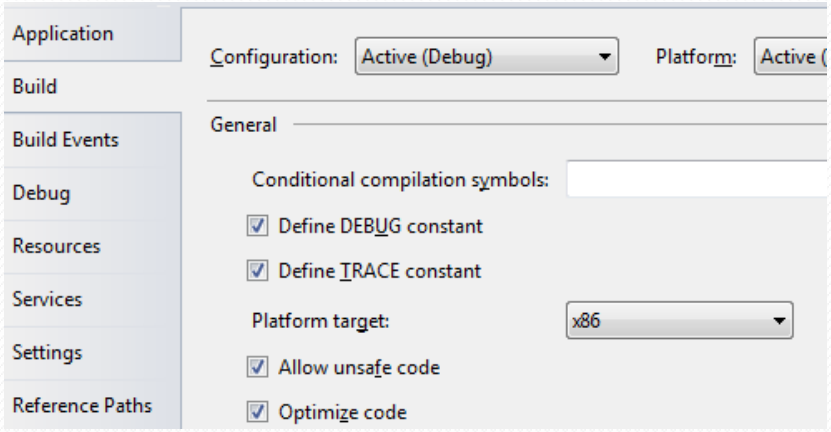
- pointer kullanımı birçok sıkıntıya yol açmaktadır. Bunlardan en önemlileri kullanım zorluğu ve bulunması zor hatalara yol açabilmesidir.

- C# 'ta pointer kullanılacağı zaman kod blokları **unsafe** anahtar kelimesi kullanılarak işaretlenmelidir. Aksi halde program derleme esnasında hata verecektir.

C#'ta Gösterici(Pointer) Kullanımı ve Emniyetsiz(unsafe) Kod

- Bir sınıfı unsafe olarak belirtirsek o sınıf içinde gösterici kullanabiliriz.
- **unsafe class** Sinif { ... }
- Herhangi bir metodun içinde bir unsafe bloğu oluşturarak o bloğun içinde gösterici kullanabiliriz.
- **int** Metot() { **unsafe** { ... } }
- Normal bir metodu unsafe olarak belirterek o metot içinde gösterici kullanabiliriz.
- **unsafe int** Metot() { ... }
- Özellikleri unsafe olarak belirterek yeni bir gösterici oluşturabiliriz. Bu durumda sınıfı unsafe olarak belirtmeye gerek kalmaz. Ancak bir metot içinde bir gösterici oluşturabilmek için ya o metodun unsafe olarak bildirilmesi ya da bir unsafe bloğunun açılması gerekir. Ayrıca içinde gösterici kullandığımız bir kaynak kod dosyası normal yollarla derlenemez. İçinde gösterici kullandığımız bir kaynak kod dosyamızı aşağıdaki iki yoldan biriyle derleriz.

- csc /unsafe KaynakKod.cs veya
- csc -unsafe KaynakKod.cs



Gösterici bildirimi

- Gösterici bildirimi ilgili yapı adının sonuna * işareti koyularak yapılır. Örnekler:
- char* gosterici1;
- int* gosterici2;
- Bu göstericilerden birincisi bir char türünden nesnenin (değişkenin), ikincisi de bir int türünden nesnenin adresini tutabilir.
- Birden fazla aynı türde gösterici oluşturmak için normal veri türlerinde olduğu gibi virgül kullanılabilir.
- Örnek:
- int* gosterici1, gosterici2;

- Göstericileri sabit ifadeler ile de bildirebiliriz. Örnek:
- **double*** gosterici=(**double***)123456;

- Burada bellekteki 123456 adresi gösterici göstericisine atanıyor. Ancak biz 123456 adresinde ne olduğunu bilmediğimiz için bu tür bir kullanım son derece tehlikelidir.

Gösterici bildirimi

- **& operatörü:**
- Bu operatör adres operatörüdür. & işlemi önünde yer aldığı değişkenin adresini döndürür. Örneğin aşağıda int türünden bir pointer' a daha önceden tanımlanan int türünden bir nesnenin adresi atanmaktadır.
  - **int sayi=99; int* p; p=&sayi;**
- *** Operatörü:**
- * operatörü bir adresteki bilgileri elde etmek ve değiştirmek için kullanılır. Aşağıda int üründe bir adres olan *sayi* nesnesinin adresine * operatörü ile yeni değer atanıyor.
  - **int sayi=99; int* p; p=&sayi; *p=100;**
- **Notlar:**
- **1-)** Henüz adresi olmayan bir göstericiye değer atanamaz. Örneğin şu kod hatalıdır:
- **int*** gosterici; *gosterici=10;
- **2-)** Ancak henüz bir değeri olmayan bir nesnenin adresi alınabilir. Yani şu kod hata vermez:
- **int** a; **int*** ptr=&a;

Göstericiler arasında tür dönüşümü

- Göstericiler arasında bilinçli tür dönüşümü yapılabilir. Ancak dikkatli olunmalıdır. Çünkü büyük türün küçük türe dönüşümünde veri kaybı yaşanabilir. Küçük türün büyük türe dönüşümünde de küçük tür bellekte kendine ait olmayan alanlardan da alır. Dolayısıyla her iki durumda da istemediğimiz durumlar oluşabilir. Göstericiler arasında bilinçsiz tür dönüşümü ise imkansızdır. Örnek bir program:
- **using** System;
- class Gostericiler {
- **unsafe static** void Main()
- { char c='A'; int i=80; char* cg=&c; int* ig=&i;
- cg=(char*)ig; //pointer adresini gösteriyor, aynı zamanda bilinçli tip dönüşümü
- Console.WriteLine(*cg);}
- }
- **Çıktı: P**
- Bu program sonucunda ekrana P yazılacaktır. (80'in Unicode karşılığı P'dir.) Bu programda herhangi bir veri kaybı gerçekleşmedi. Çünkü 80 char türünün kapasitesindeydi.

Göstericiler arasında tür dönüşümü

- Göstericiler yapı nesnelerinin bellekteki adreslerini tutarlar. İşte bu adresi elde etmek için ilgili gösterici bir tam sayı türüne dönüştürülmelidir. Örnek:
- `using System;`
- `class Gostericiler`
- `{  unsafe static void Main()`
- `{ int i=80; int* ig=&i; uint adres=(uint)ig;  Console.WriteLine("{0:X}",adres); }`
- `}`
- Bu program sonucunda benim bilgisayarım 13F478 çıktısını verir. Ancak bu sizde değişebilir. Adresi ekrana yazdırırken formatlama yapıp 16'lık düzende yazmalıyız. Çünkü bellek adresleri genellikle 16'lık düzendeki sayılarla ifade edilir.
- Göstericilerin adreslerini elde ederken uint türüne dönüşüm yapmak zorunlu değildir. İstediğiniz tam sayı türüne dönüşüm yapabilirsiniz. Ancak uint, bir bellek adresini tam olarak tutabilecek en küçük veri tipi olduğu için ilgili göstericiyi uint türüne dönüştürmeniz tavsiye edilir.

Göstericiler

- **void göstericiler**
- void tipinden göstericilere her türden adres atanabilir. void türü nesnelerdeki object türüne benzer. Örnek kod
- **int*** gosterici1;
- **void*** gosterici2;
- gosterici2=gosterici1;

- **Sizeof Operatörü:**
- Temel veri türlerinin ve yapılarının bellekte ne kadar yer kapladıklarını *sizeof* operatörü ile bulabiliriz. Örneğin
- sizeof(int)=4;

```
using System;
class Gostericiler {
 struct yapi
{ public int Ozellik1; public int Ozellik2; }
 unsafe static void Main()
{ Console.WriteLine(sizeof(yapi)); }
}
```

- Bu program 8 çıktısını verir.

Gösterici aritmetiği

```
using System;
class pointerkullanımı
        {
                unsafe public static void Main()
                { int* sayi1=(int*) 100;
                  char* sayi2=(char*) 100;
                  byte* sayi3=(byte*) 100;

                  sayi1+=2;   sayi2+=5;  sayi3+=6;

                  Console.WriteLine((uint) sayi1);
                  Console.WriteLine((uint) sayi2);
                  Console.WriteLine((uint) sayi3);
                }
        }
```

Programın çıktısı:

108
110
106

Gösterici aritmetiği

- Göstericiler yalnızca tam sayılarla matematiksel işleme girerler. +, -, --, ++, -= ve += operatörleri göstericilerle kullanılabilir. **void** göstericiler matematiksel işleme girmezler.
- İki gösterici birbirinden çıkarılabilir. Ancak bu durumda bir gösterici değil, long türünden bir nesne oluşur. Örnek:
-
- **using System;**
- **class Gosterici**
- **{**
- **unsafe static void Main()**
- **{ int* g1=(int*)500; int* g2=(int*)508;**
-
- **long fark=g2-g1; Console.WriteLine(fark);**
- **}**
- **}**
- Bu program sonucunda ekrana **2** yazılır.
- Çünkü göstericilerde çıkarma yapılırken şu formül kullanılır:

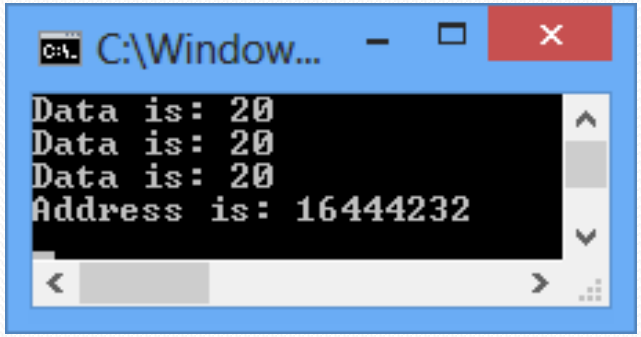
- (Birinci göstericinin adresi - İkinci göstericinin adresi)/Ortak türün byte cinsinden boyutu
- (508-500)/4=2-------→ int 4 byte yer kaplar

Gösterici aritmetiği

```
// compile with: /unsafe
 class UnsafeTest
{ // Unsafe method: takes pointer to int:
        unsafe static void SquarePtrParam(int* p)
       { *p *= *p; }
       unsafe static void Main()
      { int i = 5;
        // Unsafe method: uses address-of operator (&):
        SquarePtrParam(&i);
       Console.WriteLine(i);
      }
 }
// Output: 25
```

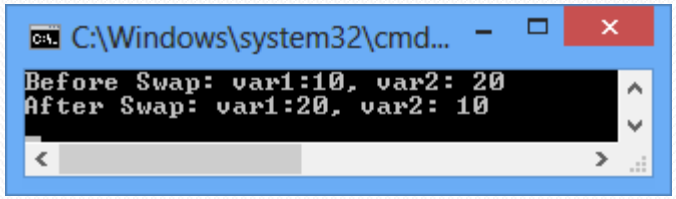
Gösterici aritmetiği

```
using System;
unsafe class Program
{
 static void Main()
 {
  int var = 20;
  int* p = &var;
  Console.WriteLine("Data is: {0} ", var);
  Console.WriteLine("Data is: {0} ", p->ToString());
  Console.WriteLine("Data is: {0} ", *p);
  Console.WriteLine("Address is: {0} ", (int)p);
 }
}
```



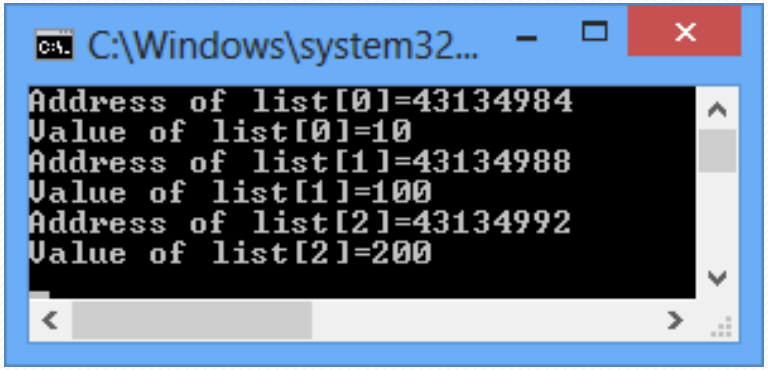
Gösterici aritmetiği

```
using System;
class TestPointer
{
    public unsafe void swap(int* p, int* q)
    {   int temp = *p;
        *p = *q;
        *q = temp;
    }
    public unsafe static void Main()
    {TestPointer p = new TestPointer();
     int var1 = 10;          int var2 = 20;
     int* x = &var1;         int* y = &var2;
     Console.WriteLine("Before Swap:var1:{0},var2: {1}", var1, var2);
     p.swap(x, y);
     Console.WriteLine("After Swap:var1:{0}, var2: {1}", var1, var2);
     Console.ReadKey();
    }
}
```



Gösterici aritmetiği

```
using System;
class TestPointer
{
 public unsafe static void Main()
 {
  int[] list = { 10, 100, 200 };
  fixed (int* ptr = list)
  /* let us have array address in pointer */
  for (int i = 0; i < 3; i++)
  {
    Console.WriteLine("Address of list[{0}]={1}", i, (int)(ptr + i));
    Console.WriteLine("Value of list[{0}]={1}", i, *(ptr + i));
   }
   Console.ReadKey();
 }
}
```



Önişlemci

- Önişlemci komutları programın derleme zamanında derleyiciyi yönlendirmemize yarayan komutlardır. İstemediğimiz satırların derleme modülü tarafından derlenmemesini sağlayabiliriz. Ayrıca, programda hata ayıklamak için ve kodun okunurluğunu arttırmak için de önişlemci komutları kullanılabilir. C#'da aslında önişlemci diye bir birim yoktur, bu birim derleyicinin bir bölümü olarak görülmekte.
- Komut çeşitleri;
  - `#define, #undef`
    `#if, #endif, #else, #elif`
    `#line,`
    `#warning, #error`
    `#region, #endregion`

Önişlemci komutları- #define

- **#define** komutu bir sembol tanımlamak için kullanılır. Sembollerin tek başlarına bir anlamı yoktur. #define ile sembol tanımlama program kodlarının en başında, (using deyimlerinden de önce) olmalıdır.

- Örnek program:
- `#define ENGLISH`
- `#define TURKISH`
- `using System;`
- `class Onislemci { static void Main() { } }`
-
- Bu programda ENGLISH ve TURKISH isimli iki sembol tanımlanmıştır. Diğer bütün önişlemci komutlarında olduğu gibi önişlemci komut parçaları arasında (örneğin #define ile TURKISH arasında) en az bir boşluk olmalıdır. Aynı sembolün tekrar tekrar tanımlanması hata verdirmez.

Önişlemci komutları-#undef

- **#undef,** #define ile tanımlanmış bir sembolü kaldırmak için kullanılır.
- Olmayan bir sembolü kaldırmaya çalışmak hata verdirmez. Programın herhangi bir satırında olabilir. Örnek program:

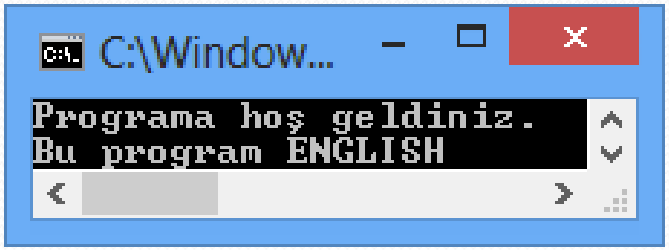
```
#define ENGLISH
using System;
#undef ENGLISH
class Onislemci
 {
    static void Main() { … }
}
```

- Sembollerin büyük ya da küçük harf olması zorunlu değildir. Ancak tamamen büyük harf okunurluğu artırdığı için tavsiye edilir. Sembollerin büyük-küçük harf duyarlılığı vardır.

Önişlemci komutları- #if , #endif

- **#if , #endif** komutları bir sembolün tanımlanmış olup olmamasına göre birtakım kodların derlenip derlenmemesinin sağlanması için kullanılır. Örnek program:

```
#define ENGLISH
using System;
class Onislemci
{static void Main()
 {
  Console.WriteLine("Programa hoş geldiniz.");
#if ENGLISH
  Console.WriteLine("Bu program ENGLISH");
#endif
}
}
```



Eğer programın başına #define ENGLISH satırını eklemeseydik program çıktısındaki ikinci satır olmayacaktı ve komut satırı pasif renkte gözükecekti

```
Console.WriteLine("Bu program ENGLISH");
```

Önişlemci komutları- #if , #endif

- #if önişlemci komutuyla mantıksal operatörleri de kullanabilir. Örnek program:

```
#define ENGLISH
#define TURKISH
using System;
class Onislemci {
  static void Main()
  { Console.WriteLine("Programa hoş geldiniz.");
#if ENGLISH && (!TURKISH)
  Console.WriteLine("Bu program ENGLISH");
#endif
 }
}
```

- Bu program sonucunda ekrana yalnızca **Programa hoş geldiniz**. satırı yazılacaktır.

- `#if` **ENGLISH && TURKISH** ENGLISH ve TURKISH varsa
  `#if` **ENGLISH || TURKISH** ENGLISH veya TURKISH varsa
  `#if` **ENGLISH && (!TURKISH)** ENGLISH var ama TURKISH yoksa
  `#if` **ENGLISH && (TURKISH==false)** ENGLISH var ama TURKISH yoksa
  Benzer şekilde çoklu koşullandırmalar da mümkündür.
- Örnek: `#if` **ENGLISH && TURKISH && FRANSIZCA**

Önişlemci komutları- #else

- **#else** önişlemci komutu C#'taki else ile aynı göreve sahiptir. Koşul sağlanmışsa bir kod bloğunun derlenmesini sağlayan #if komutu ile birlikte kullanılır. Örnek:

```
#define ENGLISH
using System;
class Onislemci
{
    static void Main()
    {
        Console.WriteLine("Programa hoş geldiniz.");
#if ENGLISH
        Console.WriteLine("Bu program ENGLISH");
#else
        Console.WriteLine("Bu program ENGLISH değil");
#endif
    }
}
```

Önişlemci komutları- #else

```
#define ENGLISH
using System;
#if ENGLISH
class Onislemci
{
    static void Main()
    {
        Console.WriteLine("Bu program ENGLISH");
    }
}
#else
class Onislemci
{
    static void Main()
    { Console.WriteLine("Bu program ENGLISH değil");
    }
}
#endif
```

Önişlemci komutları- #elif

- **#elif,** C#'taki elseif 'in kısaltılmışıdır. Yani değilse bir de bu koşula bak anlamı verir. Örnek program:

```
#define ENGLISH
using System;
class Onislemci
{ static void Main()
 {   Console.WriteLine("Programa hoş geldiniz.");
#if ENGLISH
     Console.WriteLine("Bu program ENGLISH");
#elif TURKISH
  Console.WriteLine("Bu program TURKISH");
#else
  Console.WriteLine("Bu program TURKISH ve ENGLISH değil");
#endif
 }
}
```

- #elif komutlarının sayısını bir #if-#endif aralığında istediğimiz kadar artırabiliriz.

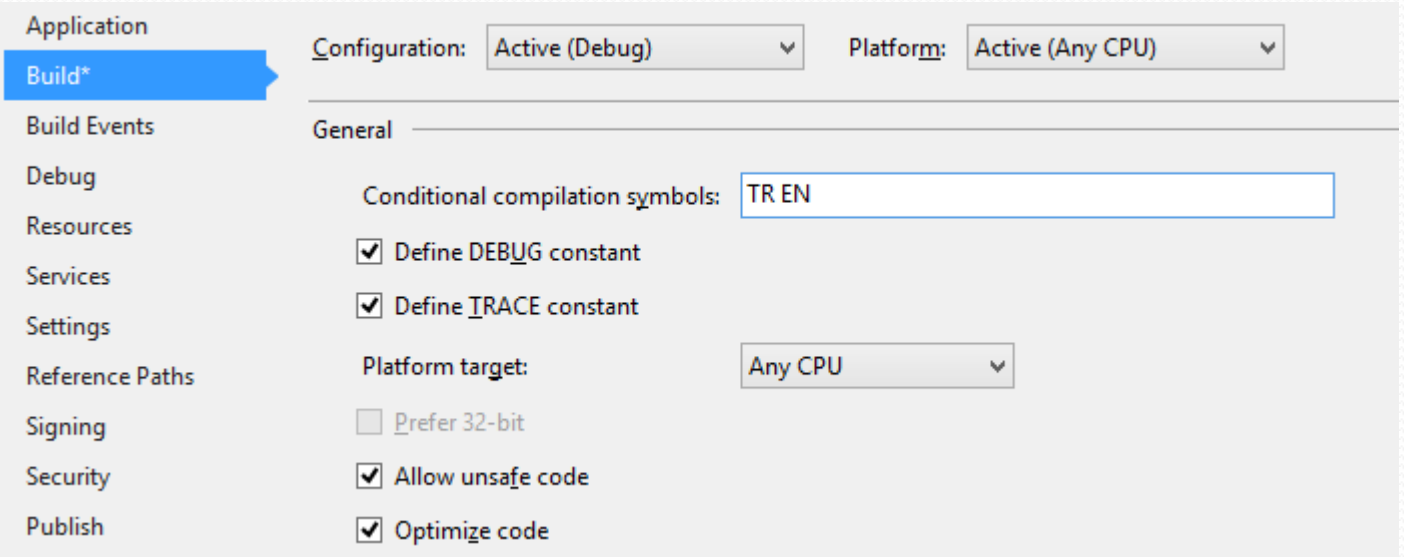
Önişlemci komutları- #elif

- İç içe geçmiş #if-#elif-#else komutları da olabilir. Örnek program:

```
#define ENGLISH
using System;
class Onislemci {
 static void Main() {
 #if ENGLISH
 #if TURKISH
     Console.WriteLine("Bu program hem ENGLISH hem TURKISH");
 #endif
 #elif TURKISH
     Console.WriteLine("Bu program sadece TURKISH");
 #endif
 }
}
```

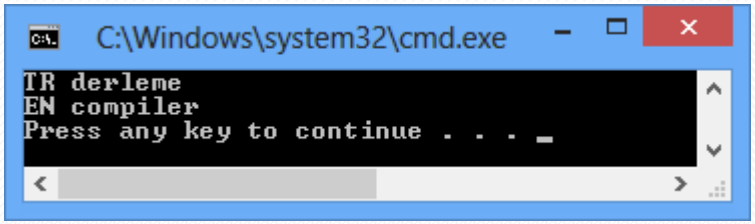
#define -1-Proje özelliklerinde koşul tanımlama

- #define önişlemci direktifi dışında, proje özelliklerinden faydalanarak da koşul tanımlaması yapılabilir.
- Bunun için proje özellikleri altındaki Build seçeneğinde, "Conditional compilation symbols: Koşullu derleme sembolleri " yazan yere derleme koşullarını söylemek gerekir. Birden fazla koşul tanımlaması yapılırken, koşullar arasında bir boşluk bırakılabilir, virgül ya da noktalı virgül konabilir.



#define -1-Proje özelliklerinde koşul tanımlama

```
using System;
class Program
{
    static void Main(string[] args)
    {
#if TR
        Console.WriteLine("TR derleme");
#endif
#if EN
        Console.WriteLine("EN compiler");
#endif
// Tanımlanan koşullar iki kelimeden oluşamazlar. Dolayısıyla aşağıdaki kod hata üretir.
        //#if TR EN
        //   Console.WriteLine("CENK OZDEMIR kosulu tanimli");
        //#endif
    }
}
```



#define -2- Komut satırında koşul tanımlama

- Komut satırından csc.exe'yi kullanarak yaptığımız derleme sırasında da koşul tanımlama imkanımız vardır. Bunun için, derlenmesini istediğimiz kaynak kod dosyasının adını söyledikten sonra /define anahtarından faydalanmak gereklidir.

```
Test.cs
using System;
class Program
{   static void Main(string[] args)
    {
#if KOSUL1
     Console.WriteLine("KOSUL1 tanimli");
#endif
#if KOSUL2
     Console.WriteLine("KOSUL2 tanimli");
#endif
    }
}
```

```
C:\>csc Test.cs /define:KOSUL1,KOSUL2
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

C:\>Test.exe
KOSUL1 tanimli
KOSUL2 tanimli

C:\>_
```

- Daha sonra kaynak kodumuzu komut satırından yandaki gibi derleyip çalıştırın

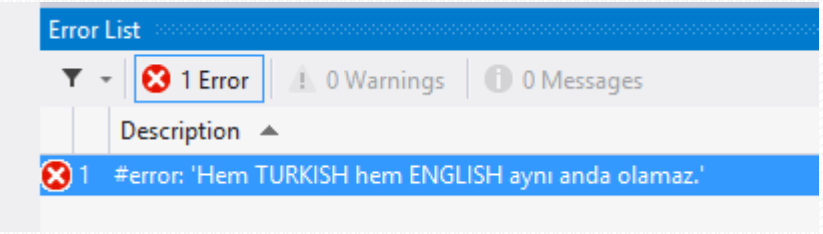
Önişlemci komutları-#error

- **#error** komutu derleyiciyi hata vermeye zorlamak için kullanılır.

- Örnek program:

```
#define ENGLISH
#define TURKISH
using System;
class Onislemci
{   static void Main()
    {
#if ENGLISH && TURKISH
#error Hem TURKISH hem ENGLISH aynı anda olamaz.
#endif
    }
}
```

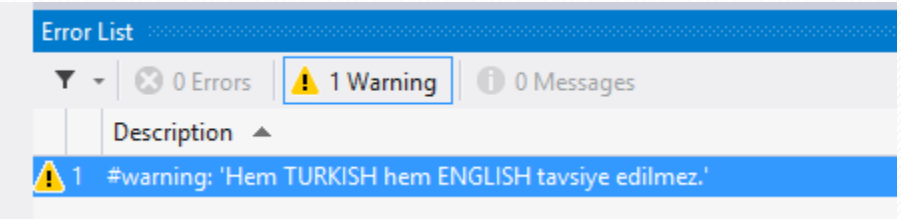
- Bu program kodu derlenmeyecek



Önişlemci komutları-#warning

- **#warning,** komutu #error komutu ile aynı mantıkta çalışır. Tek fark ekrana hata değil, uyarı verilmesidir. Yani programımız derlenir, ama uyarı verilir. Örnek program:

```
#define ENGLISH
#define TURKISH
using System;
class Onislemci {
 static void Main() {
  #if ENGLISH && TURKISH
   #warning Hem TURKISH hem ENGLISH tavsiye edilmez.
  #endif }
}
```



Önişlemci komutları

- **Örnek:**

```
class Program
{   static void Main(string[] args)
    {  int sayi = 1 / 0;  }
    static void Baglan()
    {
#error Baglan metodu henüz yazılmadı.
    }
    static bool KullaniciBul(int id)
    {
#error KullaniciBul metodu henüz yazılmadı.
    }
    static bool KullaniciSil(int id)
    {
#warning KullaniciSil metodu henüz yazılmadı.
    }
 }
```

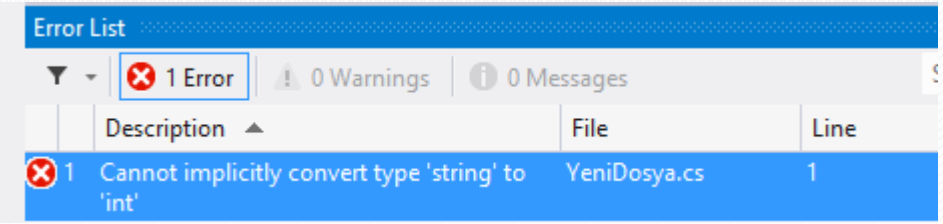
- Yukarıdaki kodu derlemeye çalıştığımızda Error List aşağıdaki gibi görünür.

- Burada dikkatinizi çekmek istediğim nokta, #error direktifini kullandığımızda, kodtaki diğer hataları ele alamıyor oluşumuzdur. Aslında kodumuz içerisinde üç tane hata mevcut (0'a bölmeye çalıştığımız satır, geriye bir şey döndürmeyen KullaniciBul(int id) ve KullaniciSil(int id) metotları) olmasına rağmen, yalnızca #error ve #warning direktiflerince üretilen hata ve uyarılar ele alınabilmektedirler.

Önişlemci komutları- #line

- **#line,** eğer programımızda bir hata var ve derleyici de (csc.exe) bu hatayı tespit etmişse bize bu hatanın hangi dosyanın hangi satırında olduğunu verir. İstersek #line komutuyla bu dosya adını ve satır numarasını değiştirebiliriz. Yani tabiri caizse derleyiciyi kandırabiliriz. #line komutu kendinden hemen sonra gelen satırın özelliklerini değiştirmek için kullanılır. Örnek program:

```
using System;
class Onislemci
{   static void Main()
    {
#line 1 "YeniDosya.cs"
        int a = "deneme";
    }
}
```

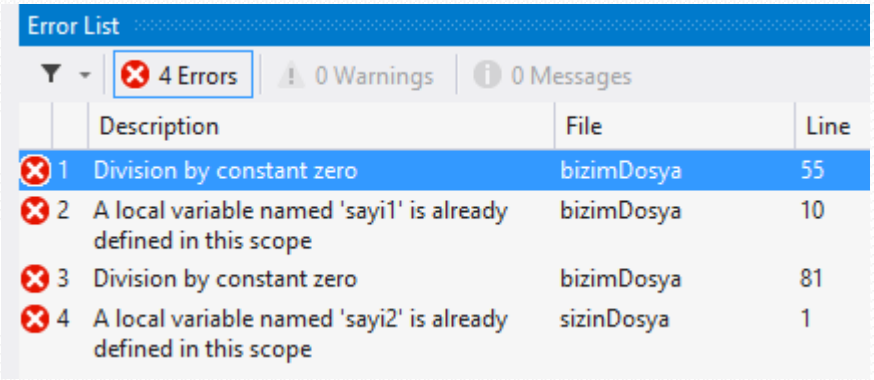


- Bu programda int türünden olan a değişkenine yanlış türde sabit atadık. Bu hatayı bilinçli olarak yaptık. Derleyici hatanın 1. satırda ve YeniDosya.cs dosyasında olduğunu iddia edecektir. Çünkü artık derleyici #line 1 "YeniDosya.cs" komutundan hemen sonra gelen satırı 1. satır ve YeniDosya.cs dosyası olarak tanıyor.

Önişlemci komutları- #line

- **Örnek:** Satır numaralarıyla oynamak

```
using System;
class Program
{
    static void Main(string[] args)
    {
#line 55 "bizimDosya"
        int sayi1 = 1 / 0;  //10
#line 10 "bizimDosya" //11
        int sayi1; //12
#line 80 "bizimDosya" //13
                      //14
        int sayi2 = 1 / 0;  //15
#line 1 "sizinDosya"  //16
        int sayi2;     //17
    }
}
```

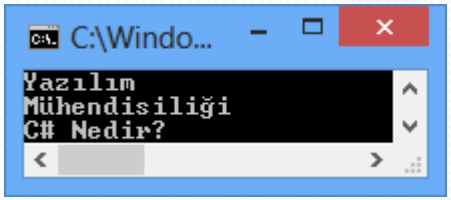


- Kod derlendikten sonra, hata oluşan satırlar 10,12,15 ve 17 olmasına rağmen aşağıdaki Error List görünür.

Önişlemci komutları- #line

- **Örnek:** Hata ayıklayıcısına yön vermek
- `using System;`
- `class Program`
- `{`
- `static void Main(string[] args)`
- `{`
- `#line hidden`
- `Console.WriteLine("Yazılım");`
- `Console.WriteLine("Mühendisiliği");`
- `#line default`
- `Console.Write("C#");`
- `#line hidden`
- `Console.WriteLine(" Nedir?");`
- `}`
- `}`

```
using System;
class Program
{
    static void Main(string[] args)
    {
#line hidden
        Console.WriteLine("Yazılım");
        Console.WriteLine("Mühendisiliği");
#line default
        Console.Write("C#");
#line hidden
        Console.WriteLine(" Nedir?");
    }
}
```



- #line önişlemci direktifinin bir diğer kullanım alanı da hata ayıklama sırasında içerisine girmek istemediğimiz kod blokları oluşturmaktır. Örnekteki kod içerisindeki hatalar ayıklanırken (debug) adım adım ilerlediğimizde(F10 veya F11), "#line hidden" satırından sonraki satırlar atlanır, "#line default" satırına gelindikten sonra ise normal davranışa geri dönülür.

Önişlemci komutları-#pragma warning

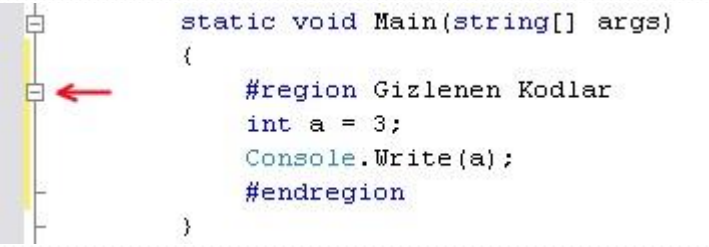
- C#'ta zaman zaman kodun çalışmasına engel teşkil etmeyen uyarılarla karşılaşırız. #pragma warning ifadesi ise bize tüm uyarıları ya da belli bir uyarıyı kapatıp/açma, kısaca uyarıları yönetme olanağı sunar.

```
static void Main(string[] args)
{
    // 0168 kodlu uyarı, tanımladığımız değişkeni kullanmadığımız durumlarda verilir.

    #pragma warning disable 0169 // 0169 kodlu uyarıyı etkisiz hale getirir
                int i;
    #pragma warning restore 0169 // 0169 kodlu uyarıyı aktif hale getirir
                int j;
    #pragma warning disable 0168 // 0168 kodlu uyarıyı etkisiz hale getirir
                int k;
    #pragma warning restore 0168 // 0168 kodlu uyarıyı aktif hale getirir
                int l;
    #pragma warning disable // Tüm uyarıları etkisiz hale getirir
                int x;
    #pragma warning restore // Tüm uyarıları aktif hale getirir
                int y;
}
```

- Tüm uyarılarlarla ilgili seçenekleri, proje özelliklerinde (Project Properties) kod analiz (Code Analysis) seçeneği altında bulabilirsiniz.

Önişlemci komutları-#region / #endregion

- **#region / #endregion,** komutları kodları bölgelere ayırmaya yarar.
- Kod bölgesinin yanındaki + işaretine tıklayınca kodlar gösteriliyor, gösterilen kodların yanındaki - işaretine tıklayınca da kodlarımız gizleniyor. Bu komut Visual Studio veya Not Defteri'nden daha gelişmiş bir editör programı kullananlar için faydalı olabilir. Örnek:
- **using** System;
- **class** Onislemci {
- **static void** Main()
- { #region YeniAlan
- *//Bu kısmı yeni bir bölge yaptık*
- #endregion
- }
- }



- Neden önişlemci komutları?
- Önişlemci komutları genellikle belirli koşullara göre programımızın yalnızca bir kısmını derlemek için kullanılır. Örneğin bir programın hem İngilizce hem Türkçe versiyonunu aynı kaynak kodda bulunduruyorsak önişlemci komutlarını programın yalnızca Türkçe ya da yalnızca İngilizce versiyonunu derlemek amacıyla kullanabiliriz.

Input (Giriş)/Output (Çıkış)

Temel I/O İşlemleri

Temel I/O işlemleri

- Programlama dilleri içerisinde yer alan önemli yapılardan biri olan I/O sistemi bilgisayarın çeşitli kaynaklarına erişmek için kullanılacak yollar anlamına gelir.
- Dosya kaydetme, ekrana ya da yazıcıya bilgi yazdırma, klavyeden bilgi girişi birer I/O işlemleridir.
- C# dilinde de I/O sistemi ile ilgili tüm sınıflar **System.IO** isim alanı altında bulunmaktadır.
- C# I/O sistemi akımlar (stream) üzerine kuruludur. Akımlar bir girdi ya da çıktı sistemi üzerinden byte düzeyinde bilgiyi okuyan sanal(soyut) birimlerdir.
- Stream sınıfı akımlar için gerekli temel metot ve özellikleri barındırır.

Directory sınıfı

- Directory sınıfının hiçbir özelliği yoktur, System.IO altında bulunur, sadece static metotlar içerir.
- **DirectoryInfo CreateDirectory (string adres):**
- adres ile belirtilen adreste bir klasör oluşturur ve bu klasör bilgilerini bir DirectoryInfo nesnesi olarak tutar. Programımızın çalıştığı klasörde bir klasör oluşturmak için sadece klasörün adını yazmak yeterlidir. Örnekler:
  - **Directory.CreateDirectory(@"C:\WINDOWS\deneme");**
  - Bu kod C:\WINDOWS altında deneme isimli bir klasör oluşturur.
  - **Directory.CreateDirectory("deneme");**
  - Bu kod programın çalıştığı klasörde deneme isimli bir klasör oluşturur.
  - **Directory.CreateDirectory(@"..\deneme");**
  - Bu kod programın çalıştığı klasörün bir üst klasöründe deneme isimli bir klasör oluşturur.
  - **Directory.CreateDirectory(@"..\..\deneme");**
  - Bu kod programın çalıştığı klasörün iki üst klasöründe deneme isimli bir klasör oluşturur. .. sayıları bu şekilde artırılabilir. Bu tür bir adres belirtme şekli bütün diğer metotlarda da geçerlidir. Ayrıca bu ve diğer bütün metotlarda da adres diye tarif ettiğimiz veriye dosya/klasörün adı da dâhildir.

Directory sınıfı

- **void Delete(string adres)**
- Belirtilen adresteki boş klasörü silmek için kullanılır. Başka bir kullanımı daha vardır.
- void Delete(string adres,bool a)
  - Bu metot ile eğer a true ise belirtilen adresteki klasör, içindeki bütün dosya ve klasörlerle birlikte silinir.
- **bool Exists(string adres)**
- Belirtilen adresteki klasörün olup olmadığını bool cinsinden tutar. Klasör varsa true, yoksa false döndürür.
- **string GetCurrentDirectory()**
- Çalışan programın hangi klasörde olduğunu verir. Örneğin Windows'taysak C:\WINDOWS'u tutar.
- **string[] GetDirectories(string adres)**
- Belirtilen adresteki bütün klasörleri adresleriyle birlikte bir string dizisi olarak tutar.
- **string GetDirectoryRoot(string adres)**
- Belirtilen adresteki klasörün kök dizin bilgisini verir. Örneğin adres C:\Program Files\Program ise C:\ değerini döndürür.

Directory sınıfı

- **string[] GetFiles(string adres)**
- Belirtilen adresteki dosyaları adresleriyle birlikte string dizisi olarak tutar. Bu ve benzer metotlarda liste İngilizce alfabetik sırasına göredir. GetFiles() metodunun bir prototipi daha vardır:
- **string[] GetFiles(string adres, string dosya)**
- Adresteki dosya(lar) adresleriyle birlikte string dizisi olarak tutulur. Dosya isminde joker karakterleri (*, ?) kullanılabilir.
- **string[] GetFileSystemEntries(string adres)**
- Belirtilen adresteki bütün dosya ve klasörleri adresleriyle birlikte bir string dizisi olarak tutar.
- **DateTime GetLastAccessTime(string adres)**
- Belirtilen adresteki dosya ya da klasöre en son ne zaman erişildiğini DateTime türünden tutar.
- **DateTime GetLastWriteTime(string adres)**
- Belirtilen adresteki dosya ya da klasörün en son ne zaman değiştirildiğini DateTime türünden tutar.

Directory sınıfı

- **DateTime GetCreationTime(string adres)**
- Belirtilen adresteki dosya ya da klasörün ne zaman oluşturulduğunu DateTime türünden tutar.
- **string[] GetLogicalDrives()**
- Bilgisayardaki bütün sürücüleri bir string dizisi olarak tutar. Bu sürücülere her türlü sabit disk, CD-ROM sürücü, flash disk vb. dâhildir.
- **DirectoryInfo GetParent(string adres)**
- Belirtilen adresin bir üst klasörünü DirectoryInfo nesnesi olarak döndürür.
- **void Move(string kaynak_adres,string hedef_adres)**
- Dosya ve klasörleri bir konumdan başka bir konuma taşır. Örnekler:
  - **Directory.Move(@"C:\Belgelerim\dosya.rtf", @"C:\Web\deneme.rtf");**
  - Bu kod C:\Belgelerim altındaki dosya.rtf dosyasını C:\Web konumuna deneme.rtf adıyla taşır.
  - **Directory.Move(@"C:\Web",@"C:\Belgelerim\internet");**
  - Bu kod sürücü klasöründeki Web klasörünü tüm içeriğiyle birlikte C:\Belgelerim klasörüne internet adıyla taşır. Eğer hedef klasörde aynı adlı dosya ya da klasör varsa çalışma zamanı hatası alırsınız.

Directory sınıfı

- **void SetLastAccessTime(string adres, DateTime zaman)**
- Belirtilen adresteki dosya ya da klasörün en son erişim zamanını zaman olarak değiştirir.
- **void SetLastWriteTime(string adres, DateTime zaman)**
- Belirtilen adresteki dosya ya da klasörün en son değiştirilme zamanını zaman olarak değiştirir.
- **void SetCreationTime(string adres, DateTime zaman)**
- Belirtilen adresteki dosya ya da klasörün oluşturulma zamanını zaman olarak değiştirir.
- **void SetCurrentDirectory(string adres)**
- Programın çalıştığı klasörü belirtilen adres ile değiştirir.

Directory sınıfı

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string yol = @"C:\Klasor";
        Directory.CreateDirectory(yol);
        Console.WriteLine("Klasör Oluşturuldu...");
        Console.ReadLine();
        Console.Clear();

        Console.WriteLine("Klasör Var mı: " + Directory.Exists(yol));
        Console.WriteLine("Oluşturulma Zamanı: " + Directory.GetCreationTime(yol));
        Console.WriteLine("Son Erişim Zamanı : " + Directory.GetLastAccessTime(yol));
        Console.WriteLine("Son Yazma Zamanı  : " + Directory.GetLastWriteTime(yol));

        Console.WriteLine("Şu anki klasör: " + Directory.GetCurrentDirectory());
        Console.ReadLine();
        Console.Clear();

        Console.WriteLine("C Klasör Listesi:");
        string[] klasorler = Directory.GetDirectories(@"C:\");

        foreach (string s in klasorler)
            Console.WriteLine(s);

        Console.ReadLine();
        Console.Clear();
```

Directory sınıfı

```
Console.WriteLine("Kök Klasör:");
Console.WriteLine(Directory.GetDirectoryRoot(yol));
Console.ReadLine();
Console.Clear();

Console.WriteLine("C Dosya Listesi");
string[] dosyalar = Directory.GetFiles(@"C:\");

foreach (string s in dosyalar)
    Console.WriteLine(s);

Console.ReadLine();
Console.Clear();

Console.WriteLine("C Dosya ve Klasör Listesi");
string[] tum = Directory.GetFileSystemEntries(@"C:\");

foreach (string s in tum)
    Console.WriteLine(s);

Console.ReadLine();
Console.Clear();

Console.WriteLine("Mantıksal Sürücüler:");
string[] mantiksalsuruculer = Directory.GetLogicalDrives();

foreach (string s in mantiksalsuruculer)
    Console.WriteLine(s);

Console.ReadLine();
Console.Clear();
```

Directory sınıfı

```
        Console.WriteLine("Bir üst klasör:");
        Console.WriteLine(Directory.GetParent(@"C:\Windows\system32"));

        Console.ReadLine();
        Console.Clear();

        Directory.Move(@"C:\Klasor", @"C:\Windows\Klasor");
        Console.WriteLine("Klasör Taşındı...");
        Console.ReadLine();
        Console.Clear();

        Directory.SetCurrentDirectory(@"C:\Windows");

        Console.WriteLine("Klasör Silindi...");
        Directory.Delete("Klasor");
    }
}
```

File sınıfı

- File sınıfındaki birçok metot Directory sınıfında da vardır, tek farkları aynı görevleri dosyalar için yerine getirmeleridir. Bu metotlar şunlardır: Exists(), Delete(), GetCreationTime(), GetLastAccessTime(), GetLastWriteTime(), Move(), SetCreationTime(), SetLastAccessTime(), SetLastWriteTime(). File sınıfının Directory sınıfında olmayan metotları:
- **StreamWriter AppendText (string adres)**
- Adreste belirtilen dosya için daha sonra göreceğimiz bir StreamWriter nesnesi oluşturur.
- **void Copy (string kaynak, string hedef)**
- Kaynakta belirtilen dosya hedefe kopyalanır. Kopyalamada bir isim çakışması söz konusuysa, yani sizin kopyalama yapmak istediğiniz klasörde zaten aynı isimli bir dosya varsa çalışma zamanı hatası alırsınız. Bunu önlemek içinse;
- void Copy(string kaynak,string hedef,bool a)
  - Burada **a true** olursa eğer hedef klasörde aynı adlı dosya varsa üstüne yazılır. **a false** yapılırsa iki parametreli hâlinden farkı kalmaz.

File sınıfı

- **FileStream Create (string adres, int tampon)**
- Belirtilen adresteki dosya oluşturulur ve dosyaya ilişkin FileStream nesnesi döndürülür. Tampon yazılmazsa yani sadece bir parametre yazılırsa varsayılan tampon miktarı kullanılır.
- **StreamWriter CreateText (string adres)**
- Belirtilen adreste üzerine yazmak için bir text dosyası oluşturulur ve dosyaya ilişkin StreamWriter nesnesi döndürülür.
- **FileAttributes GetAttributes (string adres)**
- Belirtilen adresteki dosya ya da klasörün FileAttributes enumu cinsinden özniteliği döndürülür. FileAttributes enum sabiti şu sözcükleri içerir: Archive, Compressed, Device, Directory, Encrypted, Hidden, Normal, NotContentIndexed, Offline, ReadOnly, ReparsePoint, SparseFile, System, Temporary. Bir dosya ya da klasörün birden fazla özniteliği olabilir. Bu durumda öznitelikler virgülle ayrılır.

File sınıfı

- **Open**
- Üç farklı aşırı yüklenmiş çeşidi vardır. Bunlar:
  - **FileStream Open(string adres, FileMode a)**
  - **FileStream Open(string adres, FileMode a, FileAccess b)**
  - **FileStream Open(string adres, FileMode a, FileAccess b, FileShare c)**
  - Open() metodu belirtilen adresteki dosyayı açar ve dosyaya ilişkin FileStream nesnesini döndürür. FileMode, FileAccess ve FileShare System.IO isim alanında bulunan enumlardır ve dosyanın ne şekilde açılacağını ve üzerinde ne şekilde işlem yapılacağını belirtirler.

- **FileMode enumunda bulunan sözcükler**
  - **Append** Açılan dosyanın sonuna ekleme yapmak için kullanılır. Eğer dosya bulunmazsa oluşturulur.
  - **Create** Yeni dosya oluşturmak için kullanılır. Zaten dosya varsa üzerine yazılır.
  - **CreateNew** Yeni dosya oluşturmak için kullanılır, belirtilen dosya mevcutsa çalışma zamanı hatası verdirir.
  - **Open** Dosyayı açmak için kullanılır.
  - **OpenOrCreate** Belirtilen dosya varsa açılır, yoksa yenisi oluşturulur.
  - **Truncate** Belirtilen dosya açılır ve içi tamamen silinir.

File sınıfı

- **FileAccess enumunda bulunan sözcükler**
  - **Read** Dosya okumak için kullanılır.
  - **ReadWrite** Dosya okunmak ve yazılmak üzere açılır.
  - **Write** Dosya sadece yazılmak için açılır.

- **FileShare enumunda bulunan sözcükler**
  - **Inheritable** Dosyanın child (yavru) prosesler tarafından türetilebilmesini sağlar.
  - **None** Dosyanın aynı anda başka prosesler tarafından açılmasını engeller.
  - **Read** Dosyanın aynı anda başka proseslerce de açılabilmesini sağlar.
  - **ReadWrite** Dosyanın aynı anda başka proseslerce de açılıp, okunup, yazılabilmesini sağlar.
  - **Write** Dosyaya aynı anda başka proseslerce yazılabilmesini sağlar.

File sınıfı

- **FileStream OpenRead(string adres)**
- Belirtilen dosyayı yalnızca okumak için açar ve dosyaya ilişkin FileStream nesnesini döndürür.

- **StreamReader OpenText(string adres)**
- Belirtilen dosyayı yalnızca text modunda okumak için açar ve dosyaya ilişkin StreamReader nesnesini döndürür.

- **FileStream OpenWrite(string adres)**
- Belirtilen dosyayı yazma modunda açar ve dosyaya ilişkin FileStream nesnesini döndürür.

File sınıfı

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string[] tumdosyalar = Directory.GetFileSystemEntries(@"C:\");

        foreach (string s in tumdosyalar)
        {
            Console.Write(s + " ==> ");
            Console.WriteLine(File.GetAttributes(s));
        }
    }
}
```

DirectoryInfo sınıfı

- DirectoryInfo sınıfı Directory sınıfının aksine static olmayan metot ve özellikleri içerir. Önce özellikleri bir örnek üzerinde görelim:
  - `using System;`
  - `using System.IO;`
  - `class DirectoryInfoSinifi {`
  - `static void Main() {`
  - `string adres= @"C:\WINDOWS" ;`
  - `DirectoryInfo d=new DirectoryInfo(adres);`
  - `Console.WriteLine("Özellikler: "+d.Attributes);`
  - `Console.WriteLine("Oluşturulma tarihi: "+d.CreationTime);`
  - `Console.WriteLine("Var mı? "+d.Exists);`
  - `Console.WriteLine("Uzantı: "+d.Extension);`
  - `Console.WriteLine("Tam adres: "+d.FullName);`
  - `Console.WriteLine("Son erişim zamanı: "+d.LastAccessTime);`
  - `Console.WriteLine("Son değişiklik zamanı: "+d.LastWriteTime);`
  - `Console.WriteLine("Klasör adı: "+d.Name); Console.WriteLine("Bir üst klasör: "+d.Parent);`
    `Console.WriteLine("Kök dizin: "+d.Root); } }`

DirectoryInfo sınıfı

- DirectoryInfo sınıfının metotlarının tamamı static değildir. Bu metotların çalışması için gereken adres bilgisi, kendisine ulaşılması için kullanılan DirectoryInfo nesnesindedir.
- **void Create()**
- Klasör oluşturur.

- **DirectoryInfo CreateSubdirectory(string adres)**
- Belirtilen adreste bir alt dizin oluşturur.
  - Örneğin C:\deneme altında \deneme2\deneme3 dizini oluşturmak için;
  - `string adres=@"C:\deneme";`
  - `DirectoryInfo d=new DirectoryInfo(adres);`
  - `d.Create();`
  - `DirectoryInfo alt=d.CreateSubdirectory("deneme2");`
  - `alt.CreateSubdirectory("deneme3");`
  - CreateSubdirectory metodu kendisine ulaşılan nesne içinde parametredeki klasörü oluşturuyor ve oluşturduğu klasörü de DirectoryInfo nesnesi olarak döndürüyor.

DirectoryInfo sınıfı

- **Delete**
- İki farklı aşırı yüklenmiş versiyonu vardır.
  - void Delete()
  - void Delete(bool a)
  - Birincisinde klasör boşsa silinir, ikincisinde a true ise klasör, içindeki her şeyle silinir.
- **DirectoryInfo[] GetDirectories()**
  - İlgili klasörde bulunan bütün dizinleri bir DirectoryInfo dizisinde tutar.
- **FileInfo[] GetFiles()**
  - İlgili klasörde bulunan bütün dosyaları bir FileInfo dizisinde tutar.
- **FileSystemInfo[] GetFileSystemInfos()**
  - İlgili klasördeki bütün dosya ve klasörler bir FileSystemInfo dizisinde tutulur.
- **void MoveTo(string hedef)**
  - İlgili dizin, içindeki tüm dosya ve klasörlerle beraber hedefe taşınır.
- **void Refresh()**
  - İlgili klasörün özelliklerini diskten tekrar yükler.
- **NOT:** Bütün taşıma ve kopyalama işlemlerinde kaynak ve hedef aynı sürücüde olmalıdır.

DirectoryInfo sınıfı

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string yol = @"C:\Windows\System32";

        DirectoryInfo d = new
                    DirectoryInfo(yol);

        Console.Write("Özellikler:");
        Console.WriteLine(d.Attributes);

        Console.Write("Oluşturma Tarihi:");
        Console.WriteLine(d.CreationTime);

        Console.Write("Klasör var mı?");
        Console.WriteLine(d.Exists);

        Console.Write("Uzantı:");
        Console.WriteLine(d.Extension);
```

```
        Console.Write("Tam Yol:");
        Console.WriteLine(d.FullName);

        Console.Write("Son Erişim Zamanı:");
        Console.WriteLine(d.LastAccessTime);

        Console.Write("Son Yazma Zamanı:");
        Console.WriteLine(d.LastWriteTime);

        Console.Write("Klasör Adı:");
        Console.WriteLine(d.Name);

        Console.Write("Üst Klasör:");
        Console.WriteLine(d.Parent);

        Console.Write("Kök Klasör:");
        Console.WriteLine(d.Root);
    }
}
```

FileInfo sınıfı

- FileInfo özellikleri aşağıdaki örnek üzerinde gösterilmiştir.
  - using System;
  - using System.IO;
  - class FileInfoSinifi {
  - static void Main() {
  - string adres=@"C:\WINDOWS\deneme.txt";
  - FileInfo d=new FileInfo(adres);
  - Console.WriteLine("Öznitelikler: "+d.Attributes);
  - Console.WriteLine("Oluşturulma tarihi: "+
  - d.CreationTime); Console.WriteLine("Var mı? "+d.Exists);
  - Console.WriteLine("Uzantı: "+d.Extension);
  - Console.WriteLine("Tam adres: "+d.FullName);
  - Console.WriteLine("Son erişim zamanı: "+d.LastAccessTime);
  - Console.WriteLine("Son değişiklik zamanı: "+d.LastWriteTime);
  - Console.WriteLine("Boyut: "+d.Length);
  - Console.WriteLine("Klasör adı: "+d.Name);
  - Console.WriteLine("Bulunduğu klasör: "+d.DirectoryName); } }

FileInfo sınıfı

- FileInfo sınıfı File sınıfındaki AppendText(), Create(), CreateText(), Delete(), Open(), OpenRead(), OpenText() ve OpenWrite() metotlarını içerir. Bunlara daha önce değinilmişti.
- File sınıfında olmayan metotlarsa;
- **CopyTo**
- İki aşırı yüklenmiş versiyonu vardır:
  - FileInfo CopyTo (**string** hedef)
  - FileInfo CopyTo (**string** hedef,**bool** a)
  - Birincisinde ilgili dosya hedefe kopyalanır. Hedefte aynı adlı dosya varsa çalışma zamanı hatası alırsınız. İkincisinde a true ise, hedefte aynı adlı dosya varsa üzerine yazılır.

- **void MoveTo(string hedef)**
- İlgili dosya hedefe taşınır.

- **void Refresh()**
- İlgili dosyanın bilgileri diskten tekrar alınır.

FileInfo sınıfı

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string yol = @"C:\Windows\clock.avi";

        FileInfo f = new FileInfo(yol);

        Console.Write("Özellikler:");
        Console.WriteLine(f.Attributes);

        Console.Write("Oluşturma Tarihi:");
        Console.WriteLine(f.CreationTime);

        Console.Write("Dosya var mı?");
        Console.WriteLine(f.Exists);

        Console.Write("Uzantı:");
        Console.WriteLine(f.Extension);
```

```
        Console.Write("Tam Ad:");
        Console.WriteLine(f.FullName);

        Console.Write("Son Erişim Zamanı:");
        Console.WriteLine(f.LastAccessTime);

        Console.Write("Son Yazma Zamanı:");
        Console.WriteLine(f.LastWriteTime);

        Console.Write("Boyut:");
        Console.WriteLine(f.Length);

        Console.Write("Dosya Adı:");
        Console.WriteLine(f.Name);

        Console.Write("Bulunduğu Klasör:");
        Console.WriteLine(f.DirectoryName);
    }
}
```

Path sınıfı

- Path sınıfı çeşitli işlemler yapan static üye elemanlara sahiptir. Örnek program:

```
using System; using System.IO;
class PathSinifi {
static void Main() {
string adres=@"C:\dizin\deneme.txt";
Console.WriteLine("Uzantı: "+Path.GetExtension(adres));
string yeniAdres=Path.ChangeExtension(adres,"jpg");
Console.WriteLine("Yeni uzantı: "+Path.GetExtension(yeniAdres));
string adres2=@"C:\klasör";
Console.WriteLine("Yeni adres: "+Path.Combine(adres,adres2));
Console.WriteLine("Klasör: "+Path.GetDirectoryName(adres));
Console.WriteLine("Dosya adı: "+Path.GetFileName(adres));
Console.WriteLine("Uzantısız dosya adı: "+Path.GetFileNameWithoutExtension(adres));
Console.WriteLine("Tam adres: "+Path.GetFullPath(adres));
Console.WriteLine("Kök dizin: "+Path.GetPathRoot(adres));
Console.WriteLine("Geçici dosya adı: "+Path.GetTempFileName());
```

Path sınıfı

```
• Console.WriteLine("Geçici dosya dizini: "+Path.GetTempPath());
• Console.WriteLine("Dosya uzantısı var mı? "+Path.HasExtension(adres));
• Console.WriteLine("Alt dizin ayıracı: "+Path.AltDirectorySeparatorChar);
•  Console.WriteLine("Dizin ayıracı: "+Path.DirectorySeparatorChar);
• Console.Write("Geçersiz dosya adı karakterleri: ");
• char[] dizi=Path.GetInvalidFileNameChars();
•  foreach(char b in dizi) Console.Write(b+" ");
•  Console.Write("\nGeçersiz adres karakterleri: ");
•  char[] dizi2=Path.GetInvalidPathChars();
• foreach(char b in dizi) Console.Write(b+" ");
•   Console.WriteLine("\nAdres ayırıcı karakter: "+Path.PathSeparator);
•   Console.WriteLine("Kök dizin ayıracı: "+Path.VolumeSeparatorChar); }
• }
```

Path sınıfı

```
string adres =@"C:\dizin\
string adres2=@"\klasör\deneme.txt"
Path.Combine(adres,adres2) sonucu
```

Yeni adres C:\dizin\klasör\deneme.txt olur

- Uzantı: **.txt**
- Yeni uzantı: **.jpg**
- Yeni adres: **C:\klasör**
- Klasör: **C:\dizin**
- Dosya adı: **deneme.txt**
- Uzantısız dosya adı: **deneme**
- Tam adres: **C:\dizin\deneme.txt**
- Kök dizin: **C:**
- Geçici dosya adı: **C:\Users\ERKAN TANYILDIZI\AppData\Local\Temp\tmpA303.tmp**
- Geçici dosya dizini: **C:\Users\ERKAN TANYILDIZI\AppData\Local\Temp**
- Dosya uzantısı var mı? **True**
- Alt dizin ayıracı: **/**
- Dizin ayıracı: ****
- Geçersiz dosya adı karakterleri: **" < > |** ☺ ☻ ♥ ♦ ♣ ♠ ♫ ☼ ► ◄ ↕ ‼ ¶ § ▬ ↨ ↑ ↓ → ← ∟ ↔ ▲ ▼ **: * ? \ /**
- Geçersiz adres karakterleri: " < > | ☺ ☻ ♥ ♦ ♣ ♠ ♫ ☼ ► ◄ ↕ ‼ ¶ § ▬ ↨ ↑ ↓ → ← ∟ ↔ ▲ ▼ : * ? \ /
- Adres ayırıcı karakter: **;**
- Kök dizin ayıracı: **:**

Path sınıfı

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string yol = @"C:\windows\clock.avi";

        Console.Write("Uzantı: ");
        Console.WriteLine(Path.GetExtension(yol));

        Console.Write("Klasör: ");
        Console.WriteLine(Path.GetDirectoryName(yol));

        Console.Write("Dosya Adı: ");
        Console.WriteLine(Path.GetFileName(yol));

        Console.Write("Uzantısız Dosya Adı:");
        Console.WriteLine(Path.GetFileNameWithoutExtension(yol));

        Console.Write("Tam Yol: ");
        Console.WriteLine(Path.GetFullPath(yol));

        Console.Write("Kök Dizin: ");
        Console.WriteLine(Path.GetPathRoot(yol));
```

Path sınıfı

```
        Console.Write("Geçici Dosya Adı: ");
        Console.WriteLine(Path.GetTempFileName());

        Console.Write("Geçici Dosya Dizini: ");
        Console.WriteLine(Path.GetTempPath());

        Console.Write("Uzantı var mı? : ");
        Console.WriteLine(Path.HasExtension(yol));

        Console.Write("Alternatif Alt Dizin Ayracı: ");
        Console.WriteLine(Path.AltDirectorySeparatorChar);

        Console.Write("Dizin Ayracı: ");
        Console.WriteLine(Path.DirectorySeparatorChar);

        Console.Write("Geçersiz Karakterler: ");
        Console.WriteLine(Path.GetInvalidPathChars());

        Console.Write("Yol Ayracı: ");
        Console.WriteLine(Path.PathSeparator);

        Console.Write("Sürücü Ayracı: ");
        Console.WriteLine(Path.VolumeSeparatorChar);
    }
}
```

Dosya yazma ve okuma işlemleri

- **FileStream sınıfı**
- FileStream sınıfı ile diskteki bir dosya açılır. StreamReader ve StreamWriter sınıflarıyla üzerinde işlem yapılır. Dosyalar üzerinde metin tabanlı ve bayt tabanlı işler yapabiliriz. Bir FileStream nesnesi çok değişik yollarla oluşturulabilir.
- Örnekler:

```
using System;
using System.IO;
class Program
{static void Main()
 {string adres = @"C:\Program Files\deneme.txt";
  FileStream FSnesnesi1 = new FileStream(adres, FileMode.OpenOrCreate);
  FileStream FSnesnesi2 = new FileStream(adres, FileMode.OpenOrCreate, FileAccess.Write);
  FileStream FSnesnesi3 = new FileStream(adres, FileMode.OpenOrCreate, FileAccess.Write,
FileShare.None);
  FileInfo FInesnesi1 = new FileInfo(adres);
  FileStream FSnesnesi4 = FInesnesi1.OpenRead();
  FileInfo FInesnesi2 = new FileInfo(adres);
    }
}
```

Dosya yazma ve okuma işlemleri

- `FileStream FSnesnesi5 = FInesnesi2.OpenWrite();`
- `FileInfo FInesnesi3 = new FileInfo(adres);`
- `FileStream FSnesnesi6 = FInesnesi3.Create();`
- `FileInfo FInesnesi4 = new FileInfo(adres);`
- `FileStream FSnesnesi7 = FInesnesi4.Open(FileMode.OpenOrCreate);`

- Dosyayla ilgili işlemimiz bittiğinde FileStream sınıfının **Close**() metodu ile FileStream nesnesi tarafından tutulan kaynaklar boşaltılır ve dosyayı başka prosesler kullanabilir hâle gelir. Close() metodunun kullanılışı:

  - **FSnesnesi1.Close();**

FileStream sınıfı ile yazma ve okuma

- FileStream sınıfının **Read**() ve **ReadByte**() metotları dosya akımından byte düzeyinde veri okumamızı sağlarlar.
- ReadByte() metodu akımdan okuma yapamadığı zaman geriye -1 değerini döndürür. ReadByte() metodunun prototipi:
- int **ReadByte**()
- Bu metot ile akımdan bir baytlık bilgi okunur ve akımdaki okuma pozisyonu bir artırılır ki tekrar okuma da aynı değer okunmasın. Okunan byte değeri inte dönüştürülüp int olarak tutulur.
- İkinci metodumuz ise:
- int **Read**(byte[] dizi, int baslangic, int adet)
- Bu metot ile adet kadar bayt akımdan okunur, okunan bu baytlar dizi dizisine baslangic indeksinden itibaren yerleştirilir. Geri dönüş değeri okunan byte sayısıdır.

FileStream sınıfı ile yazma ve okuma

- Şimdi komut satırı argümanı olarak adı alınan dosyanın içeriğini ekrana yazan bir program yazalım:
  - `using System; using System.IO;`
  - `class DosyaAkimi {`
  - `static void Main(string[] args) {`
  - `string adres=args[0]; FileStream fs=new FileStream(adres,FileMode.Open);`
  - `int OkunanByte;`
  - `while((OkunanByte=fs.ReadByte())!=-1) Console.Write((char)OkunanByte); }`
  - `}`
- Bir dosya akımına bir byte yazmak için
- `void WriteByte(byte veri)`
- metodu kullanılır. Eğer yazma işlemi başarısız olursa çalışma zamanı hatası oluşur.
- Dosya akımına bir bayt dizisi yazdırmak içinse
- `void Write(byte[] dizi,int baslangic,int adet)`
- metodu kullanılır. Bu metot ile byte dizisinin baslangic indeksinden itibaren adet kadar elemanı akıma yazılır. Akımın konum göstericisi yazılan byte kadar ötelenir.

FileStream sınıfı ile yazma ve okuma

- Dosya akımına yazılan veriler dosya sistemindeki dosyaya hemen aktarılmaz. Dosya akımı tamponlama mekanizması ile çalıştığı için belirli bir miktar veri yazılana kadar dosya güncellenmez. Ancak FileStream sınıfının **Flush()** metodunu kullanarak istediğimiz anda tamponu boşaltıp dosyayı tampondaki bilgilerle güncelleyebiliriz.
- Örnek:
- **using** System;
- **using** System.IO;
- **class** deneme { **static void** Main(**string**[] args)
- { **string** dosya=args[0];
- **FileStream d=new FileStream(dosya,FileMode.CreateNew,FileAccess.Write);**
- Console.Write("Dosyanın içeriğini yazın: ");
- **string** icerik=Console.ReadLine();
- **foreach**(**char** a **in** icerik)
- d.WriteByte((**byte**)a);
- d.Flush(); } }

FileStream sınıfı ile yazma ve okuma

- FileStream sınıfının önemli özellikleri:
- **bool CanRead** Akımdan okuma yapılıp yapılamayacağı öğrenilir.
- **bool CanSeek** Akımda konumlandırma yapılıp yapılamayacağı öğrenilir.
- **bool CanWrite** Akıma yazma yapılıp yapılamayacağı öğrenilir.
- **long Position** Akımdaki aktif konum bilgisi öğrenilir.
- **long Length** Akımın bayt olarak büyüklüğü öğrenilir.
- FileStream sınıfının önemli metotları:
- **void Lock(long pozisyon,long uzunluk)** Bu metotla akımın pozisyondan itibaren uzunluk kadar alanı başka proseslerin erişimine kapatılır.
- **long Seek(long ofset, SeekOrigin a)** Bu metotla akımın konumu **SeekOrigin** ile belirtilmiş konumdan **ofset** byte kadar ötelenir. **SeekOrigin** enumunun içerdiği sözcükler şunlardır:
  - **Begin** Akımın başlangıç noktası
  - **Current** Akımın o anda bulunduğu nokta
  - **End** Akımın en son noktası

FileStream Sınıfı

```
using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {
        string yol = @"Dosya.txt";
        FileStream fsw = new FileStream(yol, FileMode.Create, FileAccess.Write);

        byte[] veri = new byte[20];

        for (int i = 0; i < veri.Length; i++) veri[i] = (byte)(i + 1);

        fsw.Write(veri, 0, veri.Length);
        Console.WriteLine("{0} byte yazıldı...", veri.Length);

        fsw.Close();

        FileStream fsr = new FileStream(yol, FileMode.Open, FileAccess.Read);

        byte[] okunan = new byte[fsr.Length];

        fsr.Read(okunan, 0, (int)fsr.Length);

        foreach (byte b in okunan) Console.Write(b + " ");

        fsr.Close();
    }
}
```

StreamReader sınıfı

- StreamReader sınıfı FileStream sınıfının aksine baytlarla değil, metinlerle ilgilenir. Bir StreamReader nesnesinin oluşturulma yöntemleri:
  - `string dosya=@"C:\deneme\ornek.txt";`
  - `FileStream fs=new FileStream(dosya,FileMode.Open);`
  - `StreamReader sr1=new StreamReader(fs);`
  - `StreamReader sr2=new StreamReader(dosya);`
  - `FileInfo fi=new FileInfo(dosya);`
  - `StreamReader sr3=new StreamReader(fi);`
- Diğer sınıflarda olduğu gibi StreamReader nesneleriyle işimiz bittiğinde **Close**() metodunu kullanarak kaynakların iade edilmesi tavsiye edilir.

- StreamReader sınıfının önemli metotları:
- `string` ReadLine()
- Akımdan bir satırlık veriyi okur ve string olarak tutar. Eğer veri okunamazsa null değeri tutar. Okunan satırın sonuna "\n" eklenmez.

StreamReader sınıfı

- string ReadToEnd()
- Akımdaki verilerin tamamını string olarak tutar. Okuma işlemi aktif konumdan başlayacaktır. Eğer okuma yapılamazsa boşluk tutar.
- int Read()
- Akımdan bir karakterlik bilgi okunur ve int'e dönüştürülür. İşlem başarısız olursa -1 ile geri döner.
- int Read(char[] dizi,int indeks,int adet)
- Akımdan adet kadar karakteri dizi[indeks] elemanından itibaren diziye yerleştirir. Yerleştirilen karakter sayısını döndürür.
- int Peek()
- Akımdan bir karakterlik bilgi okunur ve bu karakterin inte dönüşmüş hâli ile geri dönülür. İşlem başarısız olursa -1 ile geri döner. En önemli nokta ise konum göstericisinin yerinin değiştirilmemesidir.

StreamReader sınıfı

- Örnek:
- using System;
- using System.IO;
- class Deneme {
- static void Main() {
- string dosya="deneme.txt";
- FileStream fs=new FileStream(dosya,FileMode.Open);
- StreamReader sr=new StreamReader(fs);
- string satir;
- while((satir=sr.ReadLine())!=null)
- Console.WriteLine(satir);
- fs.Close();
- }
- }

StreamWriter sınıfı

- StreamReader sınıfı ile dosyalardan text tabanlı veriler okunabiliyordu. StreamWriter sınıfı ise bunun tam tersini yapar. Yani StreamWriter sınıfı ile dosyalara text tabanlı veriler yazılır. Bir StreamWriter nesnesi şu yollarla oluşturulabilir:

  - `string dosya=@"C:\dosya.txt";`
  - `FileStream fs=new FileStream(dosya,FileMode.Open);`
  - `StreamWriter sw1=new StreamWriter(fs);`
  - `StreamWriter sw2=new StreamWriter(dosya);`
  - `FileInfo fi=new FileInfo(dosya);`
  - `StreamWriter sw3=fi.CreateText();`

- StreamReader sınıfında olduğu gibi StreamWriter sınıfında da **Close**() metoduyla StreamWriter nesnesine ilişkin kaynaklar iade edilir.

StreamWriter sınıfı

- StreamWriter sınıfının en önemli metotları:
- void **Write**(string str)
- Bu metotla akıma str yazısı eklenir. Yazının sonuna herhangi bir sonlandırıcı karakter konmaz. Bu metot ile diğer bütün veri türlerinden veri eklemek mümkündür. Örneğin: Write(5), Write(true), Write('c')

- void **WriteLine**(string str)
- Write metoduyla aynı işi yapar. Tek fark eklenen yazının sonuna kendisi "\n" stringini koyar. Ayrıca Write metodundan farklı olarak WriteLine() metodunu parametresiz kullanabiliriz. Bu durumda akıma sadece "\n" eklenir.

- void **Flush**()
- Tampondaki bilgilerin boşaltılmasını ve dosyanın güncellenmesini sağlar.

- Ayrıca StreamWriter sınıfının **NewLine** özelliği ile satır ayıracı olan karakterleri belirleyebiliriz. Varsayılan olarak bu karakter "\n" ve "\r"dir.

StreamWriter sınıfı

- Örnek:
- using System;
- using System.IO;
- class Deneme {
- static void Main(string[] args) {
- string dosya=args[0];
- FileStream fs=new FileStream(dosya,FileMode.Append,FileAccess.Write);
- StreamWriter sw=new StreamWriter(fs);
- Console.Write("Yazınızı girin: ");
- string yazi=Console.ReadLine();
- sw.Write(yazi);
- sw.Flush();
- }
- }

StreamWriter Sınıfı

```
using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {
        string yol = @"C:\Dosya.txt";

        FileStream fs = new FileStream(yol, FileMode.Append, FileAccess.Write);
        StreamWriter sw = new StreamWriter(fs);

        while (true)
        {
            string giris = Console.ReadLine();
            if (giris.ToLower() == "x") break;
            sw.WriteLine(giris);
        }
        sw.Flush(); sw.Close();

        Console.Clear();

        StreamReader sr = new StreamReader(yol);

        string satir; int i=1;

        while ((satir = sr.ReadLine()) != null)
        {
            Console.WriteLine("{0}:{1}",i,satir);
            i++;
        }
        sr.Close(); fs.Close();
    }
}
```

BinaryWriter ve BinaryReader sınıfları

- **BinaryWriter** ve **BinaryReader** sınıflarının yaptığı iş StreamReader ve StreamWriter sınıflarının yaptığı işin aynısıdır.

- Ancak BinaryWriter ve BinaryReader sınıflarıyla StreamReader ve StreamWriter sınıflarından farklı olarak her türde veriyi akıma yazdırabiliriz, yani verinin illaki string olma zorunluluğu yoktur.

- Bu sınıflarda sırasıyla Write(int), Write(char), Write(char[]), Write(byte) ve ReadByte(), ReadChar(), ReadUInt32, ReadString() vb. metotlar bulunmaktadır.

- Bu metotlar bütün temel veri türleri için bildirilmiştir.

BinaryWriter ve BinaryReader sınıfları

- using System;
- using System.IO;
  - class Deneme {
  - static void Main()
  - { int i=5; decimal d=15.54M; char c='A'; string dosya="deneme.txt";
  - FileStream fs1=new FileStream(dosya,FileMode.OpenOrCreate);
  - BinaryWriter bw=new BinaryWriter(fs1); bw.Write(i); bw.Write(d);
  - bw.Write(c); bw.Close();
  - FileStream fs2=new FileStream(dosya,FileMode.Open);
  - BinaryReader br=new BinaryReader(fs2); Console.WriteLine(br.ReadInt32());
  - Console.WriteLine(br.ReadDecimal());
  - Console.WriteLine(br.ReadChar()); br.Close();
- } }
- **NOT:** BinaryWriter sınıfıyla oluşturulan bir dosyayı Notepad ile okumaya kalkarsanız anlamsız simgelerle karşılaşırsınız. Çünkü BinaryWriter sınıfı dosyalara text yöntemiyle değil, binary yöntemle kayıt yapar. BinaryReader sınıfı da bu binary kayıtları okur.

Console I/O işlemleri

- I/O işlemleri için gerekli olan sınıflardan System.IO isim alanında olmayan tek sınıf Console sınıfıdır. Console sınıfını ekrana bir şeyler yazmak için ya da kullanıcıdan girdi almak için sıklıkla kullanıldı.
- Konsol I/O işlemleri için önceden tanımlanmış üç tane standart akım mevcuttur. Bu akımlar TextWriter türü olan Console.Out, Console.Error ve TextReader türünden olan Console.In 'dir. Konsol ekranına yazdığımız yazılar aslında TextWriter sınıfının metotları ile olmaktadır. Console.WriteLine ise bizim için sadece bir aracılık görevi yapar. Örneğin aşağıdaki her iki satır da eş görevdedir. İkisi de ekrana merhaba yazar.
- Console.**Out**.WriteLine("merhaba"); Console.WriteLine("merhaba");
- Yani özetle Out, Console sınıfına bağlı bir özelliktir ve geri dönüş tipi TextWriter'dır ve bu veriyle TextWriter sınıfının static olmayan bir metodu olan WriteLine()'a erişiriz.
- Konsol ekranına yazı yazdırmak için Console sınıfının WriteLine() ve Write() metotlarını kullanırız. Bu ikisi arasındaki tek fark WriteLine'ın yazının sonuna "\n" ekleyip, Write'ın eklememesidir.

Console I/O işlemleri

- Konsoldan bilgi almak için Read() ve ReadLine() metotlarını kullanırız. Eğer tamponda herhangi bir veri yoksa Read() metodu kullanıcıdan veri girişi ister ve girilen stringteki ilk karakteri int olarak tutar. Sonraki Read() metotları ise o stringteki diğer karakterleri tutar. Eğer veri okunamazsa -1 değeri tutar.
- Örnek:
- Console.Write((char)Console.Read()+" "+(char)Console.Read()+" "+(char)Console.Read());

- Bu satırda önce kullanıcıdan veri girişi istenir. Diyelim ki kullanıcı **deneme** girmiş olsun. Bu durumda ekrana
- **d e n**
- yazılır. ReadLine() metodu Read() metodunun aksine ekrandan girdileri string olarak alır. Console.In özelliğini kullanarak da Read() ve ReadLine() metotlarını çağırabiliriz. Ayrıca Console.Out özelliğini kullanarak da Write() ve WriteLine() metotlarını kullanabiliriz.

Console I/O işlemleri

- Console.In'den erişebileceğimiz yani TextReader sınıfının diğer metotları:

- int Peek()
- Bu metot ile standart girdi akımından bir karakter okunur ancak bu karakter tampondan silinmez.
- Örnek: Console.Write((char)Console.In.Peek()+" "+(char)Console.In.Peek()+" "+(char)Console.In.Peek());
- Burada kullanıcının ekrana **deneme** yazdığını varsayarsak ekrana **d d d** yazılacaktır.

- int ReadBlock(char[] dizi,int indeks,int adet)
- Bu metot ile standart girdi akımından adet kadar karakter diziye indeks elemanından itibaren yerleştirilir.

- string ReadToEnd()
- Bu metot ile standart girdi akımındaki bütün veriler okunarak tampondan temizlenir. Okunan veriler string nesnesi olarak döndürülür.

Standart akımların yönlendirilmesi

- C#'ta bütün I/O işlemleri akımlar (stream) üzerine kuruludur. Standart akımları başka akımlara yönlendirmek mümkündür. Örnek olarak komut satırında bir programı şöyle çalıştırırsak
- programadi > deneme.txt
- Bu programın, ekran çıktılarını deneme.txt dosyasına yazacağı anlamına gelir. Bu da standart çıktı olan konsolun başka akıma yönlendirilmesi anlamına gelir. Komut satırından
- programadi < deneme.txt
- Burada da Console.ReadLine() ya da Console.Read() yerine deneme.txt dosyasındaki metin kullanılır. C# programlarımız içinde akımları yönlendirmek içinse Console sınıfının şu metotları kullanılır.
- **static void SetOut(StreamWriter sw) veya static void SetOut(TextWriter tw)**
  static void SetError(StreamWriter sw) veya static void SetError(TextWriter tw)
  static void SetIn(StreamReader sr) veya static void SetIn(TextReader sr)

Standart akımların yönlendirilmesi

- Örnek:Console.In akımını SetIn metodu ile bir dosya akımına yönlendirerek girişi klavyeden değil, dosyadan alsın;

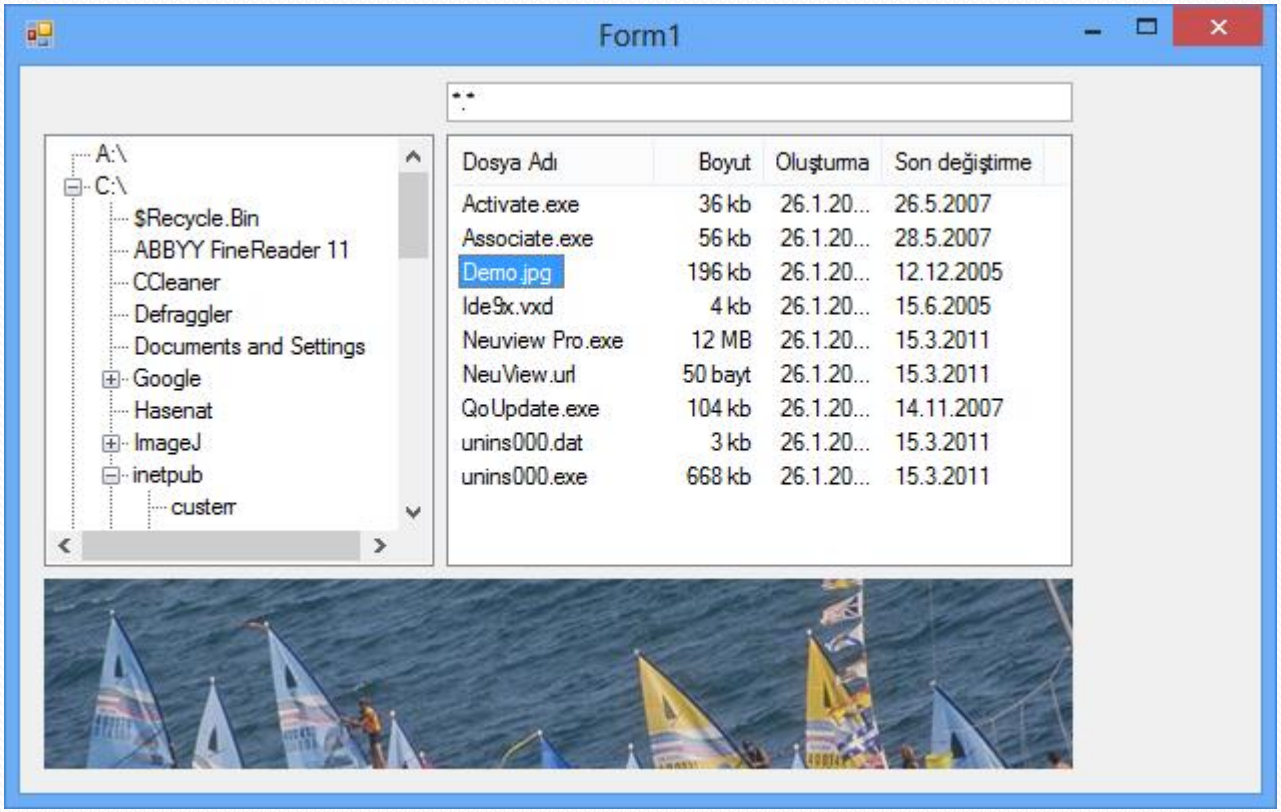
```
using System;
using System.IO;
class Deneme {
static void Main() {
FileStream fs=new FileStream("deneme.txt",FileMode.Open);
Console.SetIn(new StreamReader(fs));
Console.WriteLine(Console.ReadLine());
Console.WriteLine(Console.ReadLine()); }
}
// burada ReadLine ile kullanıcıdan giriş alınmaz, dosyadan alır
```

Standart akımların yönlendirilmesi

- Örnek: Console.WriteLine() metodu da dosyaya yönlendirilebilir. Tabii ki bunun için Console.SetOut metodu kullanılmalıdır. Programın içindeki hata mesajlarını ise SetError metodu ile bir dosyaya yazdırılabilir.
  - `using System;`
  - `using System.IO;`
  - `class Deneme {`
  - `static void Main() {`
  - `Console.WriteLine("Bu bir denemedir.");`
  - `FileStream fs1 = new FileStream("deneme.txt", FileMode.Create);`
  - `TextWriter tw = Console.Out;`
  - `StreamWriter sw = new StreamWriter(fs1);`
  - `Console.SetOut(sw); Console.WriteLine("Dosyaya yazma yapıyoruz."); //dosyaya`
  - `Console.SetOut(tw); Console.WriteLine("Merhaba dünya"); //ekrana`
  - `sw.Close(); } }`
- Burada önce ekrana Bu bir denemedir. yazılıyor. Sonra yeni bir FileStream nesnesi oluşturuluyor. Sonra hedefi konsol ekranı olan bir TextWriter nesnesi oluşturuluyor. Sonra ilişiği deneme.txt olan bir StreamWriter nesnesi oluşturuluyor. Sonra çıktı birimi deneme.txt olarak değiştiriliyor. Sonra ekrana -daha doğrusu dosyaya- Dosyaya yazma yapıyoruz. yazdırılıyor. Sonra çıktı birimi yeniden ekran yapılıyor. Sonra ekrana Merhaba dünya yazdırılıyor. En son da sw akımının kaynakları boşaltılıyor.

Visual Studio.Net -C#
12. HAFTA
FORM UYGULAMALARI

Örnek –Sadece özelliklerini ver örneği verme



Örnek

```
private void Form1_Load(object sender, System.EventArgs e)
 { string[] sürücüler = System.IO.Directory.GetLogicalDrives();
  int i;
  for (i = 0 ; i<= sürücüler.GetUpperBound(0) ; i++)
   treeView1.Nodes.Add(sürücüler[i]);
  //Detay modunda göster
  listView1.View = View.Details;
  //Colonları ekle
  listView1.Columns.Add("Dosya Ady", -2, HorizontalAlignment.Left);
  listView1.Columns.Add("Boyut", -2, HorizontalAlignment.Right);
  listView1.Columns.Add("Oluşturma", -2, HorizontalAlignment.Right);
  listView1.Columns.Add("Son değiştirme", -2, HorizontalAlignment.Left);
  listView1.Sorting = SortOrder.Ascending; //Artan syralama
   textBox1.Text="*.*";
 }
```

Örnek

```
void listele(string yol)
{   int i;   string[] k;   listView1.Items.Clear();
string[] dosyalar  = System.IO.Directory.GetFiles(yol,textBox1.Text);
for (i = 0 ; i<= dosyalar.GetUpperBound(0) ; i++)
{ ListViewItem Eleman = new ListViewItem();
 k = dosyalar[i].Split('\\');
 Eleman.Text = k[k.GetUpperBound(0)];
 System.IO.FileInfo FI = new System.IO.FileInfo(dosyalar[i]);
 long boyut = FI.Length;
 if (boyut < 1024)
  Eleman.SubItems.Add(boyut.ToString() + " bayt");
 else
  if (boyut < 1024 * 1024)
  Eleman.SubItems.Add((boyut / 1024).ToString() + " kb");
 else
 Eleman.SubItems.Add((boyut / (1024 * 1024)).ToString() + " MB");
 Eleman.SubItems.Add(FI.CreationTime.ToShortDateString());
 Eleman.SubItems.Add(FI.LastWriteTime.ToShortDateString());
```

Örnek

```
 //ReadOnly dosyalary mavi göster
  if ((FI.Attributes & System.IO.FileAttributes.ReadOnly)>0)
   Eleman.ForeColor = Color.Blue;
  //gizli ve system dosyalaryny kyrmyzy göster
  if ((FI.Attributes & System.IO.FileAttributes.Hidden)>0 ||
   (FI.Attributes & System.IO.FileAttributes.System)>0)
   Eleman.ForeColor = Color.Red;
  listView1.Items.Add(Eleman);
 }
}
private void treeView1_AfterSelect(object sender,
System.Windows.Forms.TreeViewEventArgs e)
{  //klasörleri listele
 try
 { string[] klasörler =
   System.IO.Directory.GetDirectories(e.Node.FullPath + "\\");
  int i;    string[] k;
```

Örnek

```
 //Daha önce eklenmemişse alt klasörleri ekle
 if (e.Node.GetNodeCount(true) == 0)
  for (i = 0 ; i<= klasörler.GetUpperBound(0); i++)
  {  k = klasörler[i].Split('\\');    e.Node.Nodes.Add(k[k.GetUpperBound(0)]);    }
 //dosyalary listele
 listele(e.Node.FullPath);
}
catch   {;}
}
private void listView1_SelectedIndexChanged(object sender, System.EventArgs e)
{ try
 { pictureBox1.Image = Image.FromFile(treeView1.SelectedNode.FullPath +
   "\\" + listView1.SelectedItems[0].Text);
 }
 catch
{  ;//Resim dosyası değilse hata vermesin
 }
}
```


Kaynak: http://www.tasarlagelistir.com/

Örnek

```
private void textBox1_TextChanged(object sender, System.EventArgs e)
{ try  {   listele(treeView1.SelectedNode.FullPath);   }
 catch  {;}
}
private void listView1_KeyDown(object sender, System.Windows.Forms.KeyEventArgs e)  {   if (e.KeyCode == Keys.Delete)
   {   int say, i;    DialogResult c;    string dosya;
   say = listView1.SelectedItems.Count;
   if (say > 0) //Seçili dosya varsa
   {    c = MessageBox.Show("Seçili dosyalary silmek istediğinizden emin misiniz", "Sil",MessageBoxButtons.YesNo,MessageBoxIcon.Warning, MessageBoxDefaultButton.Button2);
    if (c == DialogResult.Yes)
    {  for (i = 0 ; i<= say - 1 ; i++)
     {   dosya = treeView1.SelectedNode.FullPath + "\\" + listView1.SelectedItems[i].Text;
      try    {    System.IO.FileInfo d = new System.IO.FileInfo(dosya);
       d.Delete();//Dosyayı sil
```

Örnek

```
catch (Exception hata)
        {    MessageBox.Show(dosya + " silinemedi:\r\n" + hata.Message);    }
      }
      try //Dosya silindiği için listeyi yeniden oluştur
      {
       listele(treeView1.SelectedNode.FullPath);
      }
      catch
      {;}
     }
    }
   }
  }
}
```

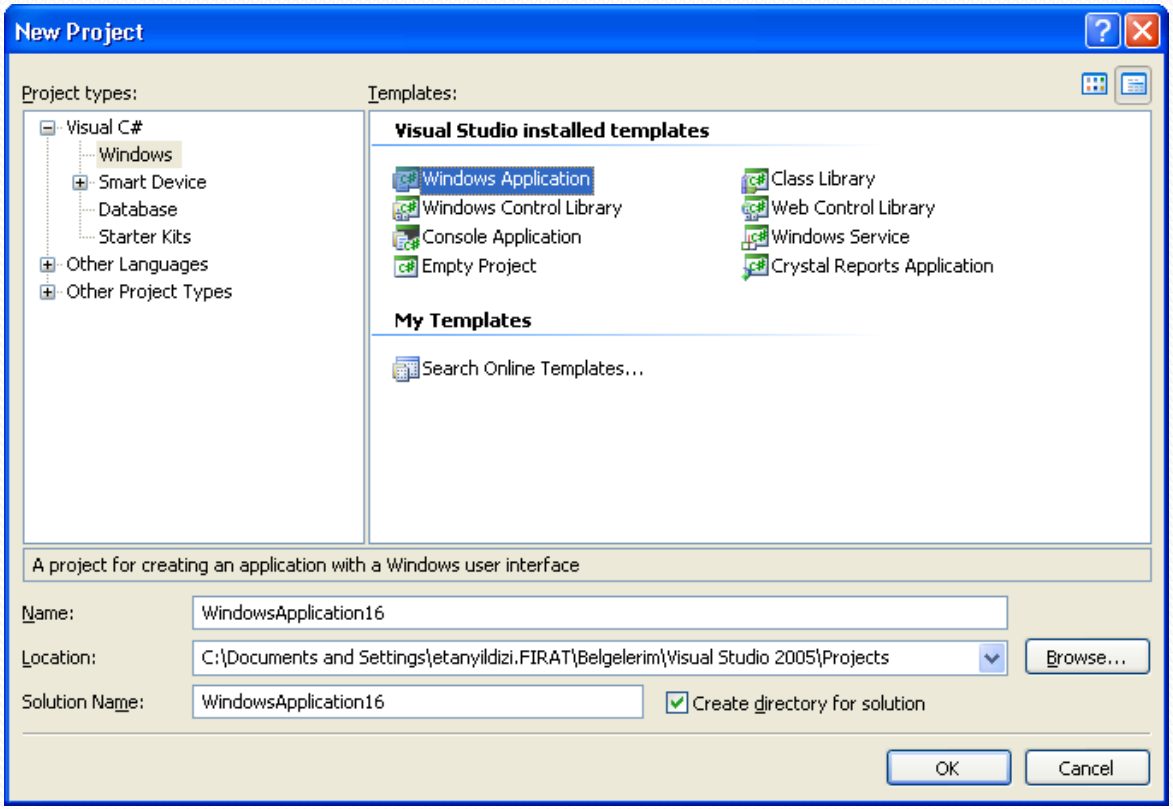
Örnek

```
public double GetTotalHDDSize(string driveName)
{
foreach (DriveInfo drive in DriveInfo.GetDrives())
 { if (drive.IsReady && drive.Name == driveName)
{ return drive.TotalSize / (1024 * 1024 * 1024); } }
return -1;
}
private long GetTotalFreeSpace(string driveName)
{ foreach (DriveInfo drive in DriveInfo.GetDrives())
{ if (drive.IsReady && drive.Name == driveName)
{ return drive.TotalFreeSpace; } }
 return -1; }
```

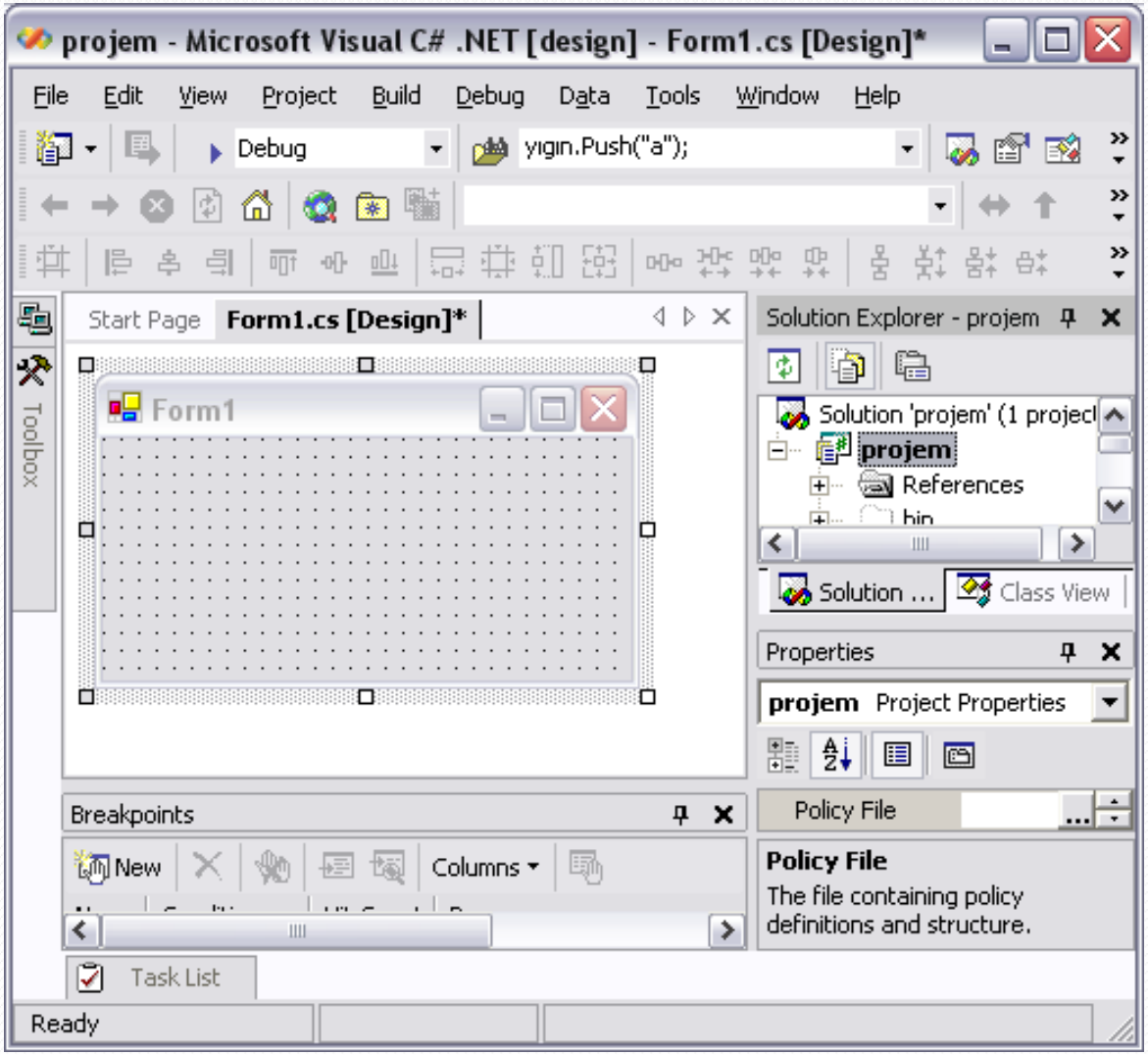
Visual Studio.Net -C#
12. HAFTA
FORM UYGULAMALARI

Proje Başlatma

- ***New Project:*** ***Yeni Proje Hazırlamak***
- ***Open Project:*** ***Mevcut Projeleri Açmak***



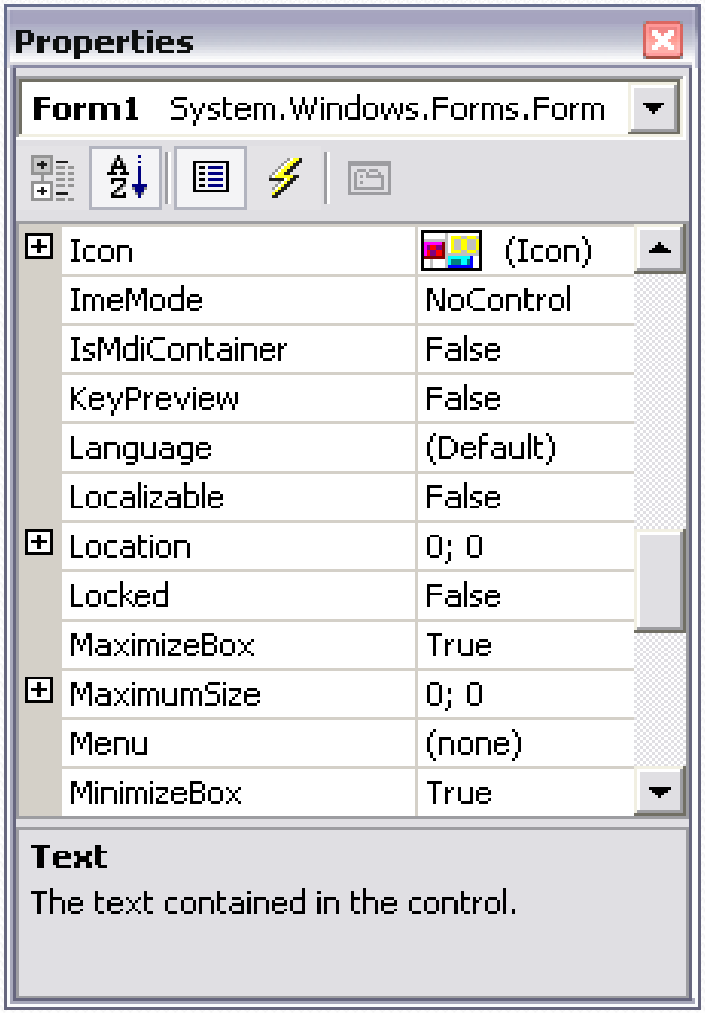
Proje Çalışma Ekranı



Proje Çalışma Ekranı

- **Properties Penceresi**
- Form ve diğer elamanlara ait özelliklerin belirlediği bölümdür.

- Properties penceresi eğer ekranda yoksa F4 fonksiyon tuşu ya da **View/ Properties Window** tıklanarak ekrana yerleştirilir.



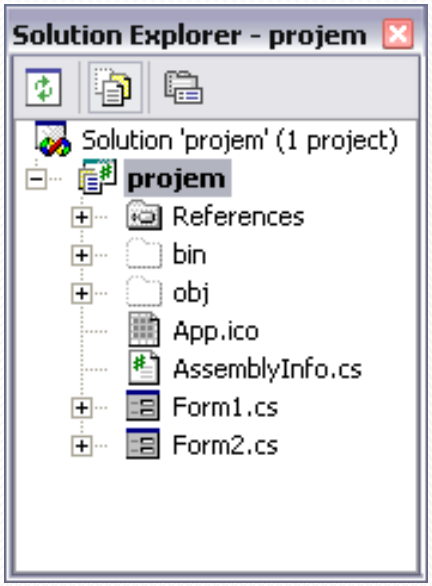
Proje Çalışma Ekranı

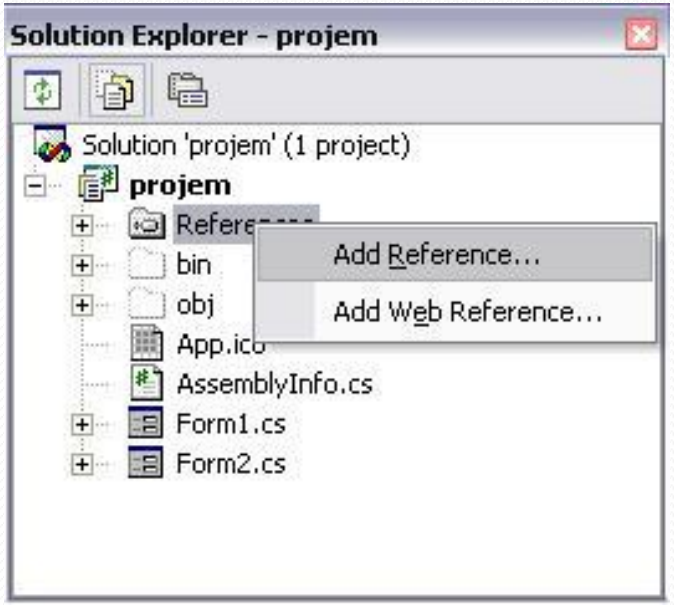
- **Solution Explorer Penceresi**

- Bu pencere proje içerisinde bulunan dosyalara erişim için kullanılır.

- Çalıştığımız projeye **ToolBox**' ta bulunmayan bir referans eklemek için Solution Explorer penceresinde **References** seçeneğine sağ tıklayarak **Add Reference** komutunu seçilmelidir.

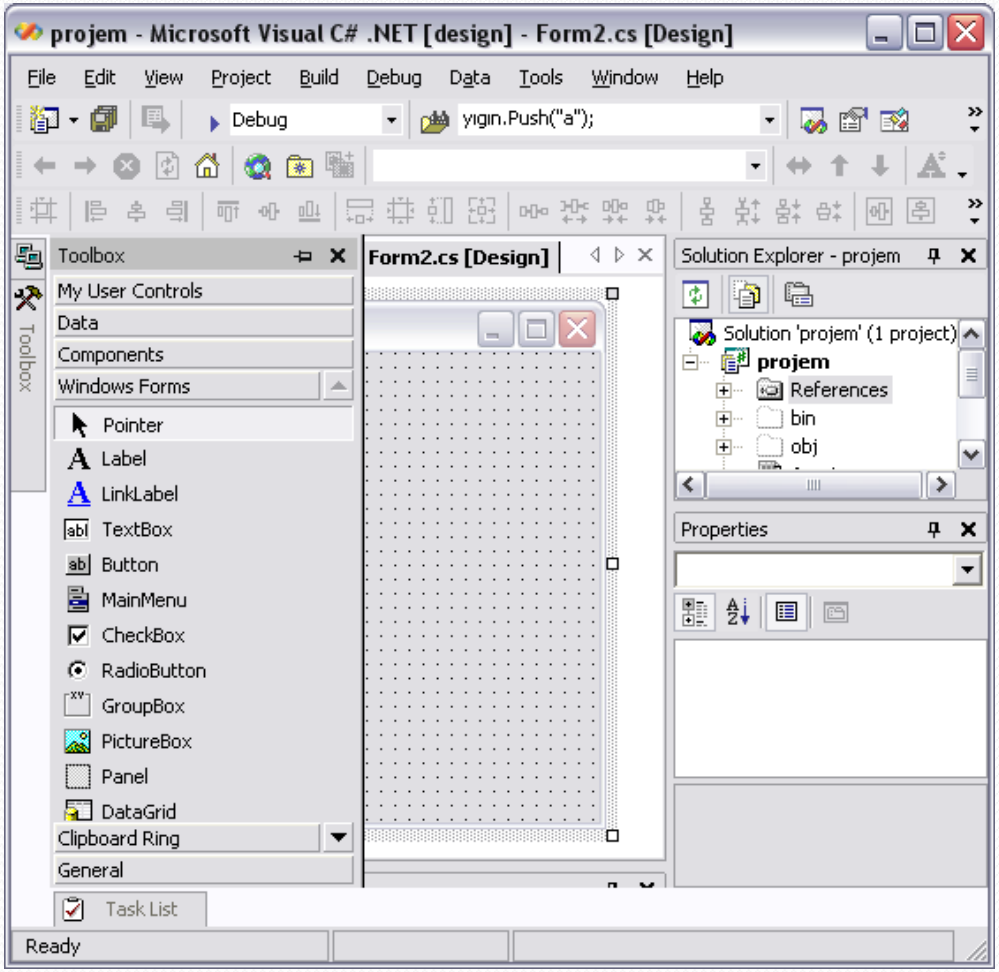
- Solution Explorer penceresini görünür hale getirmek istersek Ctrl+Alt+L kısayol tuşunu ve ya View menüsünden Solution Explorer tıklanarak seçilir.





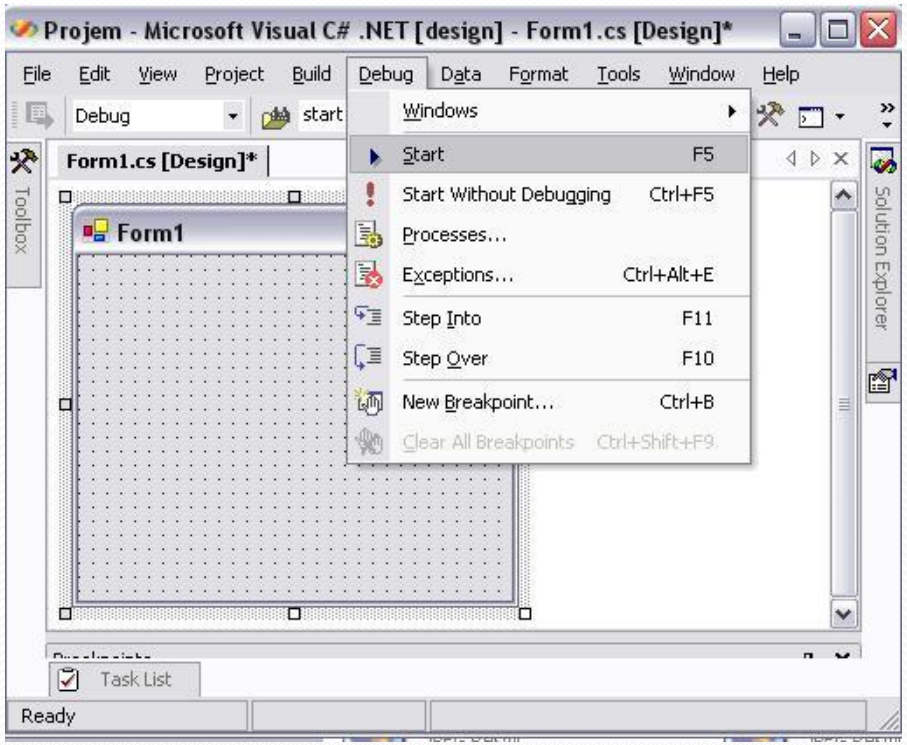
Proje Çalışma Ekranı

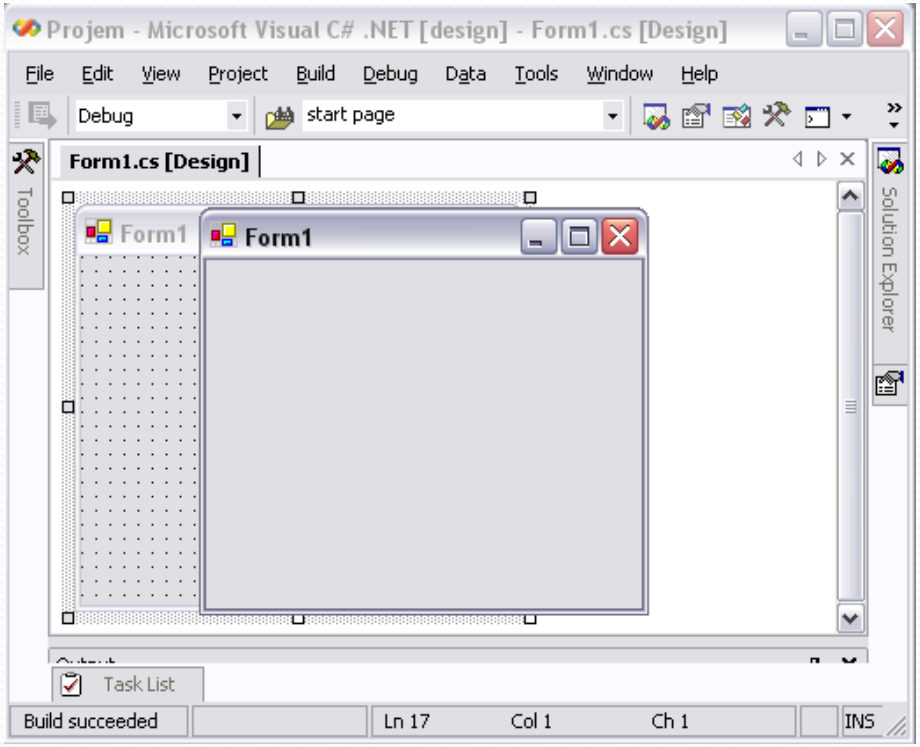
- **ToolBox Penceresi**
- Windows tabanlı uygulamalar geliştirirken sıkça kullanacağımız bir grup kontrol vardır.
- Form kontrolü hariç diğer bütün kontroller **Toolbox** panelinden seçilir.
- Bu kontroller sürüklenip Form üzerine istenilen pozisyona bırakılır.



Proje Çalışma Ekranı

- **Projeleri Çalıştırmak**





FORM UYGULAMALARI

- **Program Kodu Yazmak**

Sil_WindowsFormsApplication - Microsoft Visual Studio (Administrator)

Program.cs | Form1.cs* | Form1.cs [Design]*

Sil_WindowsFormsApplication.Program | Main()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Sil_WindowsFormsApplication
{
    0 references
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        0 references
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Sil_WindowsFormsApplication - Microsoft Visual Studio (Administrator)

Program.cs | Form1.cs* | Form1.cs [Design]*

Sil_WindowsFormsApplication.Form1 | Form1()

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Sil_WindowsFormsApplication
{
    3 references
    public partial class Form1 : Form
    {
        1 reference
        public Form1()
        {
            InitializeComponent();
        }

        1 reference
        private void Form1_Load(object sender, EventArgs e)
        {

        }
    }
}
```

Solution Explorer

Solution 'Sil_WindowsFormsApplicatio

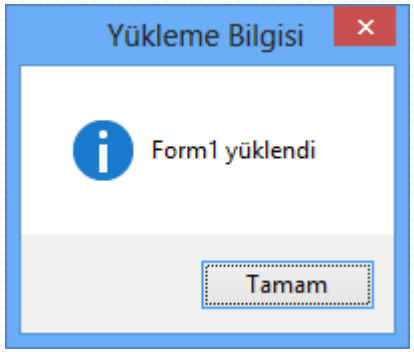
- Sil_WindowsFormsApplication
  - Properties
  - References
  - App.config
  - Form1.cs
    - Form1.Designer.cs
    - Form1.resx
    - Form1
  - Program.cs

FORM UYGULAMALARI

- **MessageBox Sınıfı**
- Proje içerisinde kullanıcıya bilgi vermek veya onay almak için kullanılan sınıftır.

- **MessageBox.Show (Açıklama, Başlık, Onay Butonu, Mesaj Sembolü)**

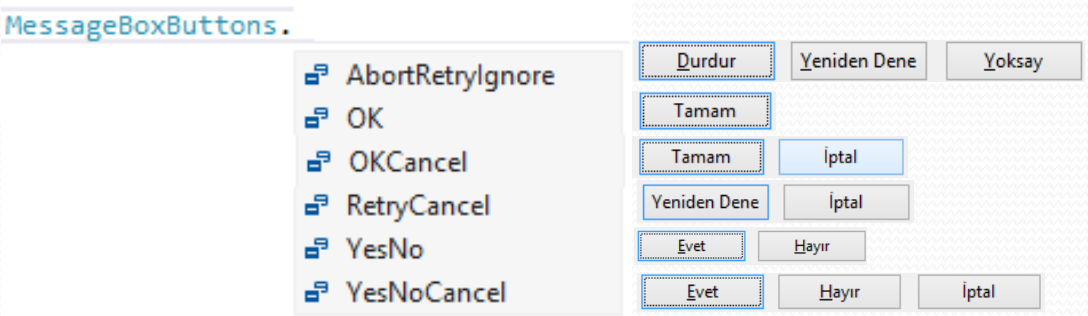
- `MessageBox.Show("Form1 yüklendi", "Yükleme Bilgisi", MessageBoxButtons.OK, MessageBoxIcon.Information);`



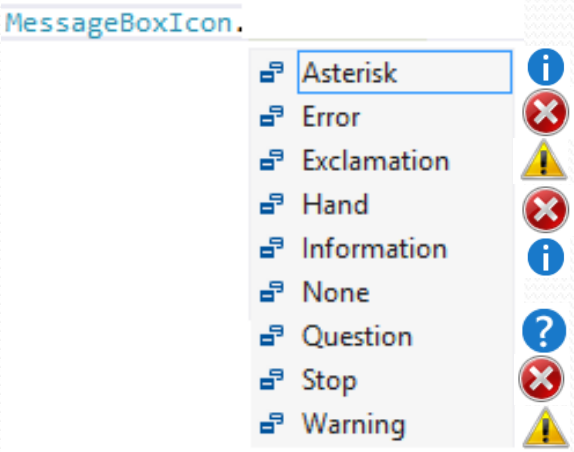
- Console.WriteLine genelde formlarda kullanılmaz.
- Ama eğer kullanmışsanız sonucu View/Output
- penceresini aktif ederek görebilirsiniz.

FORM UYGULAMALARI

- **MessageBoxButtons:** Pencerede gösterilecek buton yapılarının seçimini belirler

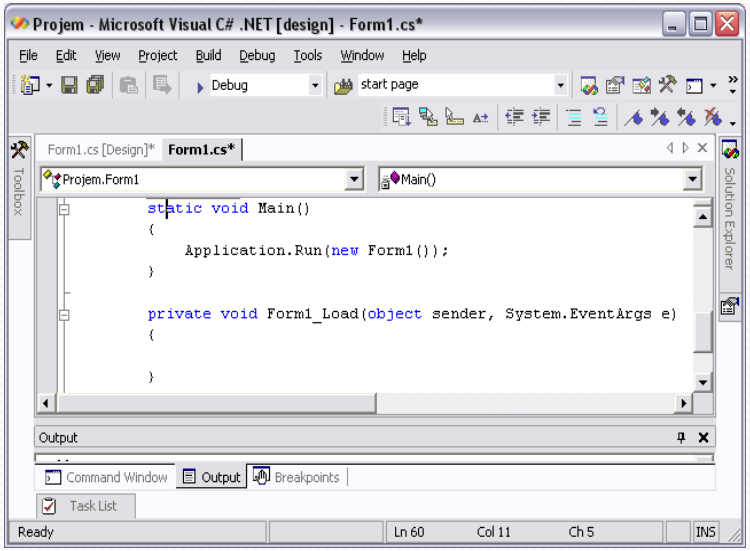


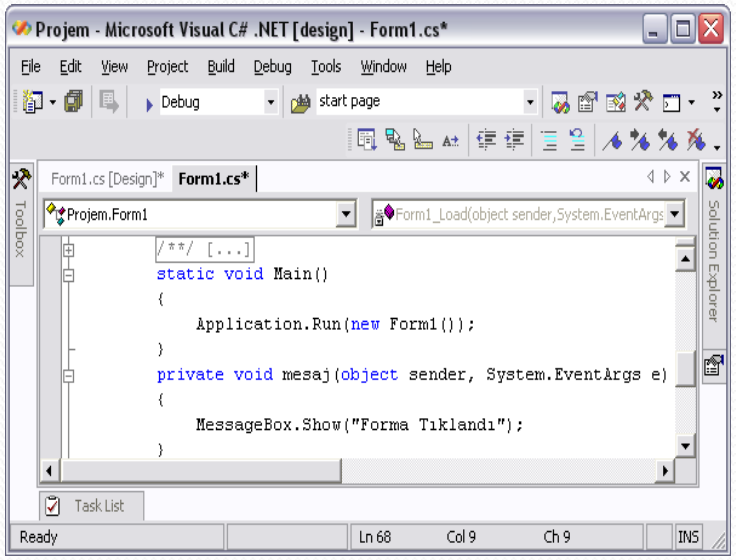
- **MessageBoxIcon:** Pencerede gösterilecek icon türlerinin seçimini belirler

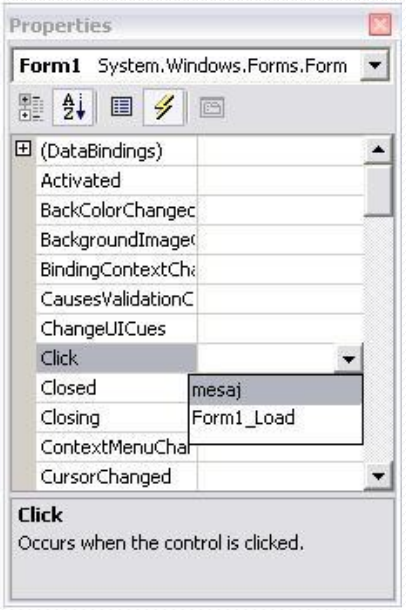


FORM UYGULAMALARI

- **Olay ve Yordam Hazırlamak**



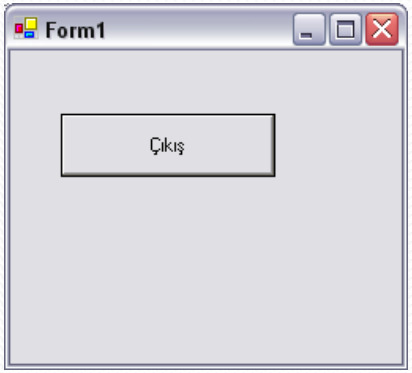


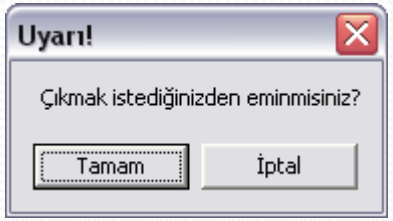


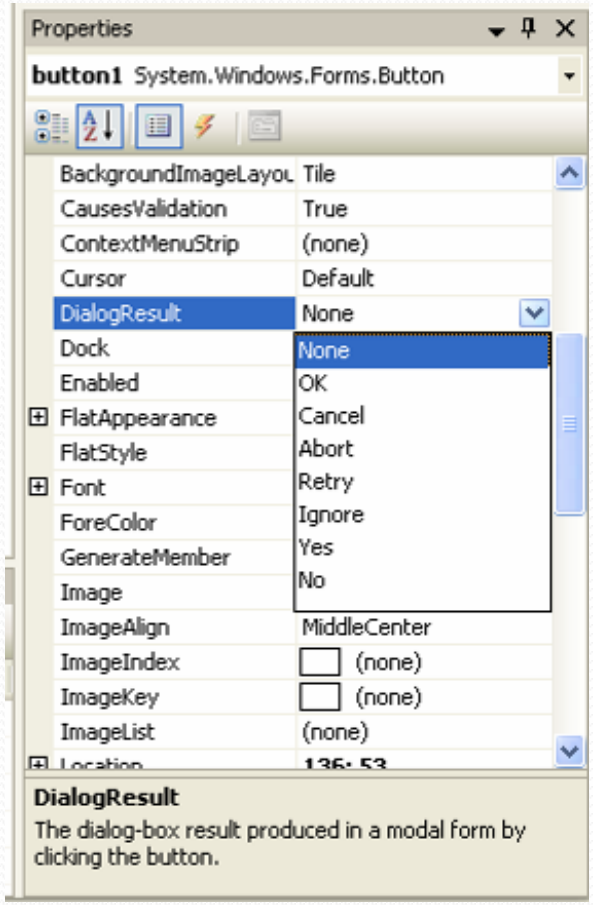
FORM UYGULAMALARI

- **MessageBox sınıfındaki onay butonları**
- **Diyalog Result: Basılan tuşa göre işlem yapma**

```
private void button1_Click(object sender,
System.EventArgs e)
{ DialogResult sonuc;
  sonuc = MessageBox.Show("Çıkmak
  istediğinizden emin misiniz?", "Uyarı!",
  MessageBoxButtons.OKCancel);
  if (sonuc == DialogResult.OK)
   { Form1.ActiveForm.Close(); }
 }
```

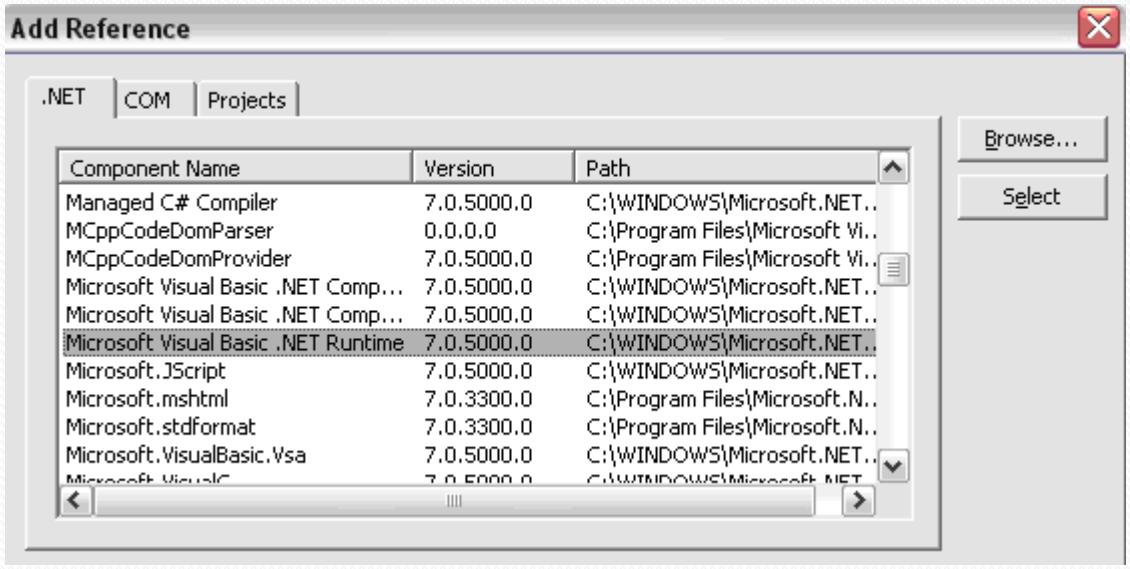






FORM UYGULAMALARI

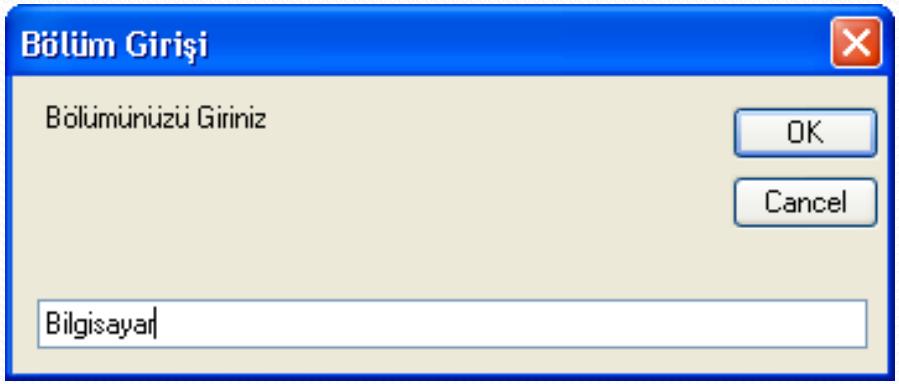
- **InputBox Sınıfı**
- Kullanıcıdan bilgi almak için kullanılan sınıftır. C# içinde direk olarak bulunmaz. Proje içerisine Add Reference ile dahil edilmesi gerekir.



FORM UYGULAMALARI

- **InputBox Sınıfı**

- private void Form1_Load(object sender, System.EventArgs e)
- {
- string bolum= **Microsoft.VisualBasic.Interaction.InputBox** ("Bölümünüzü Giriniz" ,"Bölüm Girişi","",20,20);
- MessageBox.Show(bolum);
- }



FORM UYGULAMALARI

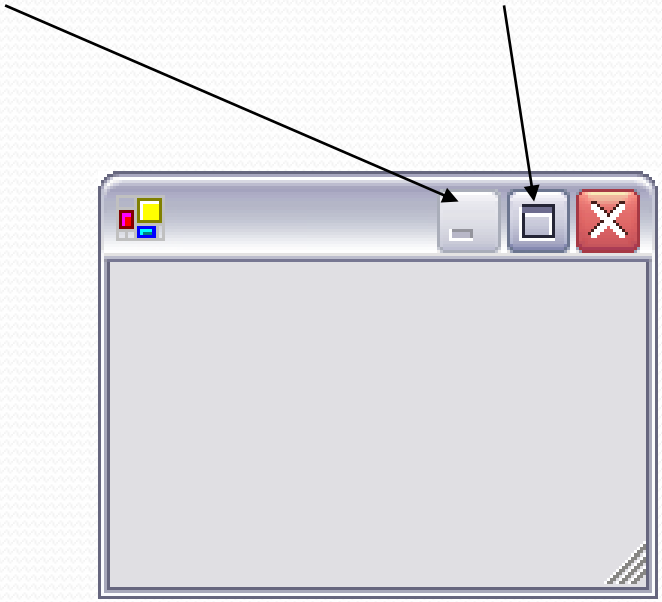
- **Kontrol elamanlarını nesne ile tanımlama**
- Toolbox içerisinden kontrol eklemesi yapmadan sadece o kontrol sınıfına ait nesne ile kontrol elamanları kullanılabilir. Örnek: Proje içerisinde yeni bir form oluşturmak.

```
private void button1_Click(object sender, System.EventArgs e)
{
        Form f = new Form();
        f.Width = Convert.ToInt32(textBox1.Text);
        f.Height = Convert.ToInt32(textBox2.Text);
        f.ShowDialog();
}
```

FORM UYGULAMALARI

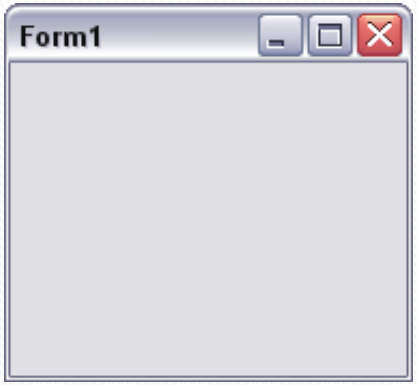
- **Form Özellikleri**
- **MinimizeBox Özelliği:** **Simge Durumuna Küçültme**
- **MaximizeBox Özelliği:** **Ekranı Kaplama**

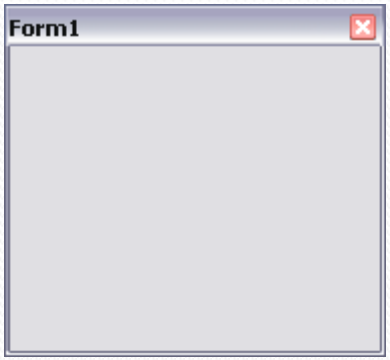
- **MinimizeBox = false, MaximizeBox=true**



FORM UYGULAMALARI

- **Form BorderStyle Özelliği:** Formun sınır özelliklerini belirler
- **Sizable** seçenekleri dışındakilerde ekran boyutu fare ile değiştirilemez.
- Örneğin
- **None** **FixedDialog** **Fixed ToolWindow**



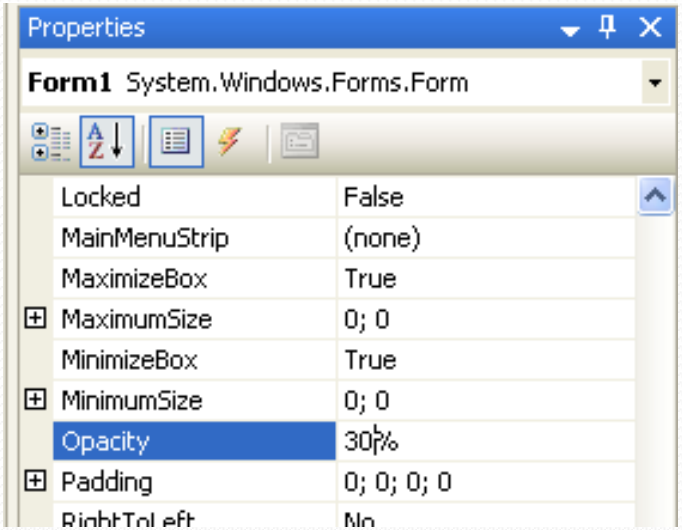


FORM UYGULAMALARI

- **Formun boyutları:** Width (x boyutu) , Height (y boyutu)

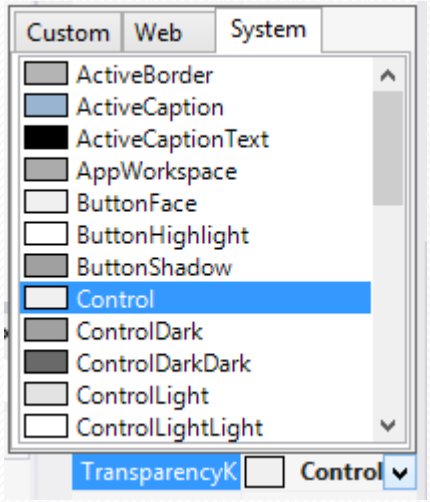
- **Formun Ekrandaki Konumu (Loaction):**
  - **private void Form1_Load (object sender, System.EventArgs e)**
  - **{ this.Location=new Point (150, 250); }**

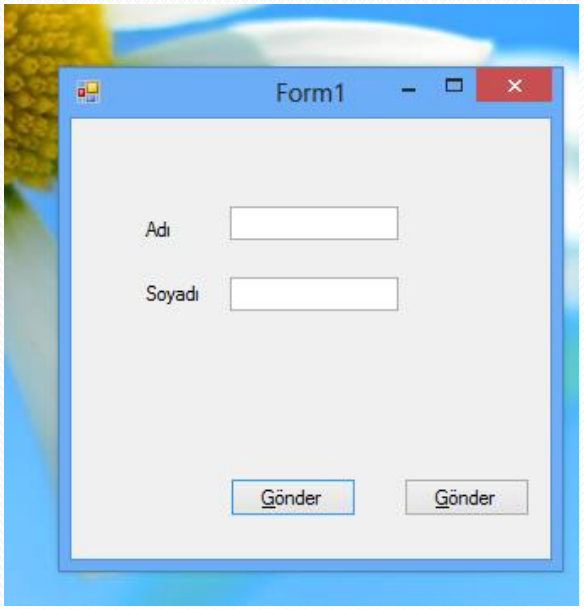
- **BackgroundImage:** Form arka planına resim ekleme
- **Oppacity:** Formun şeffaflığını ayarlama



FORM UYGULAMALARI

- **TransparencyKey:** Form içinde seçilen renk transparan yapılır. Eğer formun arka planında kullanılan renk ile aynı olursa form transparan olur. Formun varsayılan arka plan rengi **(Backcolor) Control** dur. Eğer formun **FormBorderStyle** özelliği **None** olursa formda sadece eklenen elamanlar görülür, Form transparan olur.





FORM UYGULAMALARI

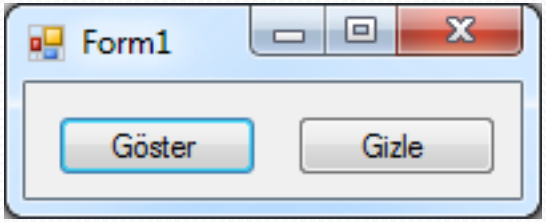
- **Show() ve ShowDialog() Metotları**
- **Show()** metodu birden fazla formu aynı anda ekrana getirmek için, **ShowDialog()** ise sadece çalışmak istediğimiz formu ekrana getirmek için kullanılır. Bu form dışındakilere erişilemez.
- Birinci Form içerisinde;

```
private void button1_Click(object sender, EventArgs e)
{// Project-Add Windows Form ile form eklendikten sonra
   Form2 yeni = new Form2();
   yeni.Show();
 }
```

FORM UYGULAMALARI

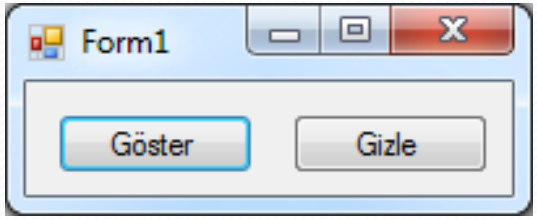
- **Örnek:** Form1 üzerinden Form2 yi çağırma. Eğer aşağıdaki kod satırları olmazsa her show metodunda form2 kapatılmadan tekrar yeni bir nesne ile açılmaya zorlanır buda hatalara neden olabilir.

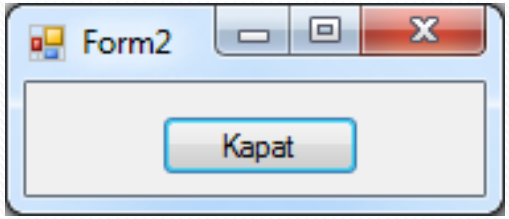
```
public partial class Form1 : Form
{ Form2 f ;
  private void button1_Click(object sender, EventArgs e){
  if (f == null || f.IsDisposed)
  { f = new Form2(); f.Show();}//Form2 kapatılmış ise gösteriyor
  else
  { f.Visible = true;//Hide ile gizlemenin etkisini kaldırıyor
    if (f.Created) //Form oluşturulmuş ise sadece aktif ediyor
        f.Activate();
  }
 }
}
```



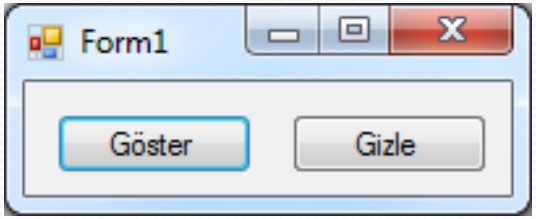
FORM UYGULAMALARI

```
private void button2_Click(object sender, EventArgs e)
{
    if (f.Created)
        f.Hide();
}
```



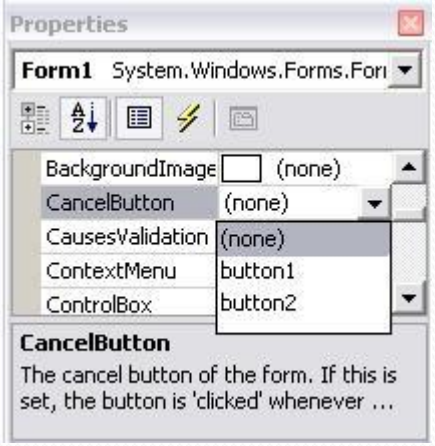



```
//Form2
public partial class Form2 : Form
private void button1_Click(object sender, EventArgs e)
{
 this.Close();    }
  }
```

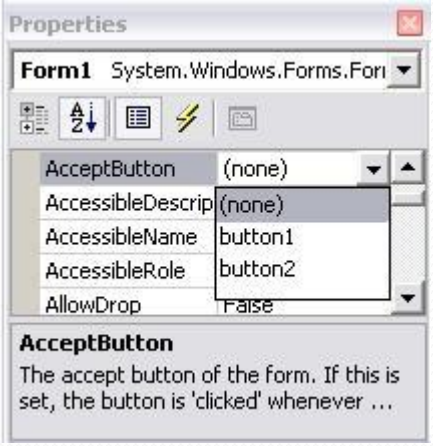


FORM UYGULAMALARI

- **İptal (CancelButton) düğmesi**
- **Esc** ile aynı işleve sahiptir. Forma yerleştirmiş olduğunuz düğmeye benzer işlevi vermek istiyorsanız formun özelliğine söz konusu düğmenin adını aktarmalısınız.



- **Tamam (AcceptButton) düğmesi**
- **Enter** ile tuşu aynı işleve sahiptir. Herhangi bir anda Enter tuşuna basıldığında formdaki düğmelerden birisi tıklanmış gibi işlem yapılmasını istiyorsanız o düğmenin adını formun AcceptButton özelliğine aktarmalısınız.



FORM UYGULAMALARI

- **Form Özellikleri:**

Özellik	Açıklama
AcceptButton	Form üzerinde Enter tuşuna basıldığı zaman “tıklanacak” Button kontrolü
CancelButton	Form üzerinde Esc tuşuna basıldığı zaman “tıklanacak” Button kontrolü
Opacity	Formun şeffaflık oranı (0 -1 arası)
MaximizeBox	Ekranı Kapla düğmesinin görünürlüğü
MinimizeBox	Simge Durumunda Küçült düğmesinin görünürlüğü
ControlBox	Close, Maximize ve Minimize düğmelerinin tümünün görünürlüğü
StartPosition	Form açıldığı zaman, ekran üzerindeki konumu
TopMost	Formun tüm pencerelerin üzerinde gözükmesi
FormBorderStyle	Formun kenar stili
MaximumSize	Formun alabileceği maksimum büyüklük
MinimumSize	Formun alabileceği minimum büyüklük

FORM UYGULAMALARI

- **Form Olayları:**

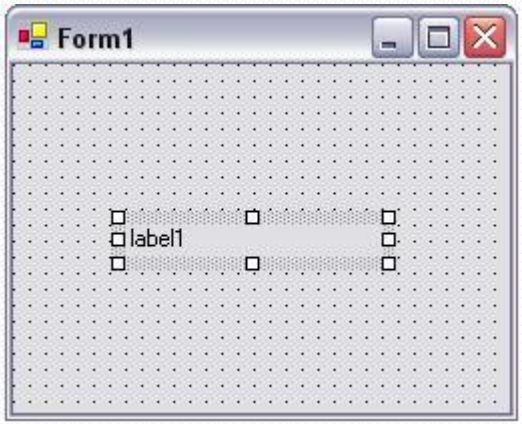
Olay	Açıklama
Click	Form üzerine tıklandığı zaman gerçekleşir
Closing	Form kapanmadan hemen önce gerçekleşir
Closed	Form kapandıktan sonra gerçekleşir
Load	Form yüklenirken gerçekleşir
KeyDown	Form üzerindeyken bir tuşun basılması ile gerçekleşir
KeyUp	Basılan tuşun kaldırılması ile gerçekleşir

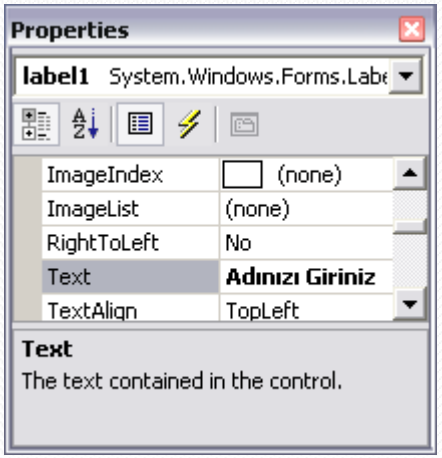
- **Form Metotları:**

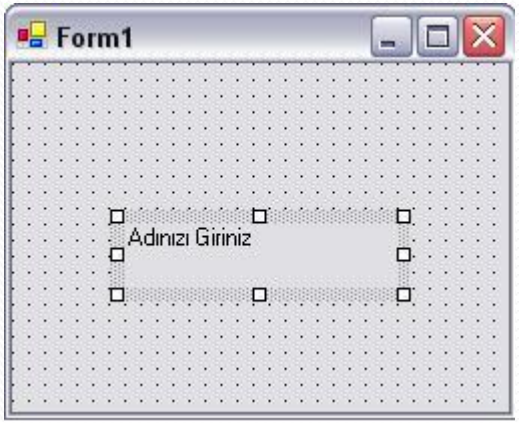
Metot	Açıklama
Hide	Formu Visible özelliğini False yaparak, gizler
Close	Formu kapatır. Eğer form başlangıç formuysa uygulama sonlanır
Show	Formu gösterir. Hide ile gizlenmişse, Visible özelliği True yapılır.
ShowDialog	Formu diyalog kutusu olarak gösterir.

FORM KONTROLLERİ- LABEL KONTROLÜ

- **Label** kontrolü Form üzerinde kullanıcıya bilgi vermek amaçlı kullanılan etikettir.







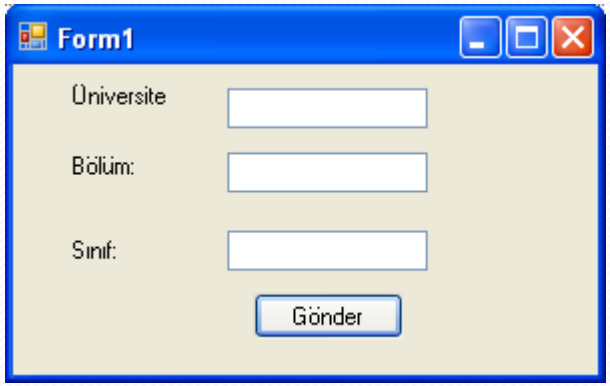
LABEL KONTROLÜ

- **Label Özellikleri :**

Özellik	Açıklama
TextAlign	Yazının, etiket üzerindeki pozisyonu belirler.
BorderStyle	Kontrolün kenar stilidir. **FixedSingle** değeri, kontrolün kenar çizgilerini gösterir. **Fixed3D** değeri, kenarların üç boyutlu olmasını sağlar
Image	Etiket üzerinde görüntülenmek istenen resmi tutar
ImageAlign	Etiket üzerindeki resmin nerede duracağını belirler
RightToLeft	Etiket üzerindeki yazının yönünü belirler. Eğer **Yes** değerini alırsa, yazılar sağdan sola gösterilir

TextBox KONTROLÜ

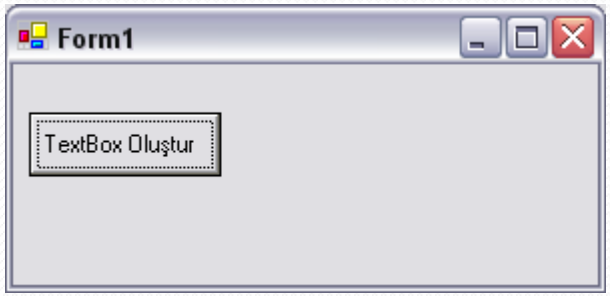
- Metin kutuları, kullanıcıdan bilgi almak için kullanılır.
- **private void button1_Click(object sender, EventArgs e)**
- **{**
- **MessageBox.Show (textBox1.Text + "\n" + textBox2.Text + "\n" + textBox3.Text);**
- **}**

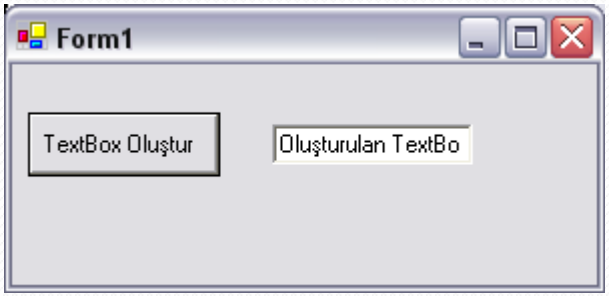


TextBox KONTROLÜ

Çalışma Anında Forma Kontrol Yerleştirmek

```
private void button1_Click(object sender, System.EventArgs e)
{
    TextBox metin = new TextBox();
    // metin kutusu nesnesi
    this.Controls.Add(metin);
    metin.Text = "Oluşturulan TextBox";
    metin.Top = 30;
    metin.Left = 130;
 }
```





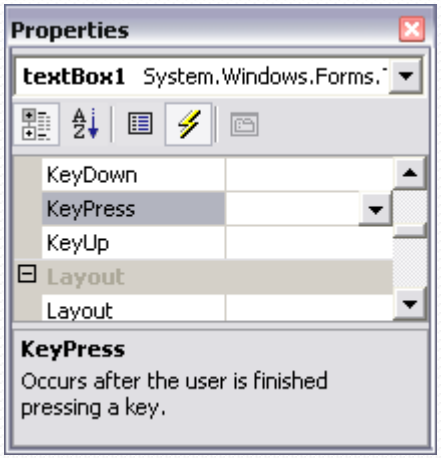
TextBox KONTROLÜ

- **KeyDown:** Bir tuşa basıldığı durumda meydana gelen olay
- **KeyUp:** Tuştan elin çekildiği durumda meydana gelen olay
- **KeyPress:** Bir tuşa basılma ve çekme anı arasında meydana gelen olaydır.
- Formun üzerine yerleştirilen **nesnelerin KeyDown** metodu yerine **formun KeyDown** metodunun işletilmesini istiyorsanız formun **KeyPreview** özelliğini **true** yapmalısınız. Bu özellik true iken öncelik formun KeyDown olayını temsil eden metoda verilir ve aktif nesnenin KeyDown metodu, formun KeyDown metodunun çalışması sona erdikten sonra işletilir. Bu durum KeyUp ve KeyPress içinde geçerlidir.

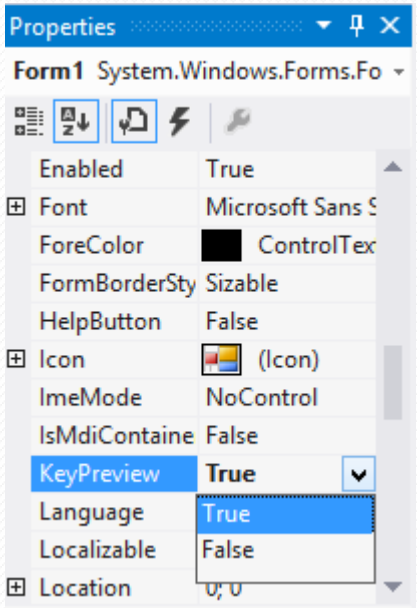
TextBox KONTROLÜ

- **Örnek:**

```
private void textBox1_KeyPress(object sender,
System.Windows.Forms.KeyPressEventArgs e)
{
     if (e.KeyChar == 13) // Enter Tuşu
        MessageBox.Show(textBox1.Text);
}
// enter tuşuna basıldığında ortaya çıkan sonuç
```



TextBox KONTROLÜ

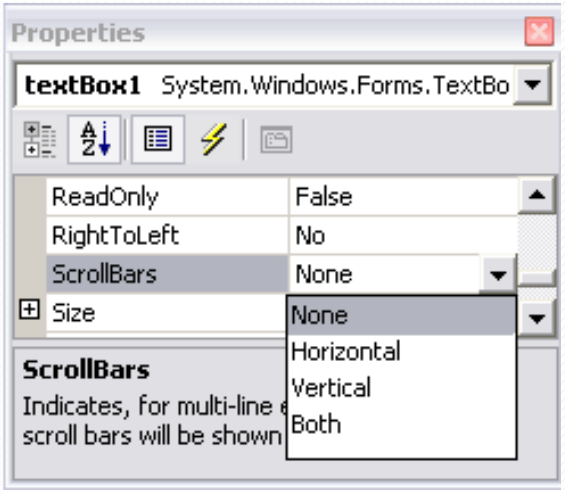


- **Örnek:** Alt tuşu ile formun kapatılması

```
private void Form1_KeyDown(object sender,
System.Windows.Forms.KeyEventArgs e)
 {
     if (e.Modifiers == Keys.Alt)
     //if(e.KeyCode==Keys.Control)
     //if (e.Control && e.Alt)                olabilir.
     // if(e.KeyCode==Keys.F1 && (e.Alt||e.Control||e.Shift))
      this.Close();
 }
```

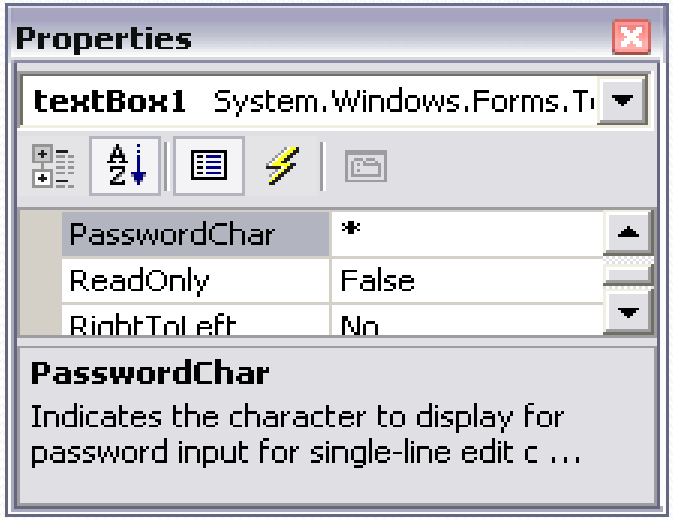
TextBox KONTROLÜ

- **MultiLine Özelliği:** Metin kutusuna bilginin çoklu satır olarak girilmesini sağlar.
- **ScrollBars Özelliği:** Eğer satır sayısı metin kutusunun boyutundan fazla ise kaydırma çubuklarına gerek duyulabilir. Kaydırma çubukları eklemek için ScrollBars özelliği kullanılır, bu özellik dört değişik değer alır.



TextBox KONTROLÜ

- **PaswordChar Özelliği:** Metin kutusuna girilen bilginin belirlenen karakter ile gizlenmesini sağlar.



TextBox KONTROLÜ

- **CharacterCasing Özelliği:** Metin kutusuna girilen karakterlerin büyük ya da küçük harfe çevrilmesini sağlar. **Upper** ve **Lower** değerleri alır.

```
private void Form1_Load(object sender, System.EventArgs e)
{
    textBox1.CharacterCasing = CharacterCasing.Upper;
}
```

TextBox KONTROLÜ

- **TextBox Özellikleri**

Özellik	Açıklama
MultiLine	Metin kutusuna birden fazla satırda değer girilebilmesini sağlar. **False** durumunda ise, metin kutusunun yüksekliği değiştirilemez
ScrollBars	Metin kutusunda kaydırma çubuklarının görünmesi. Varsayılan olarak kaydırma çubuğu görüntülenmez, ancak Horizontal, Vertical kaydırma çubukları ya da ikisi birden gösterilebilir.
PasswordChar	Metin kutusuna parola girilecekse, girilen karakterlerin hangi karakter olarak görüneceğini belirler
WordWrap	Metin kutusuna girilen değerlerin, satır sonlandığında bir alt satıra geçilmesini belirtir. Eğer MultiLine özelliği False ise, alt satırlar tanımlı olmayacağı için bu özelliğin bir etkisi görülmez.
MaxLength	Metin kutusunun alabileceği maksimum karakter sayısını belirtir.
ReadOnly	Metin kutusunun yazmaya karşı korumalı olduğunu belirtir.
CharacterCasing	Metin kutusuna karakterler girilirken büyük veya küçük harfe çevrilmesini sağlar. Upper değeri büyük, Lower değeri küçük harfe çevrimi sağlar.

TextBox KONTROLÜ

- **TextBox Olayları**

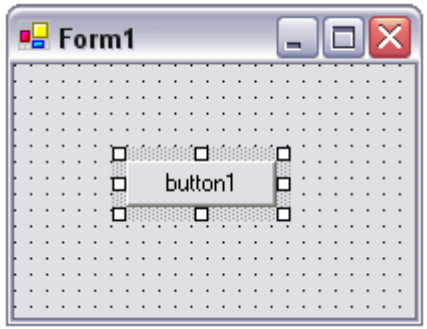
Olay	Açıklama
TextChanged	Metin kutusundaki yazı değiştiği zaman gerçekleşir.
KeyPress, KeyUp, KeyDown	Bir tuşa basılma durumunda gerçekleşir

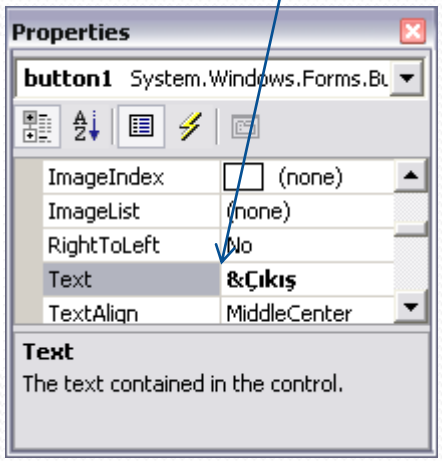
- **TextBox Metotları**

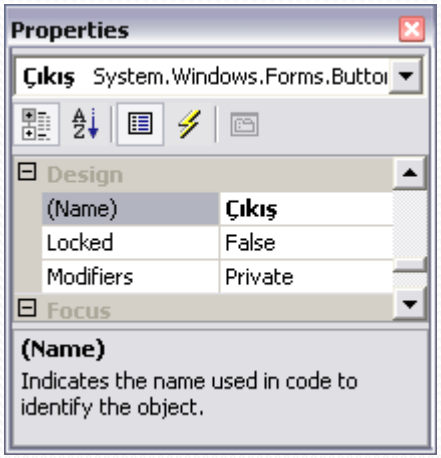
Metot	Açıklama
Cut	Seçilen karakterleri siler ancak hafızada tutar.
Copy	Seçilen karakterleri kopyalar
Paste	Hafızaya alınan karakterleri metin kutusuna yapıştırır
Clear	Metin kutundaki yazıları temizler
SelectAll	Metin kutusundaki tüm yazıyı seçer

Buton KONTROLÜ

- Windows uygulamalarında, form üzerinde komut düğmeleri olarak kullanılır. Kısayol tuşları ile buton kontrolüne erişmek için & işareti kullanılır. (Alt+ Ç tuşuna basarak butona tıklanma sağlanır)



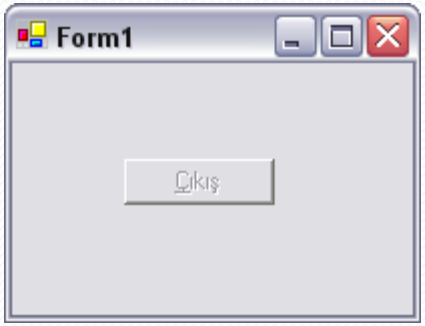




Buton KONTROLÜ

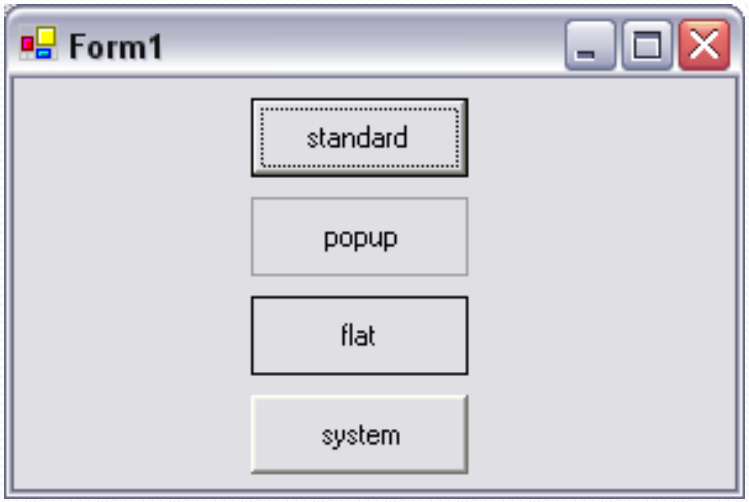
- **Enabled ve Visible Özellikleri:** Butonun aktif veya görünür olmasını kontrol eder.

```
private void Çıkış_Click(object sender, System.EventArgs e)
{ this.Close(); }
```



Buton KONTROLÜ

- **TabIndex ve TabStop Özellikleri**
- Tab tuşuna basıldığında nesnelerin hangi sırada dolaşılacağı belirlenir. TabStop false olursa o nesne atlanır
- **FlatStyle Özelliği**
- Dört farklı değer alabilir. Bunlar **Standart, Popup, Flat, System**' dir. Görünüm şekilleri aşağıdaki gibidir.



Buton KONTROLÜ

- **PerformClick() metodu:**
- Butonların çalışması için üzerine tıklamak gerekir. Ancak PerformClick() metodu ile düğme tıklanmış gibi bir etki sağlanır.
  - **private void Form1_Load(object sender, System.EventArgs e)**
  - **{ button1.PerformClick(); }**
- **Image Özelliği:** Düğmelerin üzerine resim yerleştirmek için kullanılır.

Buton KONTROLÜ

- **Button Özellikleri**

Özellik	Açıklama
DialogResult	Ait olduğu form ShowDialog metodu ile çağrıldığı zaman, dönüş değerini belirler
FlatStyle	Düğmeye basıldığında ve düğmenin üzerine gelindiğinde görünen formatı belirler

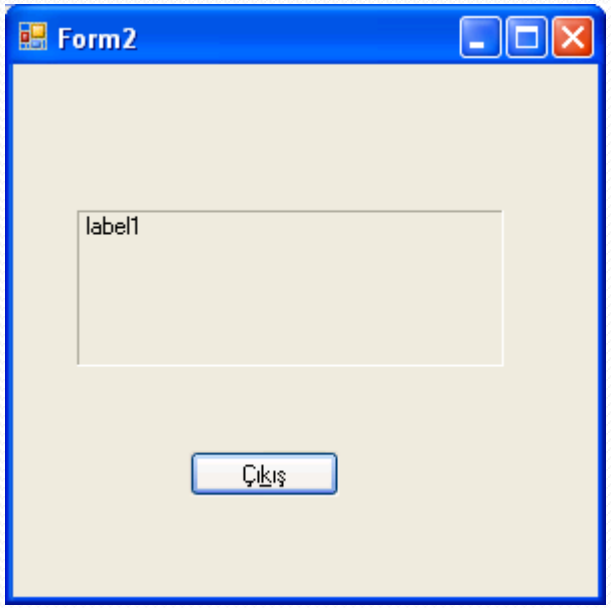
- **Button Olayları**

Olay	Açıklama
Click	Düğme üzerine tıklandığı zaman gerçekleşir

Örnek: Oluşturulan formlar arası bilgi gönderme

- **İlk Form**

Bilgi Gönderilecek Form



Örnek: Oluşturulan formlar arası bilgi gönderme

- **Formlar arası değer göndermenin farklı yolları vardır. Burada sadece bunlardan bir kaçı size verilecektir.**
- **1- Yapıcı metot kullanılarak**
- **Örnek 1:** İlk Formun kod kısmında Form2 show metodu ile gösterileceği zaman onun yapılandırıcısına Form1 de girilen değerleri parametre olarak yazıyoruz.

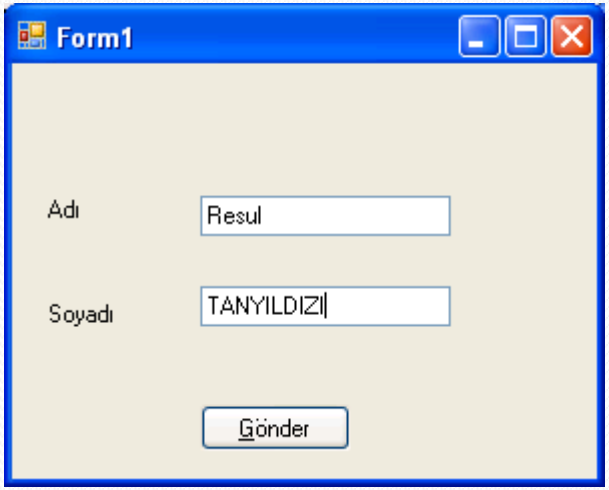
```
private void button1_Click(object sender, EventArgs e)
{
        string a, b;
        a = textBox1.Text;
        b = textBox2.Text;

        Form2 f = new Form2 (a, b);
        f.Show();
}
```

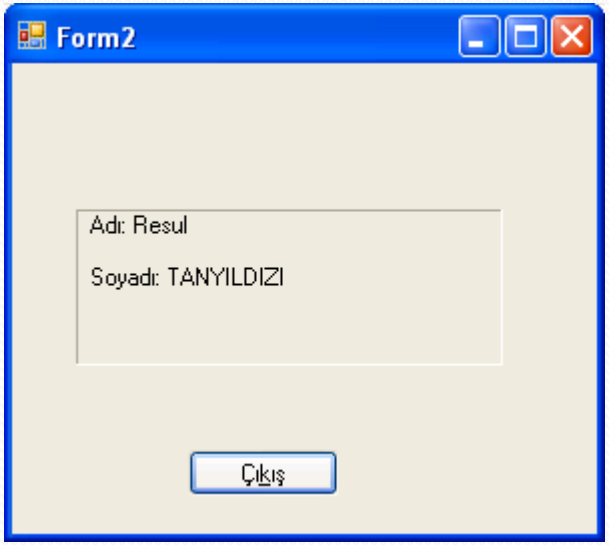
Örnek: Oluşturulan formlar arası bilgi gönderme

- **Form2 nin yapılandırıcısında alacağımız değerler için parametreleri ekliyoruz. Normal durumda burası boştur.**

```
string aa,bb;
//object yazmak bazen daha uygun olur.
public Form2(string a, string b)
{  aa = a;    bb = b;
   InitializeComponent();
}
private void Form2_Load(object sender, EventArgs e)
{
  label1.Text = " Adı: "+aa + "\n \n" + " Soyadı: "+ bb;
}
private void button1_Click(object sender, EventArgs e)
{        this.Close();        }
```

Örnek: Oluşturulan formlar arası bilgi gönderme





Örnek: Oluşturulan formlar arası bilgi gönderme

- **Örnek 2:**
- **Bu örnekte Form2 den ana forma değer gönderilmektedir. Form1 nesnesi yapıcı metotla From2 ye tanıtılarak Form1 deki public olan elemanlara erişim yapılması sağlanır. Form1 içindeki textbox, label gibi elamanlara erişmek için formun designer bölümünden bunların erişimi public yapılmalıdır.**

Server Explorer Toolbox

Form1.Designer.cs* Form2.Designer.cs Form2.cs* Fo

WindowsFormsApplication1.Form1

```
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed re
        protected override void Dispose(bool disposing
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        Windows Form Designer generated code

        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label2;
        public System.Windows.Forms.TextBox textBox1;
        public System.Windows.Forms.TextBox textBox2;
    }
}
```

Örnek: Oluşturulan formlar arası bilgi gönderme

- **Örnek 2: Form2 yapıcısına Form1 nesnesi this ile gönderiliyor.**

```
public void button1_Click(object sender, EventArgs e)
 {
    Form2 f=new Form2(this);
    f.Show();
 }
```

Örnek: Oluşturulan formlar arası bilgi gönderme

- **Örnek 2:**

```
  Form1 frm;
  public Form2(Form1 f1)
  {
   InitializeComponent();
   this.frm = f1;
  }

  public void button1_Click(object sender, EventArgs e)
  {
    frm.textBox1.Text = "deneme";
    this.Close();
  }
```

Örnek: Oluşturulan formlar arası bilgi gönderme

- **2- Static Değer Taşıma ile değer gönderme**

```
public partial class Form1 : Form
    {
        public static string  bolum;
        private void button2_Click(object sender, EventArgs e)
        {
            Form2 frm2 = new Form2();
            frm2.Show();
        }

        private void Form1_Activated(object sender, EventArgs e)
        {
            textBox1.Text = bolum;
        }
}
```

Örnek: Oluşturulan formlar arası bilgi gönderme

- **2- Static Değer Taşıma ile değer gönderme**

```
public partial class Form2 : Form
{
 private void button1_Click(object sender, EventArgs e)
 {
    Form1.bolum =  "deneme";
    this.Close();
 }
}
```

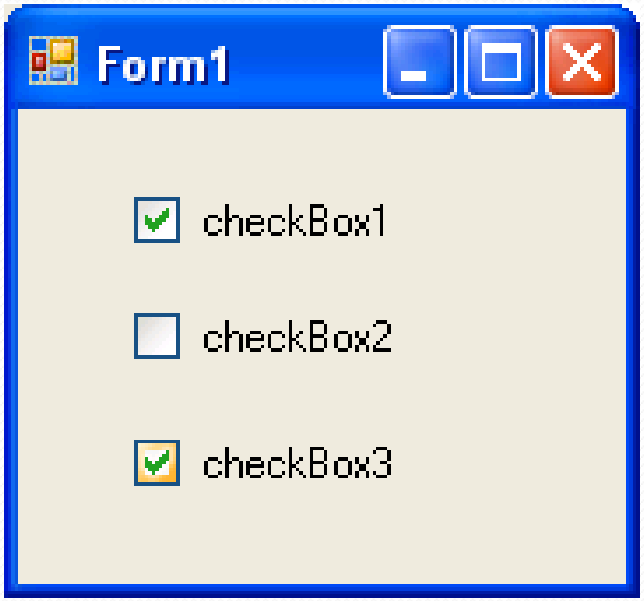
Örnek: Oluşturulan formlar arası bilgi gönderme

- 3- Uygulama (Application) nesnesi ile forma değer gönderme

```
public partial class Form2 : Form
{
 private void button1_Click(object sender, EventArgs e)
 {
   Form1 frm1 = (Form1)Application.OpenForms["Form1"];
   frm1.textBox1.Text="deneme";
 }
}
```

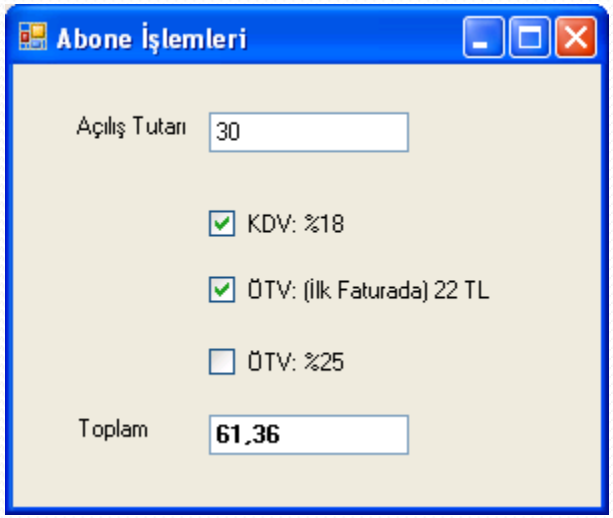
CheckBox KONTROLÜ

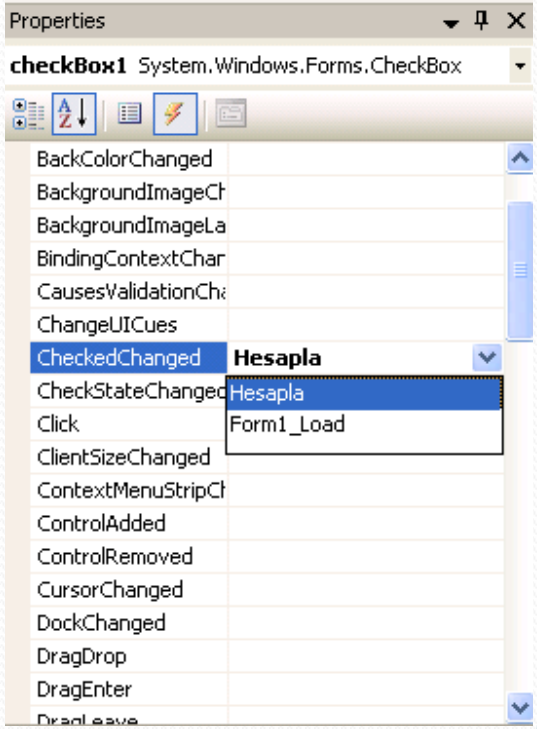
- Kontrol Kutusu, kullanıcıya birden çok seçeneği seçme imkanı sağlar. Checked özelliği kontrol kutusunun seçilip seçilmediğini kontrol eder. Seçili ise , true değilse false değerini alır.



Örnek:

- Bir GSM şebekesinden faturalı hat açılışında toplam tutar hesaplanması.
- Öncelikle CheckBox özelliğinden kontrol ifadelerinin her birinin **CheckedChange** olayına Hesapla diye bir metot tanımlayınız.





Örnek :

```
using System;
namespace uygulama {
 public partial class Form1 : Form     {
  public Form1()      { InitializeComponent();  }
  private void Hesapla(object sender, EventArgs e)
   {  /* Form üzerindeki tüm seçme kutularının durumu değiştiği
     zaman, toplam fiyat tekrar hesaplanır*/
     double toplam = Convert.ToDouble(textBox1.Text);
     // İlk faturada 22 YTL açılış bedeli eklenir
     if (checkBox2.Checked) { toplam += 22; }   // KDV eklenir
     if (checkBox1.Checked) { toplam *= 1.18;}
     // Özel İletişim vergisi eklenir
     if (checkBox3.Checked) { toplam *= 1.25;}
     textBox2.Text = Convert.ToString(toplam);
   }
 }
}
```

CheckBox KONTROLÜ

- **CheckBox Özellikleri**

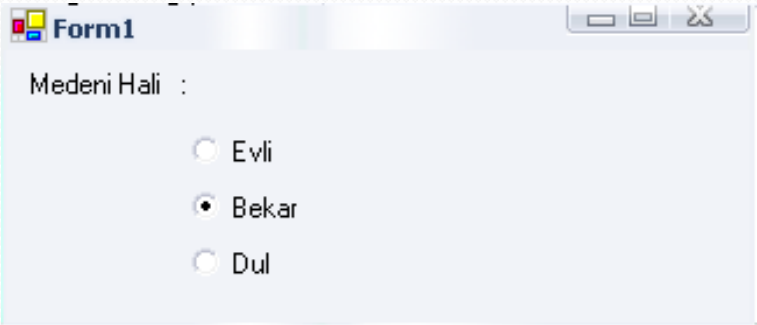
Özellik	Açıklama
Checked	Kontrolün seçili olup olmadığını belirler
CheckAlign	Seçme kutusunun ve üzerinde yazan metnin birbirlerine göre konumlarını belirler.
Appearance	Kontrolün seçme kutusu ya da düğme şeklinde olmasını belirler
ThreeState	Seçili olup olmaması dışında, Intermediate durum da eklenir. Eğer kontrol Intermediate durumundaysa Checked özelliği True olur.
AutoChecked	Kontrole basıldığı zaman seçili duruma geçileceğini belirtir. Eğer bu özellik False ise, kontrolün durumunu değiştirmek için, Click olayında, Checked özelliğini güncellemek gerekir

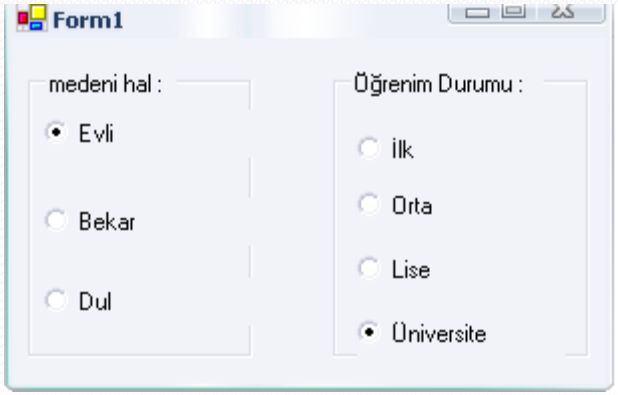
- **CheckBox Olayları**

Olay	Açıklama
CheckChanged	Seçme kutusunun durumu değiştiği zaman gerçekleşir.

Radiobutton KONTROLÜ

- RadioButton kontrolü CheckBox'tan farklı olarak birkaç seçenekten sadece birini seçme imkanı veren bir onay kontrolüdür.
- Bu kontrolün tek başına kullanılması anlamsızdır. Bir kaç seçenekten birini seçme imkanı veren bir kontrol olduğu için en az iki tane birlikte verilmelidir.
- Formda düğmelerden biri seçildiğinde seçilmiş olan kendiliğinden kalkacaktır.
- **GroupBox** kontrolleri sayesinde farklı seçim butonları yapabilirsiniz.





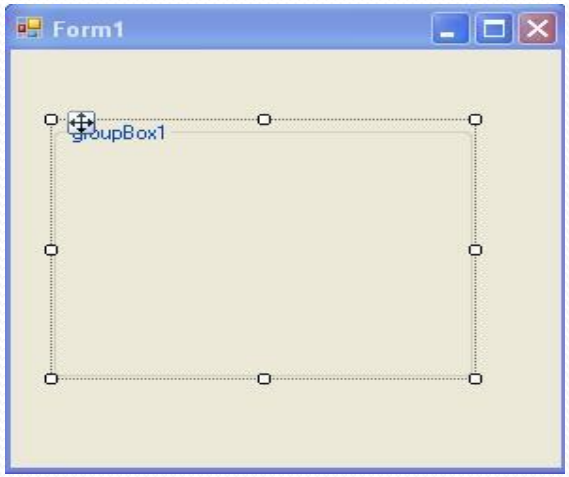
Radiobutton KONTROLÜ

- Bu kontrolün özellikleri CheckBox'a çok benzemektedir. CheckBox'ta olduğu gibi;

- **Appearance** özelliği ile komut düğmesi görünümüne sahip RadioButton düğmeleri oluşturulabilir,
- **Text özelliği** ile içindeki metni yazabilir,
- **TextAlign** özelliği ile içindeki metnin yerleşmesini belirleyebilir,
- **Image** özellikleri ile içinde resim gösterebilir,
- **CheckAlign** özelliği ile seçenek düğmesini sola, sağa alınabilir.

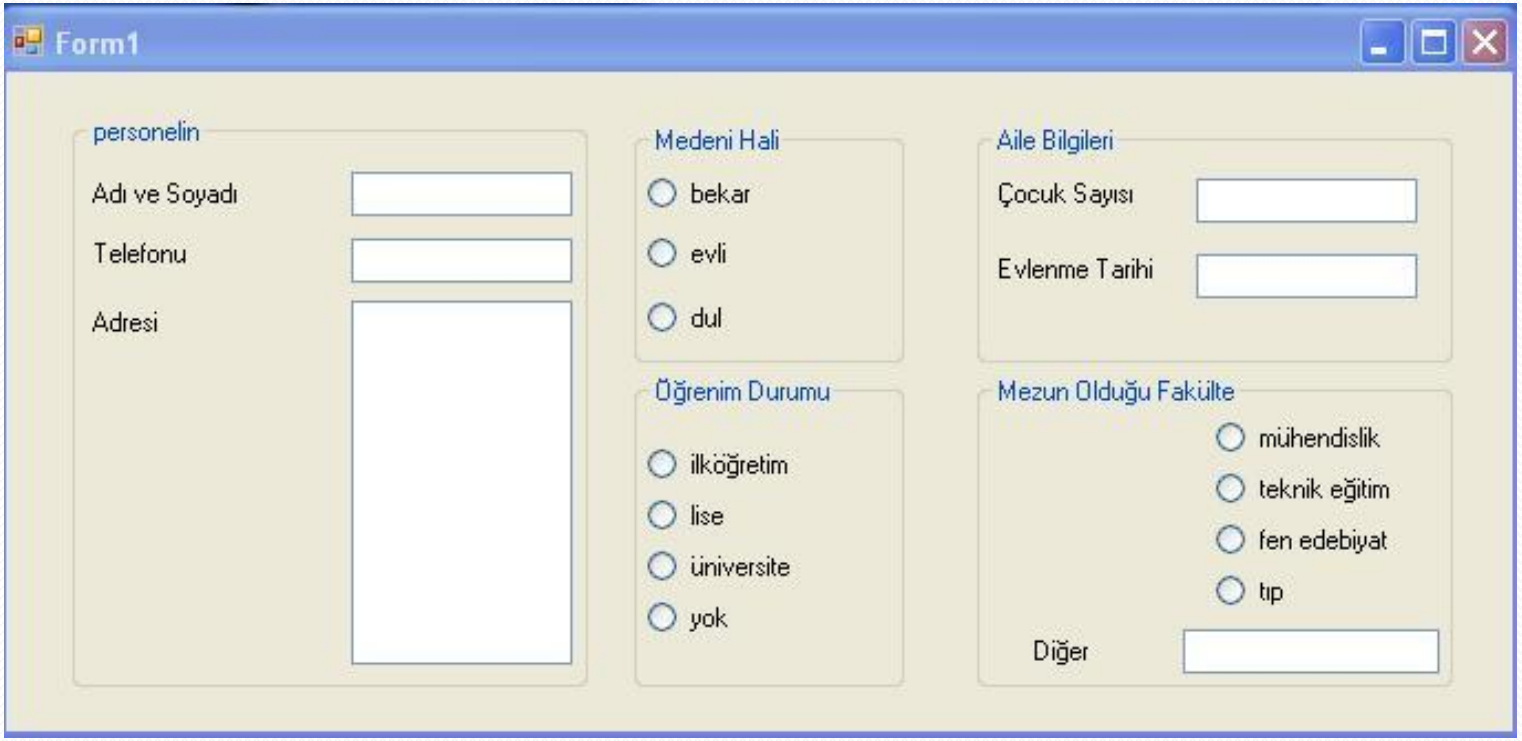
GroupBox KONTROLÜ

- Bu kontrol tek başına değil, diğer kontrolleri gruplamak için kullanılır. Kontrolleri bu GroupBox ile gruplamanın birçok avantajı vardır.
- GroupBox içine yerleştirilen kontroller, GroupBox'a bağımlıdırlar ve konumları bu çerçeve dışına taşamaz. Özellikle birkaç kontrolü birden görünür ya da görünmez yapmak için hepsinin Visible özelliğini tek tek değiştirmek yerine çerçevenin Visible özelliğini değiştirilerek çerçeve içindeki tüm kontroller aynı anda değiştirilebilir. Her bir kontrol tek tek taşınmak yerine çerçeve taşınır. Çerçevelerin buna benzer pek çok faydaları vardır. Ayrıca radioButton' ların gruplanmasında çerçeve kullanmak kaçınılmaz olabilir.



Örnek

- Aşağıdaki form dizaynında aile bilgileri bölümünün medeni hali bekar olanlar için gösterilmemesi, mezun olduğu fakülte bölümünün ise sadece öğrenim durumu üniversite olması durumunda gösterilmesi gerekir. Aile bilgileri ve mezun olduğu fakülte içindeki bütün kontrolleri tek seferde gizleyip, göstermek için sadece onun içinde bulunduğu GroupBox kontrolünü gizleyip göstermek yeterlidir.



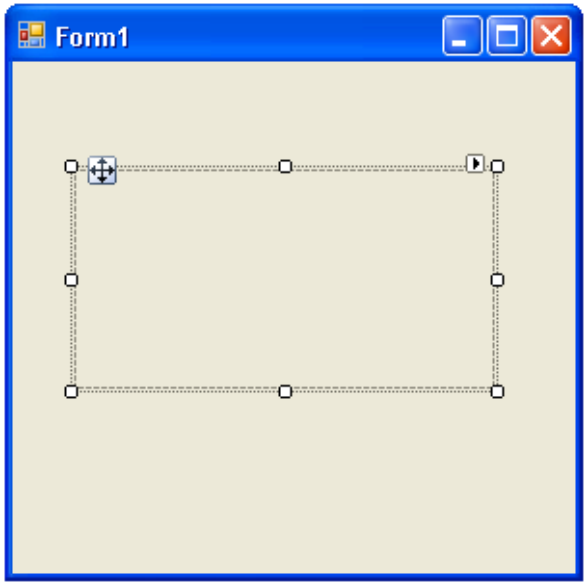
Örnek

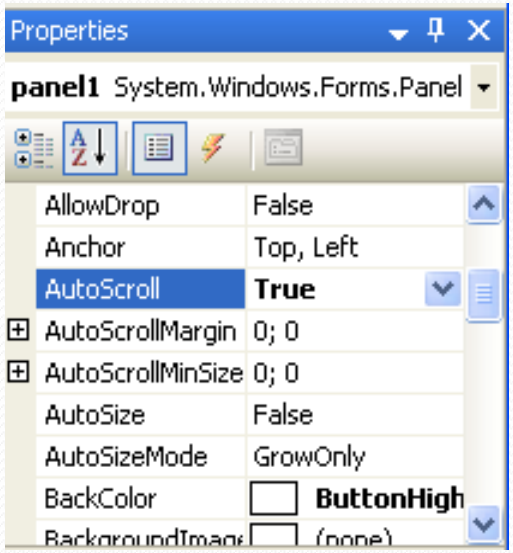
- Bekar seçeneği işaretli ise (radioButton1) Aile Bilgilerini (groupBox4) gizlememiz diğer durumlarda göstermemiz gerekir. Fakülte seçeneği işaretli ise(radioButton7) mezun olduğu fakülteyi (groupBox5) göstermemiz, diğer durumlarda gizlememiz gerekir. Buna göre;

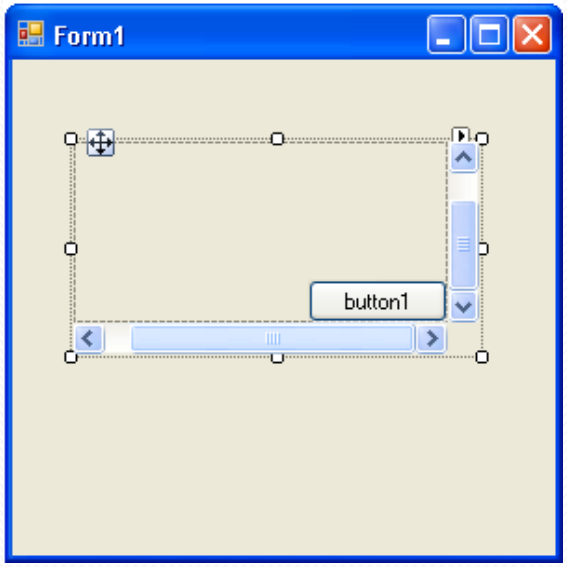
```
private void radioButton1_CheckedChanged(object sender, EventArgs e)
{
  if (radioButton1.Checked == true) groupBox4.Visible = false;
  else groupBox4.Visible = true;
}
private void radioButton6_CheckedChanged(object sender, EventArgs e)
{ if (radioButton6.Checked == true) groupBox5.Visible = true; }
private void radioButton4_CheckedChanged(object sender, EventArgs e)
{ if (radioButton4.Checked == true) groupBox5.Visible = false; }
private void radioButton5_CheckedChanged(object sender, EventArgs e)
{ if (radioButton5.Checked == true) groupBox5.Visible = false; }
private void radioButton7_CheckedChanged(object sender, EventArgs e)
{ if (radioButton7.Checked == true) groupBox5.Visible = false; }
```

Panel KONTROLÜ

- Kullanıma sokulan grupları kontrol etmek, üzerinde daha rahat işlem yapabilmek için bir arada tutan bir toolbox komutudur. Panel üzerine istediğimiz toolbox komutunu rahatça bırakabilir üzerlerinde işlem yapabiliriz. GroupBox gibi ayarlama yapan bir kontroldür. GroupBox'a göre en büyük artısı kaydırma çubuklarını (AutoScroll) desteklemesidir.





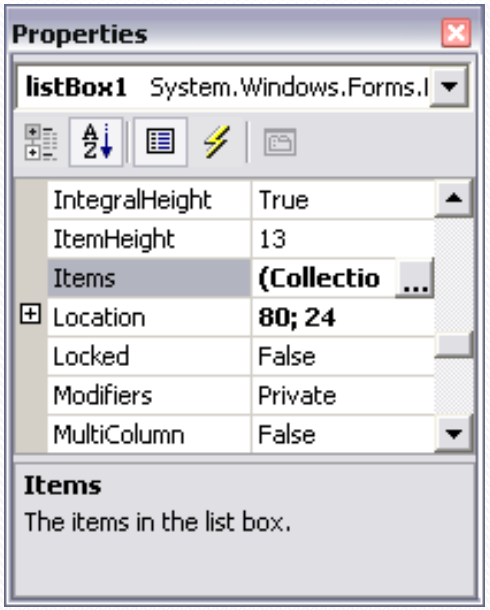


Visual Studio.Net -C#
13. HAFTA
FORM UYGULAMALARI

Visual Studio.Net -C#
13. HAFTA
FORM UYGULAMALARI

ListBox KONTROLÜ

- Kullanıcıya sunulan seçeneklerin bir liste halinde görünmesini sağlar. Liste kutusundan istenen sayıda öğe seçilebilir.



- Kod ortamında ListBox'a elaman ekleme

```
private void Form1_Load(object sender, System.EventArgs e)
{   listBox1.Items.Add("Elazığ");
    listBox1.Items.Add("Bursa");
    listBox1.Items.Add("Manisa");
}
```

ListBox KONTROLÜ

- **AddRange() metodu:** Birden fazla eleman bir seferde listBox nesnesine ekler. String bir dizinin elemanlarını da ekleyebilirsiniz.
  - listBox2.Items.**AddRange**(listBox1.Items);
  - listBox1.Items.**AddRange**(string dizi adı);

- **CopyTo() metodu:** listBox ın içeriğini bir dizi değişkene aktarır
  - string[] sehir; sehir=new string[3];
  - listBox1.Items.**CopyTo**(sehir,0);

- **Insert(sıra_no, eleman) metodu:** İstediğimiz bir sıraya eleman yerleştirmek
  - listBox1.Items.**Insert**(1, "eleman");

- **Remove() metodu:**Listboxtan bir eleman silmek
  - listBox1.Items.**Remove**("eleman");

ListBox KONTROLÜ

- **Clear() metodu :** Liste kutusunu tamamen boşaltmak

- **Contains (string) metodu:** Liste içerisinde bir elamanın olup olmadığını kontrol eder varsa true yoksa false değeri geri döner
  - **listBox1.Items.Contains("a");**

- **FindStringExact (string) metodu:** Contains ile aynı işleve sahip fakat elamanın ilk geçtiği index numarasını geri döndürür.
  - **listBox1. FindStringExact("a");**

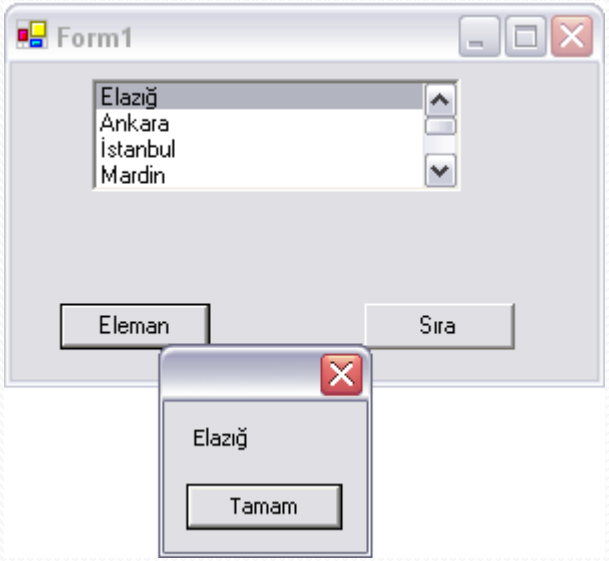
- **Count Özelliği:** Liste kutusundaki elaman sayısını verir.

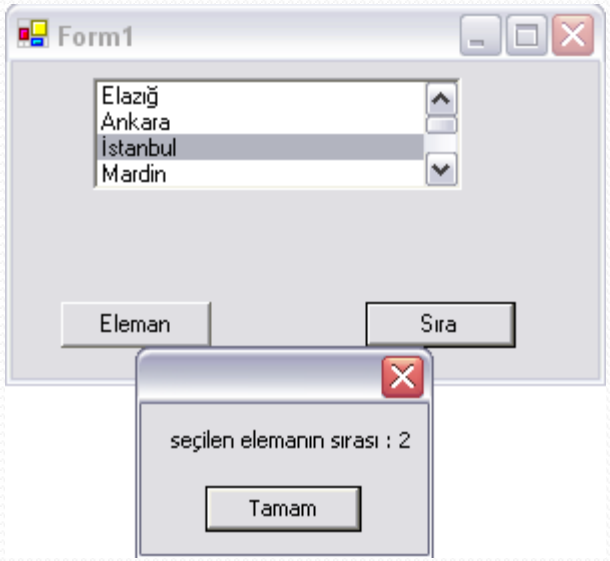
```
private void button1_Click(object sender, EventArgs e)
{
 for (int i = 0; i < listBox1.Items.Count; i++)
 { MessageBox.Show("" + listBox1.Items[i]); }
}
```

ListBox KONTROLÜ

- **Text Özelliği:** Liste kutusundaki seçilen elamanın değerini döndürür.
- **SelectedIndex Özelliği:** Liste kutusundaki elamanının indis numarasını verir.

```
• private void button1_Click(object sender, System.EventArgs e)
• { MessageBox.Show(listBox1.Text); }

• private void button2_Click(object sender, System.EventArgs e)
• { MessageBox.Show("seçilen elemanın sırası : " +
•                                   listBox1.SelectedIndex.ToString()); }
```





ListBox KONTROLÜ

- **SelectedIndexChanged() metodu:** Liste kutusundan herhangi bir eleman seçildiğinde çalışan olaydır.

```
private void listBox1_SelectedIndexChanged(object sender,
System.EventArgs e)
{
  MessageBox.Show(listBox1.Text);
}
```

ListBox KONTROLÜ

- **ListBox Özellikleri**

Özellik	Açıklama
Items	Liste kutusuna eklenen öğelerin tutulduğu koleksiyon nesnesidir.
SelectedItem	Liste kutusundan seçilen öğeyi alır.
SelectedItems	Liste kutusundan seçilen öğeleri alır. Seçilen öğeler dinamik bir dizide tutulur.
SelectedIndex	Liste kutusundan seçilen öğenin indisini alır.
SelectedIndices	Liste kutusundan seçilen öğelerin indislerini bir koleksiyon nesnesinde tutar.
DataSource	Listenin öğelerinin tutulduğu veri kaynağıdır. Veri kaynağı boş geçilirse Items koleksiyonuna eklenen öğeler görüntülenir.
DisplayMember	Veri kaynağından gelen öğelerin, kullanıcıya gösterilecek özelliğidir.
ValueMember	Veri kaynağından gelen öğelerin, dönüş değerini belirleyen özelliğidir.
SelectedValue	Seçilen öğenin, liste kutusunun ValueMember ile belirtilen özelliğidir.
SelectionMode	Liste kutusundan kaç tane öğe seçilebileceğini belirtir. None değeri 0, One değeri 1, MultiSimple ve MultiExtended değerleri birden fazla öğenin seçilebileceğini belirtir.
MultiColumn	Liste kutusundaki öğelerin biden fazla kolonda görüntülenmesini belirler.

ListBox KONTROLÜ

- **ListBox Olayları**

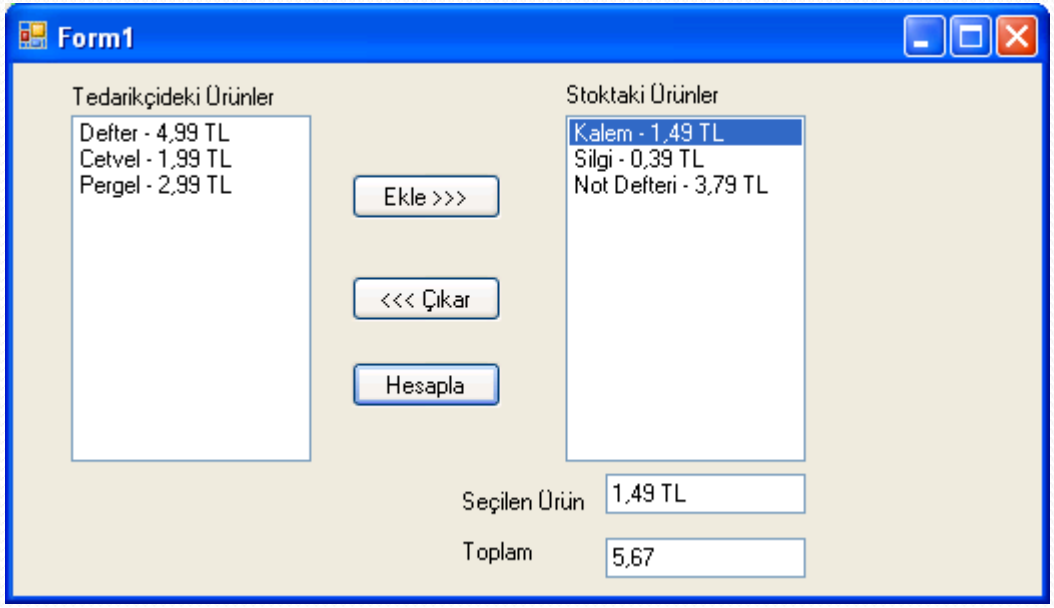
Olay	Açıklama
SelectedIndexChanged	Liste kutusunda bir öğe seçildiği zaman gerçekleşir.

- **ListBox Metotları**

Metot	Açıklama
GetItemText	Parametre olarak verilen nesnenin liste kutusunda gösterilen yazısını döndürür.
GetSelected	Parametre olarak verilen indisteki öğenin seçili olup olmadığını döndürür.
FindStringExact	Parametredeki String ifadesini liste kutusunda arayarak, bulduğu ilk öğenin indisini döndürür

Örnek:

- Tedarikçiden alınacak ve stokta bulunan ürünleri listelemek ve alım satım işlemi yapmak için ListBox kontrolleri kullanma.



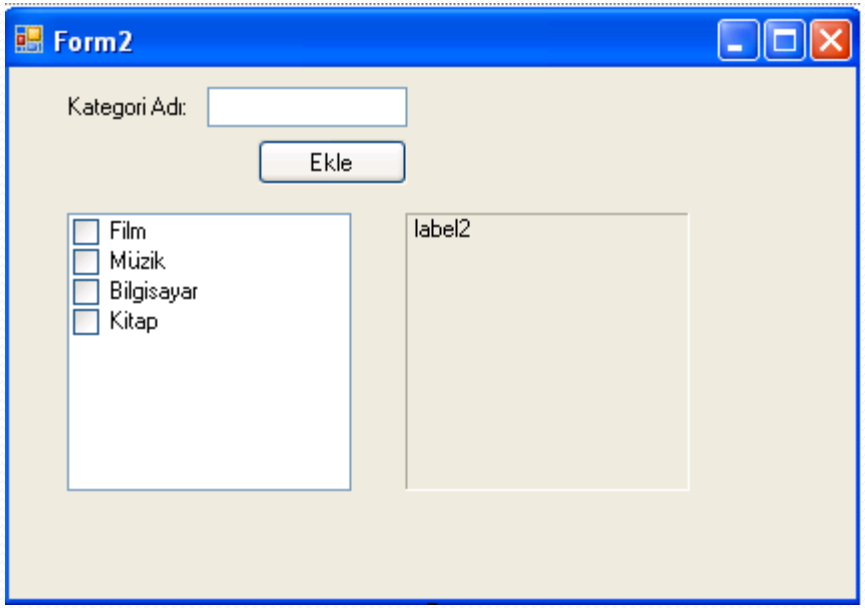
CheckedListBox KONTROLÜ

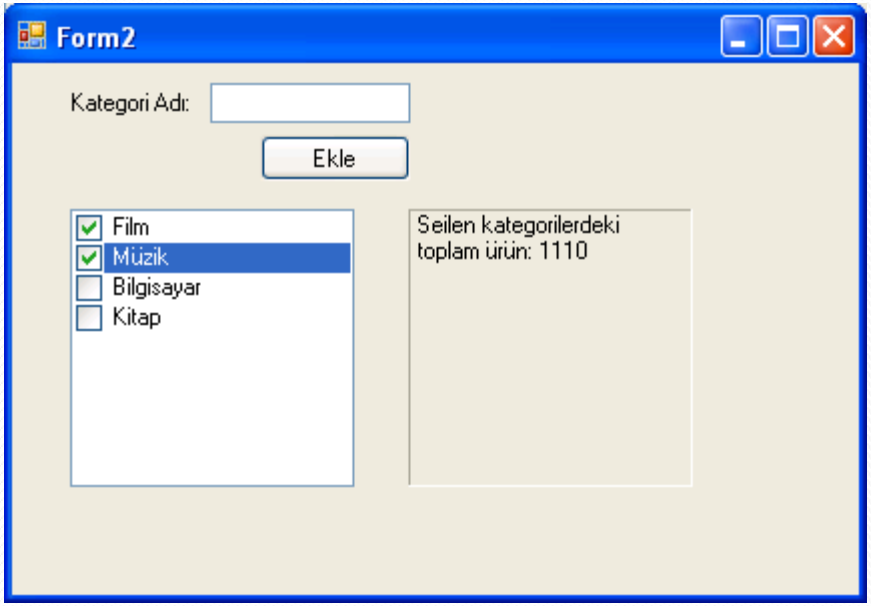
- Liste kutusunun tüm özellik, metot ve olaylarını alır ve listedeki öğelerin işaret kutusu ile gösterilmesini sağlar.

```
private void button1_Click(object sender, System.EventArgs e)
{   int eleman = checkedListBox1.Items.Count;
    for (int i = 0; i < eleman; i++)
    {  if (checkedListBox1.GetItemChecked(i) == true)
       listBox1.Items.Add(checkedListBox1.Items[i]);
    }
}
```

Örnek:

- Kategori başına stoktaki toplam ürünlerin gösterildiği bir uygulamada listelenen kategorileri seçmek için CheckedListBox kullanımı.





CheckedListBox KONTROLÜ

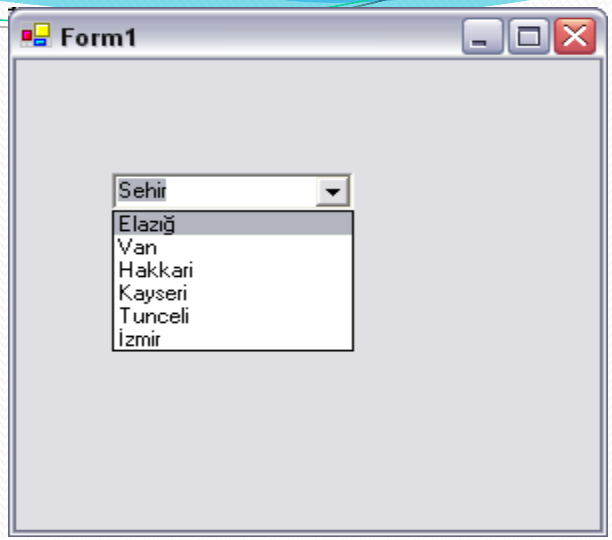
- **CheckedListBox Özellikleri**

Özellik	Açıklama
CheckedItems	Liste kutusunda işaretlenmiş öğeleri tutar
CheckedIndices	Liste kutusunda işaretlenmiş öğelerin indislerini tutar
CheckOnClick	Liste kutusunda öğeye tıklandığı zaman işaretlenmesini belirler. False ise ilk tıklamada öğe seçilir, ikinci tıklamada seçme kutusu işaretlenir.

- **CheckedListBox Metotları**

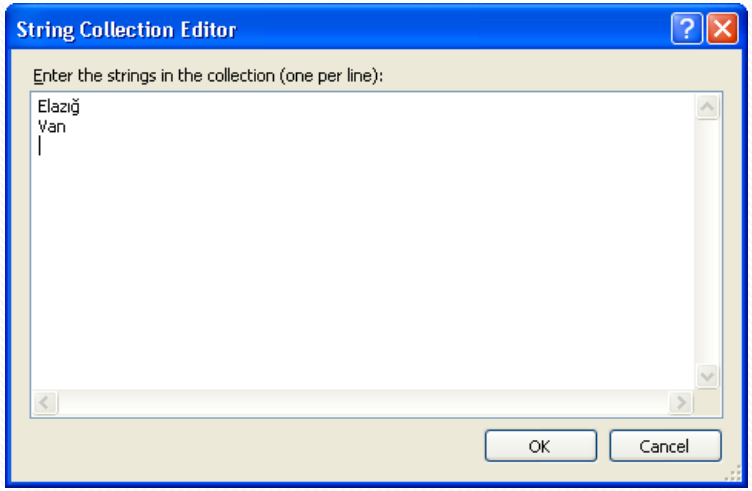
Metot	Açıklama
GetItemSelected	Parametre olarak verilen indisteki öğenin seçili olup olmadığını döndürür
SetItemSelected	İlk parametrede verilen indisteki elemanın seçili olup olmadığını, ikinci parametrede verilen Boolean değeri ile belirler

ComboBox KONTROLÜ



- 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, kullanıcın açılan kutu üzerinde değer girebilmesidir. Dolayısıyla bir TextBox kontrolü gibi de davranabilir.



- Listede bulunmasını istediğimiz elemanlar Items-> String Collection Editor diyalog kutusuna yazılır.

ComboBox KONTROLÜ

- Sayfa içerisinde kodlar ile ComboBox a eleman eklenebilir.

```
private void Form1_Load(object sender, System.EventArgs e)
{
 string[] meslekler = { "İşçi", "Memur", "Mühendis",
"Eğitimci", "Programcı", "Tekniker", "Veteriner" };

 comboBox1.Items.AddRange(meslekler);
}
```

ComboBox KONTROLÜ

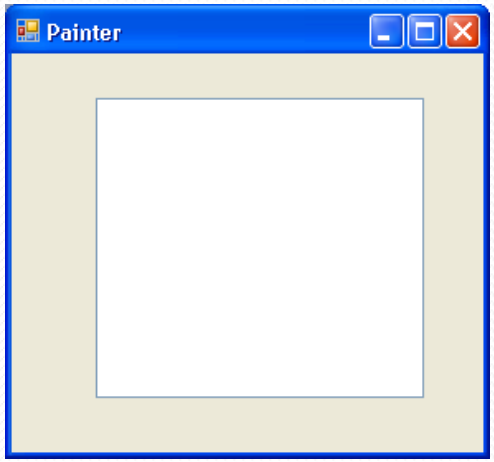
- **ComboBox Özellikleri**

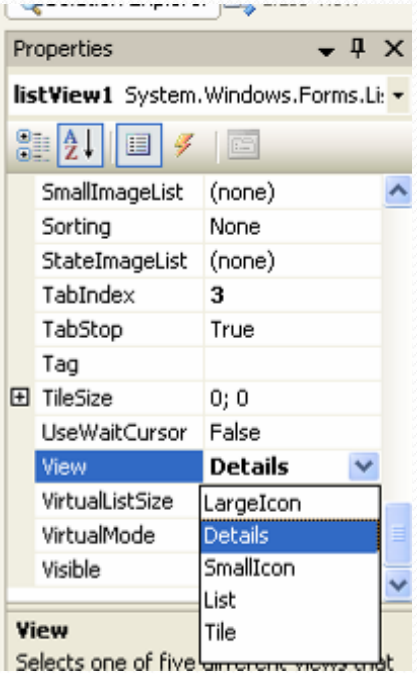
Özellik	Açıklama
DropDownStyle	Kontrolün listeleme stilini belirler. Simple stil, listedeki sadece bir öğeyi görüntüler. DropDown stili, listenin tüm elemanlarını görüntüleyerek seçilmelerini ve kullanıcının değer girmesini sağar. DropDownList kullanıcının değer girmesini engeller.
DropDownWidth	ComboBox kontrolünün açılan listesinin genişliğini belirler.
MaxDropDownItems	Kontrole eklenebilecek maksimum öğe sayısını belirler.
MaxLength	Kullanıcının girebileceği maksimum karakter sayısını belirler.
SelectedText	Seçilen öğenin görüntülenen yazısını belirler.

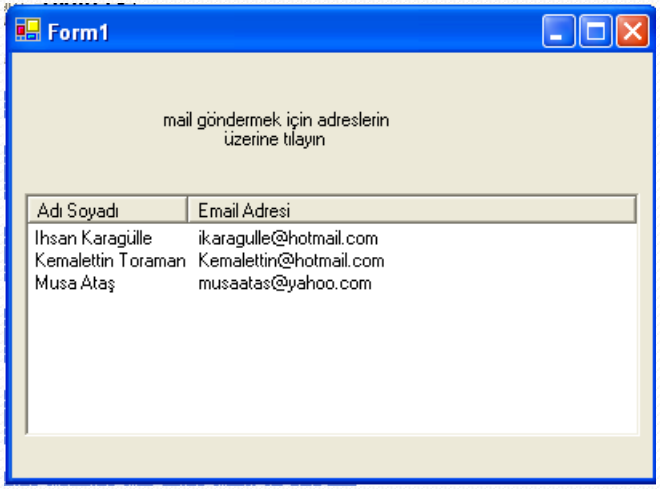
Örnek

- Tarih ve sayı formatlarını, kullanıcının seçimine bırakarak bir sayı veya tarih yazdırma işlemi ComboBox kontrolleri ile yapılabilir.

- **Listview Kontrolleri**
- **Listview** gelişmiş bir listeleme kontrolüdür. Listbox'ta olduğu gibi içine elemanlar eklenebilir, her elemana bir resim verilebilir ve listedeki elemanlar farklı biçimlerde listelenebilir. Form uygulamasına Listview özelliği eklendikten sonra **Properties-View** kısmından Details seçeneğini seçerek oluşturacak sütunlar görüntülenir.

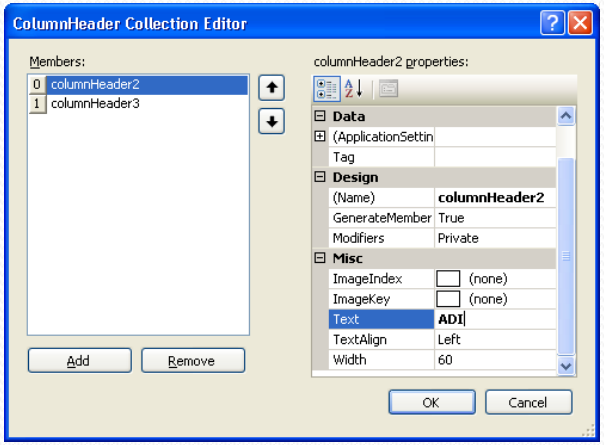


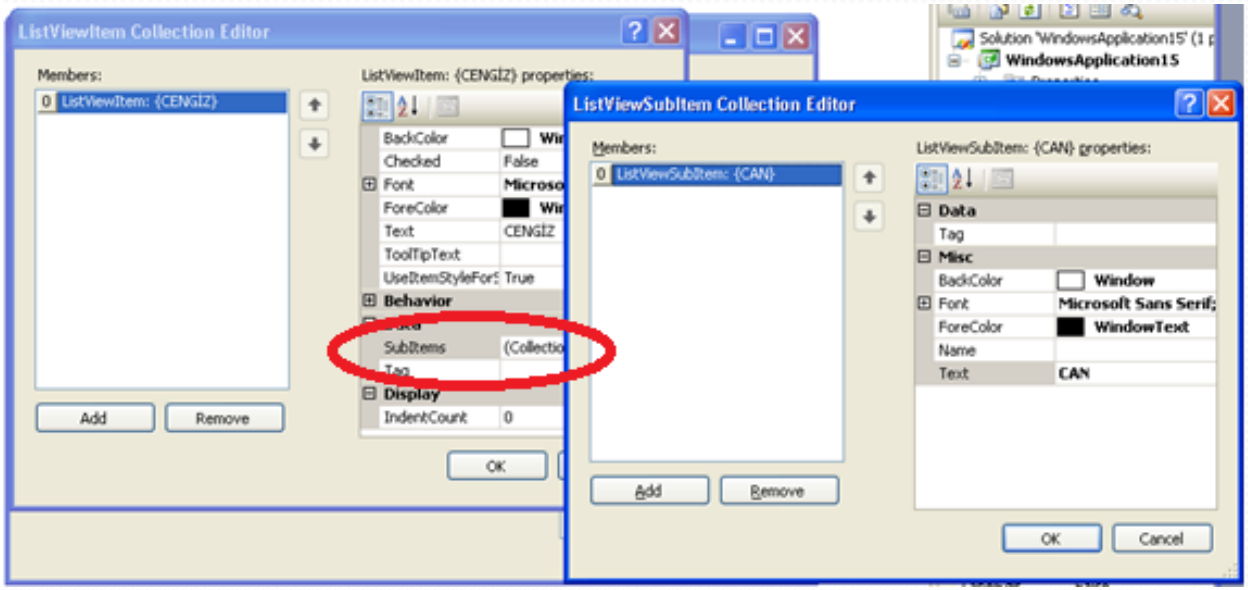




VISUAL STUDIO.NET ve FORM UYGULAMALARI

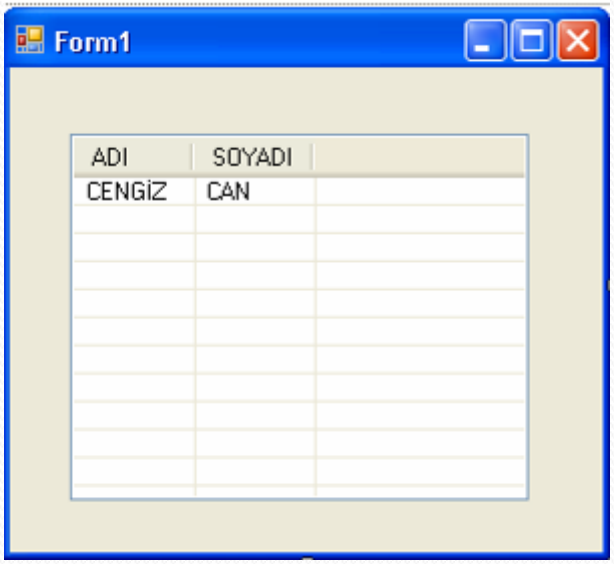
- Listview içerisindeki verileri sütunlar halinde gruplamak için **Properties/Columns (Collection)** seçeneği seçilir ve sütun ifadeleri girilir.
- Sütunlar içerisinde görülecek ifadeler içinde **Items (Collection)** seçeneğini seçilip ifadeler girilir. Burada dikkat edilmesi gereken birinci sütuna gelecek ifadeyi ekledikten sonra bu pencerede iken diğer sütunlara gelecek ifadeler için **SubItems (Collection)** seçeneğini kullanmaktır.





VISUAL STUDIO.NET ve FORM UYGULAMALARI

- Listview içerisine eklemeler kod ortamında da gerçekleştirilebilir.

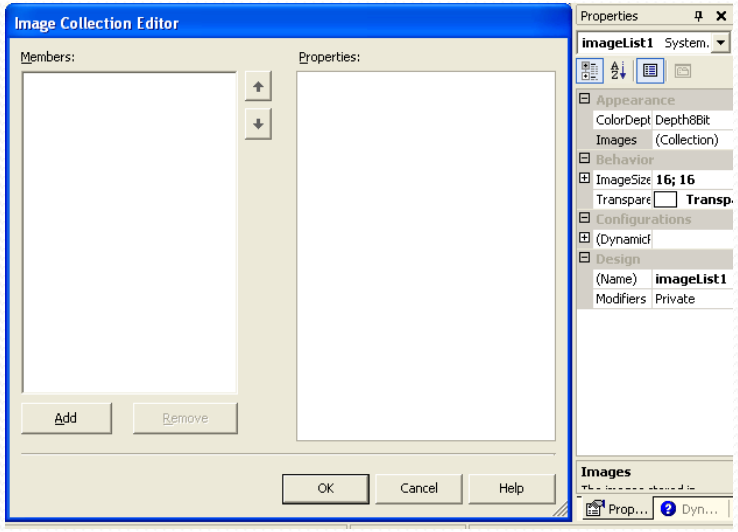


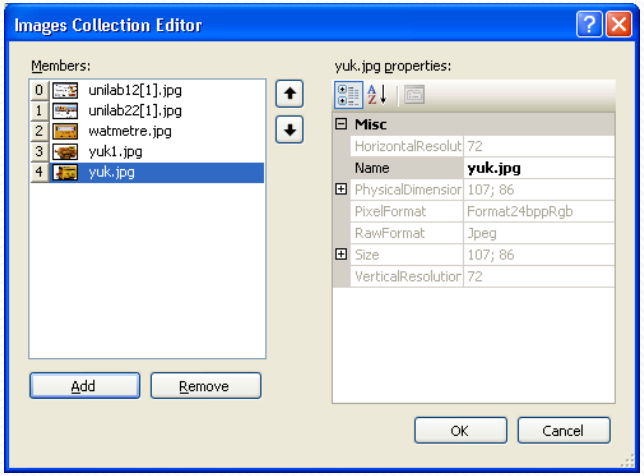
- private void Form1_Load(object sender, System.EventArgs e)
- {
- listView1.Activation = ItemActivation.OneClick;
- // tek tıklamada aktif hale getir.
- listView1.Columns.Add(“ADI", -2, HorizontalAlignment.Left);
- // Adı sütun başlığı, -2, Girilen ifadenin tam olarak sığması için gereken ifade, sola hizalama
- listView1.Columns.Add(“SOYADI",-2,HorizontalAlignment.Left);
- listView1.Items.Add(“CENGİZ");
- listView1.Items[0].SubItems.Add(“CAN");
- // eğer 3. bir sütun (adres) olsaydı //listView1.Items[0].SubItems.Add(“Elazığ, 23200"); aynı şekilde eklemeye devam edilecekti
- // listView1.Items[0].SubItems[1].Text="Ali";
- // Can değeri Ali olarak değişti. (Subitems indisi 1 den başlar)

- listView1.View = View.Details;
- listView1.Sorting = SortOrder.Ascending;// Sıralama }

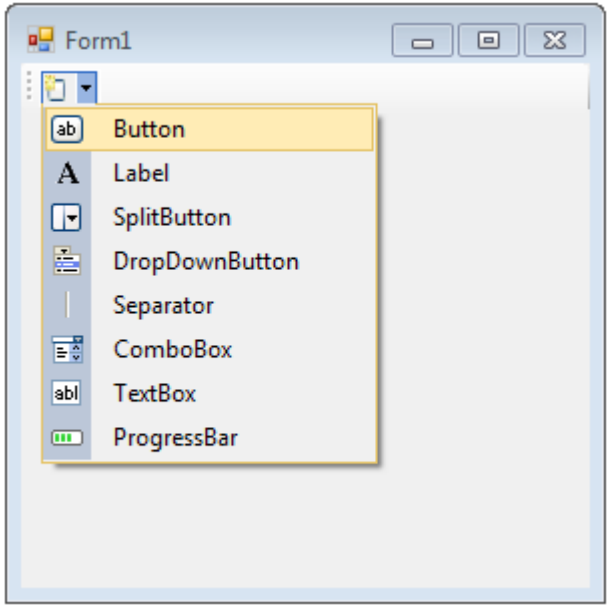
- **Imagelist Kontrolleri**

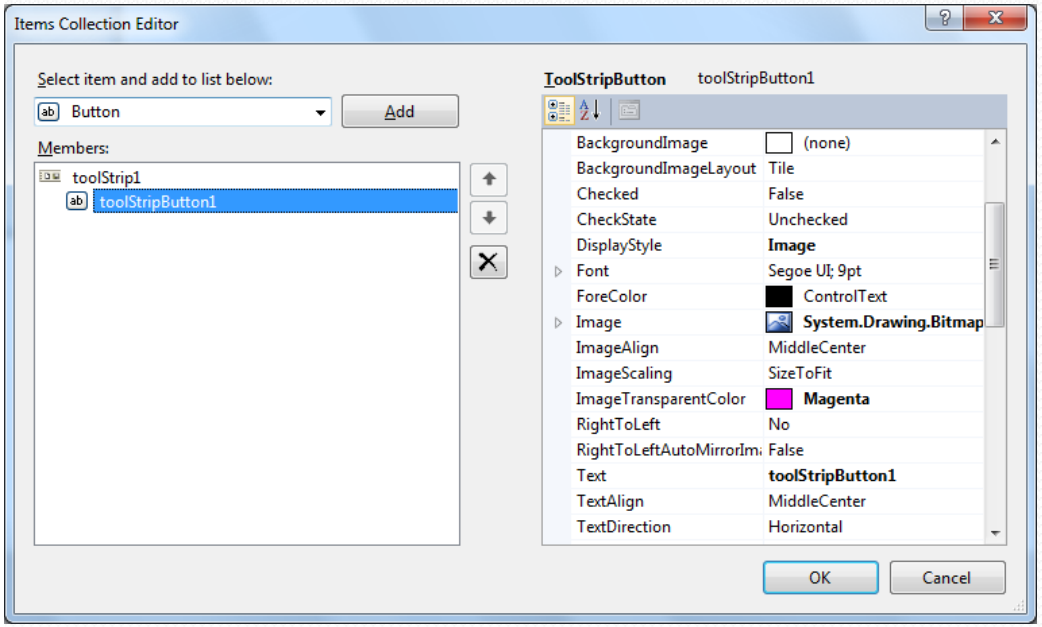
- Birden fazla resmi bir arada tutmak için kullanılan bir kontroldür. Daha çok Listview, ToolBar , TreeView gibi birden fazla resmin bulunabileceği kontrollerdeki resimleri bir arada tutmak içim kullanılır.
- Bu kontrolü form üzerine yerleştirdikten sonra Properties penceresindeki Images özelliğinin yanında bulunan ... düğmesine tıklayarak resimleri belirleyebilirsiniz.





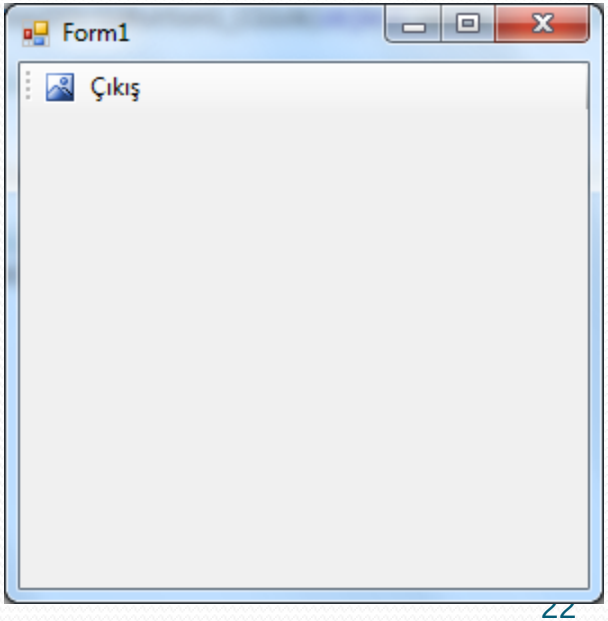
- **ToolStrip Kontrolleri**
- Araç çubukları için hazır düğmeler sunan bir kontroldür. ToolStrip kontrolü içine düğmeler ekleyip üzerlerine resim yerleştirilebilir veya yazı yazılabilir.



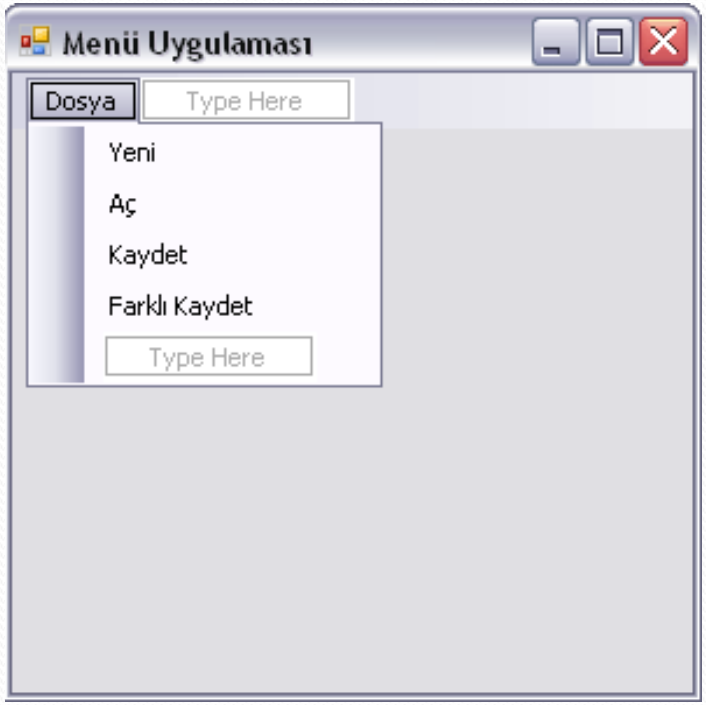


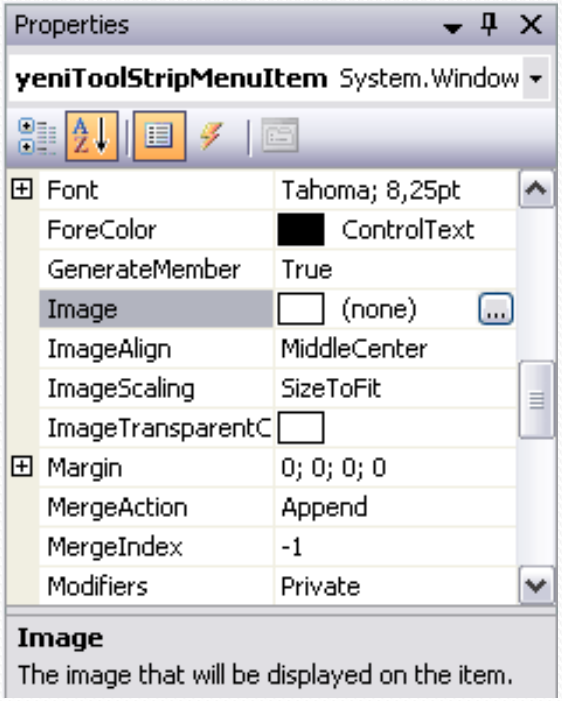
- **ToolStrip Kontrolleri**

```
private void toolStripButton1_Click(object sender, EventArgs e)
{ MessageBox.Show("Toolsript Buton 1 e tıklandı"); }

private void toolStripLabel1_Click(object sender, EventArgs e)
{ MessageBox.Show("Toolsript Label 1 e tıklandı");
  this.Close();
}
```



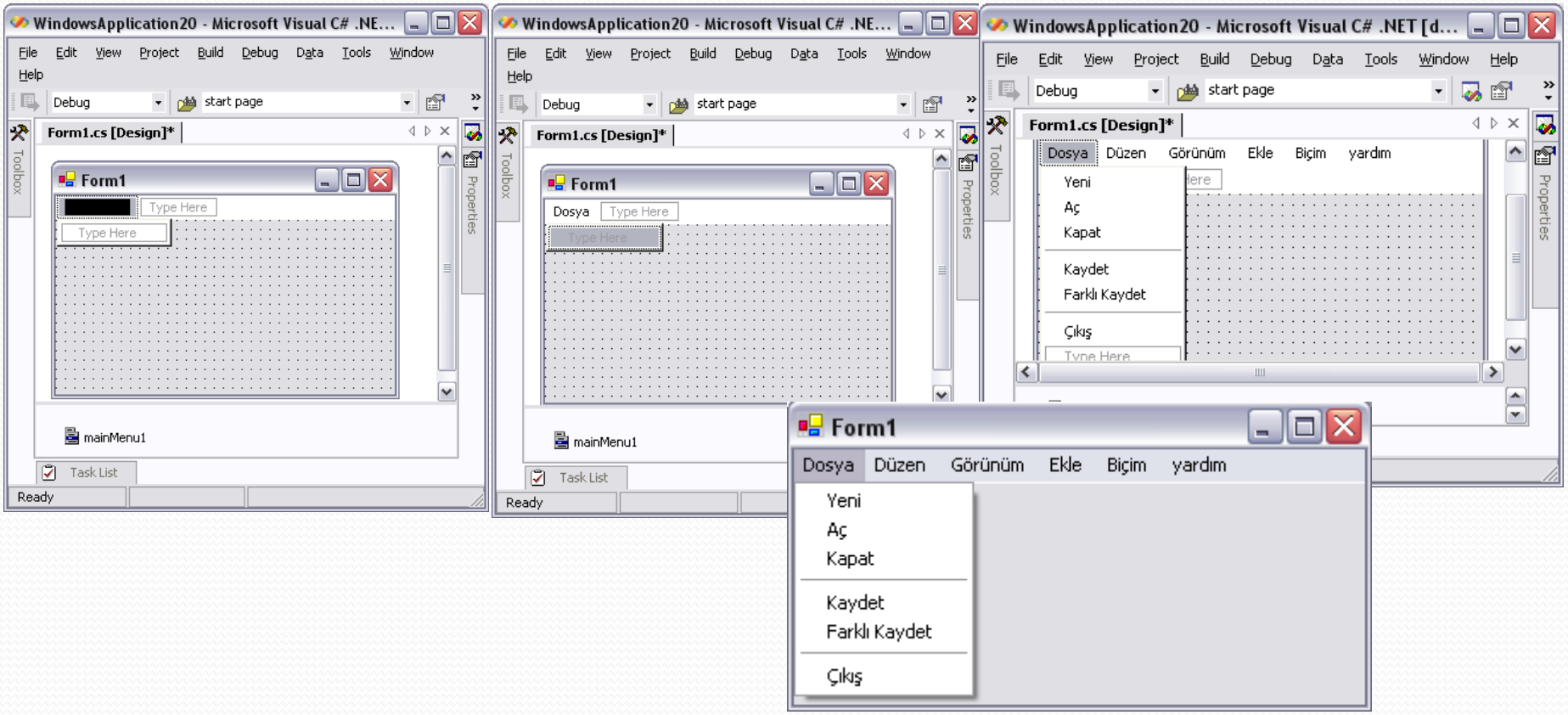
- **Menü Oluşturmak -MenuStrip**
- Araç kutusundan (ToolBox) MenuStrip öğesini seçip forma eklenerek menü oluşturulur. Menu isimleri *Type Here* yazan kutucuğa yazılır. Alt menü oluşturmak için *Type Here* öğesinin sağındaki ok işaretine tıklanır ve aynı işlem tekrar edilir.





VISUAL STUDIO.NET ve FORM UYGULAMALARI

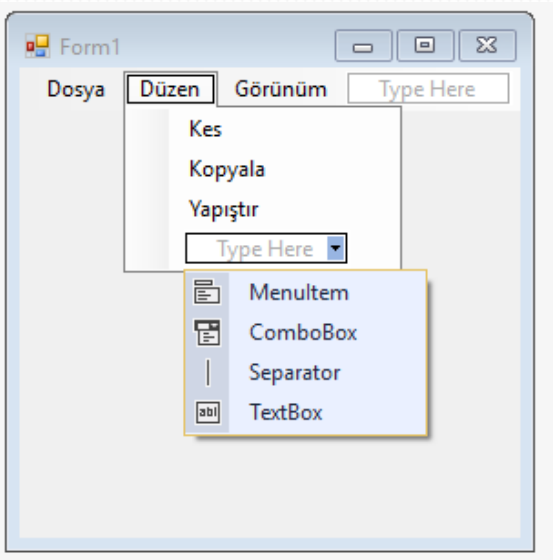
- **MenuStrip**



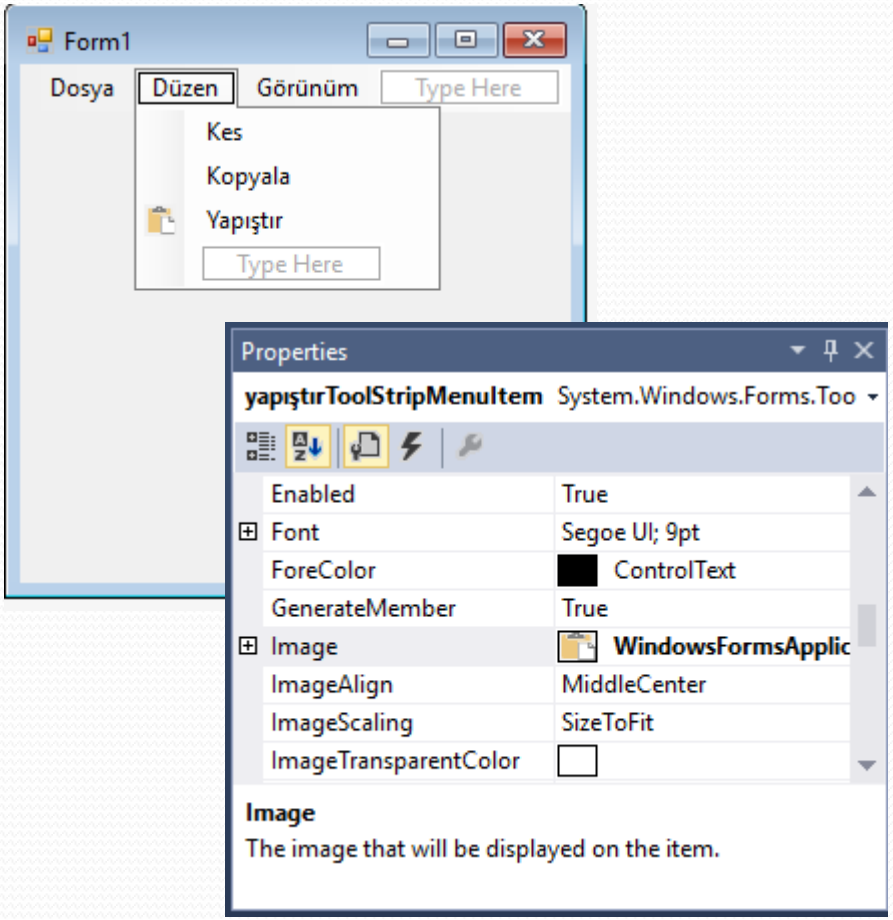

```
private void yeniToolStripMenuItem_Click(object sender, EventArgs e)
{MessageBox.Show("Yeni  Komutunu verdiniz"); }
```

- **MenuStrip**

Menü öğelerine ayırıcı çubuk (seperator bar) eklemek için,

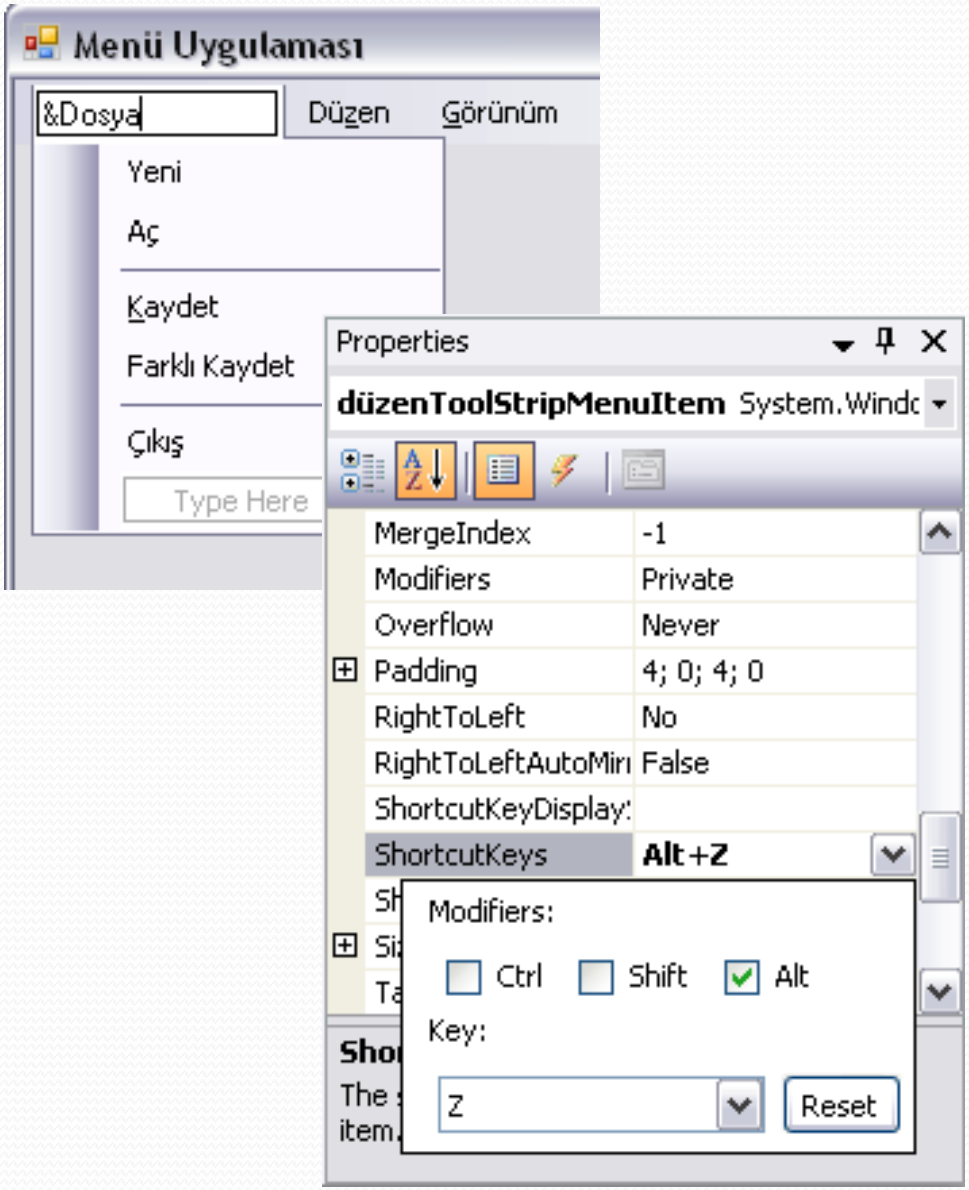


Menü öğelerine resim ekleme: Properties/Image



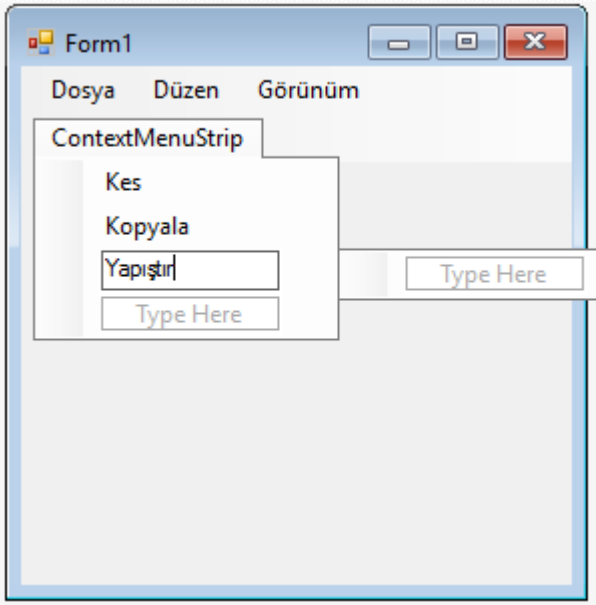
VISUAL STUDIO.NET ve FORM UYGULAMALARI

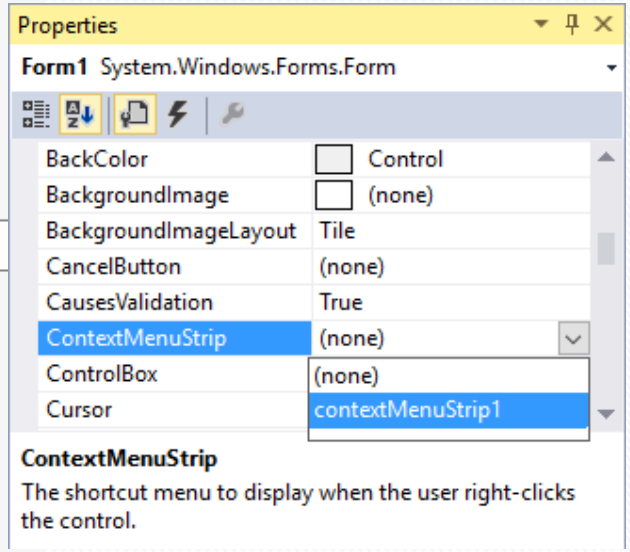
- **Menü Erişimi:**
- & (Ampersand) işareti eklenen karaktere ALT+karakter ile klavyeden erişebilirsiniz (ALT+D)

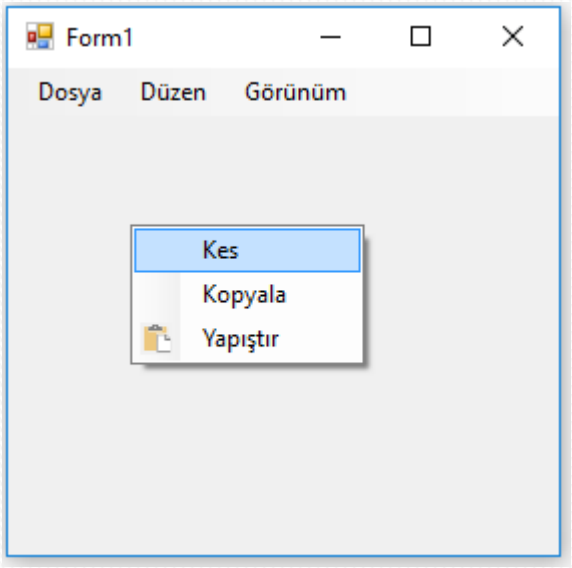


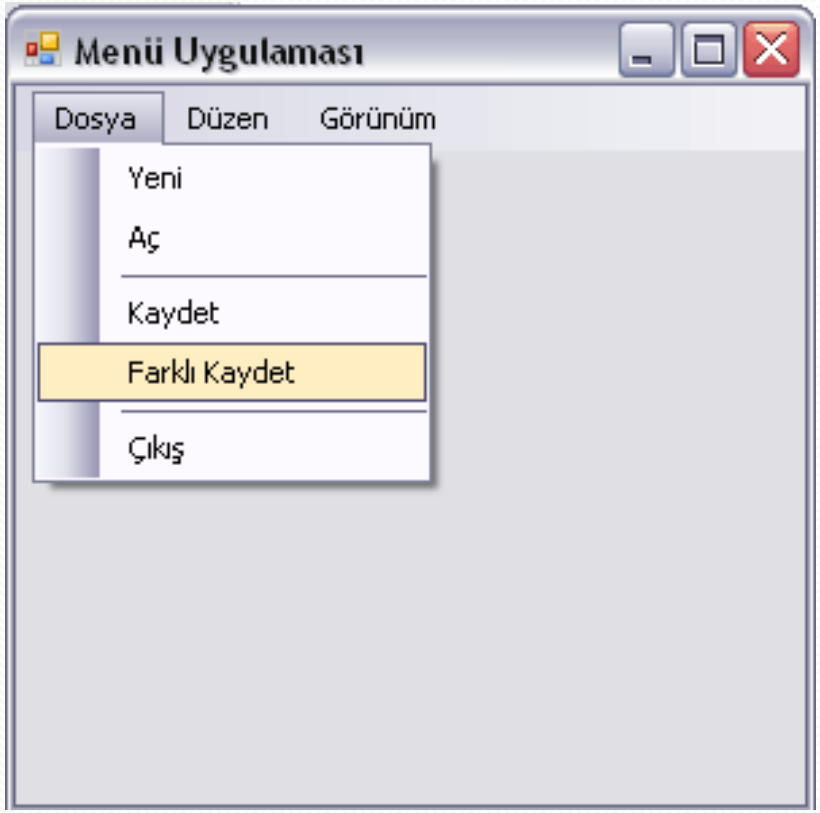
- **Kısa yol Tuşları :**
- Properties-ShortcutKeys özelliği ile menü ve altında bulunan elamanlara faklı kısa yollar ataya bilirsiniz.

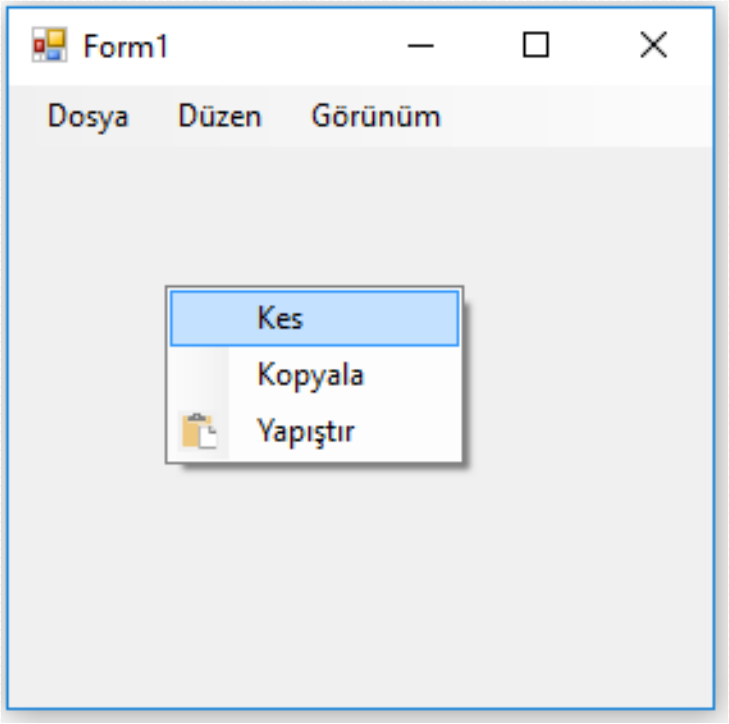
- **Menü Oluşturmak -ContexMenuStript**
- İçerik menüleri genellikle, öğe üzerinde sağ tık yapıldığında açılan menüler için kullanılır. İçerik menüsünü oluşturduktan sonra kullanabilmek için formun özelliklerinde bunun tanımlanması gerekir.



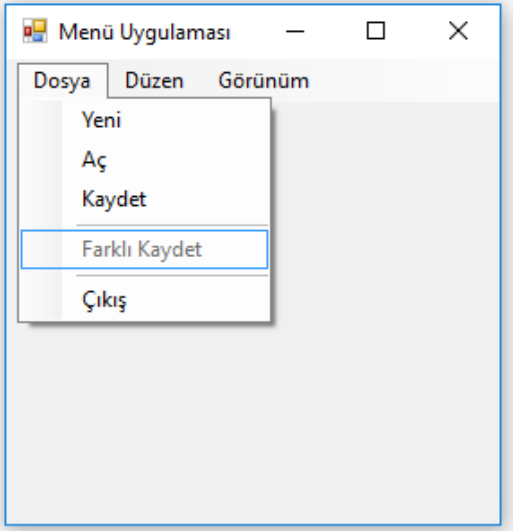





Menü Uygulaması
Dosya
Düzen
Görünüm
Yeni
Aç
Kaydet
Farklı Kaydet
Çıkış

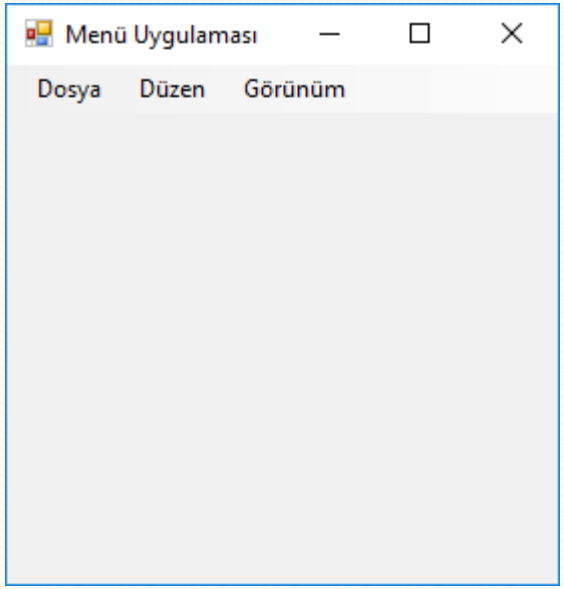

Form1
Dosya
Düzen
Görünüm
Kes
Kopyala
Yapıştır

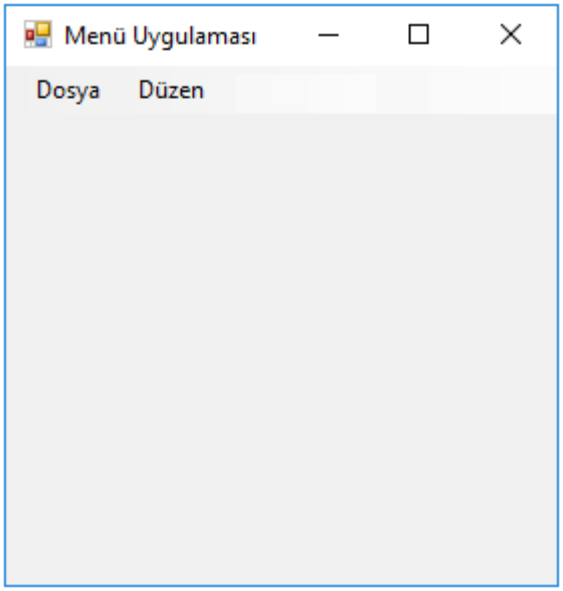
- **Çalışma Zamanında Menüleri Değiştirmek**
- Menüleri çalışma zamanındaki şartlara bağlı olarak dinamik bir şekilde yönetilebilir. Örneğin programın belirli bir görevi yerine getiremediği zamanlarda, çağrılan bir komutla bu menü pasif duruma getirilebilir veya gizlenebilir. Çalışma zamanında bir menü öğesi eklenebilir veya başka bir menü öğesiyle birleştirilebilir.
- **Aktif/Pasif Yapma**
- `private void Form1_Load(object sender, EventArgs e)`
- `{farklıKaydetToolStripMenuItem.Enabled = false; }`



- **Menü Öğelerini Gizleme**

```
private void Form1_Load(object sender, EventArgs e)
{ GörünümToolStripMenuItem.Visible = false;}
```





VISUAL STUDIO.NET ve FORM UYGULAMALARI

- **NumericUpDown**
- Bu kontrol kullanıcının, sayısal bir değeri girmesini veya yukarı aşağı okları ile seçmesini sağlar.

- **Örnek:** Alarm kurarken, tarih ve zaman değerlerinin ayarlanması NumericUpDown kontrolü ile yapılabilir.

VISUAL STUDIO.NET ve FORM UYGULAMALARI

```
string tarih; string zaman;
private void Form4_Load(object sender, EventArgs e)
{numeric_Ay.Minimum = 1; numeric_Gun.Minimum = 1;
 numeric_Yil.Minimum = 1; numeric_Yil.Maximum = 9999;
 numeric_Ay.Maximum = 12; numeric_Gun.Maximum = 31;
 numeric_Saat.Minimum = 0; numeric_Dakika.Minimum = 0;
 numeric_Saat.Maximum = 23; numeric_Dakika.Maximum = 59;
 numeric_Yil.Value = DateTime.Now.Year;
 numeric_Ay.Value = DateTime.Now.Month;
 numeric_Gun.Value = DateTime.Now.Day;
 numeric_Saat.Value = DateTime.Now.Hour;
 numeric_Dakika.Value = DateTime.Now.Minute;
}

private void button1_Click(object sender, EventArgs e)
{
 tarih = numeric_Gun.Value + "." + numeric_Ay.Value + "." +
         numeric_Yil.Value + " ";
 zaman = numeric_Saat.Value + ":" + numeric_Dakika.Value;
 label3.Text = "Alarm Saati :" + tarih + zaman;
}
```

VISUAL STUDIO.NET ve FORM UYGULAMALARI

- **NumericUpDown Özellikleri**

Özellik	Açıklama
`HexaDecimal`	Sayıların on altılık tabanda görüntülenmesini belirler.
`Increment`	Aşağı yukarı oklar kullanıldığında, sayıların artma ve azalma adımlarını belirler.
`Maximum`	Kontrolde gösterilen sayıların alabileceği maksimum değeri belirler.
`Minimum`	Kontrolde gösterilen sayıların alabileceği belirler minimum değeri belirler.
`ThousandSeparators`	Sayıların basamak ayracını gösterilmesini belirler.
`Value`	Kontrolü gösterdiği sayı değerini belirler.
`ReadOnly`	`True` değerini alırsa kullanıcının giriş yapmasını engeller.

VISUAL STUDIO.NET ve FORM UYGULAMALARI

- **NumericUpDown Olayları**

Olay	Açıklama
`ValueChanged`	Aşağı yukarı oklar kullanıldığında meydana gelen değişim için tetiklenen olay.

- **NumericUpDown Metotları**

Metot	Açıklama
`DownButton`	Aşağı düğmesine basar ve sayı değerini düşürür.
`UpButton`	Yukarı düğmesine basar ve sayı değerini artırır. numericUpDown1.UpButton();

VISUAL STUDIO.NET ve FORM UYGULAMALARI

- **DomainUpDown Kontrolü**
- 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	Açıklama
Items	Kontrolün öğelerinin tutulduğu dinamik bir listedir
SelectedItem	Kontrolde seçilen öğeyi tutar
Wrap	Liste sonuna gelindiğinde baştaki veya sondaki öğeye geri dönülmesini belirler.

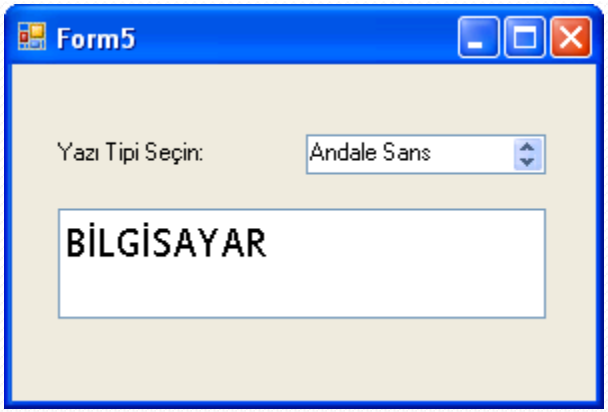
- **DomainUpDown Olayları**

Olay	Açıklama
SelectedItemChanged	Kontrolde seçilen öğe değiştiği zaman gerçekleşir.

VISUAL STUDIO.NET ve FORM UYGULAMALARI

- Örnek: Metin kutularının değiştirilmek istenen yazı tipleri DomainUpDown kontrolünde tutulabilir.

```
private void Form5_Load(object sender, EventArgs e)
{
  for (int i = 10; i < 100; i++)
  {domainUpDown1.Items.Add(System.Drawing.FontFamily.Families[i].Name);   }
  domainUpDown1.Wrap = true;
}
private void domainUpDown1_SelectedItemChanged(object sender, EventArgs e)
{ if (domainUpDown1.SelectedIndex >= 0)
 {textBox1.Font = new Font(Convert.ToString(domainUpDown1.SelectedItem), 15);}
}
```



VISUAL STUDIO.NET ve FORM UYGULAMALARI

- **HScrollBar / VscrollBar Kontrolleri**

- **Horizontal - Vertical ScrollBar** kontrolleri, sayısal bir değer taşıyan kaydırma çubuklarıdır. Tuttukları değerlerin sayısal olması bakımından **NumericUpDown** kontrolüne benzer.

- Bu kontroller, üzerlerinde 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.

VISUAL STUDIO.NET ve FORM UYGULAMALARI

- **ScrollBar Özellikleri**

Özellik	Açıklama
Value	Kaydırma çubuğunun pozisyonuna göre alınan değeri tutar.
SmallChange	Kontrolü, üstündeki oklar ile kaydırıldığı zaman eklenecek ya da çıkartılacak değeri tutar.
LargeChange	Kontrolü, kaydırma çubuğundaki boşluğa tıklanarak kaydırıldığında zaman eklenecek ya da çıkartılacak değeri tutar.
Minimum	Value özelliğinin alabileceği değeri minimum tutar
Maximum	Value özelliğinin alabileceği maksimum değeri tutar

- **ScrollBar Olayları**

Olay	Açıklama
Scroll	Çubuklar kaydırıldıkları zaman gerçekleşir.
ValueChanged	Kod ile ya da çubuklar kaydırılınca Value özelliği değiştiği zaman gerçekleşir.

VISUAL STUDIO.NET ve FORM UYGULAMALARI

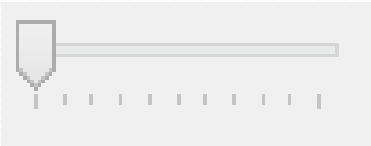
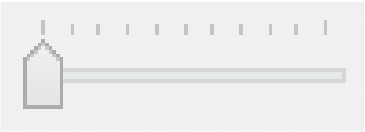
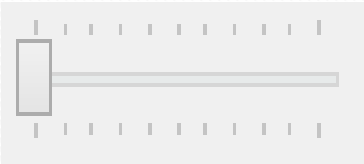
- **Örnek:** Bir **ComboBox** kontrolünün öğelerini listelemek için, aşağıya doğru bir kaydırma çubuğu görüntülenir. Ancak listedeki bazı elemanların kontrole sığmıyorsa, çalışma anında bu kontrolün genişliği artırılabilir.

```
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 Kontrolleri**

- 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** kontrolünün birçok özelliği **ScrollBar** kontrollerinin özellikleriyle aynıdır. Fakat kontrolü daha esnek hale getiren birkaç özelliği vardır.

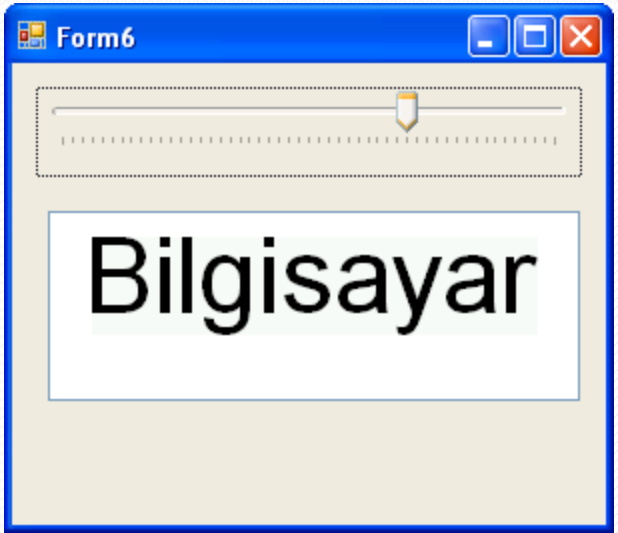




- **TrackBar Özellikleri**

Özellik	Açıklama
`TickStyle`	Kontrolün değerini gösteren çizgilerin pozisyonunu belirler
`TickFrequency`	Çizgiler arasında kalan değerlerin sayısını belirler
`Orientation`	Kontrolün yönünün yatay veya dikey olmasını sağlar.

VISUAL STUDIO.NET ve FORM UYGULAMALARI

```
private void Form6_Load(object sender, EventArgs e)
{
    trackBar1.Minimum = 5;
    trackBar1.Maximum = 55;
}
private void trackBar1_Scroll(object sender, EventArgs e)
{
    textBox1.Font = new Font("Arial", trackBar1.Value);
}
```



- **TabControl Kontrolleri**

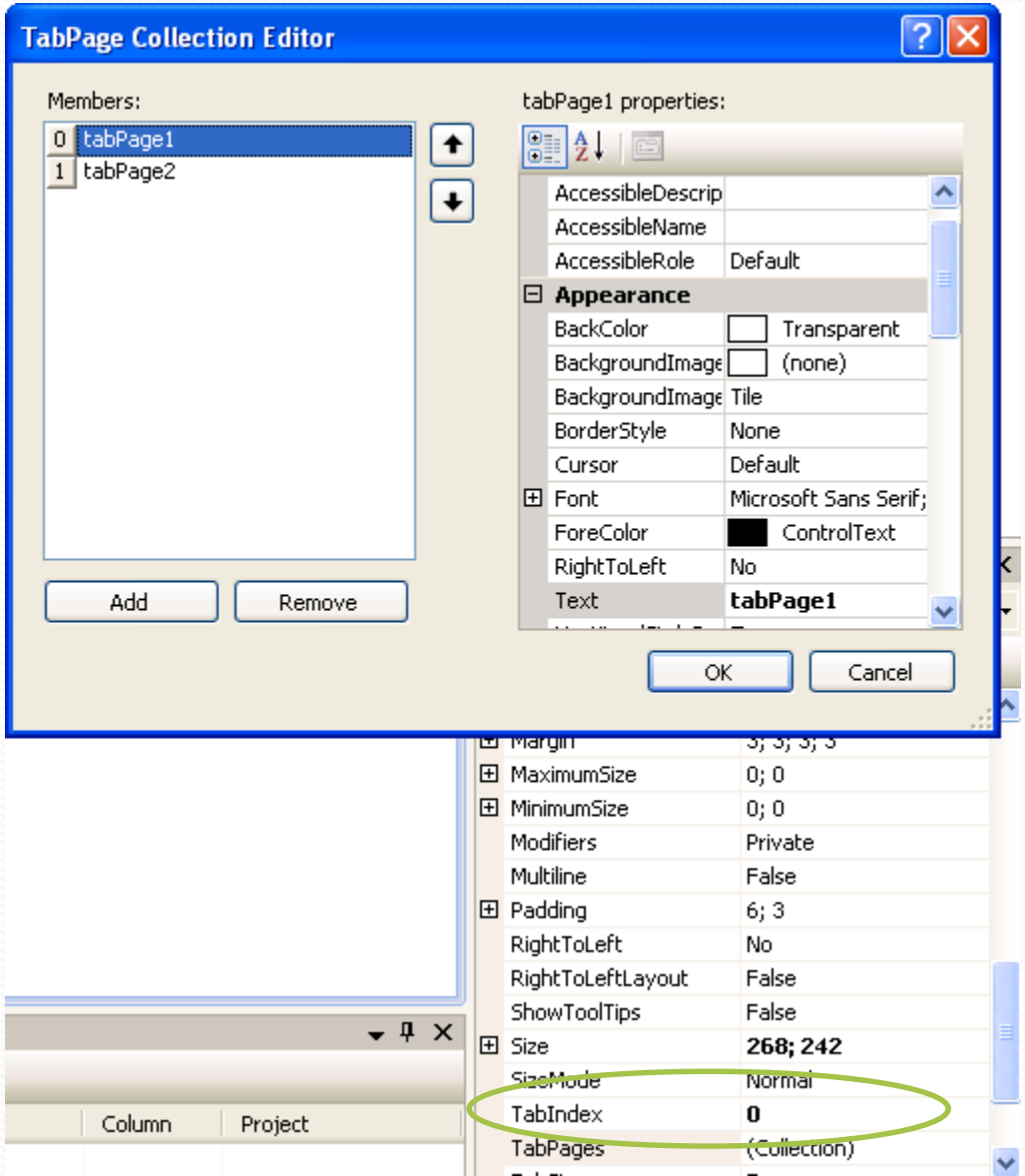
- **TabControl** nesnesi, içinde sekme sayfaları tutan yapıdır. Bu sayfalar, **TabPage** nesneleri olarak oluşturulup yapılandırıldıktan sonra **TabControl** nesnesinin **TabPages** koleksiyonuna eklenir. Ekleme işlemi, **Properties** paneli ile tasarım anında da yapılabilir.

VISUAL STUDIO.NET ve FORM UYGULAMALARI

Özellik	Açıklama
HotTrack	Fare ile sekme sayfalarının üzerine gelindiğinde, isimlerinin görsel olarak değişmesini (arka zemin mavi ton) belirler.
ItemSize	Sekme sayfalarının boyutunu belirler
Multiline	Eklenen sekmelerin birden fazla satırda üst üste gözükmesini belirler
ShowToolTips	Fare sekme sayfalarının üzerindeyken bilgi mesajının gösterilmesini belirler. ToolTipText değer girilmesi gerekir.
SelectedTab	Seçilen sekme sayfasını belirler
SelectedIndex	Seçilen sekme sayfasının indisini belirler
TabCount	Sekme sayısını belirler
TabPages	Kontrolün içinde bulunduğu sekme sayfalarının koleksiyonudur.

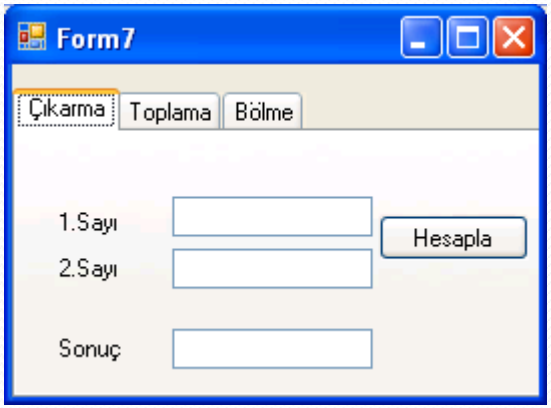
VISUAL STUDIO.NET ve FORM UYGULAMALARI

- **TabControl** nesnesine **TabPage** sayfaları eklemek için tasarım anında **TabPages Collection Editor** penceresinden yararlanılabilir.



- **TabPage Özellikleri**
- Sekme sayfaları, normal form tasarımları gibi kontroller eklenerek yapılır .
- **TabPage** kontrolü **Panel** kontrolünden türer ve **Panel** kontrolünün tüm özelliklerini alır.

Özellik	Açıklama
`ToolTipText`	Bu özelliğin değeri, fare sayfanın üzerindeyken, bilgi mesajı olarak gösterilir. Ait olduğu TabControl nesnesinin ShowToolTip özelliği True olmalıdır.



- **Timer Kontrolleri**

- Zaman değeri ayarlanabilen sayaçtır.
- 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.

VISUAL STUDIO.NET ve FORM UYGULAMALARI

- **Timer Özellikleri**

Özellik	Açıklama
`Enabled`	Kontrolün aktif olup olmadığını belirler.
`Interval`	Sayacın hangi zaman aralığında bir çalışması gerektiğini belirler. Milisaniye cinsindedir.

- **Timer Olayları**

Olay	Açıklama
`Tick`	`Interval` özelliğinde belirtilen zaman değeri geçtiğinde gerçekleşir.

- **Timer Metotları**

Metot	Açıklama
`Start`	Sayacı başlatır
`Stop`	Sayacı durdurur

VISUAL STUDIO.NET ve FORM UYGULAMALARI

```
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(); }
```

- Uygulama: Kronometre yapınız

VISUAL STUDIO.NET ve FORM UYGULAMALARI

- **ProgressBar**

- Yapılan işlemlerin ilerleyişini gözlemeyi sağlar.
- Maksimum ve minimum değerleri arasındaki pozisyonu gösterir.
- **ProgressBar Özellikleri**
-

Özellik	Açıklama
`Minimum`	Kontrolün alabileceği minimum değer belirler
`Maximum`	Kontrolün alabileceği minimum değer belirler
`Value`	Kontrolün verilen değer aralığındaki pozisyonunu belirler

VISUAL STUDIO.NET ve FORM UYGULAMALARI

- **Örnek: ProgressBar** bir sayım 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.

VISUAL STUDIO.NET ve FORM UYGULAMALARI

```
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 = ProgressBar1.Maximum - ProgressBar1.Minimum;
 int oran = (aralik - kalan) / aralik * 100; Label1.Text = oran + "% tamamlandı";
 ProgressBar1.Value = ProgressBar1.Maximum - kalan; kalan -= 1;
}
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(); }
}
```

- **Treeview Kontrolleri**

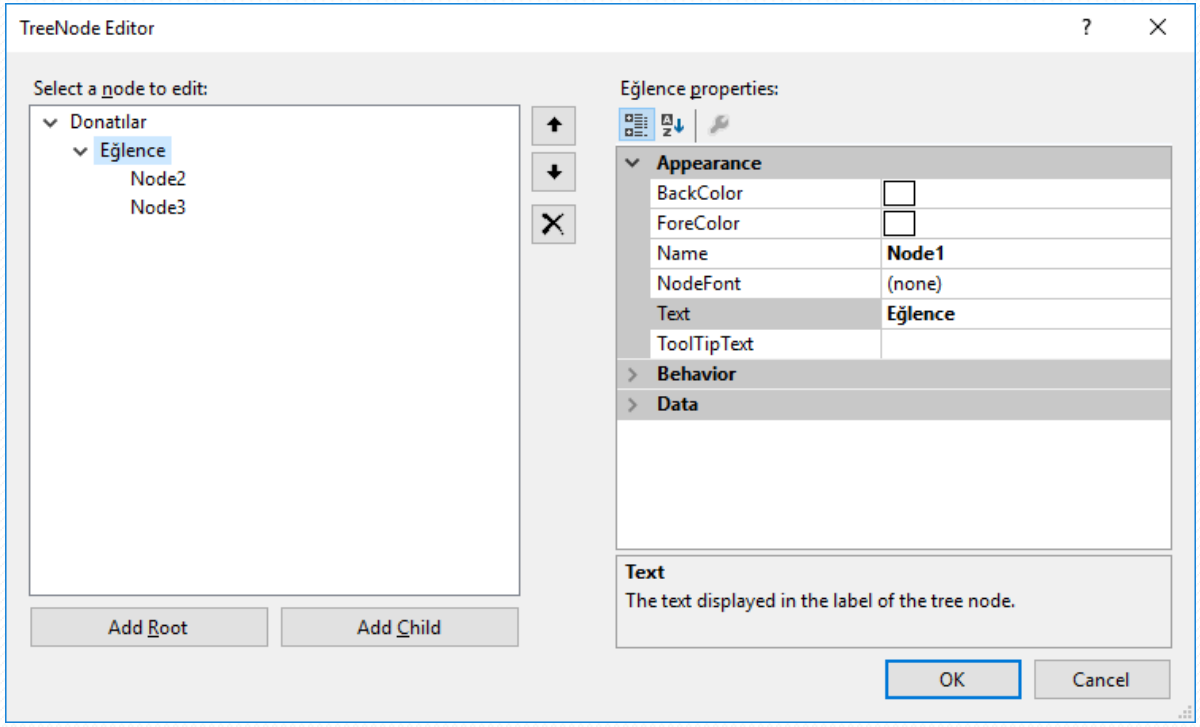
- Windows'un kullandığı ve ağaç yapısı olarak adlandırılan bir yapı içerir. Kullanıcıların bazı detayları daha net anlayabilmesi ve görebilmesi için oluşturulur.

- **Özellikleri:**
- **treeView1.Nodes:** treeView satır işlemleri için kullanılır.
- **treeView1.Nodes.Add():** Ağaç yapısına düğüm noktası eklemek için kullanılır. Daha sonra bu düğüm noktalarına alt bileşenler eklenir.

- Properties / Nodes (Collections) kısmından eklenebileceği gibi kodlarla ekleme yapılabilir.

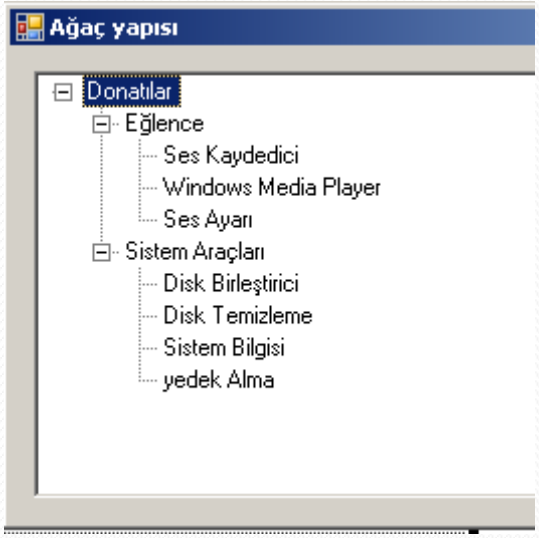
- **Treeview Kontrolleri**

- **AddRoot** ile kök düğüm eklenir. Kök düğümler birden fazla olabilir.
- **AddChild** ile seçilen düğümün alt düğümleri eklenir.



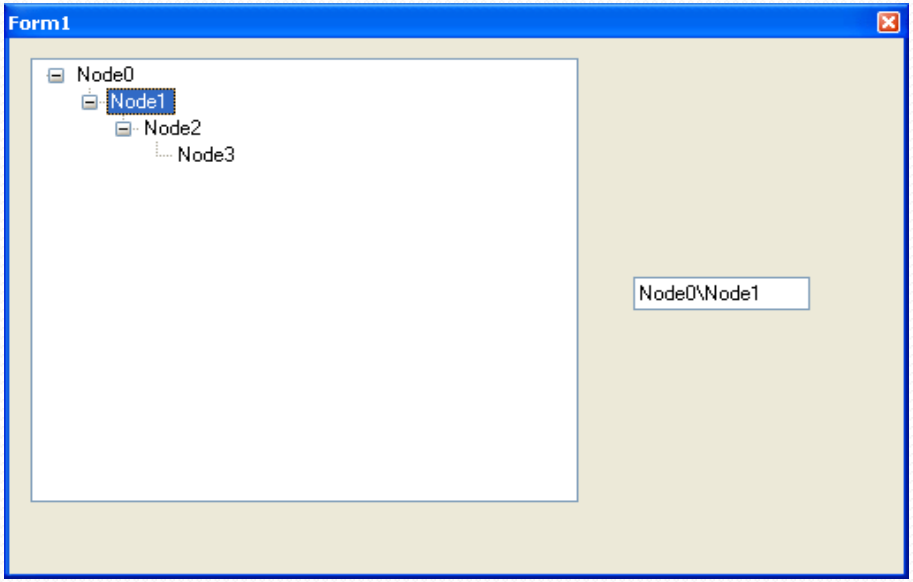
VISUAL STUDIO.NET ve FORM UYGULAMALARI

- private void Form1_Load(object sender, EventArgs e)
- {
- treeView1.Nodes.Add("Donatılar");
- treeView1.Nodes[0].Nodes.Add(new TreeNode("Eğlence"));
- treeView1.Nodes[0].Nodes[0].Nodes.Add(new TreeNode("Ses Kaydedici"));
- }



- **SelectedNode :** Bu özellik ile seçilen elaman ile ilgili işlemler yapılabilir.

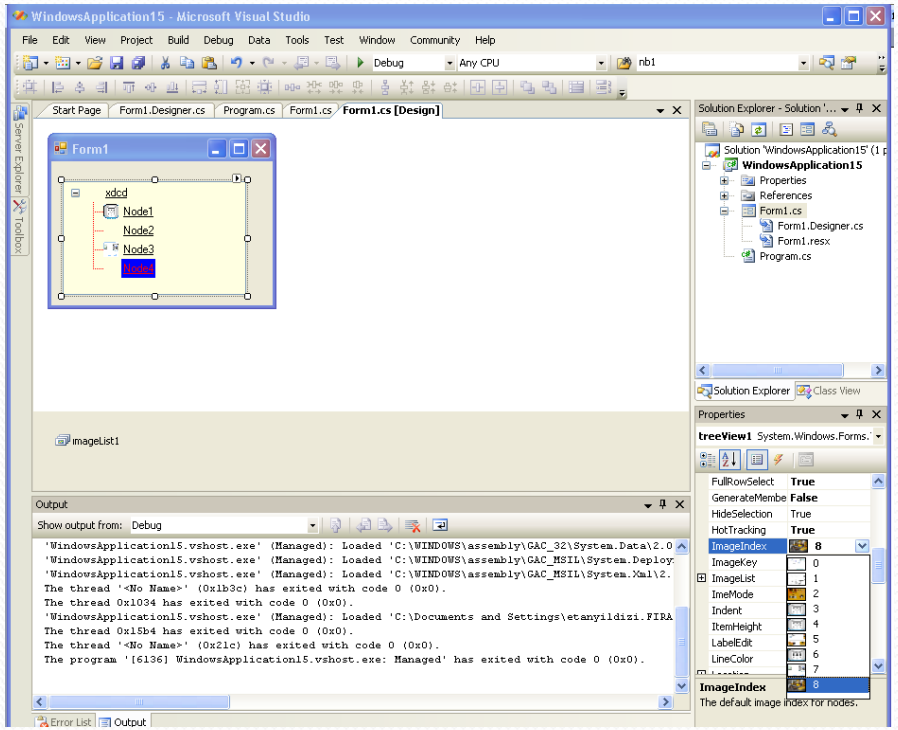
- private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
- { textBox1.Text = treeView1.SelectedNode.FullPath; }
- //SelectedNode.Text değerini
- //SelectedNode.Index indisini verir



VISUAL STUDIO.NET ve FORM UYGULAMALARI

- **treeView1.CheckBoxes:** true değeri aktarılırsa her satırın başına checkbox eklenir. Kutucukların işaretlenmesi ile aktif hale gelir.
- **private void Form1_Load(…)**
- **{**
- **treeView1.CheckBoxes = true;**
- **}**

- **ImageList** veya **Image** özelliği ile ağacın düğümlerine resim eklenebilir.

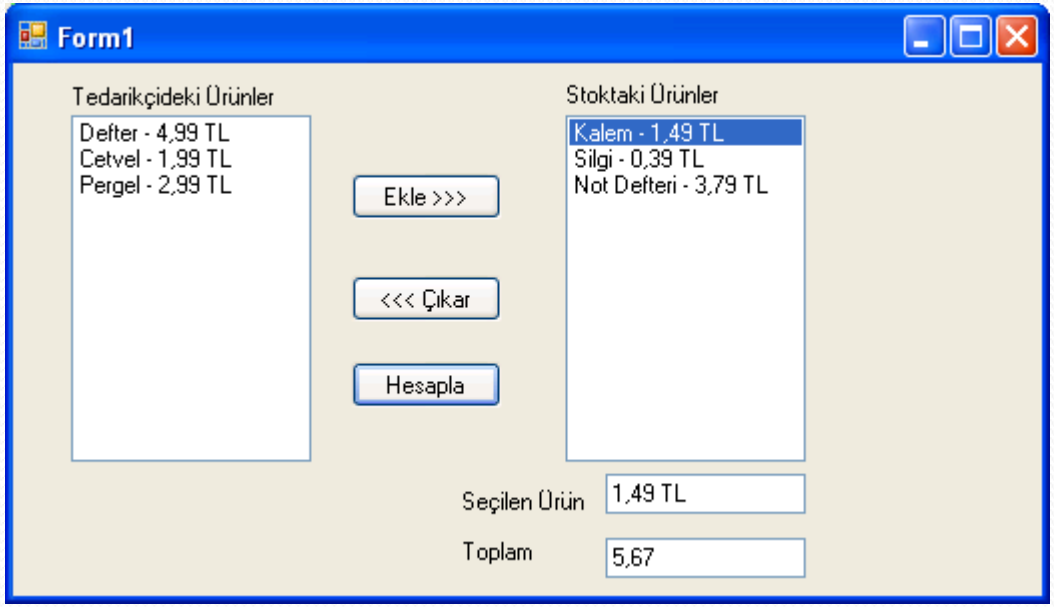


Visual Studio.Net -C#
14. HAFTA
Veritabanı Uygulamaları
ADO.NET

Örnekler

Örnek:

- Tedarikçiden alınacak ve stokta bulunan ürünleri listelemek ve alım satım işlemi yapmak için ListBox kontrolleri kullanma.



Örnek

- **Uygulama Adımları**
- **1-** Ürünlerin tutulması için bir Struct oluşturulur. Bu ürün yapısının ToString metodu tekrar yazılmıştır. Bunun nedeni, ListBox kontrolünde listelenen nesnelerin görüntülendiği değer ToString metodu çağırılarak belirlenir. Dolayısıyla liste kutularında istenen formatta değerin gözükmesini sağlamak için ToString metodunun tekrar yazılması gerekir.
-
- **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); }**
- **}**

Örnek

- **2-** Liste kutularının özellikleri ayarlanır ve içine eleman doldurulur.

```
private void Form1_Load(object sender, EventArgs e)
        {   listBox1.SelectionMode = SelectionMode.MultiExtended;
            listBox2.SelectionMode = SelectionMode.MultiExtended;
            UrunEkle();    }
        public void UrunEkle()   {
            Urun u = new Urun();
            u = new Urun("Kalem", 1.49);        listBox1.Items.Add(u);
            u = new Urun("Silgi", 0.39);        listBox1.Items.Add(u);
            u = new Urun("Defter", 4.99);        listBox1.Items.Add(u);
           u = new Urun("Cetvel", 1.99);        listBox1.Items.Add(u);
           u = new Urun("Pergel", 2.99);        listBox1.Items.Add(u);
           u = new Urun("Not Defteri", 3.79);    listBox1.Items.Add(u);
        }
```

Örnek

- **3-**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 button2_Click(object sender, EventArgs e)
        {   // Tedarikçiden alınan ürünler stok listesine eklenir
            foreach (object item in listBox1.SelectedItems)
            {       listBox2.Items.Add(item);       }
            // Stok listesine eklenen tüm ürünler  tedarikçi listesinden çıkartılır
            foreach (object item in listBox2.Items)
            {               listBox1.Items.Remove(item);          }
            button3.Enabled = true;
            button4.Enabled = true;
        }
```

Örnek:

- **4-**Stok listesinden öğe çıkarmak için, ekleme işlemine benzer kodlar çalıştırılır.

```
private void button3_Click(object sender, EventArgs e)
    {   // Tedarikçiden alınan ürünler stok listesine eklenir
        foreach (object item in listBox2.SelectedItems)
        {    listBox1.Items.Add(item);
             // Stok listesine eklenen tüm ürünler tedarikçi listesinden çıkartılır
        }
        foreach (object item in listBox1.Items )
        {            listBox2.Items.Remove(item);          }
        if ( listBox2.Items.Count == 0 )
        {   button3.Enabled = false;       button4.Enabled = false;          }
    }
```

Örnek

- **5-**Stoktaki toplam fiyatın hesaplanması işlemi, ürünlerin fiyatlarının alınıp toplanması ile gerçekleşir.

```
private void button4_Click(object sender, EventArgs e)
        {   double toplam = 0;
            for (int i=0; i<=listBox2.Items.Count - 1; i++ )
            {   Urun urun = (Urun)( listBox2.Items[i]) ;
                toplam += urun.Fiyat;
            }
            textBox2.Text = System.Convert.ToString( toplam );
        }
```

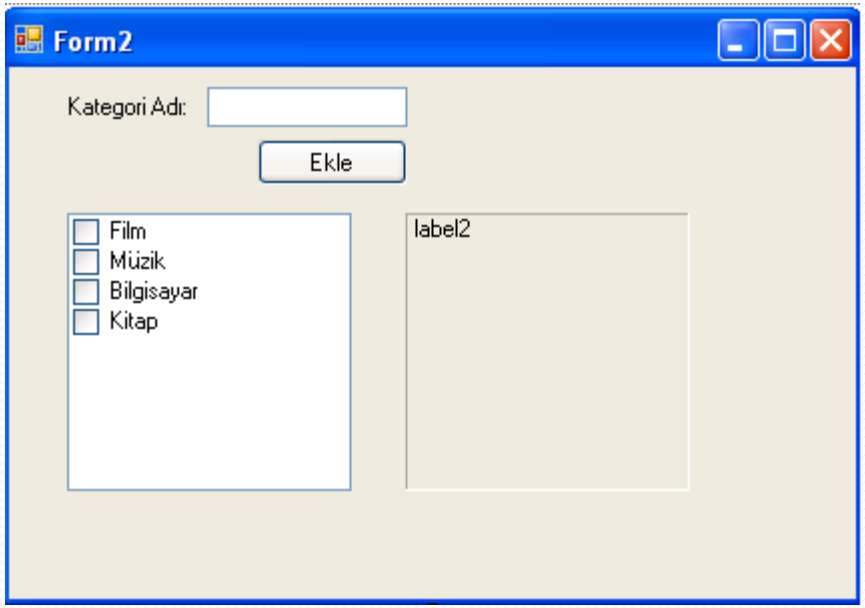
Örnek

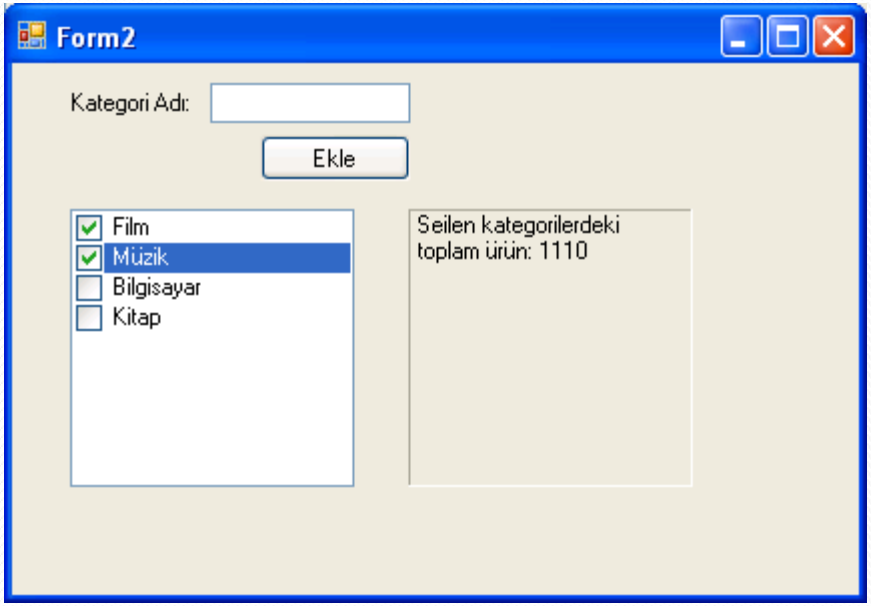
- **6-**Stok listesindeki bir öğenin seçildiği durumda, bu öğenin fiyatı görüntülenir.

```
private void listBox2_SelectedIndexChanged(object sender, EventArgs e)
        {
            Urun secilen = new Urun();
             secilen = ( (Urun )( listBox2.SelectedItem ) );
             textBox1.Text = string.Format( "{0:C}", secilen.Fiyat );
        }
    }
}
```

Örnek:

- Kategori başına stoktaki toplam ürünlerin gösterildiği bir uygulamada listelenen kategorileri seçmek için CheckedListBox kullanımı.



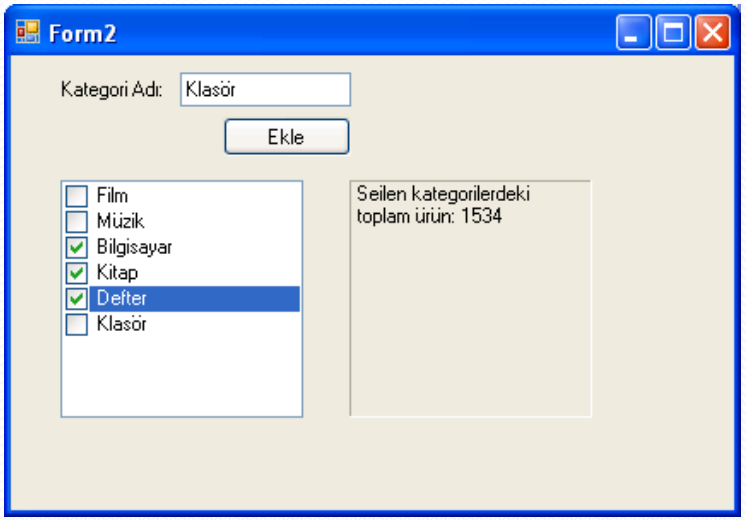


Örnek:

```
private void chlistKategoriler_SelectedIndexChanged(object sender, EventArgs e)
    {   int toplam=0;
        // Listedeki seçilen öğelerin ürün adeti toplanır.
        for (int i = 0; i <= chlistKategoriler.Items.Count - 1; i++)
        {  if (chlistKategoriler.GetItemChecked(i))
           {  object secilen = null;
              secilen = chlistKategoriler.Items[i];
              // Stok durumunu gsteren fonksiyon arlr
              toplam += StokDurumu(secilen.ToString());
           }
        }
       lblToplamUrun.Text = "Seilen kategorilerdeki toplam ürün: " + toplam;
    }
```

Örnek:

```
// Kategoriye göre, stoktaki ürünlerin belirlenmesi
    public int StokDurumu(string kategori
    {   switch (kategori)
      {  case "Film":   return 1100;
         case "Müzik":   return 982;
         case "Bilgisayar":    return 302;
         case "Kitap":    return 1222;
         default:   return 10;
       }
    }
//Ekleme İşlemi
    private void btnKategoriEkle_Click(object sender, EventArgs e)
    {       chlistKategoriler.Items.Add(txtKategoriAdi.Text);      }
```



Örnek

- Tarih ve sayı formatlarını, kullanıcının seçimine bırakarak bir sayı veya tarih yazdırma işlemi ComboBox kontrolleri ile yapılabilir.

Örnek

- **1-**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");
- }

Örnek

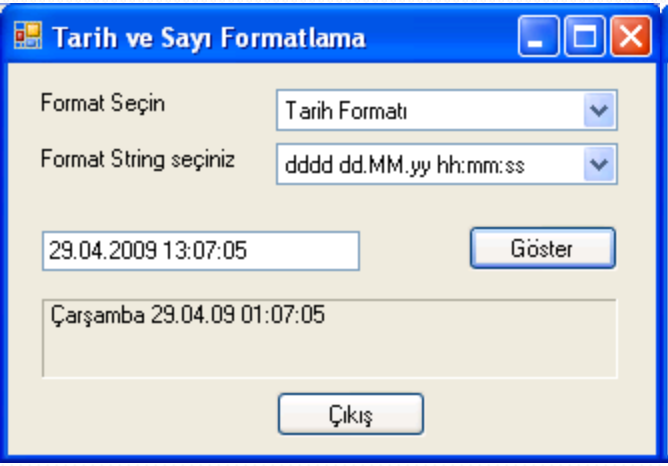
- **2-** Tarih ya da sayı formatlarından biri seçildiği zaman, ikinci ComboBox kontrolüne değişik format seçenekleri eklenir.

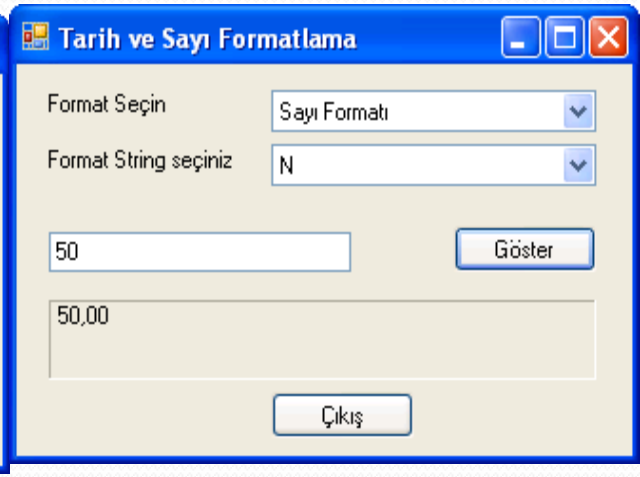
```
private void cmbFormat_SelectedIndexChanged (object sender, EventArgs e)
    {   cmbFormatString.Items.Clear();
        switch (cmbFormat.SelectedIndex)
        {   case 0:
                cmbFormatString.Items.Add("dd - MM - yyyy");
                cmbFormatString.Items.Add("yyyy*MM*dd hh:mm");
                cmbFormatString.Items.Add("dddd dd.MM.yy hh:mm:ss");    break;
            case 1:
                cmbFormatString.Items.Add("C");
                cmbFormatString.Items.Add("P");
                cmbFormatString.Items.Add("N");                         break;
        }
    }
```

Örnek

- **3-** Format seçildikten sonra metin kutusuna girilen değer alınır ve ilgili formatta gösterilir.
- private void btnGoster_Click(object sender, 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; }
- }

VISUAL STUDIO.NET ve FORM UYGULAMALARI





Visual Studio.Net -C#
14. HAFTA
Veritabanı Uygulamaları
ADO.NET

VERİTABANI UYGULAMALARI

- **Veri Merkezli Uygulamalar:**
  - Veri Depolama,
  - Bağlantılı (Connected) ve Bağlantısız (Disconnected) Veri Ortamları,
  - Veri Erişim Yöntemleri, olarak ele alınmaktadır.
- **Veri Depolama**
- Veriye erişmek için çeşitli veri depolama yöntemleri geliştirilmiştir.
  - Yapısal Olmayan Yöntem: Not defteri
  - Yapısal Yöntem: Excel
  - Hiyerarşik Yöntem: Xml
  - İlişkisel Veritabanı Yöntemi: Sql, Access,vs.
  - Nesne Yönelimli Veri Tabanı Yöntemi: Ado.Net

Veri Depolama

- **Yapısal Olmayan:** Bu yöntem ile depolanan veriler için belirli bir sınıflandırma ve sıralama yoktur. Veriler düz bir şekilde kaydedilir. Örneğin basit not dosyaları.
- **Yapısal:** Bu yöntem ile depolanan veriler çeşitli gruplara ayrılarak saklanır fakat bu gruplar arasında bir alt-üst ayrımı yapılmaz. Örneğin virgülle ayrılmış dosyalar (csv), Excel belgeleri.
- **Hiyerarşik:** Hiyerarşik depolama yöntemini ağaç yapısına benzetebiliriz. Bu yöntemde veriler çeşitli kategorilere bölünerek depolanır. Her bir kategorinin içerisinde alt kategorilerde olabilir. Örneğin XML (eXtensible Markup Language) dosyalar.
- **İlişkisel Veritabanı:** İlişkisel veritabanlarında veriler tablolar üzerinde depolanır. Tablo içerisindeki her bir satır kaydı, her bir sütun ise veriyi ifade eder. Örneğin SQL Server, Oracle, Access.
- **Nesne Yönelimli Veritabanı:** En gelişmiş veri depolama yöntemidir. Bu yöntemde veriler; ihtiyaca göre gruplandırılarak, nesneler içerisinde saklanır. Örneğin Versant, AOL.
- **ADO.NET bu depolama tekniklerinin tümünü destekler.**

Bağlantılı ve Bağlantısız Veri Ortamları

- **Bağlantılı (Connected) Veri Ortamları**

- Bağlantılı veri ortamları, uygulamaların veri kaynağına sürekli bağlı kaldığı ortamlardır. Bu ortamlarda veri alma ve değiştirme işlemleri uygulama ile veri kaynağı arasında bağlantı kurulduktan sonra gerçekleştirilir. Bağlantılı veri ortamlarında, veri işlemleri gerçekleştiği sürece bağlantı açık kalır.

- **Avantajları:**
  - En güvenli veri ortamı.
  - Erişimler eş zamanlı.

- **Dezavantajları:**
  - Sabit bir ağ bağlantısının olması gerekir.
  - Ağ trafiğinin yoğunluğunu artırır.

Bağlantılı ve Bağlantısız Veri Ortamları

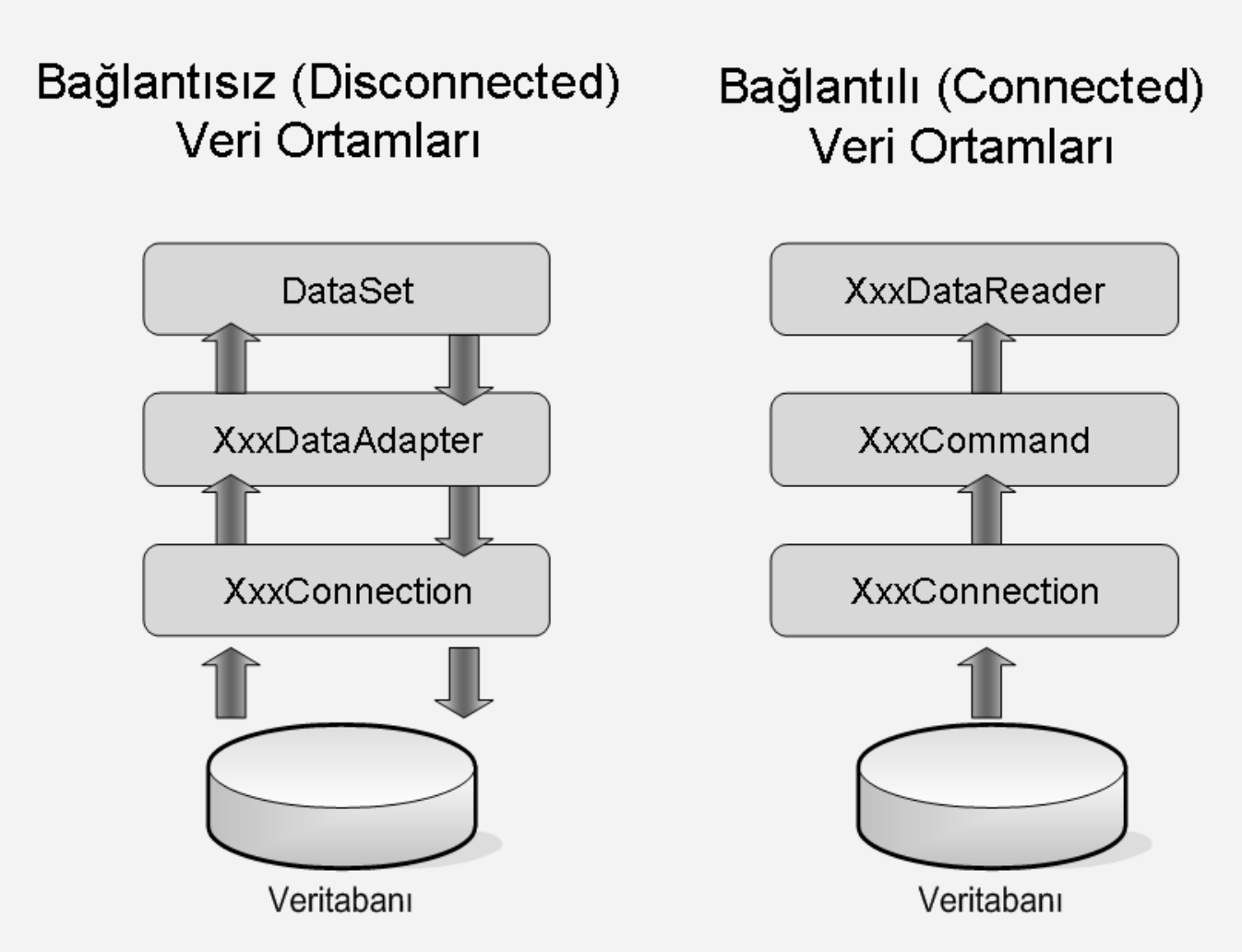
- **Bağlantısız (Disconnected) Veri Ortamları**

- Bağlantısız veri ortamı, uygulamanın veri kaynağına sürekli bağlı kalmadığı veri ortamıdır. Bağlantı, veri alış verişi yapılırken açılır, işlem bittikten sonra kapatılır.

- **Avantajları:**
  - Taşınabilen aygıtlarla (Laptop, Pocket PC) girilen veriler, istenilen zamanda veri ortamlarına aktarılabilir.
  - Uygulama performansını arttırır.

- **Dezavantajları:**
  - Verinin güncelliği sağlanmalıdır.
  - Veri çakışması önlenmelidir.

Bağlantılı ve Bağlantısız Veri Ortamları



ADO.NET

- **ADO (ActiveX Data Objects),** farklı veri kaynaklarına hızlı ve güvenli erişim için Microsoft tarafından geliştirilen nesne modelidir.

- **ADO.NET** ise ADO teknolojisinin en yeni versiyonudur. ADO ile aynı programlama modelini kullanmamakla birlikte, ADO modelinden gelen pek çok çözüm yolunu da beraberinde getirir.

- ADO.NET nesne modeli iki ana bölümden oluşmaktadır.
  - .NET Veri Sağlayıcı Sınıfları
  - DataSet Sınıfları

ADO.NET Nesne Modeli

- .NET Veri Sağlayıcı Sınıfları:
  - .NET veri sağlayıcı sınıfları, farklı türdeki veritabanlarına bağlanmak için kullanılır. Bu sınıflar sayesinde istenilen türdeki veri kaynağına kolayca bağlantı kurulabilir, veri çekilebilir ve gerekli güncelleme işlemleri yapılabilir. ADO.NET nesne modeli, aşağıdaki veri sağlayıcı sınıflarını içerir: SQL Server .NET Veri Sağlayıcısı, OLE DB .NET Veri Sağlayıcısı, Diğer .NET Veri Sağlayıcıları
- DataSet Sınıfları:
  - Çevrimdışı ortamlar için veri depolama ve yönetme işlemlerini sağlar. DataSet sınıfları veri kaynağından bağımsız her tür uygulama ve veritabanı için kullanılabilir. Özellikle İlişkisel Veritabanı, XML ve XML Web Servisleri üzerinden veri çekmek için kullanılır.

ADO.NET Veri Sağlayıcıları

- NET veri sağlayıcıları, **System.Data** isim alanı içinde tanımlanmıştır.
  - **SQL Server .NET:** SQL Server 7.0 ve SQL Server 2000 veritabanlarına hızlı bağlantı sağlar. SQL Server bağlantı nesneleri **System.Data.SqlClient** isim alanında bulunur.
  - **OLE DB .NET:** SQL Server 6.5 ve daha öncesi sürümlerine, Oracle, Sybase, DB2/400 ve Microsoft Access veri tabanlarına bağlantı kurmayı sağlar. OLE DB bağlantı nesneleri **System.Data.OleDb** isim alanında bulunur.
  - **ORACLE .NET:** Oracle veritabanlarına bağlantı için tasarlanmış veri sağlayıcısıdır. Oracle bağlantı nesneleri **System.Data.OracleClient** isim alanında bulunur.
  - **ODBC .NET:** Diğer veritabanlarını destekleyen genel bir veri sağlayıcıdır. ODBC bağlantı nesneleri **System.Data.ODBC** isim alanında bulunur.

ADO.NET Veri Sağlayıcıları

- Her bir veri sağlayıcısı içerisinde, birçok bağlantı nesnesi bulunur.
  - Connection: Bağlantı kurmak için kullanılır.
  - Command : Veritabanına sorgu yollamak için kullanılır.
  - DataReader : Çevrim içi bağlantı ile sadece veri okuma.
  - DataAdapter : Çevrim dışı bağlantılarda veri işleme nesnesi.

- Veri Kaynaklarına Bağlanmak
  - Veri Sağlayıcı Seçmek
  - Bağlantı oluşturmak
  - Bağlantı Yönetimini seçmek

Veri Sağlayıcı Seçmek

- Uygulama ile veritabanı arasında bağlantı kurmak ve kurulan bağlantı üzerinden kayıtları almak, değiştirmek ve silmek için veri sağlayıcılarını kullanır.

- Microsoft .NET Framework, veritabanları ile bağlantı kurmak için farklı veri sağlayıcılarını destekler.
  - SQL Server .NET
  - OLEDB .NET
  - ODBC .NET

Veri Sağlayıcı Seçmek

- **System.Data.SqlClient** isim alanı içerisinden çevrimiçi bağlantılar geliştirmek için SqlConnection, SqlCommand, SqlDataReader sınıfları kullanılır.

- **SqlConnection;** MS SQL Server üzerinde bağlantı açmak ve kapatmak için kullanılan sınıfdır.
- **SqlCommand;** MS SQL Server üzerinde Stored Procedure (Saklı Yordamlar) veya SQL Cümleleri çalıştırmak için kullanılan sınıftır.
- **SqlDataReader;** MS SQL Server üzerinde SqlCommand ile çalıştırılan SELECT sorguların sonuçlarını geri döndürmek için kullanılan sınıftır.

Veri Sağlayıcı Seçmek

- **System.Data.SqlClient isim alanı içerisinden <u>çevrimdışı</u> bağlantılar geliştirmek için SqlConnection, SqlDataAdapter, DataSet sınıfları kullanılır.**
-
- **SqlConnection;** MS SQL Server üzerinde bağlantı açmak ve kapatmak için kullanılan sınıftır.
- **SqlDataAdapter;** MS SQL Server'dan çekilen verileri DataSet içerisine ve DataSet'e çevrimdışı eklenmiş verileri MS SQL Server'a aktarmak için kullanılan sınıftır.
- **DataSet;** SQLDataAdapter nesnesinden gelen kayıtları çevrimdışı depolamak ve yönetmek için kullanılan sınıftır. DataSet tüm veri sağlayıcı sınıflar için ortaktır.

Veri Sağlayıcı Seçmek

- **System.Data.OleDb isim alanı içerisinden çevrimiçi bağlantılar geliştirmek için OleDbConnection, OleDbCommand, OleDbDataReader sınıfları kullanılır.**
-
- **OleDbConnection;** Access veya diğer veritabanları üzerinde bağlantı açmak ve kapatmak için kullanılan sınıftır.
- **OleDbCommand;** Access veya diğer veritabanları üzerinde Stored Procedure (Saklı Yordamlar) veya SQL Cümleleri çalıştırmak için kullanılan sınıftır.
- **OleDbDataReader;** Access veya diğer veritabanları üzerinde OleDbCommand ile çalıştırılan SELECT sorguların sonuçlarını geri döndürmek için kullanılan sınıftır.

Veri Sağlayıcı Seçmek

- **System.Data.OleDb isim alanı içerisinden <u>çevrimdışı</u> bağlantılar geliştirmek için OleDbConnection, OleDbDataAdapter, DataSet sınıfları kullanılır.**
-
- **OleDbConnection;** Access veya diğer veritabanları üzerinde bağlantı açmak ve kapatmak için kullanılan sınıftır.
- **OleDbDataAdapter;** Access veya diğer veritabanlarından çekilen verileri DataSet içerisine ve DataSet'e çevrimdışı eklenmiş verileri ilgili veritabanına aktarmak için kullanılan sınıftır.

Bağlantı Oluşturmak

- **Ms Access ile OLEDB Bağlantı Cümleleri**
- **Access’e Bağlantı:**
- "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=DB_Name.mdb;"
- **Access’e Çalışma Grubu dosyası üzerinden Bağlantı:**
- "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=Db_Name.mdb; Jet OLEDB:System Database=Db_Name.mdw"
- **Access’e Parola Korumalı Bağlantı:** "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=Db_Name.mdb; Jet OLEDB:Database Password=sifreniz"
- **Network’teki Access’e Bağlantı:** "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=\\Server_Name\Share_Name\Share_Path\Db_Name.mdb"
- **Remote Server(UzakServer) üzerindeki bir Access’e Bağlantı:** "Provider=MS Remote; Remote Server=http://Your-Remote-Server-IP; Remote Provider=Microsoft.Jet.OLEDB.4.0; Data Source=Db_Name.mdb"

Bağlantı Oluşturmak

- **Ms Access 2007 (ve sonrası) ile OLEDB Bağlantı Cümleleri**

- **Provider=Microsoft.ACE.OLEDB.12.0;Data Source=myDB.accdb;**
- **Persist Security Info=False;**

- **Database password**
- **Provider=Microsoft.ACE.OLEDB.12.0;Data Source=myDB.accdb;Jet OLEDB:**
  Database Password=MyDbPassword;

Bağlantı Oluşturmak

- **SQL Server ile ODBC Bağlantı Cümleleri**

- **SQL Server sunucusuna SQL Authentication (Kimlik Doğrulama) ile bağlanmak:** "Driver={SQL Server}; Server= Server_Name; Database=Db_Name; Uid=Username; Pwd= sifreniz;"

- **SQL Server sunucusuna Windows Authentication ile bağlanmak:**
- "Driver={SQL Server}; Server= Server_Name; Database=DB_Name; Trusted_Connection=yes;"

Bağlantı Oluşturmak

- **SQL Server ile Sql Server Bağlantı Cümleleri**

- **SQL Server sunucusuna Authentication ile bağlanmak**
- "Data Source=_Server_Name; Initial Catalog=Db _Name; User SQL Id= Username; Password=sifreniz;"
- **SQL Server sunucusuna SQL Authentication ile bağlanmak**
- "Server= Server_Name;Database=Db_Name;User ID= Username; Password=sifreniz;Trusted_Connection=False”
- **SQL Server sunucusuna Windows Authentication ile bağlanmak**
- "Data Source= Server_Name;Initial Catalog=Db_Name;Integrated Security=SSPI;”
- **SQL Server sunucusuna SQL Authentication ile bağlanmak**
- "Server=Server_Name;Database=Db_Name;Trusted_Connection=True;"

Bağlantı Yönetimi

- Bağlantı cümlesini oluşturduktan sonra, bağlantıyı açmak ve kapamak için **Connection** sınıfının iki önemli metodu kullanılır.

- **Open ()**
- **Close ()**
- (Bağlantı nesnesinin **Dispose ()** metodu da bağlantıyı kapatmak için kullanılabilir.)

Bağlantı Yönetimi

```
 System.Data.OleDb.OleDbConnection cnNorthwind;
try
{ cnNorthwind = new System.Data.OleDb.OleDbConnection();
  cnNorthwind.ConnectionString = @"Provider=Microsoft.Jet.  OLEDB.4.0;Data Source=Northwind.mdb";
  cnNorthwind.Open();// Veritabanı işlemleri gerçekleştirilir.
}
catch (InvalidOperationException XcpInvOp)
{  // İlk önce bu tipte hata yakalanır.
   MessageBox.Show("Önce veri tabanı bağlantısını kapatın");
 //Hata Mesajının içeriğini görmek için kullanılır.
   MessageBox.Show(XcpInvOp.ToString()); }
catch (Exception Xcp)  { //Diğer hatadan farklı bir tipte hata burada yakalanır.
    MessageBox.Show(Xcp.ToString()); }
finally {   cnNorthwind.Close();  //ya da
            cnNorthwind.Dispose();
        }
```

Bağlantı Durumlarını Kontrol Etmek

- Bağlantı sınıfının durumu hakkında bilgi almak için, bağlantı sınıfının **State** özelliği kullanılır.

İsim	Açıklama	Değeri
Broken	Yalnızca, açık bir bağlantının kopup tekrar bağlanıldığı durum	16
Closed	Bağlantı kapalı	0
Connecting	Veri kaynağına bağlanma aşamasında	2
Executing	Bağlantı üzerinden bir komutu çalıştırılıyor	4
Fetching	Bağlantı üzerinden veri çekiliyor	8
Open	Bağlantı açık	1

```
private void BaglantiAc(System.Data.OleDb.OleDbConnection con)
{    //Connection, sadece kapalı ise açılacak
     if (con.State == ConnectionState.Closed)
         con.Open();
}
```

Bağlantılı (Connected) Veritabanı İşlemleri

- Bağlantılı veri ortamları ile veritabanı üzerinde, gerekli tüm veritabanı işlemleri yapılabilir (Veritabanından tek değer çekme, Sadece okunabilir kayıt kümeleri döndürme, Kayıt ekleme, Kayıt silme, Kayıt güncelleme)

- **Command ile Çalışmak**
- Command, veritabanı üzerinde Stored Procedure (Saklı Yordam) ve Sorgu çalıştırmak için kullanılır. Command Nesneleri ile veritabanı tablolarında; sorgu, ekleme, silme ve güncelleme işlemleri yapılabilir. Command Nesnelerinin özellikleri aşağıda belirtilmiştir.
- **Name:** Command nesnesinin kod içerisindeki ismidir.
- **Connection:** Command nesnesinin hangi Connection üzerinde çalışacağını belirler.
- **CommandType:** Çalıştırılacak komutun türünü belirtir. Text, Stored Procedure ve TableDirect olmak üzere üç değeri vardır. TableDirect ile tablonun tamamı elde edilmekte. SQL Server tarafından desteklenmez.
- **CommandText:** Stored Procedure adını veya Sorgu cümlesini tutar.
- **Parameters:** İsteğe bağlı parametrelerin kullanımı

Bağlantılı (Connected) Veritabanı İşlemleri

- **CommandType**

Komut Türü	Örnek
Text (Varsayılan)	String select = “SELECT ContactName FROM Customers”; SqlCommand cmd = new SqlCommand(select,conn)
StoredProcedure	SqlCommand cmd = new SqlCommand (“CustOrderHist”,conn); cmd.CommandType=CommandType.StoredProcedure; cmd.Parameters.AddWithValue(“@CustomerID,”QUICK”);

TableDirect	OleDbCommand cmd=new OleDbCommand(“Categories”,conn);
	cmd.Commandtype=CommandType.TableDirect
	//Tablonun tamamı elde edilmekte
	//Sadece OleDb’de kullanılabilmektedir.

Bağlantılı (Connected) Veritabanı İşlemleri

- **Command ile Çalışmak**
- Command, kod içerisinden veya ToolBox üzerinden oluşturulabilir. Bu yöntemler ile kullanılan veritabanına göre, OleDbCommand veya SqlCommand nesneleri oluşturulur.

```
using System.Data.OleDb;
 //Access Veritabanına bağlanmak için Command tanımlanır.
OleDbCommand cmd = new OleDbCommand();

//Command Sınıfının CommandText özelliğine universiteler tablosu
cmd.CommandText = "select * from universiteler";

//Command Sınıfının Connection özelliğine aktif connection aktarılır
cmd.Connection = conn;

//Command Sınıfına, sorgu cümlesi yazılacağını belirler.
cmd.CommandType = CommandType.Text;
```

Bağlantılı (Connected) Veritabanı İşlemleri

- **Command ile Çalışmak**

- Command özelliklerine değer girildikten sonra, Command'ı çalıştırmak için Command sınıfı metotlarından uygun olan seçilir.

- **ExecuteScalar:** Çalıştırılan Command nesnesinden geriye tek değer döndürmek için kullanılır.

- **ExecuteReader:** Çalıştırılan Command nesnesinden geriye kayıt kümesi döndürmek için kullanılır.

- **ExecuteNonQuery:** Command Nesnesi üzerinde veri güncelleme, değiştirme ve silme işlemleri yapmak için kullanılır. Bu işlemin sonucunda etkilenen kayıt sayısı geriye döndürür.

- **ExecuteXmlReader:** Çalıştırılan Command Nesnesinden geriye XML döndürmek için kullanılır. Sadece SQL Server 7.0 ve sonraki versiyonları için kullanılır.

Bağlantılı (Connected) Veritabanı İşlemleri

- **Command ile Geriye Değer Döndürmek**

- OleDbCommand veya SqlCommand nesnesi ile geriye değer döndürmek için, **ExecuteScalar** metodu kullanılır. ExecuteScalar metodu ile geriye değer döndürmek için, **Sum, Min, Max** veya **Count** gibi fonksiyonların yanında Select cümlesi veya Stored Procedure ile geriye tek değer döndürülebilir.

- Örnekte OleDbCommand nesnesinin ExecuteScalar metodu ile Universiteler tablosundaki toplam kayıt sayısı geri döndürülmektedir.

```
using System.Data.OleDb;
OleDbConnection conn = new OleDbConnection(@"provider = Microsoft.JET.OLEDB.4.0; Data source=..\universiteler.mdb");
OleDbCommand cmd = new System.Data.OleDb.OleDbCommand("select count(*) from universiteler", conn);
 conn.Open();
MessageBox.Show("" + cmd.ExecuteScalar().ToString());
```

Bağlantılı (Connected) Veritabanı İşlemleri

- **Command ile Geriye Değer Döndürmek**

- **Örnek:** Ürün Tablosundaki Stok Miktarı SqlCommand nesnesi ile geriye döndürülmektedir.

```
using System.Data.SqlClient;
string sql ="SELECT StokMiktari FROM Urun WHERE UrunID="+ "@UrunID";
SqlConnection cnAlisveris = new SqlConnection("Driver={SQL Server}; Server= Server_Name; Database=Db_Name; Uid=Username; Pwd= sifreniz;");
SqlCommand cmUrun = new SqlCommand (sql, cnAlisveris);
SqlParameter prmID = new SqlParameter("@UrunID", System.Data.SqlDbTypeDbType.Int, 4);
  cmUrun.Parameters["@UrunID"].Value = 42;
  cnAlisveris.Open();
  int adet = Convert.ToInt32(cmUrun.ExecuteScalar());
  cnAlisveris.Close();  MessageBox.Show("Quantity in stock: " + adet.ToString());
```

Bağlantılı (Connected) Veritabanı İşlemleri

- **Command ile Geriye Değer Döndürmek**

- OleDbCommand veya SqlCommand nesnesi ile geriye kayıt döndürmek için, **ExecuteReader** metodu kullanılır. ExecuteReader ile dönen kayıtlar DataReader nesnesine aktarılır.

- Örnekte Ürün Tablosundaki tüm ürünler, OleDbDataReader ile form üzerindeki
- ListBox kontrolüne eklenir.

```
using System.Data.OleDb;
OleDbCommand cmUrun = new OleDbCommand ("SELECT UrunAdi, StokMiktari FROM Urun", cnAlisveris);
  cnAlisveris.Open();
  OleDbDataReader rdrUrun = cmUrun.ExecuteReader();
while (rdrUrun.Read())
{
  listBox1.Items.Add(rdrUrun.GetString(0)+" - "+  rdrUrun.GetInt16(1));
}
rdrUrun.Close();
```

Bağlantılı (Connected) Veritabanı İşlemleri

- **Command ile Kayıt Döndürmeyen Sorgular Çalıştırmak**

- Command ile veritabanı yapısında değişiklik yapılabilir (Tablo, View ve Stored Procedure oluşturmak, değiştirmek ve silmek), güvenlik seçenekleri ayarlanabilir (Tablo ve View izinleri) ve veritabanı içerisindeki veri değiştirilebilir (Kayıt ekleme, silme ve güncelleme). OleDbCommand veya SqlCommand nesnesi ile bu tür işlemlerin yapılabilmesi için, **ExecuteNonQuery** metodu kullanılır.

- ExecuteNonQuery metodu ile **INSERT**, **UPDATE** ve **DELETE** sorguları çalıştırılabilir.

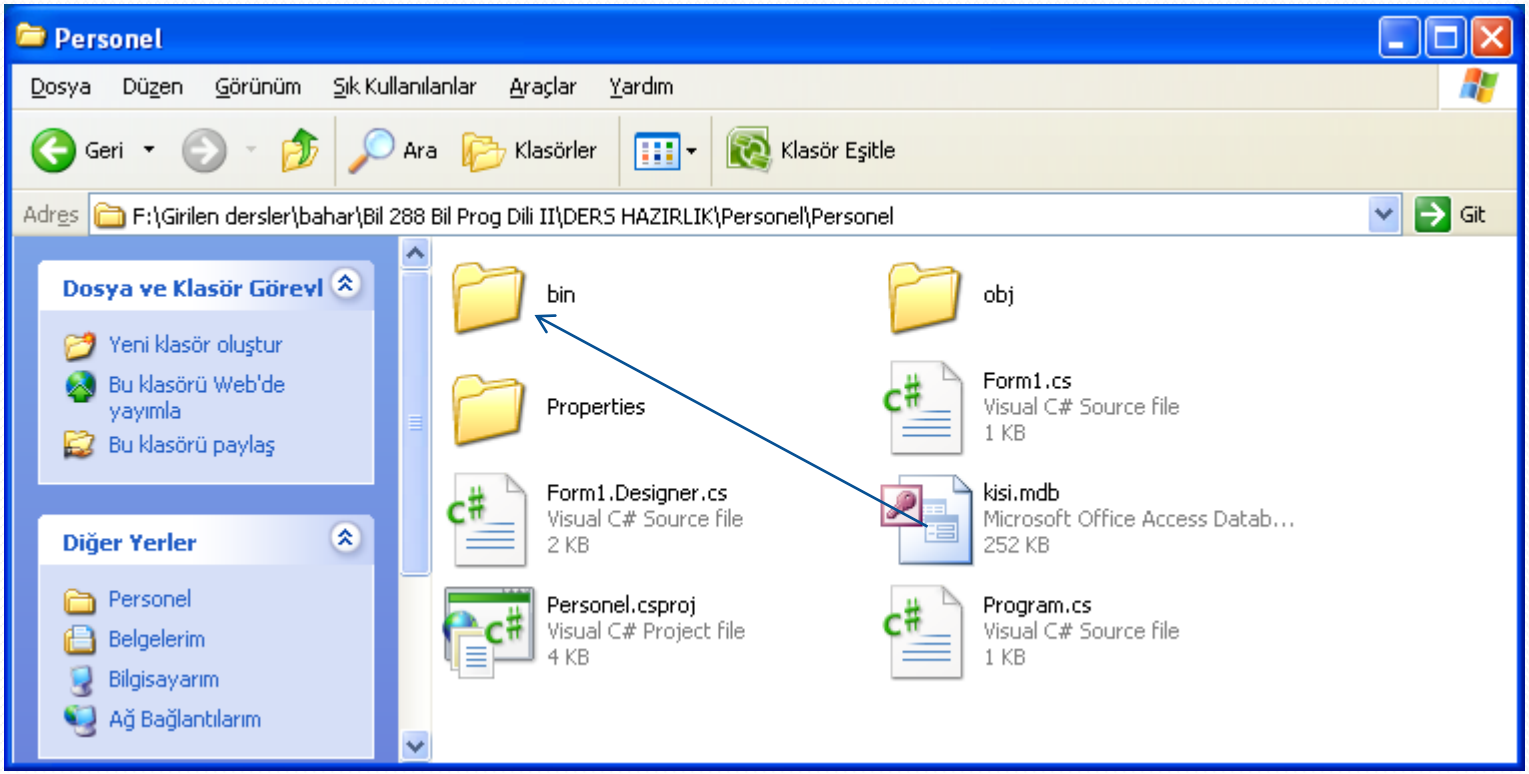
Örnek

- **Veritabanının oluşturulması**
- Bu uygulamada kullanılacak Personel tablosu için bir veritabanı oluşturulması gerekir.
- 1. Microsoft Access ile “kisi.mdb” isminde bir veritabanı oluşturun.
- 2. Veritabanına Personel isminde bir tablo ekleyin ve tabloda belirtilen kolonları ekleyin. (Dikkat tablo ve alan adlarının tamamı büyük harf olmayacak bağlantıda hata verebilir.)

Personel

Alan Adı	Veri Türü
Numara	Otomatik Sayı
Ad	Metin
Soyad	Metin
DogumTarihi	Tarih/Saat
Adres	Metin
Sehir	Metin

- **Kontrollerin eklenmesi**
- Personel isminde yeni bir Windows projesi açın. Veritabanını bin/debug klasörüne kopyalayın



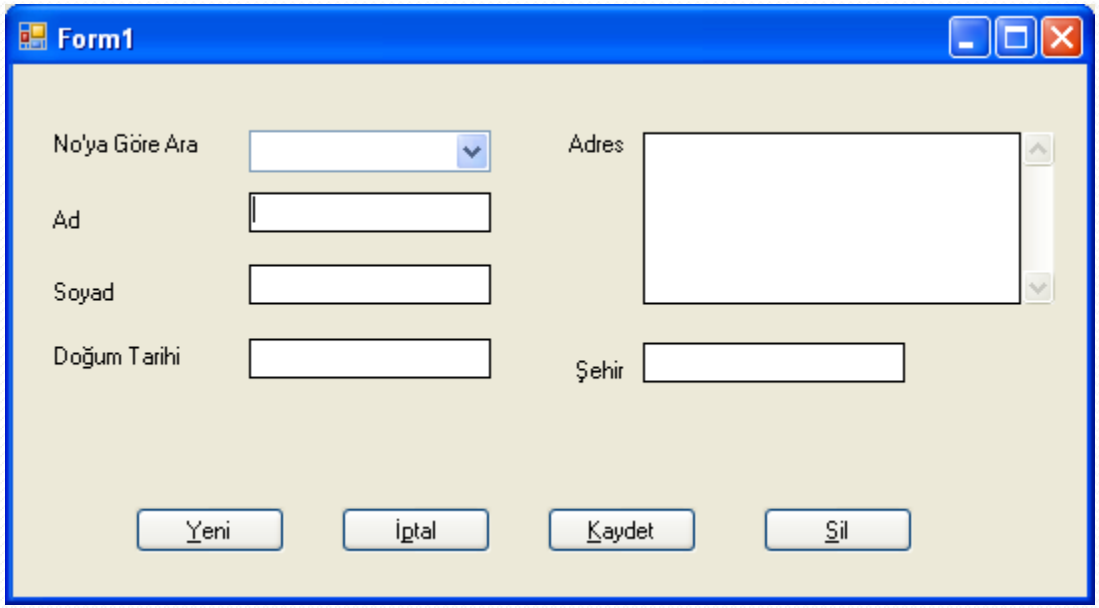
Örnek

- **Kontrollerin eklenmesi**
- Personel isminde yeni bir Windows projesi açın. Aşağıdaki özellikleri girin.

Kontrol – Kontrol İsmi	Özellik	Değer
TextBox – txtAd	BorderStyle	FixedSingle
TextBox – txtSoyad	BorderStyle	FixedSingle
TextBox – txtDTarihi	BorderStyle	FixedSingle
TextBox – txtSehir	BorderStyle	FixedSingle
TextBox – txtAdres	BorderStyle	FixedSingle
	Multiline	True
	ScrollBars	Vertical
ComboBox – cbNo	DropDownStyle	DropDownList
Button – btnYeni	Text	Yeni
Button – btnIptal	Text	İptal
Button – btnKaydet	Text	Kaydet
Button – btnSil	Text	Sil

Örnek

- **Kontrollerin eklenmesi**



Örnek

- **Kodların yazılması**
- Personel tablosu üzerinde işlem yapılması için veritabanına bağlantı açılması gerekir. Bu bağlantı için gereken Connection String ifadesinin merkezi bir yerden alınması, değişiklik durumunda kolaylık sağlayacaktır. Veritabanı işlemleri son olarak debug klasöründe oluşturulacaktır.

```
using System.Data.OleDb;

namespace Personel
{
    public partial class Form1 : Form
    {
        OleDbConnection conn= new OleDbConnection ("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=kisi.mdb");
```

- `using System.Data.OleDb;`
- `namespace Personel`
- `{public partial class Form1 : Form`
- `{ OleDbConnection conn= new OleDbConnection ("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=kisi.mdb");`
- `...`

Örnek

- public void Kaydet()
- {
- OleDbCommand kaydet = new OleDbCommand("**INSERT INTO** Personel(Ad,Soyad,DogumTarihi,Adres,Sehir) **VALUES** (' "
- + txtAd.Text + " ',' " + txtSoyad.Text + " ',' " + txtDTarihi.Text + " ',' " + txtAdres.Text + " ',' " + txtSehir.Text + " ')", conn);
- //Tırnaklara dikkat
- //değerler string ise **' "+…+" '** arasına numeric veya int ise **"+…+"** arasında yazılmalı
- // yoksa veri türü uyuşmazlığı ortaya çıkar
- try {
- conn.Open();
- kaydet.ExecuteNonQuery(); }
- catch (Exception ex) { MessageBox.Show(ex.Message); }
- finally
- { conn.Close(); }
- }

Örnek

```
public void Sil(int ID)     {
OleDbCommand sil = new OleDbCommand("DELETE  FROM Personel WHERE Numara=" + ID + "", conn);
  try
  {
   conn.Open();
   sil.ExecuteNonQuery();
  }
  catch(Exception ex)   {     MessageBox.Show(ex.Message);    }
  finally   {    conn.Close();   }
 }
 public void Temizle()
    {   txtAd.Text = ""; txtSoyad.Text = "";
        txtAdres.Text = ""; txtSehir.Text = ""; txtDTarihi.Text = "";
        txtAd.Focus();
    }
```

Örnek

```
public void IDDoldur()    {
        cbNo.Items.Clear();
        OleDbCommand veri = new OleDbCommand("SELECT Numara FROM Personel ORDER BY Numara", conn);
        OleDbDataReader oku = null;
        try   {
            conn.Open();
            oku=veri.ExecuteReader();
            while(oku.Read())
            { cbNo.Items.Add(oku.GetInt32(0));        }
        }
        catch (Exception ex)       {       MessageBox.Show(ex.Message);       }
        finally       {           oku.Close();           conn.Close();       }
    }
```

Örnek

```
public void IDyeGoreFormDoldur(int ID)    {
        OleDbCommand veri = new OleDbCommand("SELECT Numara, Ad, Soyad, DogumTarihi, Adres, Sehir FROM Personel where Numara="+ID+"", conn);
        OleDbDataReader oku=null;
        try    {
            conn.Open();        oku=veri.ExecuteReader();
            if (oku.Read())        {
                txtAd.Text = oku["Ad"].ToString();
                txtSoyad.Text = oku["Soyad"].ToString();
                txtAdres.Text = oku["Adres"].ToString();
                txtSehir.Text = oku["Sehir"].ToString();
                txtDTarihi.Text = oku["DogumTarihi"].ToString();        }
        }
        catch (Exception ex)        {        MessageBox.Show(ex.Message);        }
        finally        {        oku.Close();        conn.Close();        }
    }
```

Örnek

```
public bool Kontrol()    {
        if (txtAd.Text == "")
        {   MessageBox.Show("Adı Giriniz");
            txtAd.Focus();        return false;
        }
        else if (txtSoyad.Text == "")
        {  MessageBox.Show("Soyadı Giriniz");
            txtSoyad.Focus();
            return false;
        }
        else if (txtDTarihi.Text== "")
        {
 MessageBox.Show("Doğum Tarihini Giriniz");
            txtDTarihi.Focus();
            return false;
        }
else if (txtAdres.Text == "")
        {
MessageBox.Show("Adresi Giriniz");
            txtAdres.Focus();
            return false;
        }
        else if (txtSehir.Text == "")
        {
MessageBox.Show("Şehiri Giriniz");
            txtSehir.Focus();
            return false;
        }
        else      {      return true;        }
    }
```

Örnek

```
private void btnKaydet_Click(object sender, EventArgs e)
        {     if(Kontrol() == true)
              {
                  Kaydet();
                  btnYeni.Enabled = true;
                  btnKaydet.Enabled = false;
                  btnIptal.Enabled = false;
                  IDDoldur();
                  cbNo.SelectedIndex = cbNo.Items.Count - 1;
              }
        }
```

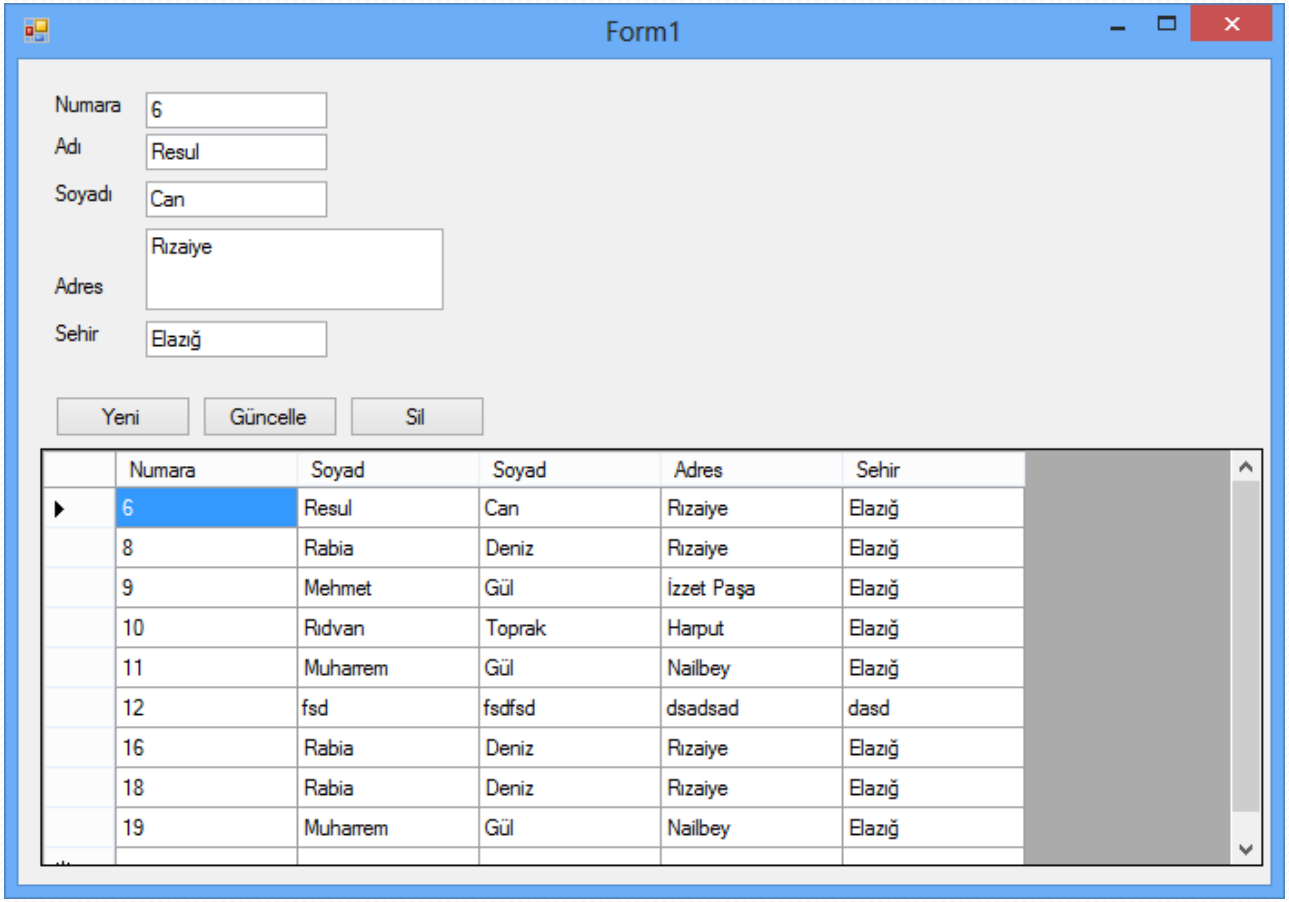
```
private void btnYeni_Click(object sender, EventArgs e)       {
            Temizle();
            btnYeni.Enabled = false;
            btnKaydet.Enabled = true;
            btnIptal.Enabled = true;
            cbNo.SelectedIndex = -1;        }
 private void btnIptal_Click(object sender, EventArgs e)       {
            Temizle();
            btnYeni.Enabled = true;
            btnKaydet.Enabled = false;
            btnIptal.Enabled = false;
            cbNo.SelectedIndex = 0;        }
```

Örnek

```
private void btnSil_Click(object sender, EventArgs e)
 {
 if(MessageBox.Show(cbNo.SelectedItem + " nolu kaydı silmek istiyor musunuz?",
this.Text, MessageBoxButtons.YesNo, MessageBoxIcon.Question,
MessageBoxDefaultButton.Button2) == DialogResult.Yes)
    {
        Sil(Convert.ToInt32(cbNo.SelectedItem));
        IDDoldur();
        cbNo.SelectedIndex = cbNo.Items.Count - 1;
    }
}
private void Form1_Load(object sender, EventArgs e)
  {        IDDoldur();            cbNo.SelectedIndex = 0;        }

private void cbNo_SelectedIndexChanged(object sender, EventArgs e)
   {       IDyeGoreFormDoldur(Convert.ToInt32(cbNo.SelectedItem));        }
```

Örnek

- Veri tabanından alınan değerlerin dataGridView üzerinde gösterilmesi



Örnek

```
public partial class Form1 : Form
{   OleDbConnection conn = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=db/kisi.mdb");

public void Form1_Load(object sender, EventArgs e)
        { //Form üzerinde Add Column ile eklenebilir veya çalışma zamanında oluşturulabilir.
            dataGridView1.Columns.Add("Column_No", "Numara");
                                    //(Column name, HeaderText)
            dataGridView1.Columns.Add("Column_Ad", "Ad");
            dataGridView1.Columns.Add("Column_Soyad", "Soyad");
            dataGridView1.Columns.Add("Column_Adres", "Adres");
            dataGridView1.Columns.Add("Column_Sehir", "Sehir");

            veri_al();
}
```

Örnek

```
public void veri_al()
{   OleDbCommand veri = new OleDbCommand("SELECT Numara, Ad, Soyad, Adres, Sehir FROM Personel ORDER BY Numara", conn);
 OleDbDataReader oku = null;
 try
 { conn.Open();
   oku = veri.ExecuteReader();
    while (oku.Read())
        {
        dataGridView1.Rows.Add(oku["Numara"].ToString(), oku["Ad"].ToString(), oku["Soyad"].ToString(), oku["Adres"].ToString(), oku["Sehir"].ToString());
         }
  }
  catch (Exception ex)   {         MessageBox.Show(ex.Message);      }
  finally      {       oku.Close();     conn.Close();    }
}
```

Örnek

```
private void dataGridView1_SelectionChanged(object sender, EventArgs e)
{//   Data gridden seçilen bilgileri textbox içerisine yazar
 txtNo.Text = dataGridView1.CurrentRow.Cells[0].Value.ToString();
 txtAd.Text = dataGridView1.CurrentRow.Cells[1].Value.ToString();
 txtSoyad.Text = dataGridView1.CurrentRow.Cells[2].Value.ToString();
 txtAdres.Text = dataGridView1.CurrentRow.Cells[3].Value.ToString();
 txtSehir.Text = dataGridView1.CurrentRow.Cells[4].Value.ToString();
}

private void btnYeni_Click(object sender, EventArgs e)
{//Yeni kayıt ekleme için metin kutularının boş yapılması
  txtNo.Text = "";
  txtAd.Text = "";
  txtSoyad.Text = "";
  txtAdres.Text = "";
  txtSehir.Text = "";
  txtNo.Visible = false;
  btnYeni.Visible = false;
}
```

Örnek

```
private void btnEkle_Click(object sender, EventArgs e)
{ // Kayıt ekleme
  if ((txtAd.Text != "") && (txtSoyad.Text != "") && (txtAdres.Text != "") && (txtSehir.Text != ""))
  {    OleDbCommand kaydet = new OleDbCommand("INSERT INTO Personel(Ad, Soyad,Adres,Sehir)  VALUES ('" + txtAd.Text + "','" + txtSoyad.Text + "','" + txtAdres.Text + "','" + txtSehir.Text + "')", conn);
     try        {  conn.Open();   kaydet.ExecuteNonQuery(); }
     catch (Exception ex) { MessageBox.Show(ex.Message); }
     finally {
         dataGridView1.Rows.Clear();
         conn.Close(); btnYeni.Visible = true; txtNo.Visible = true;
         veri_al();
          }
     }
 }
```

Örnek

```
private void btnGuncel_Click(object sender, EventArgs e)
{ int ID = Convert.ToInt16(txtNo.Text);
string sorgu = "UPDATE Personel SET Ad='" + txtAd.Text + "',
Soyad='" + txtSoyad.Text + "', Adres='" + txtAdres.Text + "',
Sehir='" + txtSehir.Text + "' WHERE Numara=" + ID ;
 if ((textBox2.Text != "") && (textBox3.Text != "") &&
(textBox4.Text != "") && (textBox5.Text != ""))
  {OleDbCommand guncelle = new OleDbCommand(sorgu, conn);
  try   {conn.Open(); guncelle.ExecuteNonQuery(); }
  catch (Exception ex) { MessageBox.Show(ex.Message); }
  finally
  { dataGridView1.Rows.Clear(); conn.Close();
    btnYeni.Visible = true; txtNo.Visible = true;
    veri_al();
  }
 }
}
```

Örnek

```
private void btnSil_Click(object sender, EventArgs e)
{ if (txtNo.Text != "")
  { int ID= Convert.ToInt32(txtNo.Text);
    OleDbCommand sil = new OleDbCommand("DELETE  FROM Personel WHERE Numara=" + ID + "", conn);
    try {
         conn.Open();
         sil.ExecuteNonQuery();
        }
    catch (Exception ex) { MessageBox.Show(ex.Message); }
    finally { conn.Close();
              dataGridView1.Rows.Clear();
              veri_al();
             }
 }
}
```

dataGridView özellikleri

- Seçilen satırın indis numarası
  - **int index =dataGridView1.CurrentRow.Index;**
- Seçilen satırın hücre değerini
  - **dataGridView1.CurrentRow.Cells[index].Value.ToString()**
  - (index değeri =0 ise seçilen satırın ilk sütün değeri alınır)
- Satır ve sütun sayılarının alınması
  - **int satir = dataGridView1.RowCount;**
  - **int sutun = dataGridView1.DisplayedColumnCount(true);**
- dataGridView içindeki tüm satırların silinmesi
  - **dataGridView1.Rows.Clear();**
- dataGridView içindeki verilen indisteki satırın silinmesi
  - **this.dataGridView1.Rows.RemoveAt(index);**

Bağlantısız (Disconnected) Veritabanı İşlemleri

- Bağlantısız veri ortamları, uygulamaların veri tabanından bağımsız çalıştığı ortamlardır.

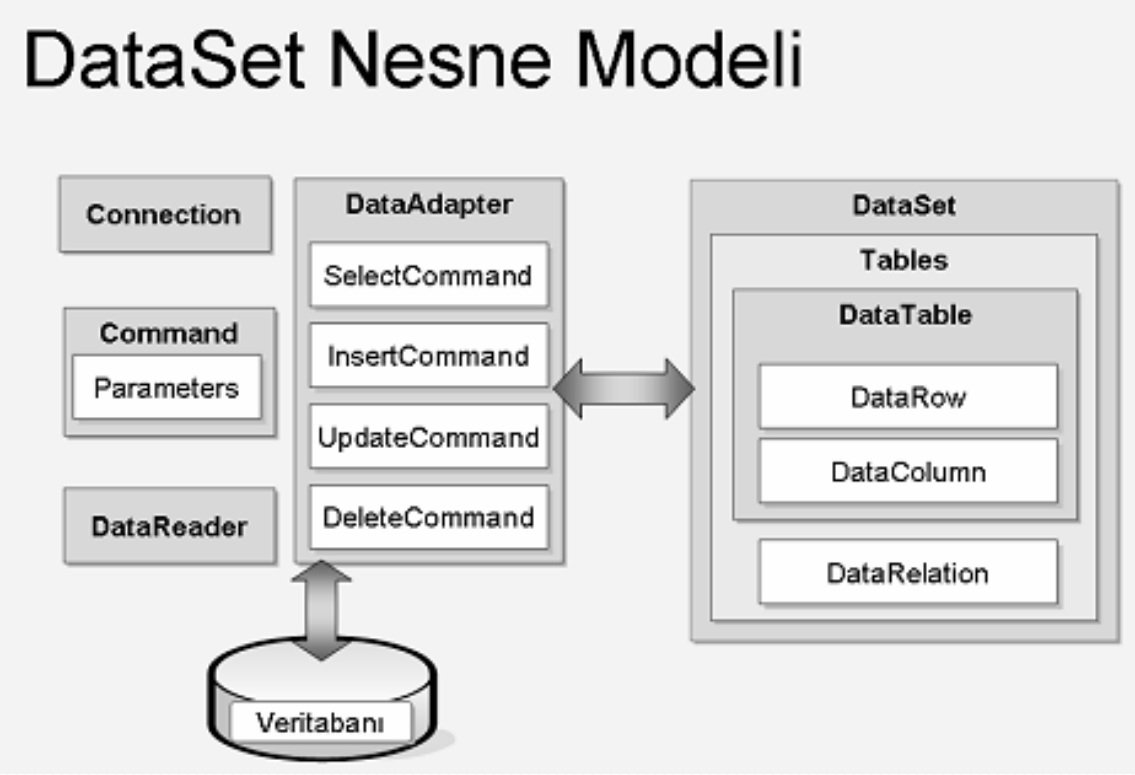
- Veritabanı sunucusunun uzak olması, veri işlemlerinin uzun sürmesi ve mobil çalışma ihtiyacı, bağlantısız veri ortamlarına olan ihtiyacı artırmıştır.
  - **DataAdapter** nesnesi, **DataSet** nesne modeli, **DataTable** nesne modeli

- **DataAdapter nesnesi:** Connection, Command ve DataReader sınıflarını kullanarak, verilerin DataSet'e doldurulmasını ve DataSet de yapılan değişikliklerin veri tabanına kaydedilmesini sağlar.

- Veri kaynağından DataAdapter ile çekilen verilerin, çekirdek belleğe atılan kopyası DataSet içerisinde saklanır. DataSet ile bu veriler üzerinde gerekli düzenlemeler yapıldıktan sonra, veriler aynı DataAdapter ile veritabanına aktarılır.

Bağlantısız (Disconnected) Veritabanı İşlemleri

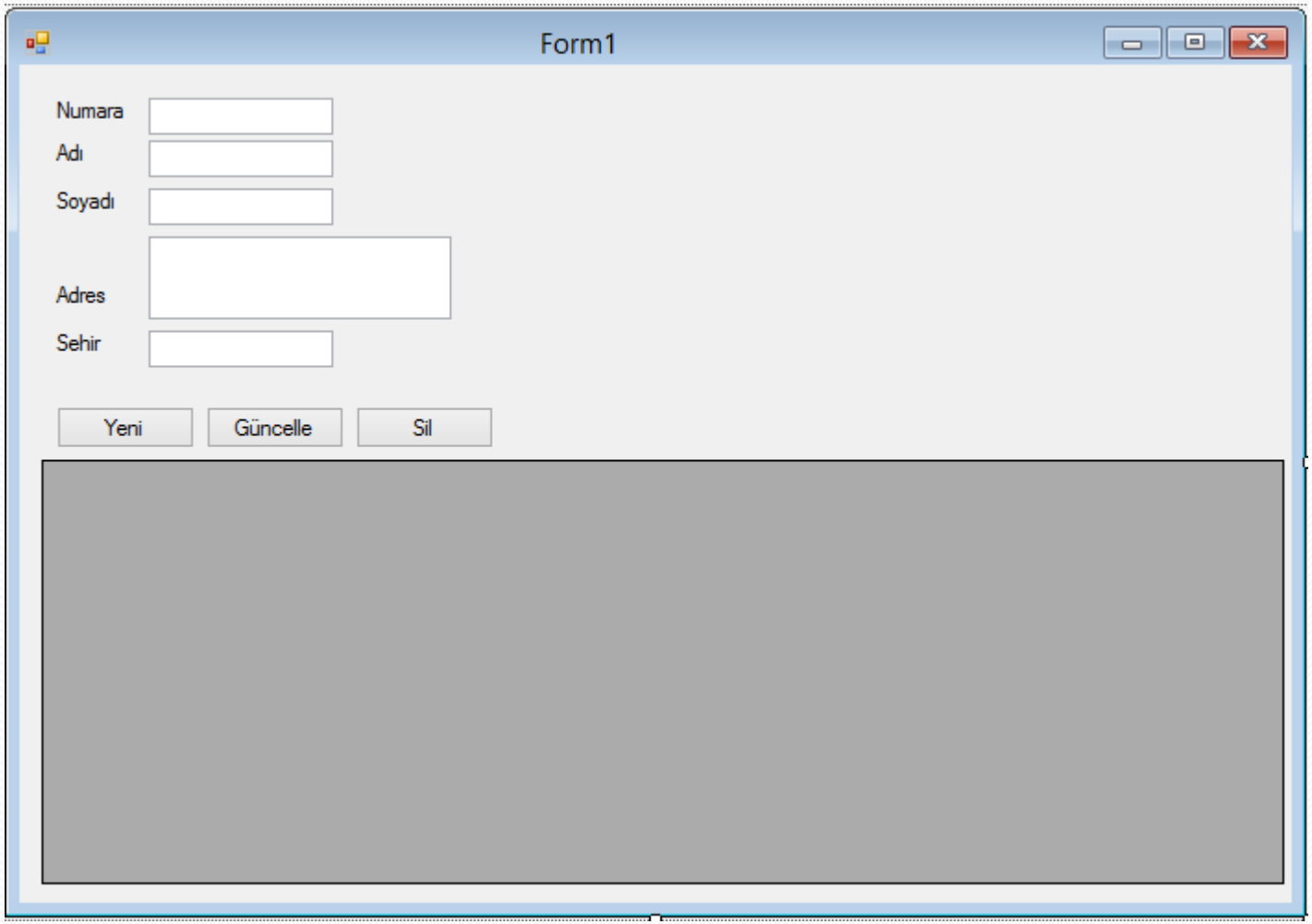
- **DataSet Nesne Modeli**
- **DataSet**, Sanal bir veritabanı yapısını temsil eder. DataTable nesnelerinden oluşur. Bu tablolar arasında ilişkiler tanımlanabilir. DataSet'i oluşturan nesneler: **DataTable**, **DataColumn**, **DataRow**, **DataRelation** nesneleridir.



Bağlantısız (Disconnected) Veritabanı İşlemleri

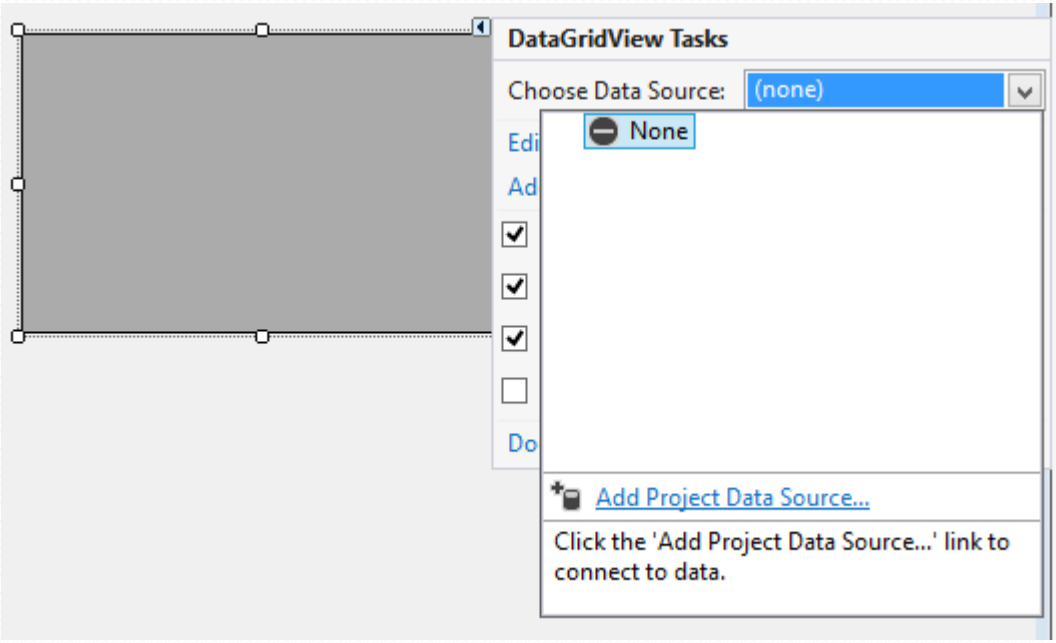
- **DataTable :** Veritabanı tablolarını temsil eder. **DataColumn**, **DataRow** nesnelerinden oluşur. **Primary Key** alanı tanımlanabilir.
- **DataColumn: DataTable** nesnelerini oluşturmak için gereken kolonları temsil eder.
- **DataRow : DataTable** nesneleri için veri satırlarını temsil eder.
- **DataRelationship :** Tablolar arasındaki ilişkileri temsil eder.
- **DataView : DataTable** nesneleri üzerinde filtreleme, veri güncellemeleri işlemleri yapmak için kullanılır.

Bağlantısız (Disconnected) Veritabanı İşlemleri



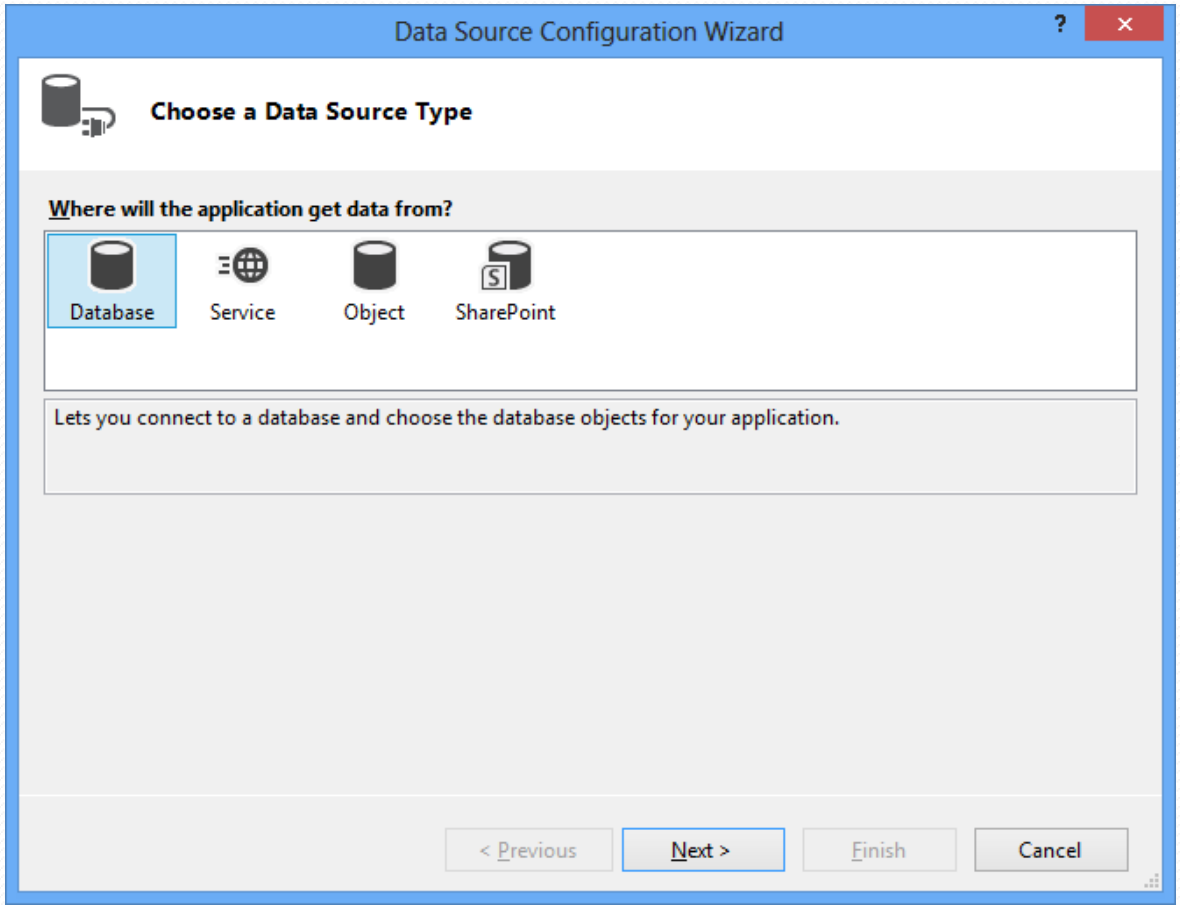
Bağlantısız (Disconnected) Veritabanı İşlemleri

- Bağlantısız veritabanı için forma dataGridView ekleyin. Daha sonra Data Source seçin (Visaul Studio yardımıyla dataset hazırlanması)

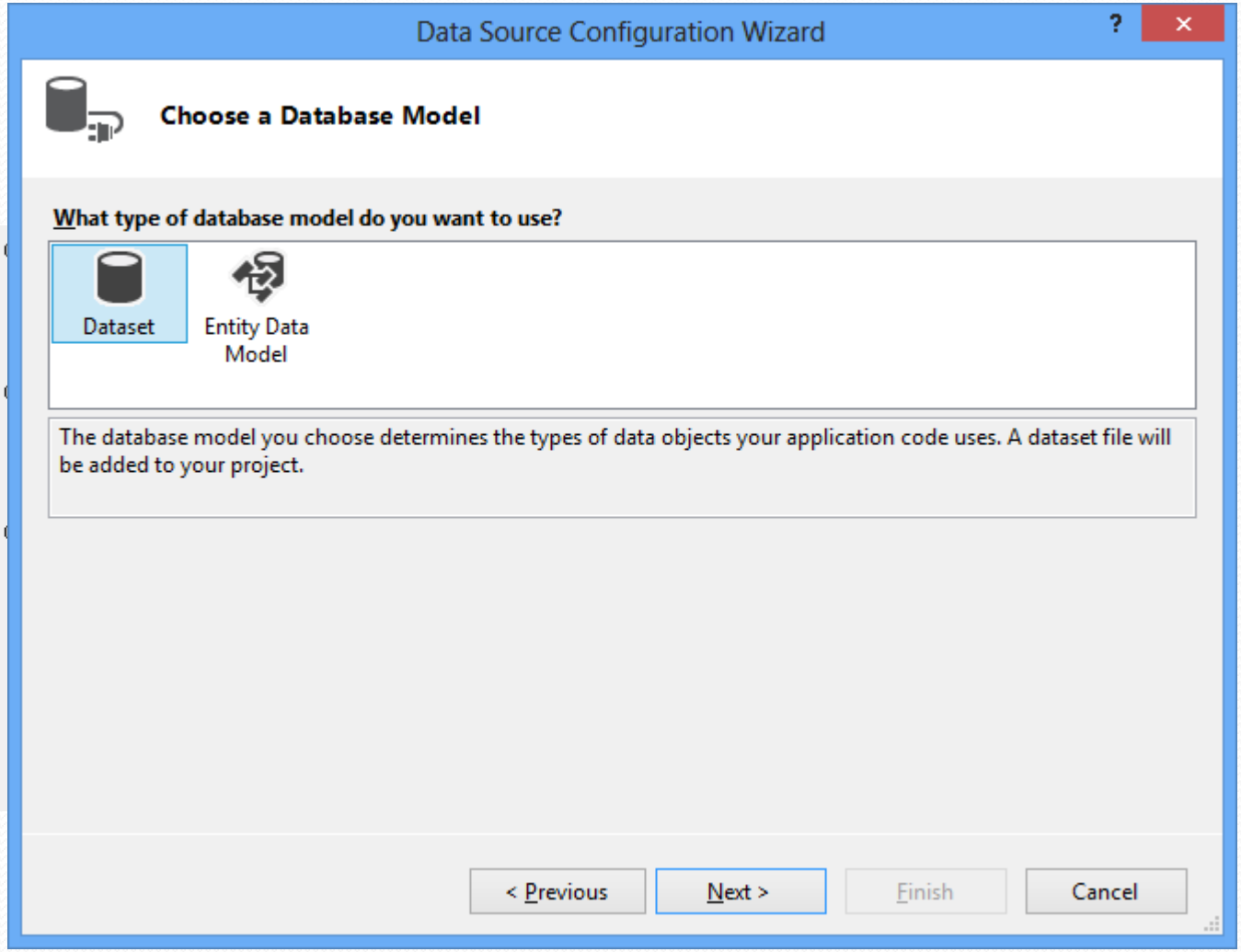


Bağlantısız (Disconnected) Veritabanı İşlemleri

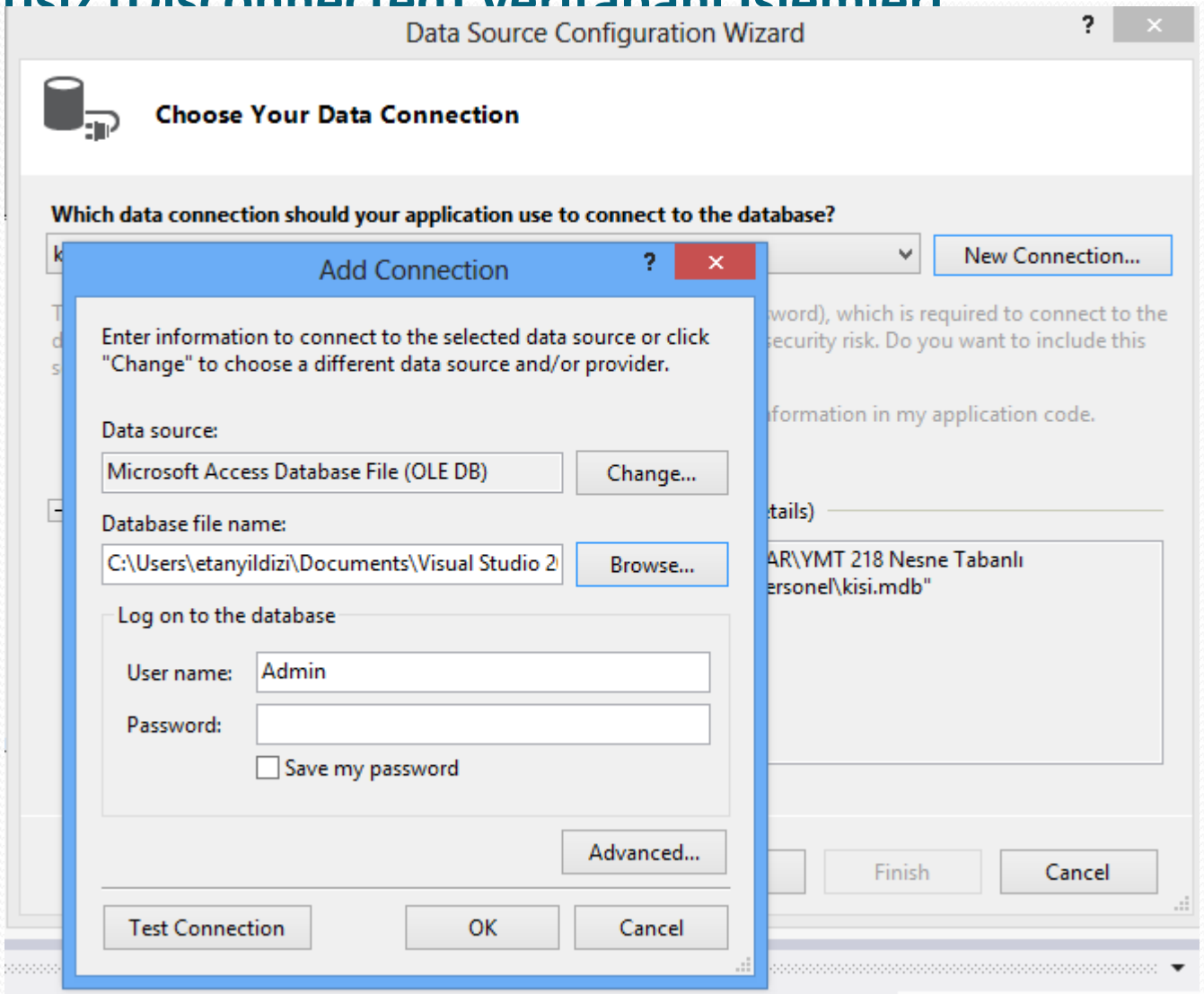
- Sihirbaz yardımıyla veri tabanını ve dataseti belirleyin



Bağlantısız (Disconnected) Veritabanı İşlemleri

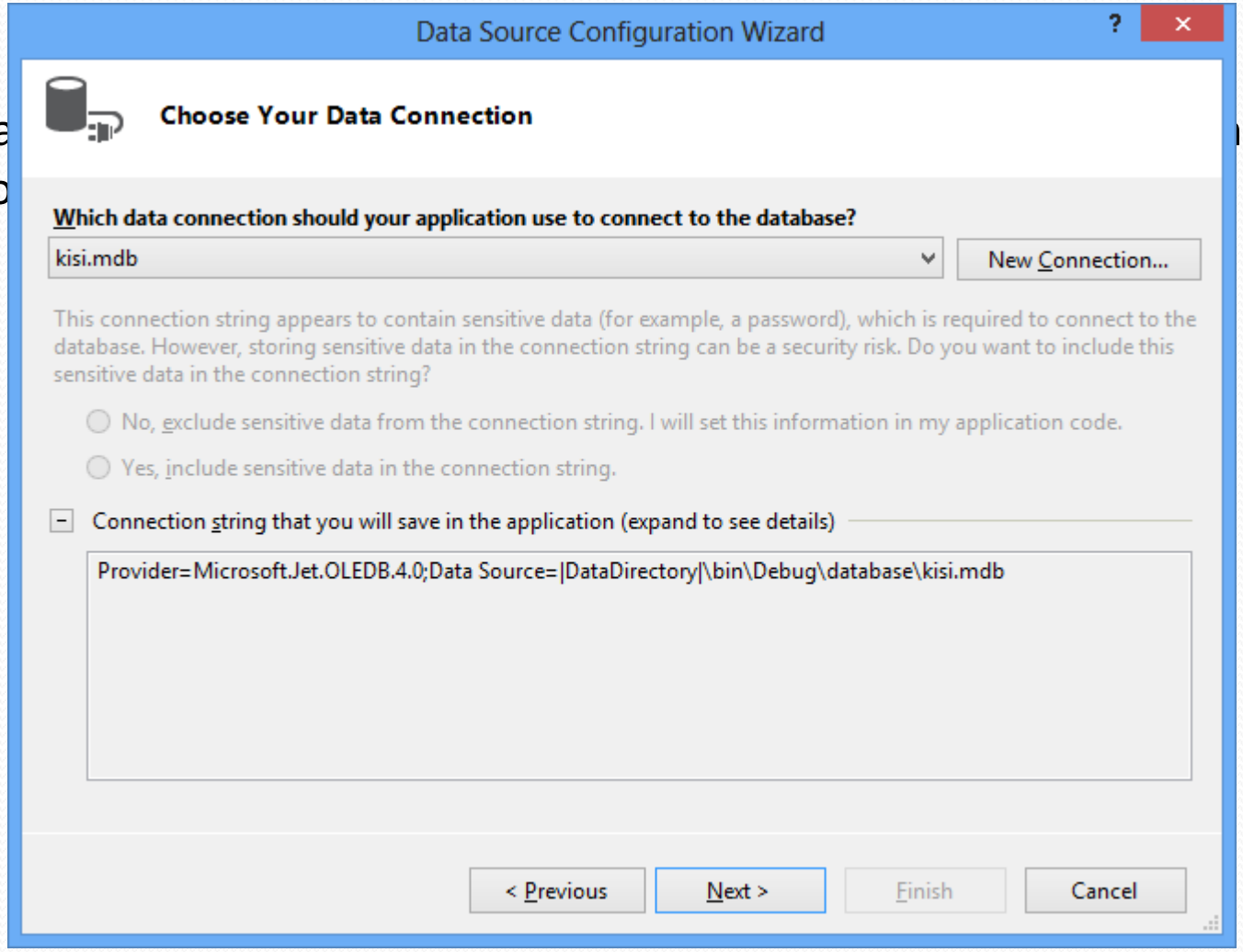


Bağlantısız (Disconnected) Veritabanı İşlemleri

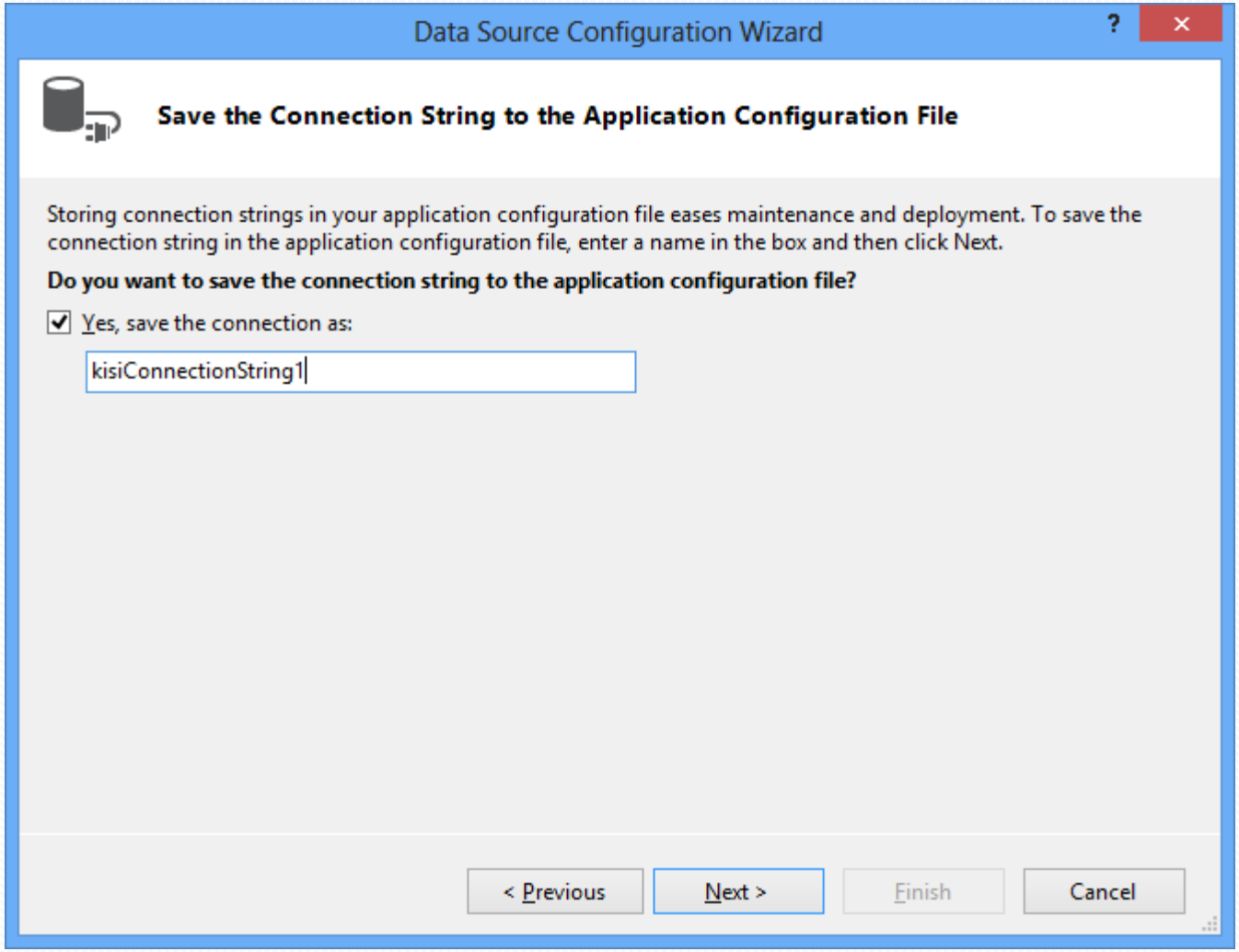


Bağlantısız (Disconnected) Veritabanı İşlemleri

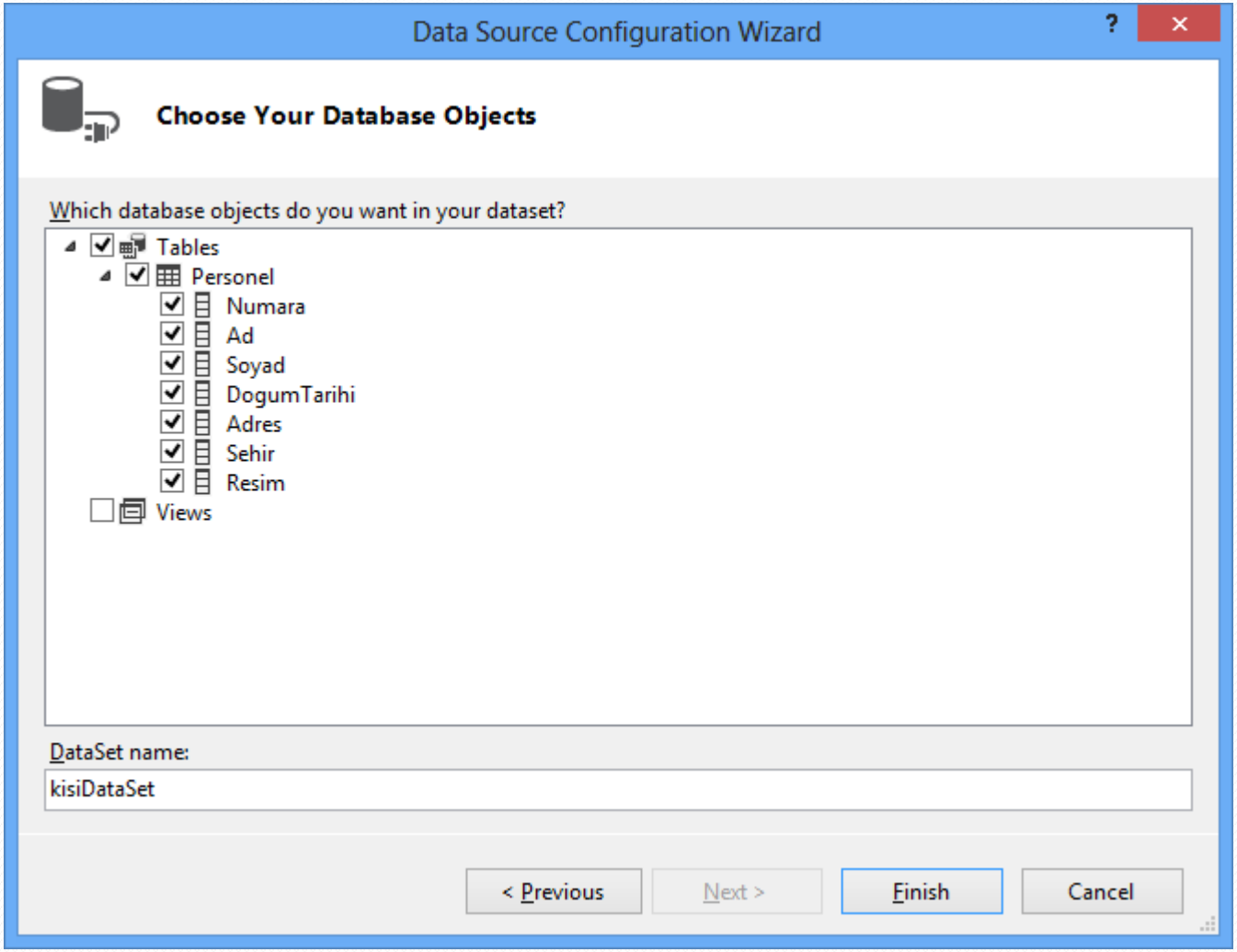
- Ba ra Data So



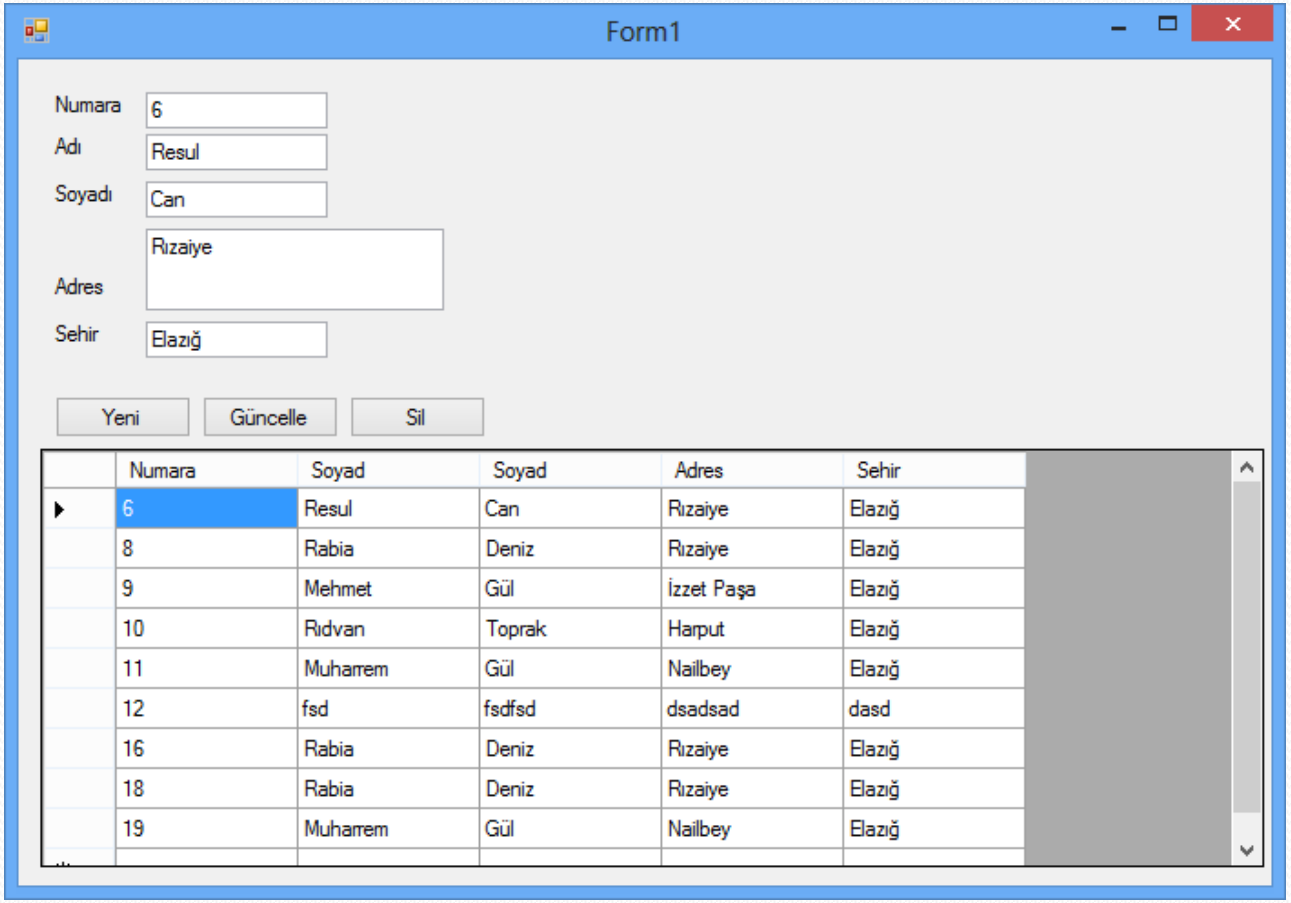
Bağlantısız (Disconnected) Veritabanı İşlemleri



Bağlantısız (Disconnected) Veritabanı İşlemleri



Bağlantısız (Disconnected) Veritabanı İşlemleri



Numara	Soyad	Soyad	Adres	Sehir
6	Resul	Can	Rızaiye	Elazığ
8	Rabia	Deniz	Rızaiye	Elazığ
9	Mehmet	Gül	İzzet Paşa	Elazığ
10	Rıdvan	Toprak	Harput	Elazığ
11	Muharrem	Gül	Nailbey	Elazığ
12	fsd	fsdfsd	dsadsad	dasd
16	Rabia	Deniz	Rızaiye	Elazığ
18	Rabia	Deniz	Rızaiye	Elazığ
19	Muharrem	Gül	Nailbey	Elazığ

Bağlantısız (Disconnected) Veritabanı İşlemleri

- **DataAdapter ile kayıtları Dataset'e doldurmak**

- Hazır bağlantı kullanmadan **DataAdapter** ve **DataSet** tanımlanabilir.
- Örnekte **OleDbDataAdapter** ile çekilen veriler, **ds** ismindeki **DataSet** nesnesine aktarılır. **DataSet** içerisindeki veriler, **DataGrid** ile ekranda gösterilir.
  - `OleDbConnection conn = new OleDbConnection ("provider = “  + microsoft.jet.oledb.4.0; data source=C:\Stok.mdb");`
  - `OleDbDataAdapter da = new OleDbDataAdapter("select * from kitaplar", conn);`
  - `DataSet ds = new DataSet();`
  - **da.Fill (ds, "kitaplar ");**
  - dataGridView1.DataSource= ds.Tables["kitaplar"];

- Fill metodu ile belirli kayıt aralığı DataSet içerisine aktarılabilir. Örnekte da isimli DataAdapter ile çekilen ilk altı kayıt, Kitaplar tablosuna aktarılır.
  - **da.Fill(ds, 0, 5, "kitaplar");**

Bağlantısız (Disconnected) Veritabanı İşlemleri

- Örnekte **ds** ismindeki **DataSet** nesnesinin tüm tablo, ilişki ve verileri **ds_yeni** ismindeki **DataSet** nesnesinin içerisine aktarılmıştır. Yeni_DataSet, tablo adı veya veritabanı adı olabilir.
  - **DataSet** ds = new DataSet("Yeni_DataSet");
  - **DataSet** ds_yeni;
  - ds_yeni= **ds.Copy**(); //veya **ds.Copy ()** => **ds.Clone()**;

- **DataSet** sınıfının **Tables** koleksiyonu ile DataSet içerisine bir veya birden çok **DataTable** eklenebilir. Örnekte **dtKitaplar** isminde yeni bir DataTable oluşturulmaktadır.
  - **DataTable dtKitaplar = new DataTable("Kitaplar");**

- Oluşturulan tabloyu DataSet içerisine eklemek için DataSet nesnesinin Tables koleksiyonu kullanılır.
  - **ds.Tables.Add(dtKitaplar);**

Bağlantısız (Disconnected) Veritabanı İşlemleri

- DataTable nesnesinin içerisine kolon eklenebilir. Örnekte dtKitaplar ismindeki DataTable nesnesinin içerisine, yeniId isminde yeni bir kolon eklenmektedir. Yeni kolon eklemek için, DataTable nesnesinin **Columns** koleksiyonu kullanılır.
  - **DataColumn colKitapId = dtKitaplar.Columns.Add("yeniId");**

- Örnekte DataTable nesnesi için Ucret, KDV ve Tutar isminde 3 adet kolon oluşturulmuştur. Örnekteki KDV kolonu, Ucret kolonun %17 değeri üzerinden hesaplanır. Tutar kolonu ise Ucret ve KDV değerinin toplamı ile hesaplanır.

Bağlantısız (Disconnected) Veritabanı İşlemleri

```
• DataTable dtKitaplar = new DataTable("Kitaplar");
• this.kisiDataSet.Tables.Add(dtKitaplar);
• DataColumn colKitapId = dtKitaplar.Columns.Add("yeniId");
• DataColumn colUcret = dtKitaplar.Columns.Add("Ucret");
• colUcret.DataType = System.Type.GetType("System.Double");
• //"System.Int32"
• DataColumn colKdv = dtKitaplar.Columns.Add("KDV");
• // dtKitaplar.Columns.Add("OrderID", typeof(Int32));
• colKdv.DataType = System.Type.GetType("System.Double");
• colKdv.Expression = "Ucret * 0.18";
• DataColumn colTutar = dtKitaplar.Columns.Add("Tutar");
• colTutar.DataType = System.Type.GetType("System.Double");
• colTutar.Expression = "Ucret +KDV";
• dtKitaplar.Rows.Add(5, 25); //dtKitaplar.Rows.Add("yeniId değeri", "Ucret değeri")
• dtKitaplar.Rows.Add(20, 50);
• dataGridView2.DataSource = this.kisiDataSet;
• dataGridView2.DataMember = "Kitaplar";
```

Bağlantısız (Disconnected) Veritabanı İşlemleri

- **Tabloları ilişkilendirme**

```
DataSet customerOrders = new DataSet("CustomerOrders");
DataTable ordersTable = customerOrders.Tables.Add("Orders");
DataColumn pkOrderID = ordersTable.Columns.Add("OrderID", typeof(Int32));
ordersTable.Columns.Add("OrderQuantity", typeof(Int32));
ordersTable.Columns.Add("CompanyName", typeof(string));
ordersTable.PrimaryKey = new DataColumn[] { pkOrderID };

DataRelation customerOrdersRelation = customerOrders.Relations.Add("CustOrders",
customerOrders.Tables["Customers"].Columns["CustomerID"],
customerOrders.Tables["Orders"].Columns["CustomerID"]);
foreach (DataRow custRow in customerOrders.Tables["Customers"].Rows)
{ Console.WriteLine(custRow["CustomerID"].ToString());
  foreach (DataRow orderRow in custRow.GetChildRows(customerOrdersRelation))
  { Console.WriteLine(orderRow["OrderID"].ToString()); }
 }
```

Bağlantısız (Disconnected) Veritabanı İşlemleri

- DataSet üzerinde yapılan değişiklikleri veri kaynağına aktarmak için, DataAdapter sınıfının **Update** metodu kullanılır. DataAdapter nesnesinin **DeleteCommand**, **UpdateCommand** ve **InsertCommand** nesneleri içinde tutulan sorgular ile güncelleme işlemi gerçekleştirilir. Örnekte Sipariş tablosundaki tüm değişiklikler veri kaynağına aktarılmaktadır.
- `OleDbCommandBuilder cb = new OleDbCommandBuilder(da);`
- `// Ekleme, silme güncelleme için DataAdapter nesnesini ayarlar.`
  - `da.Update(ds, "siparisler");`

- Örnekte Dataset içerisindeki kitap_baslik kolonun değeri, TextBox kontrolünün **Text** özelliğine aktarılır.
  - `textBox1.Text = ds.Tables["kitaplar"].Rows[2].Item["kitap_baslik"];`

- Örnekte ComboBox ve ListBox kontrolunun **DataSource** ve **DisplayMember** özellikleri kullanılmaktadır.
  - `comboBox1.DataSource = ds.Tables["kitaplar"];`
  - `comboBox1.DisplayMember = ds.Tables["kitaplar"].Columns["kitap_baslik"].ToString();`
  - `listBox1.DataSource = ds.Tables["kitaplar"];`
  - `listBox1.DisplayMember = ds.Tables["kitaplar"].Columns["kitap_baslik"].ToString();`

Bağlantısız (Disconnected) Veritabanı İşlemleri

- Örnekte TreeView kontrolüne veri bağlamak için, TreeNode nesnesinin **Text** özelliği kullanılır.
  - **treeView1.Nodes[0].Text = ds.Tables["kitaplar"].Rows[1].Item["kitap_baslik"];**

- Örnekte DataSet nesnesinden gelen veriler ListView ve CheckedListBox kontrollerine aktarılmıştır.
  - **int count ;**
  - **Count = ds.Tables["kitaplar"].Columns.Count();**
  - **for (int i=0;i< count;i++)**
  - **{ listView1.Items.Add(ds.Tables["kitaplar"].Rows[i][0].ToString()); }**
  -
  - **for (int i=0;i<count;i++) {**
  - **checkedListBox1.Items.Add(ds.Tables["kitaplar"].Rows[i][0].ToString()); }**
-

Bağlantısız (Disconnected) Veritabanı İşlemleri

- **DataRow** ile DataTable içerisindeki kayıtlar değiştirilebilir. DataRow nesnesi ile satır düzenleme işlemleri için aşağıdaki metodlar kullanılır.
- **BeginEdit**, veriyi düzenlerken oluşabilecek olayları askıya alır. Veriyi düzenlemek için Items koleksiyonu kullanılır.
- **EndEdit** metodu ile, askıya alınan olaylar yeniden aktif edilir.
- **CancelEdit** metodu ile değişikliklerden ve askıya alınan olaylardan vazgeçilir.

Bağlantısız (Disconnected) Veritabanı İşlemleri

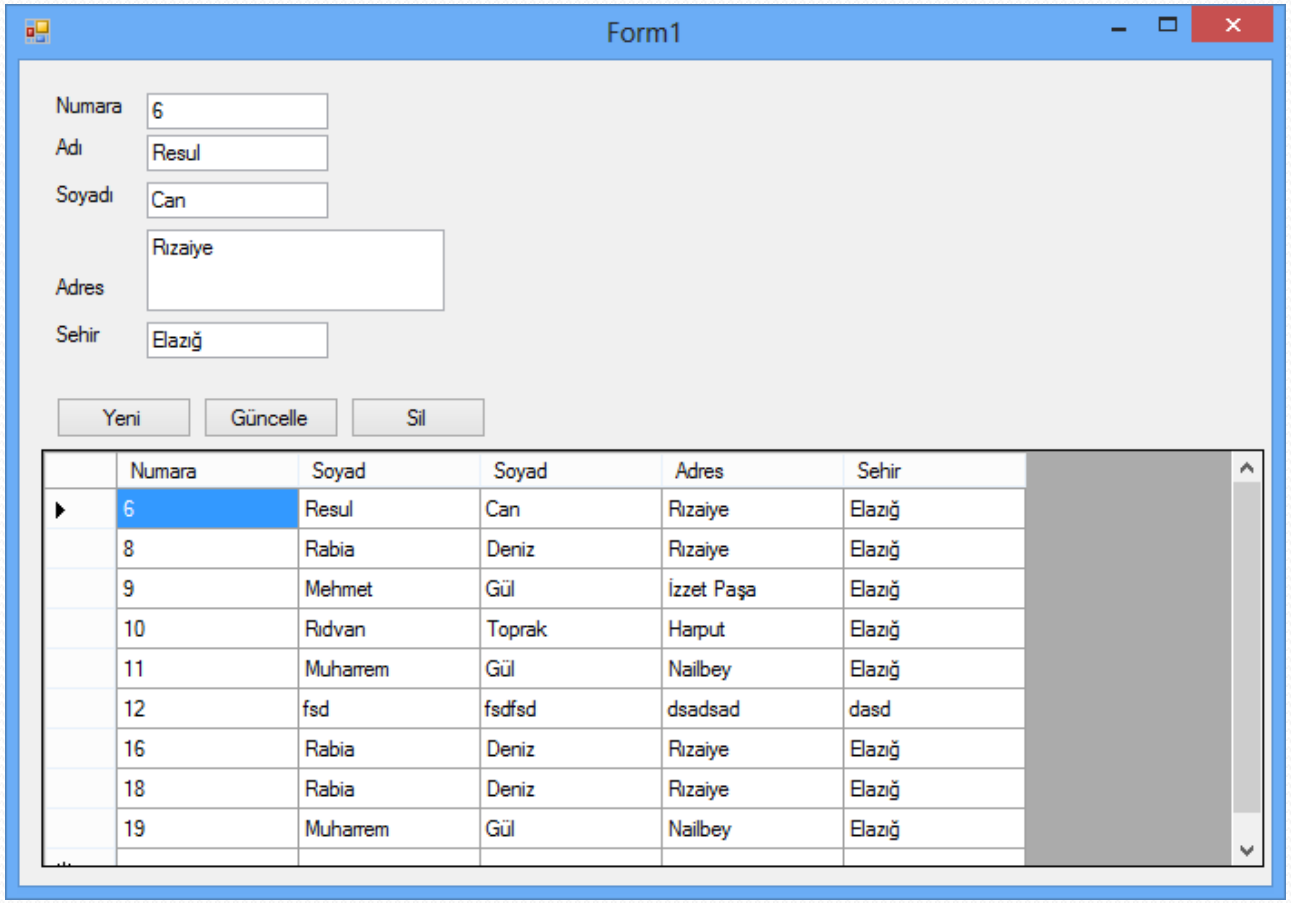
- Örnekte DataTable içerisindeki dördüncü kayıt için güncelleme işlemi yapılmıştır.
  - **DataRow drNew = dtKitaplar.Rows[3];**
  - **drNew.BeginEdit();**
  - **drNew["kitap_baslik"] = "Yeni Dünya"; drNew["kitap_yazar"] = "Can Yılmaz";**
  - **drNew.EndEdit();**
- DataRow ile DataTable içerisindeki belirli bir satır silinebilir. Örnekte DataTable içerisindeki dördüncü kayıt silinmiştir.
  - **DataRow drSil = dtKitaplar.Rows[3];**
  - **dtKitaplar.Rows.Remove(drSil);**
  - **DataRow nesnesinin Delete metodu kullanılarak aktif kayıt silinebilir.**
  - **drSil.Delete();**

Bağlantısız (Disconnected) Veritabanı İşlemleri

```
using System.Data.OleDb;
using System.Data;
class vt
{  static void Main()
    {   string kaynak = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=dosya.mdb";
        OleDbConnection baglanti = new OleDbConnection(kaynak);
        string sorgu = "SELECT * FROM tablo";
        OleDbDataAdapter odda = new OleDbDataAdapter(sorgu, baglanti);
        DataSet ds = new DataSet();
        odda.Fill(ds, "tablo");
        DataRow r = ds.Tables["tablo"].NewRow();
        r["id"] = 21;
        r["ad"] = "hatice";
        r["soyad"] = "özdoğan";
        odda.InsertCommand = new OleDbCommand("INSERT INTO tablo (id,ad,soyad) values (?,?,?)", baglanti);
        odda.UpdateCommand = new OleDbCommand("UPDATE tablo SET id=?, ad=?, soyad=? WHERE id=?", baglanti);
        odda.DeleteCommand = new OleDbCommand("DELETE FROM tablo WHERE id=?", baglanti);
        odda.InsertCommand.Parameters.Add("@id", OleDbType.Integer, 2, "id");
        odda.InsertCommand.Parameters.Add("@ad", OleDbType.VarChar, 10, "ad");
        odda.InsertCommand.Parameters.Add("@soyad", OleDbType.VarChar, 10, "soyad");
        odda.UpdateCommand.Parameters.Add("@id", OleDbType.Integer, 2, "id");
        odda.UpdateCommand.Parameters.Add("@ad", OleDbType.VarChar, 10, "ad");
        odda.UpdateCommand.Parameters.Add("@soyad", OleDbType.VarChar, 10, "soyad");
        odda.UpdateCommand.Parameters.Add("@oldid", OleDbType.Integer, 2, "id").SourceVersion = DataRowVersion.Original;
        odda.DeleteCommand.Parameters.Add("@id", OleDbType.Integer, 2, "id").SourceVersion = DataRowVersion.Original;
        ds.Tables["tablo"].Rows.Add(r);Ekleme
        ds.Tables["tablo"].Rows[0]["id"] = 98;//Değiştirme
        ds.Tables["tablo"].Rows[1].Delete();//Silme
        odda.Update(ds, "tablo");//Veri tabanını Günceleme

    }
}
```

Bağlantısız (Disconnected) Veritabanı İşlemleri

- Örnek



Bağlantısız (Disconnected) Veritabanı İşlemleri

```
public partial class Form1 : Form
{OleDbConnection conn = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; Data Source=db/kisi.mdb");
DataSet ds = new DataSet();     OleDbDataAdapter da_veri;
private void Form1_Load(object sender, EventArgs e)
{ da_veri = new OleDbDataAdapter("SELECT Numara, Ad, Soyad, Adres, Sehir FROM Personel ORDER BY Numara", conn);
da_veri.Fill(ds, "Personel");
dataGridView1.DataSource = ds.Tables["Personel"];
}
 private void button1_Click(object sender, EventArgs e)
 {    textBox1.Text = "";   textBox2.Text = "";  textBox3.Text = "";
      textBox4.Text = "";  textBox5.Text = "";
      textBox1.Visible = false; label5.Visible = false;
      button1.Visible = false;
 }
```

Bağlantısız (Disconnected) Veritabanı İşlemleri

```
private void dataGridView1_SelectionChanged(object sender, EventArgs e)
{
textBox1.Text = dataGridView1.CurrentRow.Cells[0].Value.ToString();
textBox2.Text = dataGridView1.CurrentRow.Cells[1].Value.ToString();
textBox3.Text = dataGridView1.CurrentRow.Cells[2].Value.ToString();
textBox4.Text = dataGridView1.CurrentRow.Cells[3].Value.ToString();
textBox5.Text = dataGridView1.CurrentRow.Cells[4].Value.ToString();
}
public void guncelle()
{
        da_veri.Update(ds, "Personel");
        ds = new DataSet();
        da_veri.Fill(ds, "Personel");
        dataGridView1.DataSource = ds.Tables["Personel"];
 }
```

Bağlantısız (Disconnected) Veritabanı İşlemleri

```
private void button4_Click(object sender, EventArgs e)
{ if ((textBox2.Text != "") && (textBox3.Text != "") && (textBox4.Text != "") &&
(textBox5.Text != ""))
 {  DataRow dr = ds.Tables["Personel"].NewRow();
     dr["Ad"] = textBox2.Text;        dr["Soyad"] = textBox3.Text;
     dr["Adres"] = textBox4.Text;     dr["Sehir"] = textBox5.Text;
     ds.Tables["Personel"].Rows.Add(dr);
     da_veri.InsertCommand = new OleDbCommand("INSERT INTO Personel(Ad,
Soyad,Adres,Sehir)  values (?,?,?,?)", conn);
da_veri.InsertCommand.Parameters.Add("@Ad", OleDbType.VarChar, 50, "Ad");
da_veri.InsertCommand.Parameters.Add("@Soyad", OleDbType.VarChar, 50, "Soyad");
da_veri.InsertCommand.Parameters.Add("@Adres", OleDbType.VarChar, 500, "Adres");
da_veri.InsertCommand.Parameters.Add("@Sehir", OleDbType.VarChar, 50, "Sehir");
guncelle();
 button1.Visible = true; textBox1.Visible = true; label5.Visible = true;
}
}
```

Bağlantısız (Disconnected) Veritabanı İşlemleri

```
 private void button3_Click(object sender, EventArgs e)
 {if (textBox1.Text != "")
  { int ID = Convert.ToInt32(textBox1.Text);
    da_veri.DeleteCommand = new OleDbCommand("DELETE  FROM Personel WHERE Numara=?", conn);
    da_veri.DeleteCommand.Parameters.Add("@Numara", OleDbType.Integer,2, "Numara");
    int satir = ds.Tables["Personel"].Rows.Count;
    for (int i = 0; i < satir; i++)
    {
     if (ID.ToString()==ds.Tables["Personel"].Rows[i]["Numara"].ToString())
     {
       ds.Tables["Personel"].Rows[i].Delete();
       guncelle();
     }
    }
  }
 }
```

Bağlantısız (Disconnected) Veritabanı İşlemleri

```
private void button2_Click(object sender, EventArgs e)
{ int ID = Convert.ToInt16(textBox1.Text);
 if ((textBox2.Text != "") && (textBox3.Text != "") && (textBox4.Text != "") &&
(textBox5.Text != ""))
 {da_veri.UpdateCommand = new OleDbCommand("UPDATE Personel SET Ad=?,Soyad=?,
Adres=?, Sehir=? WHERE Numara=?", conn);
 da_veri.UpdateCommand.Parameters.Add("@Ad", OleDbType.VarChar,50, "Ad");
 da_veri.UpdateCommand.Parameters.Add("@Soyad", OleDbType.VarChar,50, "Soyad");
 da_veri.UpdateCommand.Parameters.Add("@Adres", OleDbType.VarChar,500, "Adres");
 da_veri.UpdateCommand.Parameters.Add("@Sehir", OleDbType.VarChar, 50, "Sehir");
 da_veri.UpdateCommand.Parameters.Add("@Numara", OleDbType.Integer,5,"Numara");
  int satir = ds.Tables["Personel"].Rows.Count;
  for (int i = 0; i < satir; i++)
    if (ID.ToString() == ds.Tables["Personel"].Rows[i]["Numara"].ToString())
       {   DataRow dr = ds.Tables["Personel"].Rows[i];
           dr["Ad"] = textBox2.Text;        dr["Soyad"] = textBox3.Text;
           dr["Adres"] = textBox4.Text;     dr["Sehir"] = textBox5.Text;
           guncelle();   }
  } }
```

Bağlantısız (Disconnected) Veritabanı İşlemleri

- **Windows Form ile Kayıt Üzerinde Hareket Sağlamak**

- Verileri düzenlemeden önce, hangi veri üzerinde düzenleme yapılacağının
- tespit edilmesi gerekir. DataSet, DataTable veya DataView ile kayıtlar üzerinde hareket sağlayan nesneye **CurrencyManager** denir.

- DataSet içinde çoklu veri kaynağı tutulabildiği için, birden fazla CurrencyManager nesnesi içerebilir.

- Belirli bir satıra gidebilmek için, CurrencyManager nesnesinin **Position** özelliği kullanılır.

Bağlantısız (Disconnected) Veritabanı İşlemleri

- Örnekte dtKitaplar tablosunun kayıtları arasında ilk, son, önceki ve sonraki
- satıra hareket sağlanmıştır.
  - CurrencyManager cmKitaplar = new CurrencyManager();
  - private void Form1_Load() {
  - txtKitapAdi.DataBindings.Add("Text", dtKitaplar, "kitap_baslik");
  - cmKitaplar = (CurrencyManager)BindingContext[dtKitaplar];
  - cmKitaplar.Position = 0; }
  - private void btnMoveNext()
  - { if (cmKitaplar.Position != cmKitaplar.Count) { cmKitaplar.Position+= 1;} }
  - private void btnMoveFirst() { cmKitaplar.Position = 0; }
  - private void btnMovePrevious() {
  - if (cmKitaplar.Position != 0) { cmKitaplar.Position -= 1; } }
  - private void btnMoveLast() { cmKitaplar.Position = cmKitaplar.Count-1; }

Visual Studio.Net -C#
14. HAFTA
Veritabanı Uygulamaları
Örnekler

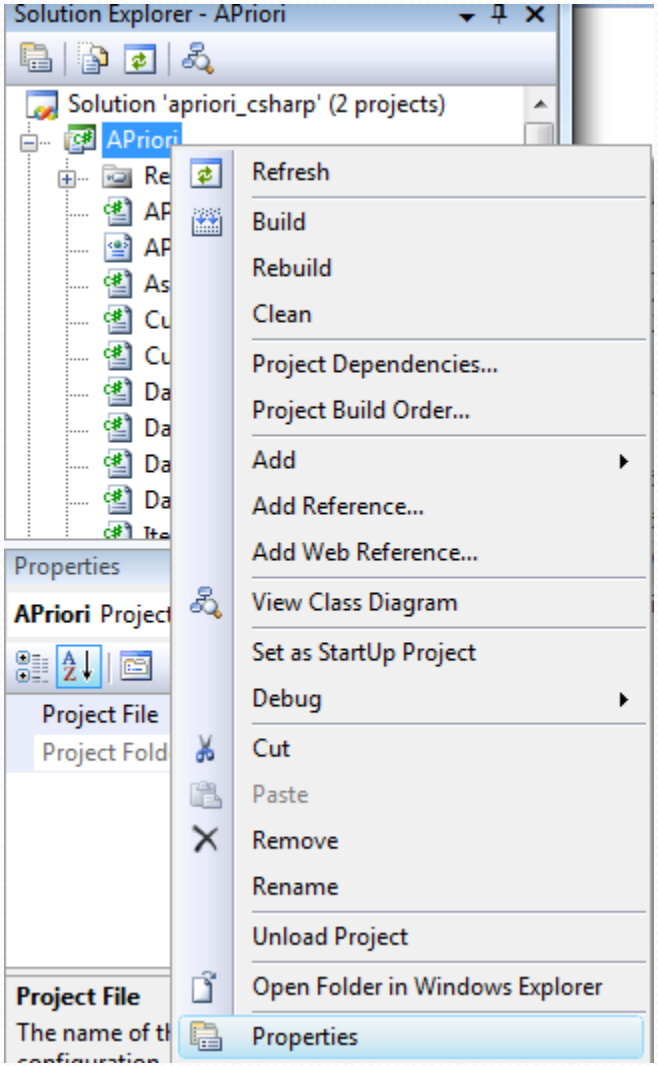
Visual Studio.Net -C#
14. HAFTA
Veritabanı Uygulamaları
Örnekler

VISUAL STUDIO.NET ve Veritabanı

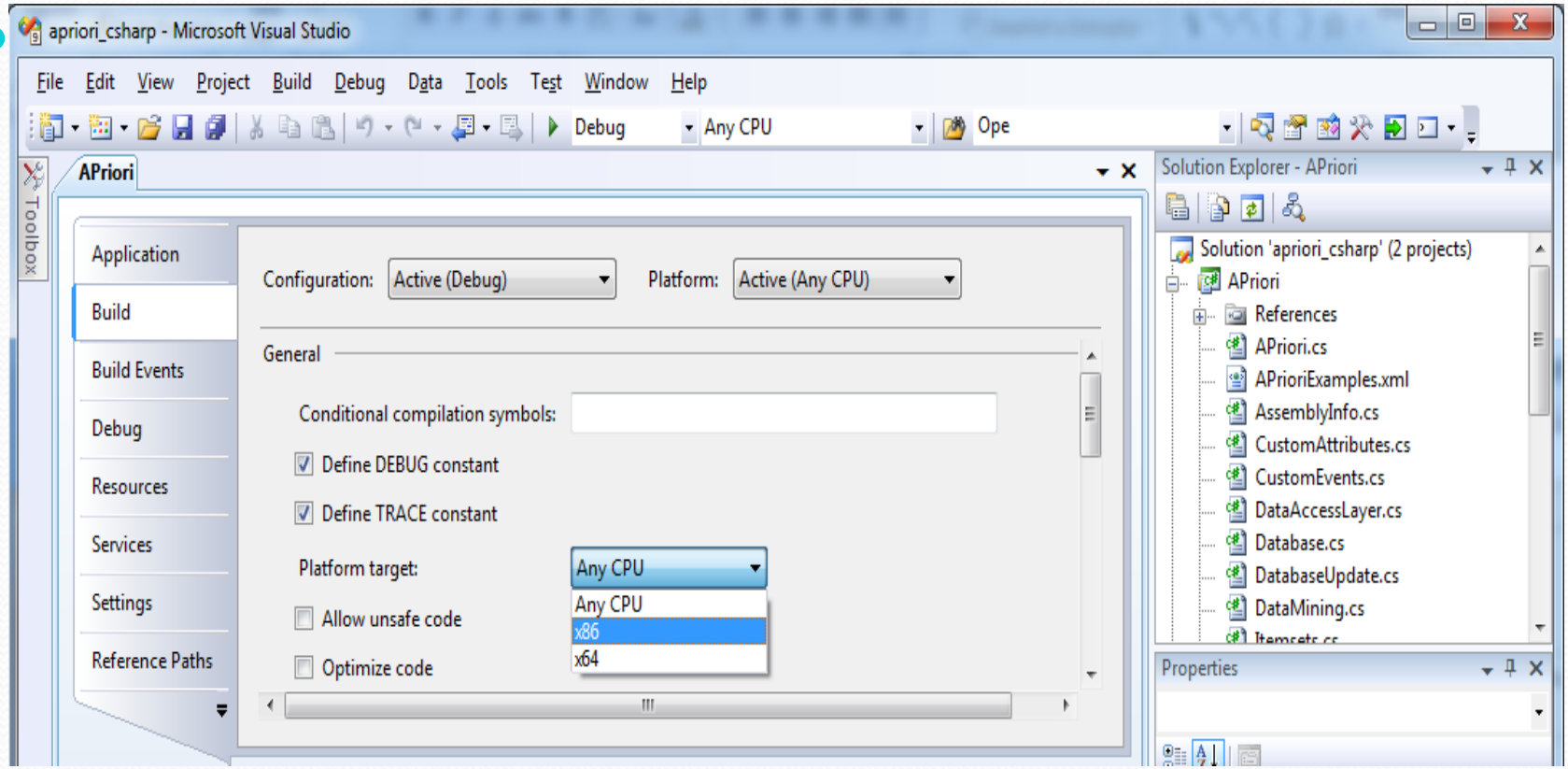
- **C# Veritabanı (Bilge ADAM) notlarına bakınız.**
- **Ms Access ile OLEDB Bağlantı Cümleleri**
- **Access'e Bağlantı**
- **"Provider=Microsoft.Jet.OLEDB.4.0; Data Source=DB_Name.mdb;"**
- **Access'e Parola Korumalı Bağlantı**
- **"Provider=Microso 4.0; Data Source=Db _Name.mdb; Jet OLE ase Password=sifreniz"**
- **Ms Access 2007 ile OLEDB Bağlantı Cümleleri**
- **Standard güvenlik**
- **Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\myFolder\myAccess2007file.accdb;Persist Security Info=False;**
- **Database password**
- **Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\myFolder\myAccess2007file.accdb;Jet OLEDB:Database Password=MyDbPassword;**

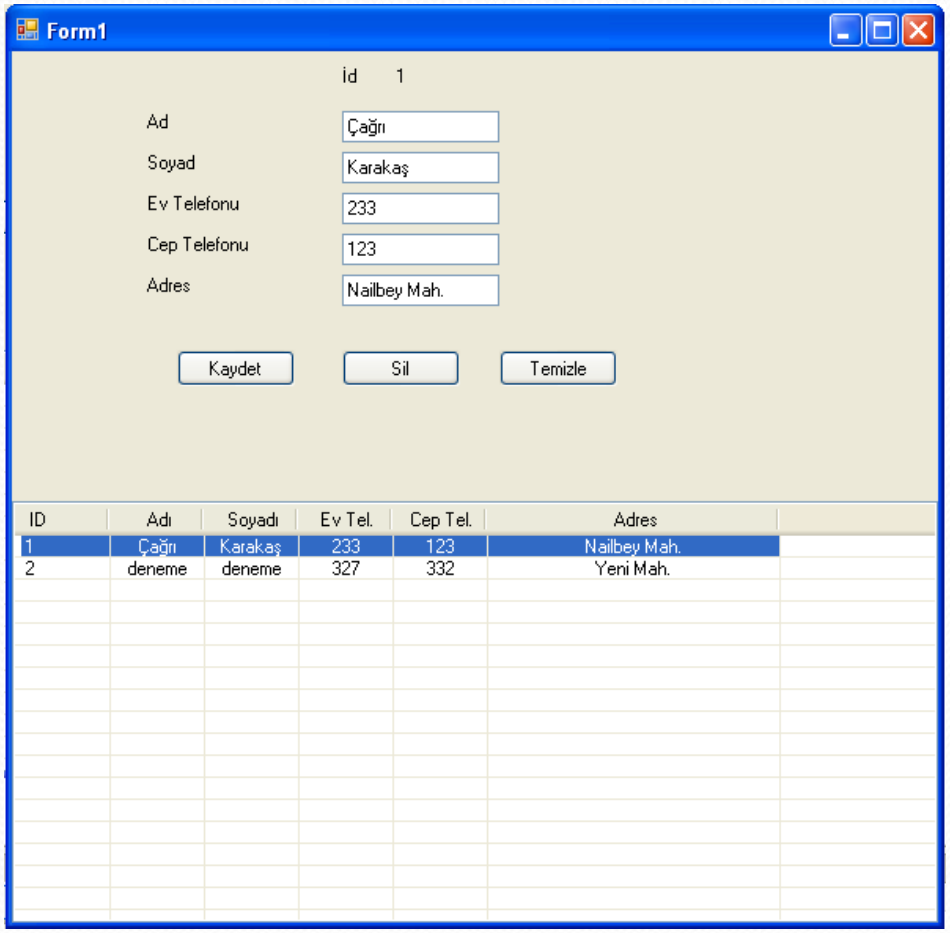
VISUAL STUDIO.NET ve Veritabanı

- Win7 (64 bit) kullanıyorsanız mdb dosyalarında hata alabilirsiniz. Bunu gidermek için Solution Explorer da projenizde sağ tıklayıp Properties ten Build x86 yapınız.



VISUAL STUDIO.NET ve Veritabanı





- Yandaki şekildeki görüldüğü gibi kaydet, sil ve temizle fonksiyonlarını yerine getirebilen basit bir C Sharp veri tabanı uygulaması anlatılacaktır.
- Uygulamamızdaki bağlantılar tümüyle SQL diliyle sağlanmıştır. Uygulamamızın C Sharp kısmına başlamadan önce uygun şekilde veri tabanımızı hazırlamamız gerekir.

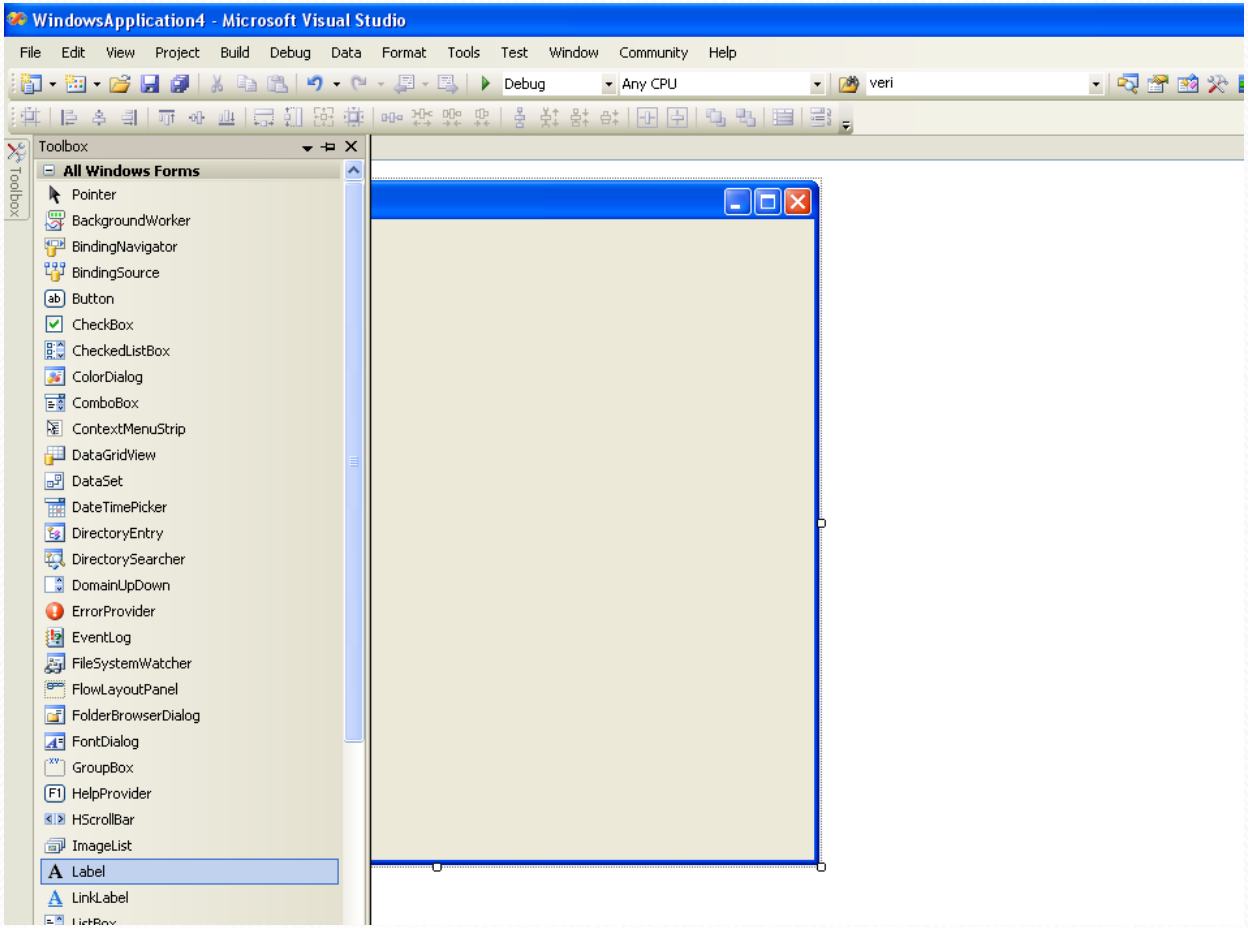
Veri tabanımızı inceleyecek olursak;

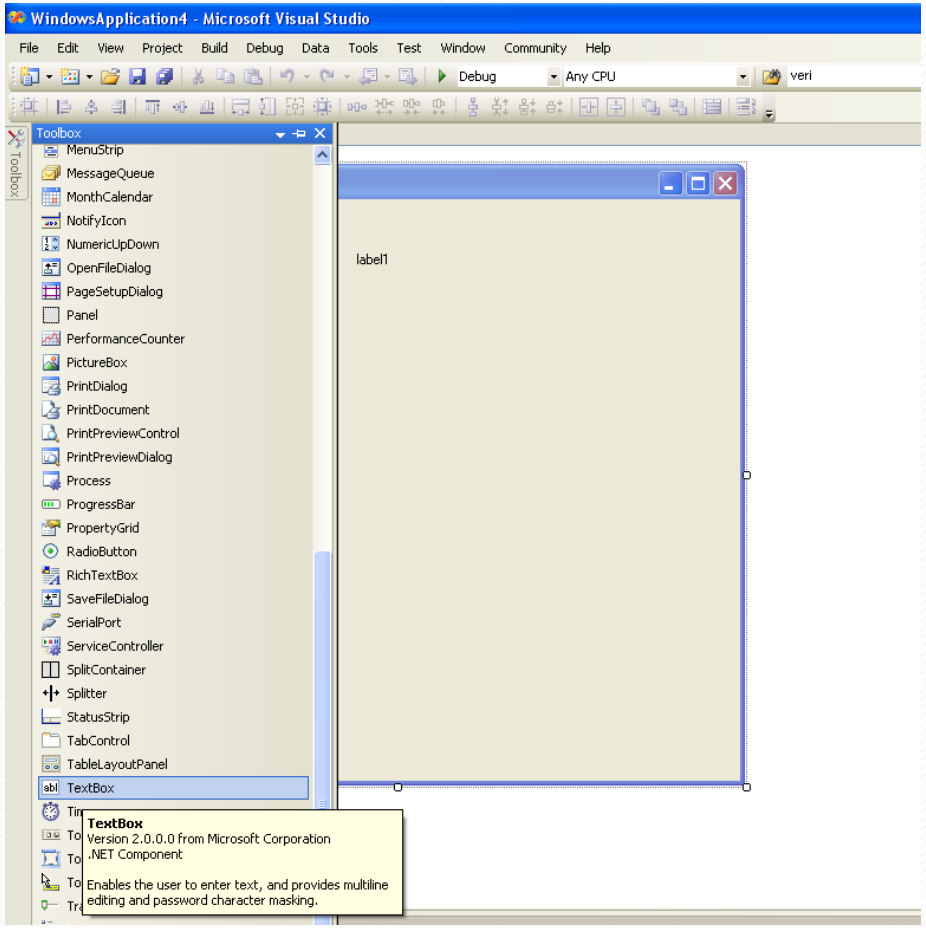
İd numarası, ad, soyad, ev telefonu, cep telefonu ve adres şeklindeki kayıtları tutabilecek şekilde veri tabanımızı hazırlıyoruz. Ben burada veri tabanı parolası olarak “cagri” değerini verdim. Bunu vermemizin amacı veri tabanının başka bir programla açılıp ulaşılmasını engellemektir. Böylece veri tabanını sadece kendi programımızla kontrol edebiliriz. Bu da uygulamamızda “kullanıcı adı” veya “şifre” girişi kullanacaksak yapmamız gereken temel işlevlerden birisidir. Ayrıca bütün sütunları metin olarak ayarladım.

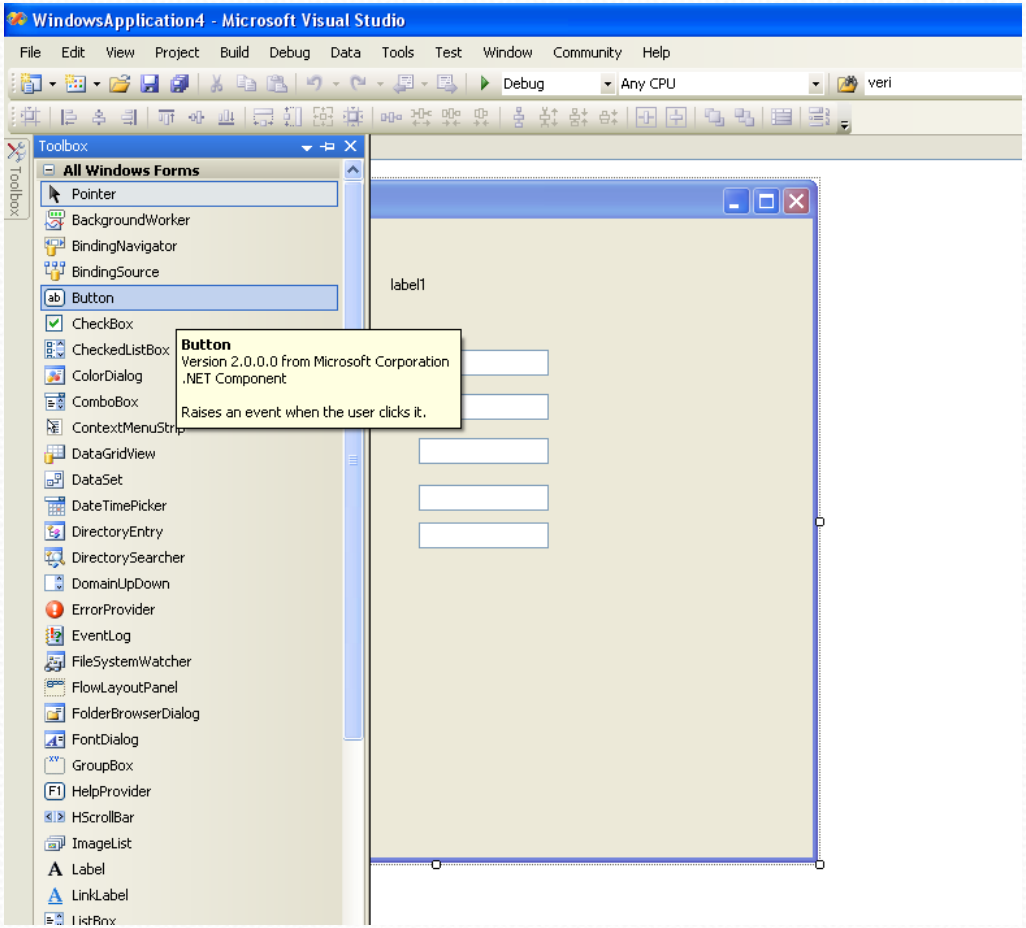
data : Tablo

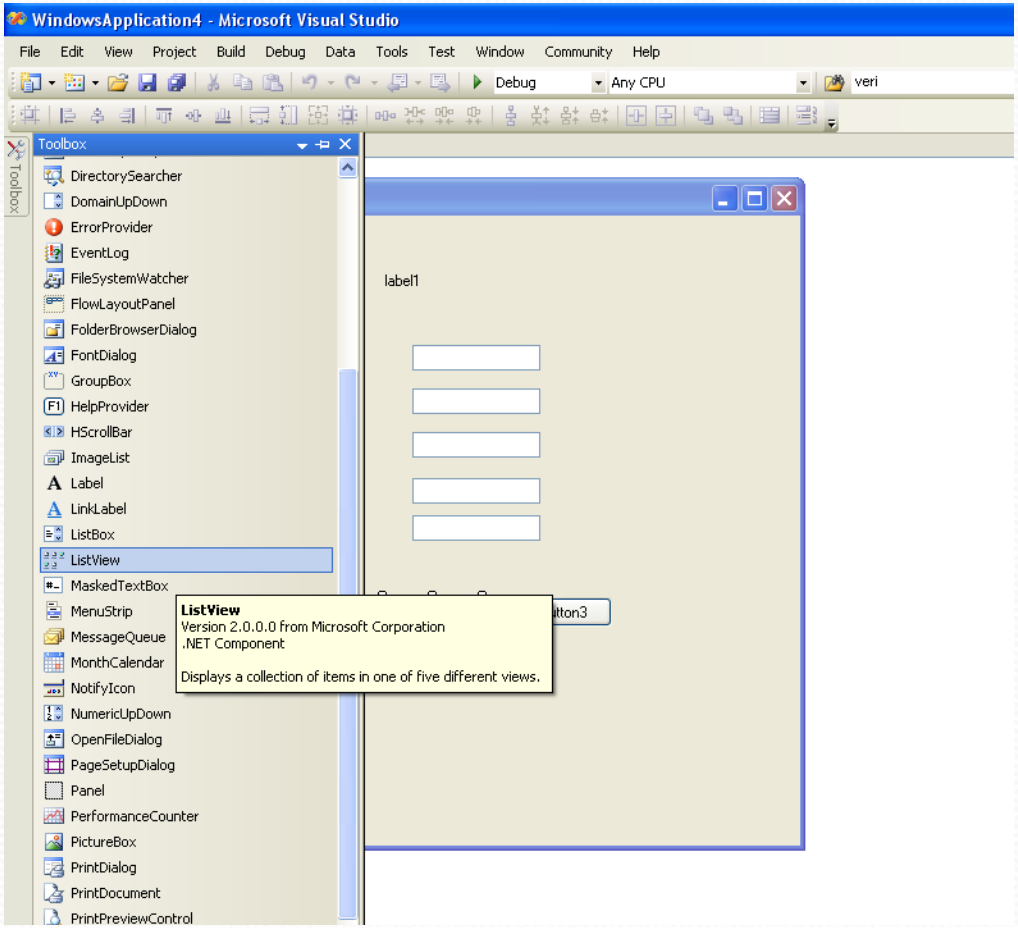
id	ad	soyad	evtel	ceptel	adres
1	Çağrı	Karakaş	233	123	Nailbey Mah.
2	deneme	deneme	327	332	Yeni Mah.
(OtomatikSayı)					

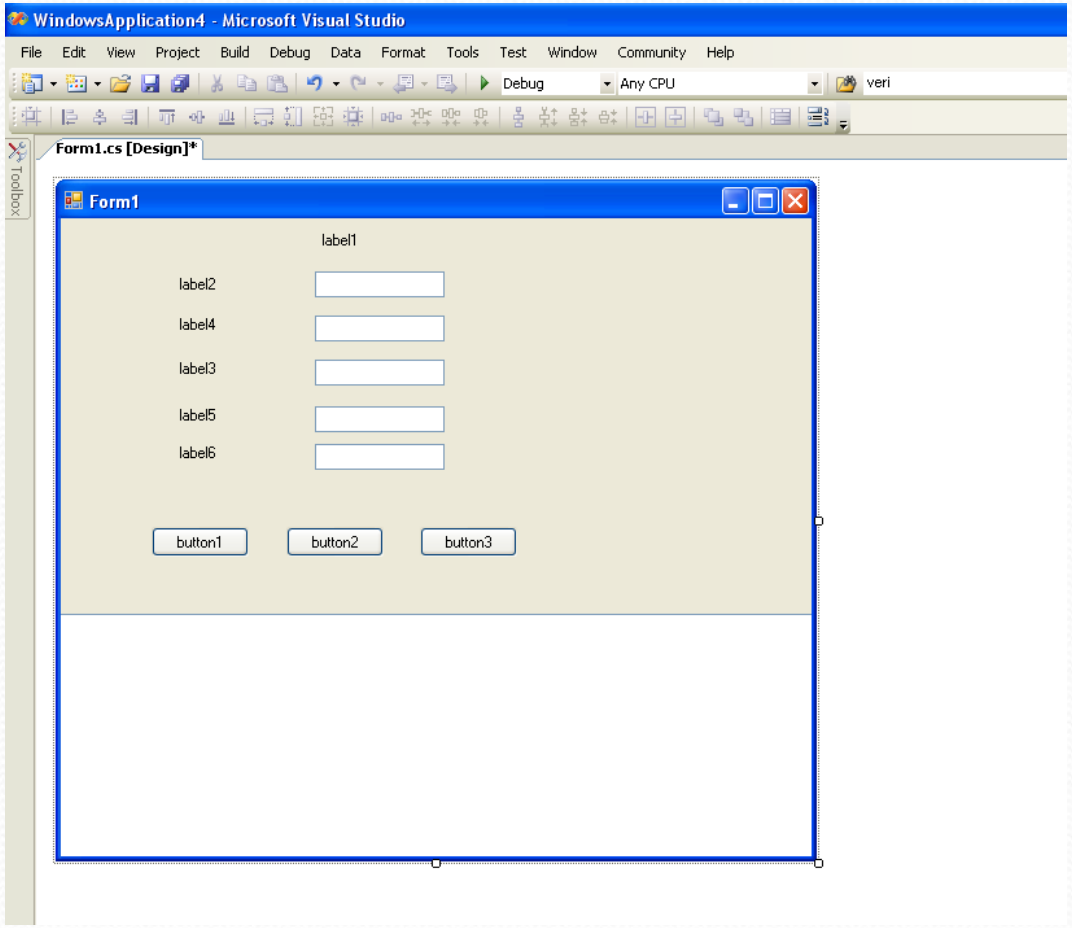
Şimdi hazırladığımız veri tabanımızı daha önceden kaydettiğimiz uygulamamızın “bin/Debug” klasörüne atıyoruz. Ardından ilk slayttaki resmimizdeki gibi Label, TextBox, ListView ve butonlarımızı formumuza yerleştiriyoruz.




WindowsApplication4 - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Window Community Help
Debug
Any CPU
veri
Toolbox
MenuStrip
MessageQueue
MonthCalendar
NotifyIcon
NumericUpDown
OpenFileDialog
PageSetupDialog
Panel
PerformanceCounter
PictureBox
PrintDialog
PrintDocument
PrintPreviewControl
PrintPreviewDialog
Process
ProgressBar
PropertyGrid
RadioButton
RichTextBox
SaveFileDialog
SerialPort
ServiceController
SplitContainer
Splitter
StatusStrip
TabControl
TableLayoutPanel
TextBox
label1
TextBox
Version 2.0.0.0 from Microsoft Corporation
.NET Component
Enables the user to enter text, and provides multiline editing and password character masking.


WindowsApplication4 - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Window Community Help
Debug
Any CPU
veri
Toolbox
All Windows Forms
Pointer
BackgroundWorker
BindingNavigator
BindingSource
Button
CheckBox
CheckedListBox
ColorDialog
ComboBox
ContextMenuStrip
DataGridView
DataSet
DateTimePicker
DirectoryEntry
DirectorySearcher
DomainUpDown
ErrorProvider
EventLog
FileSystemWatcher
FlowLayoutPanel
FolderBrowserDialog
FontDialog
GroupBox
HelpProvider
HScrollBar
ImageList
Label
LinkLabel
ListBox
Button
Version 2.0.0.0 from Microsoft Corporation
.NET Component
Raises an event when the user clicks it.
label1

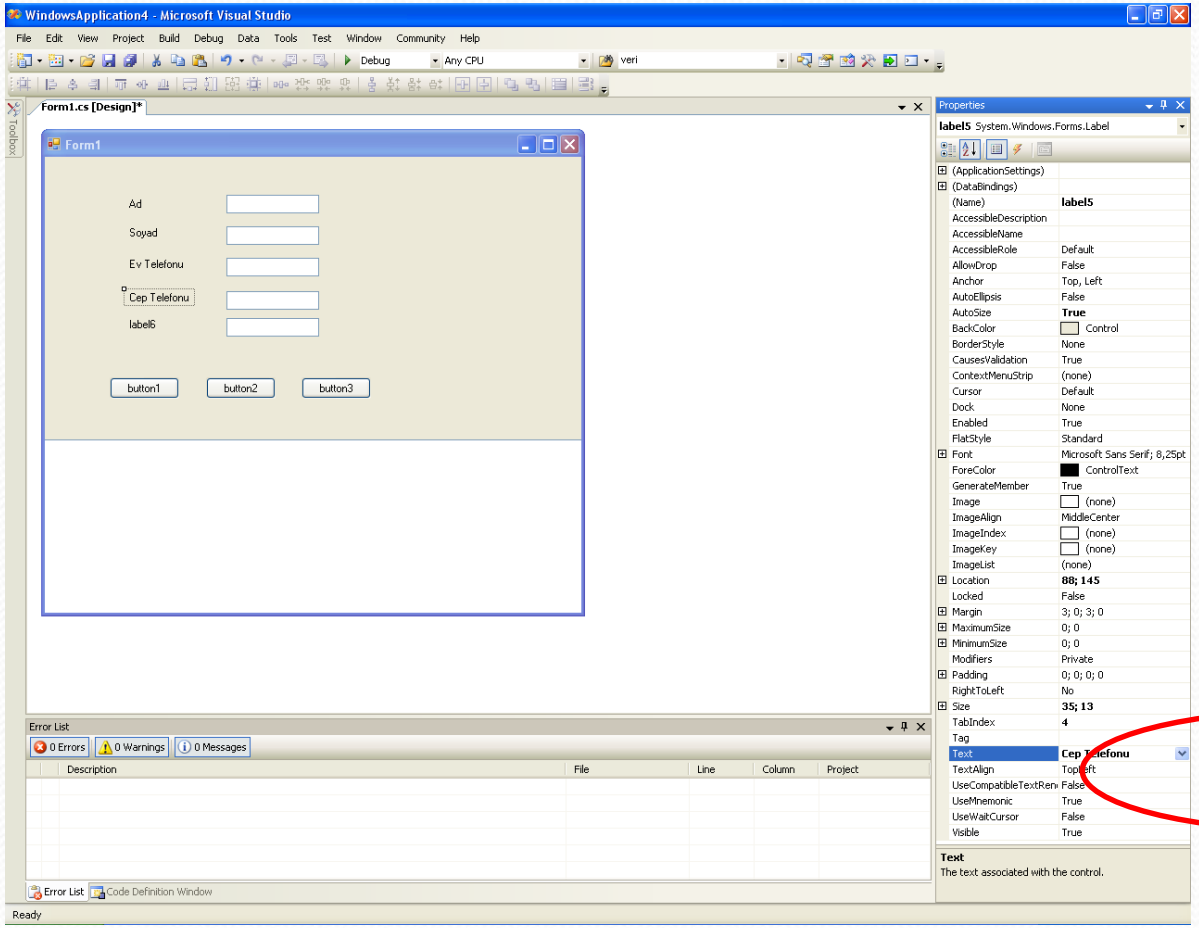

WindowsApplication4 - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Window Community Help
Debug
Any CPU
veri
Toolbox
DirectorySearcher
DomainUpDown
ErrorProvider
EventLog
FileSystemWatcher
FlowLayoutPanel
FolderBrowserDialog
FontDialog
GroupBox
HelpProvider
HScrollBar
ImageList
Label
LinkLabel
ListBox
ListView
MaskedTextBox
MenuStrip
MessageQueue
MonthCalendar
NotifyIcon
NumericUpDown
OpenFileDialog
PageSetupDialog
Panel
PerformanceCounter
PictureBox
PrintDialog
PrintDocument
PrintPreviewControl
ListView
Version 2.0.0.0 from Microsoft Corporation
.NET Component
Displays a collection of items in one of five different views.
label1


WindowsApplication4 - Microsoft Visual Studio
File Edit View Project Build Debug Data Format Tools Test Window Community Help
Debug
Any CPU
veri
Toolbox
Form1.cs [Design]*
Form1
label1
label2
label4
label3
label5
label6
button1
button2
button3

Bütün araçlarımızı ekledikten sonra labellerimizin text özelliklerini de “ad, soyad, ev telefonu…” şeklinde değiştirebiliriz. Label1’ in text özelliğine ise “id” değerini veriyoruz. Karşısına bir label daha ekliyoruz(Label7) ve bu yeni labelimizin text içeriğini boş bırakıyoruz. Label 7’ye ve textboxlarımıza kod kısmında veritabanından okuttuğumuz değerleri yazdıracağız.

Aynı şekilde butonlarımızı da isimlendiriyoruz. Sonra formumuza sağ tıklayıp “view code” seçeneğinden kod kısmına geliyoruz.

id


WindowsApplication4 - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Window Community Help
Debug
Any CPU
veri
Form1.cs [Design]*
Form1
Ad
Soyad
Ev Telefonu
Cep Telefonu
label5
button1
button2
button3
Properties
label5 System.Windows.Forms.Label
(ApplicationSettings)
(DataBindings)
(Name) label5
AccessibleDescription
AccessibleName
AccessibleRole Default
AllowDrop False
Anchor Top, Left
AutoEllipsis False
AutoSize True
BackColor Control
BorderStyle None
CausesValidation True
ContextMenuStrip (none)
Cursor Default
Dock None
Enabled True
FlatStyle Standard
Font Microsoft Sans Serif; 8,25pt
ForeColor ControlText
GenerateMember True
Image (none)
ImageAlign MiddleCenter
ImageIndex (none)
ImageKey (none)
ImageList (none)
Location 88; 145
Locked False
Margin 3; 0; 3; 0
MaximumSize 0; 0
MinimumSize 0; 0
Modifiers Private
Padding 0; 0; 0; 0
RightToLeft No
Size 35; 13
TabIndex 4
Tag
Text Cep Telefonu
TextAlign TopLeft
UseCompatibleTextRen False
UseMnemonic True
UseWaitCursor False
Visible True
Text
The text associated with the control.
Error List
0 Errors
0 Warnings
0 Messages
Description
File
Line
Column
Project
Error List
Code Definition Window
Ready

Kod bölümünde ilk olarak en yukarıda “Using System” den sonra aşağıdaki kodu ekliyoruz.

using System.Data.OleDb;

sonra ana sınıfımıza aşağıdaki kodu ekliyoruz:

OleDbConnection baglantı = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Jet OLEDB:Database Password=cagri;Data Source=cagri.mdb");

Burada “baglantı” isminde yeni bir veritabanı nesnesi oluşturduk. “Bin/Debug” klasörleri altındaki “cagri.mdb” veritabanına bağlantı sağlanacaktır. Password bölümüne de kullandığımız parolayı yazıyoruz.

Form1.cs | Form1.cs [Design]

cagrikarakas.Form1 | baglantı

```
using System;
using System.Data.OleDb;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace cagrikarakas
{
    public partial class Form1 : Form
    {
        OleDbConnection baglantı = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Jet OLEDB:Database Password=cagri;Data Source=cagri.mdb");
```

İlerde kullanmak üzere 2 adet de integer tipinde değişken tanımladıktan sonra fonksiyonlarımızı yazmaya başlayabiliriz.

cagrikarakas - Microsoft Visual Studio

File Edit View Refactor Project Build Debug Data Tools Test Window Community Help

Debug | Any CPU | id1

Form1.cs | Form1.cs [Design]

cagrikarakas.Form1 | baglanti

```
using System;
using System.Data.OleDb;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace cagrikarakas
{
    public partial class Form1 : Form
    {
        OleDbConnection baglanti = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Jet OLED
        public Form1()
        {
            InitializeComponent();
        }
        int a,b;
        private void Form1_Load(object sender, EventArgs e)
        {
            vericekme();
        }
```

```
public void vericekme() // databaseden veri cekmek ve guncellemek için kullandığım fonksiyon
    {
        try
        {
            //data baseimizdeki verileri okuma komutu
            OleDbCommand veri = new OleDbCommand("SELECT id, ad, soyad, evtel, ceptel, adres FROM data ORDER BY id",
baglantı);
            OleDbDataReader oku = null;
            baglantı.Open();//baglantıyı acar
            oku = veri.ExecuteReader();//verileri cekme komutunu calıstırır
            listView1.Items.Clear();//listview in içeriğini temizler
            while (oku.Read())//database deki verileri listview e yazdırır
            {
                ListViewItem rehberItem = new ListViewItem(oku["id"].ToString());
                rehberItem.SubItems.Add(oku["ad"].ToString());
                rehberItem.SubItems.Add(oku["soyad"].ToString());
                rehberItem.SubItems.Add(oku["evtel"].ToString());
                rehberItem.SubItems.Add(oku["ceptel"].ToString());
                rehberItem.SubItems.Add(oku["adres"].ToString());
                listView1.Items.Add(rehberItem);
            }
            oku.Close();
            baglantı.Close();//baglantıyı kapatır
        }
        catch
        {
            MessageBox.Show("Veri Tabanıyla Bağlantı Kurulamadı.\nProgramın bulunduğu dizinde cagri.mdb\ndosyasının
bulunduğundan emin olun\nve programı tekrar başlatın");
        }
    }
```

“Vericekme” fonksiyonunu formun load yordamında ve kaydet fonksiyonunda kullandım. Yani hem form yüklendiğinde ListView’e veritabanındaki bilgileri yazdıracak hem de kaydet fonksiyonunu çağırdığımda güncelleme yapacaktır. Ayrıca herhangi bir sorundan dolayı fonksiyon çalışmakta başarısız olursa “catch” bölümü devreye girecek ve yazmış olduğum mesaj ekrana gelecektir.

İkinci olarak uygulamamızdaki en basit fonksiyon olan “temizle” fonksiyonumuz:

```
private void temizle()//textboxların içeriğini temizler
    {
        textBox1.Clear(); textBox2.Clear(); textBox3.Clear(); textBox4.Clear(); textBox5.Clear();
    }
```

Bu fonksiyonumuzla da TextBox’larımızın içeriği temizlenir.

```
private void kaydet()//database e veri kaydetmek için kullandığım fonksiyon
    {
        try
        {//kayıt ekleme komutu
            OleDbCommand kaydet = new OleDbCommand("INSERT INTO data (ad, soyad, evtel, ceptel, adres) VALUES ('" + textBox1.Text + "','" + textBox2.Text + "','" + textBox3.Text + "','" + textBox4.Text + "','" + textBox5.Text + "')", baglantı);
            baglantı.Open();//database baglantısını acar
            kaydet.ExecuteNonQuery();//kaydetme komutumuzu calıstırır
            baglantı.Close();//database baglantısını kapatır
            vericekme();//listview deki verileri gunceller
            temizle();//textboxlarının içini temizler
        }
        catch
        {
            MessageBox.Show("Kayit yapılamadı. Lutfen tekrar deneyiniz.");
        }
    }
```

Bu fonksiyonumuzda veri kaydı işlemini gerçekleştiriyoruz. Yani TextBox'larımızın içeriğini SQL diliyle veritabanımızdaki ad, soyad, evtel, ceptel ve adres sütunlarımıza yazdırıyoruz. Sonra "vericekme" ve "temizle" fonksiyonlarımızı da çağırıyoruz. Böylece hem TextBox'larımızın içeriğini temizleyeceğiz hem de veritabanımızın son halini ListView'imizde görüntüleyeceğiz.

```
private void sil()//silme fonksiyonu
    {
        try
        {
            a = Convert.ToInt32(listView1.SelectedItems[0].Text);//listview de secili elemanın id sini
değişkene atar
            OleDbCommand sil = new OleDbCommand("DELETE  FROM data WHERE id=" + a + "",
baglantı);//databaseden silme komutu
            baglantı.Open();//baglantıyı acar kapaıtr
            sil.ExecuteNonQuery();//silme komutumuzu calıstırır
            baglantı.Close();//baglantıyı kapatır
            vericekme();//listview dedi verileri gunceller
        }
        catch
        {
            MessageBox.Show("Lütfen listeden silmek istediğiniz kişiyi seçiniz");
        }
```

“Sil” fonksiyonumuzda önceden belirlediğimiz a değişkenimize ListView’de tıkladığımız satırın “id” numarasını atayacağız. Sonra SQL diliyle belirlediğimiz “id” numarasına göre veritabanından silme işlemini gerçekleştiriyoruz. Son olarak yine ListView’imizi güncelliyoruz.

```
public void oku()
    {
        try
        {
            b= Convert.ToInt32(listView1.SelectedItems[0].Text);
            OleDbCommand verimiz = new OleDbCommand("SELECT id, ad, soyad, evtel, ceptel, adres FROM data WHERE id=" + b + "" , baglantı);
            OleDbDataReader oku1 = null;
            baglantı.Open();
            oku1 = verimiz.ExecuteReader();
            oku1.Read();
            {
                label7.Text=oku1["id"].ToString();
                textBox1.Text=oku1["ad"].ToString();
                textBox2.Text=oku1["soyad"].ToString();
                textBox3.Text=oku1["evtel"].ToString();
                textBox4.Text=oku1["ceptel"].ToString();
                textBox5.Text=oku1["adres"].ToString();
            }
            oku1.Close();
            baglantı.Close();//baglantıyı kapatır
        }
        catch
        {
            MessageBox.Show("Veri Tabanıyla Bağlantı Kurulamadı.\nProgramın bulunduğu dizinde cagri.mdb\ndosyasının bulunduğundan emin olun\nve programı tekrar başlatın");
        }
    }
```

Son olarak “oku” adında bir fonksiyon oluşturuyoruz. Bu fonksiyonda da yine ListView’de seçtiğimiz elemanın “id” numarasını bir değişkene atıyoruz. Bu sefer veritabanına değilde, veri tabanından okuttuğumuz verileri Label ve TextBox’larımıza yazdırıyoruz.

Son olarak oluşturmuş olduğumuz bu fonksiyonlarımızı uygun nesnelerimizin uygun yordamlarına ekliyoruz:

```
private void Form1_Load(object sender, EventArgs e)
    {
        vericekme();
    }
```

Burada formumuzun (çift tıklayarak da ulaşabileceğimiz) load yordamında (yani form yüklendiği zaman yapacağı işlem için) vericekme fonksiyonumuzu çağırıyoruz. Böylece form yüklenince vericekme çalışacak ve veritabanındaki bilgiler ListView'imize yazdırılacak.

Aynı şekilde buton1, buton2, buton3 ve ListView'in click yordamlarına uygun fonksiyonlarımızı yazıyoruz.

cagrikarakas - Microsoft Visual Studio

File Edit View Project Build Debug Data Format Tools Test Window Community Help

Debug | Any CPU | id1

Form1.cs | Form1.cs [Design]

Form1

id

Ad

Soyad

Ev Telefonu

Cep Telefonu

Adres

Kaydet | Sil | Temizle

ID	Adı	Soyadı	Ev Tel.	Cep Tel.	Adres

Properties

listView1 System.Windows.Forms.ListView

(DataBindings)
AfterLabelEdit
BackColorChanged
BackgroundImageChan
BeforeLabelEdit
BindingContextChanged
CacheVirtualItems
CausesValidationChang
ChangeUICues
Click listView1_Click
ClientSizeChanged
ColumnClick
ColumnReordered
ColumnWidthChanged
ColumnWidthChanging
ContextMenuStripChan
ControlAdded
ControlRemoved
CursorChanged
DockChanged
DoubleClick
DragDrop
DragEnter
DragLeave
DragOver
DrawColumnHeader
DrawItem
DrawSubItem
EnabledChanged
Enter
FontChanged
ForeColorChanged
GiveFeedback
HelpRequested
ImeModeChanged
ItemActivate
ItemCheck
ItemChecked
ItemDrag
ItemMouseHover
ItemSelectionChanged
KeyDown
KeyPress

Click
Occurs when the component is clicked.

Error List

0 Errors | 0 Warnings | 0 Messages

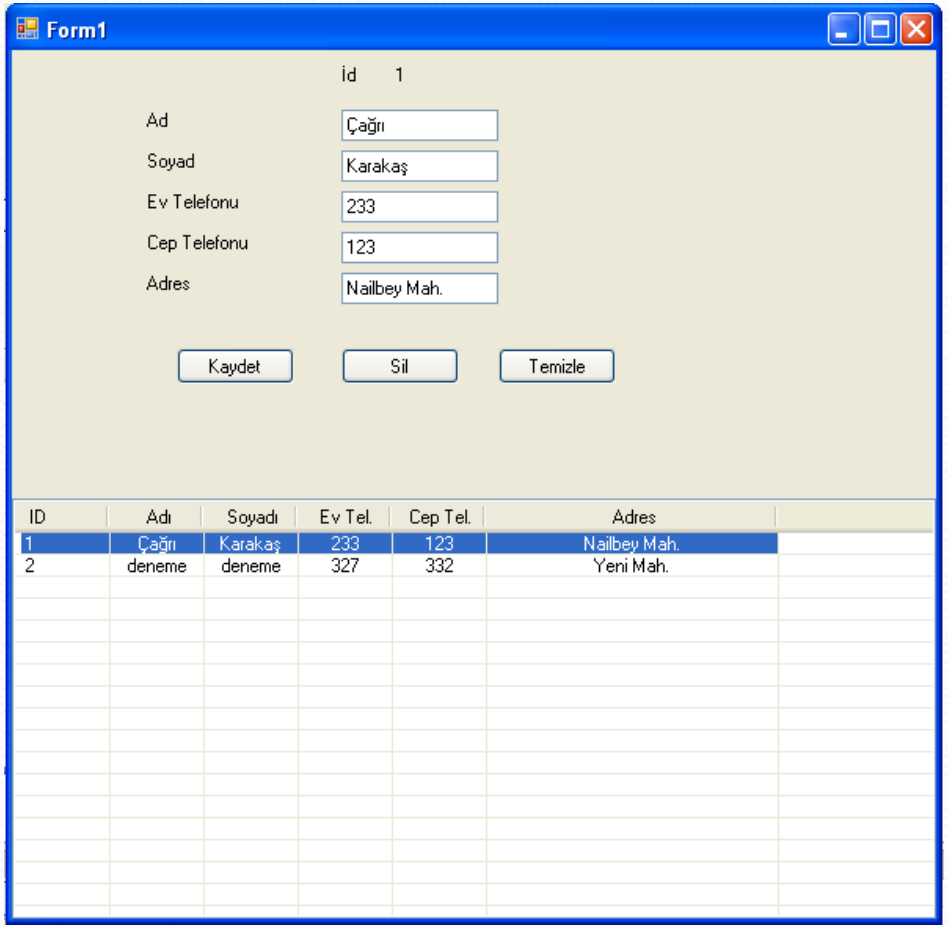
Description	File	Line	Column	Project

Error List | Code Definition Window

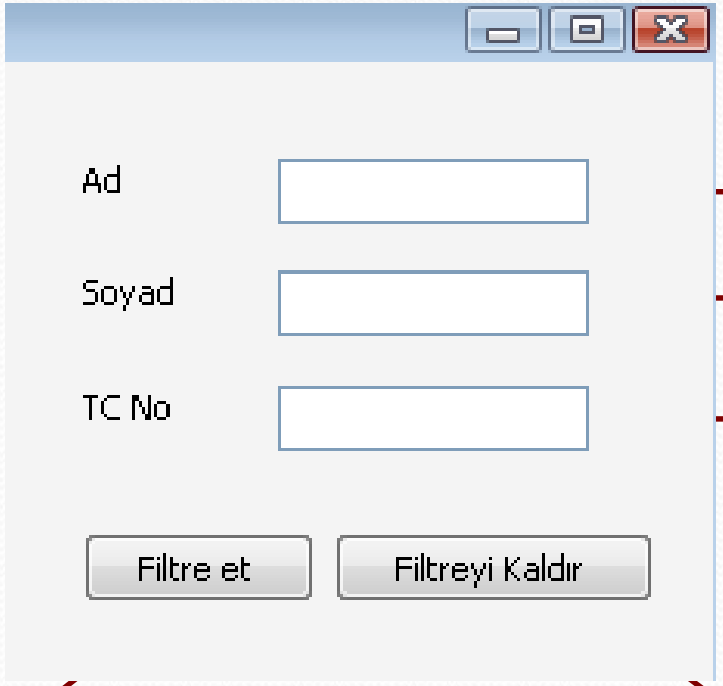
Ready

```
private void button1_Click_1(object sender, EventArgs e)
        {
            kaydet();//kaydetme fonksiyonunu calıstırır
        }

private void button2_Click(object sender, EventArgs e)
        {
            sil();//silme fonksiyonunu calıstırır
        }

        private void button3_Click(object sender, EventArgs e)
        {
            temizle();
        }

private void listView1_Click(object sender, EventArgs e)
        {
            oku();
        }
```

Sonunda formumuzun kod kısımlarını da bitirmiş olduk ve son görüntümüz aşağıdaki gibi oldu. Artık uygulamamız kullanıma ve geliştirilmeye hazır. Umarım sizlere yardımcı ve de yol gösterici olmuştur. Teşekkürler…



C# ile Filtreleme



Text7

Text8

Text9

private void button5_Click

private void button4_Click

Hangi Alana Göre Filtreleme Yapılacağını Belirleyen Kısım

```
private void button5_Click(object sender, EventArgs e)
    {
        int x=0;
        if (textBox7.Text != "") { x++; }
        if (textBox8.Text != "") { x++; }
        if (textBox9.Text != "") { x++; }
        if ((x==2) || (x==3)) { MessageBox.Show("Sadece bir türe göre arama
  yapabilirsiniz!"); }
        else
        {
            if (textBox7.Text != "") { oku1(); }
            if (textBox8.Text != "") { oku2(); }
            if (textBox9.Text != "") { oku3(); }
        }
    }
```

Ada Göre Filtreleme Yapan Bölüm

```
public void oku1()
    {
        try
        {
            OleDbCommand verimiz = new OleDbCommand("SELECT id, ad, soyad, telno, tcno, sehir, adres FROM ogrenci WHERE ad Like '" +
    textBox7.Text + "%'", baglantı);
            OleDbDataReader oku = null;
            baglantı.Open();
            oku = verimiz.ExecuteReader();
            listView1.Items.Clear();                    //listview in içeriğini temizler
            while (oku.Read())                          //database deki verileri listview e yazdırır
            {
                ListViewItem rehberItem = new ListViewItem(oku["id"].ToString());
                rehberItem.SubItems.Add(oku["ad"].ToString());
                rehberItem.SubItems.Add(oku["soyad"].ToString());
                rehberItem.SubItems.Add(oku["telno"].ToString());
                rehberItem.SubItems.Add(oku["tcno"].ToString());
                rehberItem.SubItems.Add(oku["sehir"].ToString());
                rehberItem.SubItems.Add(oku["adres"].ToString());
                listView1.Items.Add(rehberItem);
            }
            oku.Close();
            baglantı.Close();                                   //baglantıyı kapatır
        }
        catch
        {
            MessageBox.Show("Veri Tabanıyla Bağlantı Kurulamadı.\nProgramın bulunduğu dizinde 03531048.mdb\ndosyasının bulunduğundan
    emin olun\nve programı tekrar başlatın");
        }
    }
```

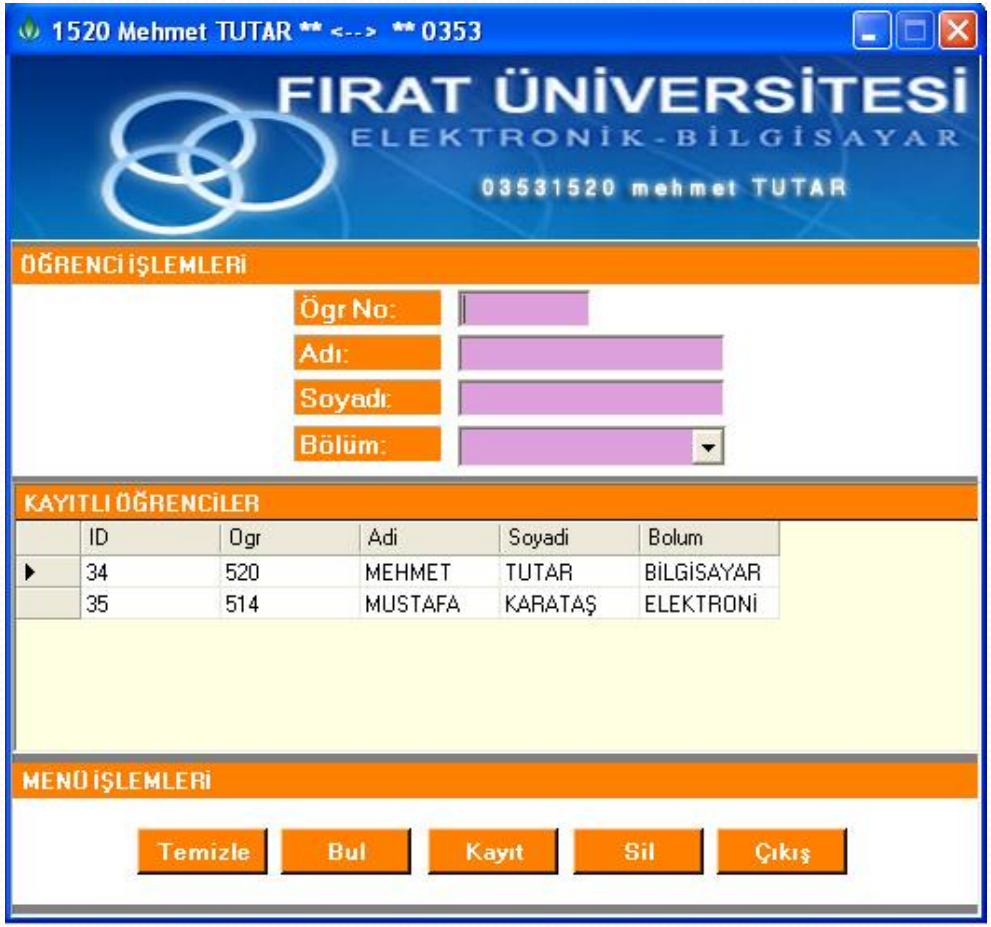
Soyada Göre Filitreleme Yapan Bölüm

```
public void oku2()
    {
      try
      {
        OleDbCommand verimiz = new OleDbCommand("SELECT id, ad, soyad, telno, tcno, sehir, adres FROM data WHERE soyad Like '" +
    textBox8.Text + "%'", baglantı);
        OleDbDataReader oku = null;
        baglantı.Open();
        oku = verimiz.ExecuteReader();
        listView1.Items.Clear();//listview in içeriğini temizler
        while (oku.Read())//database deki verileri listview e yazdırır
        {
          ListViewItem rehberItem = new ListViewItem(oku["id"].ToString());
          rehberItem.SubItems.Add(oku["ad"].ToString());
          rehberItem.SubItems.Add(oku["soyad"].ToString());
          rehberItem.SubItems.Add(oku["telno"].ToString());
          rehberItem.SubItems.Add(oku["tcno"].ToString());
          rehberItem.SubItems.Add(oku["sehir"].ToString());
          rehberItem.SubItems.Add(oku["adres"].ToString());
          listView1.Items.Add(rehberItem);
        }
        oku.Close();
        baglantı.Close();//baglantıyı kapatır
      }
      catch
      {
        MessageBox.Show("Veri Tabanıyla Bağlantı Kurulamadı.\nProgramın bulunduğu dizinde cagri.mdb\ndosyasının bulunduğundan emin
    olun\nve programı tekrar başlatın");
      }
    }
```

TC Kimlik Bölümüne Göre Filtreleme Yapan Bölüm

```
public void oku3()
    {
        try
        {
            OleDbCommand verimiz = new OleDbCommand("SELECT id, ad, soyad, telno, tcno, sehir, adres FROM data WHERE tcno Like '" +
    textBox9.Text + "%'", baglantı);
            OleDbDataReader oku = null;
            baglantı.Open();
            oku = verimiz.ExecuteReader();
            listView1.Items.Clear();//listview in içeriğini temizler
            while (oku.Read())//database deki verileri listview e yazdırır
            {
                ListViewItem rehberItem = new ListViewItem(oku["id"].ToString());
                rehberItem.SubItems.Add(oku["ad"].ToString());
                rehberItem.SubItems.Add(oku["soyad"].ToString());
                rehberItem.SubItems.Add(oku["telno"].ToString());
                rehberItem.SubItems.Add(oku["tcno"].ToString());
                rehberItem.SubItems.Add(oku["sehir"].ToString());
                rehberItem.SubItems.Add(oku["adres"].ToString());
                listView1.Items.Add(rehberItem);
            }
            oku.Close();
            baglantı.Close();//baglantıyı kapatır
        }
        catch
        {
            MessageBox.Show("Veri Tabanıyla Bağlantı Kurulamadı.\nProgramın bulunduğu dizinde cagri.mdb\ndosyasının bulunduğundan emin
    olun\nve programı tekrar başlatın");
        }
    }
```

GÖRÜNÜM

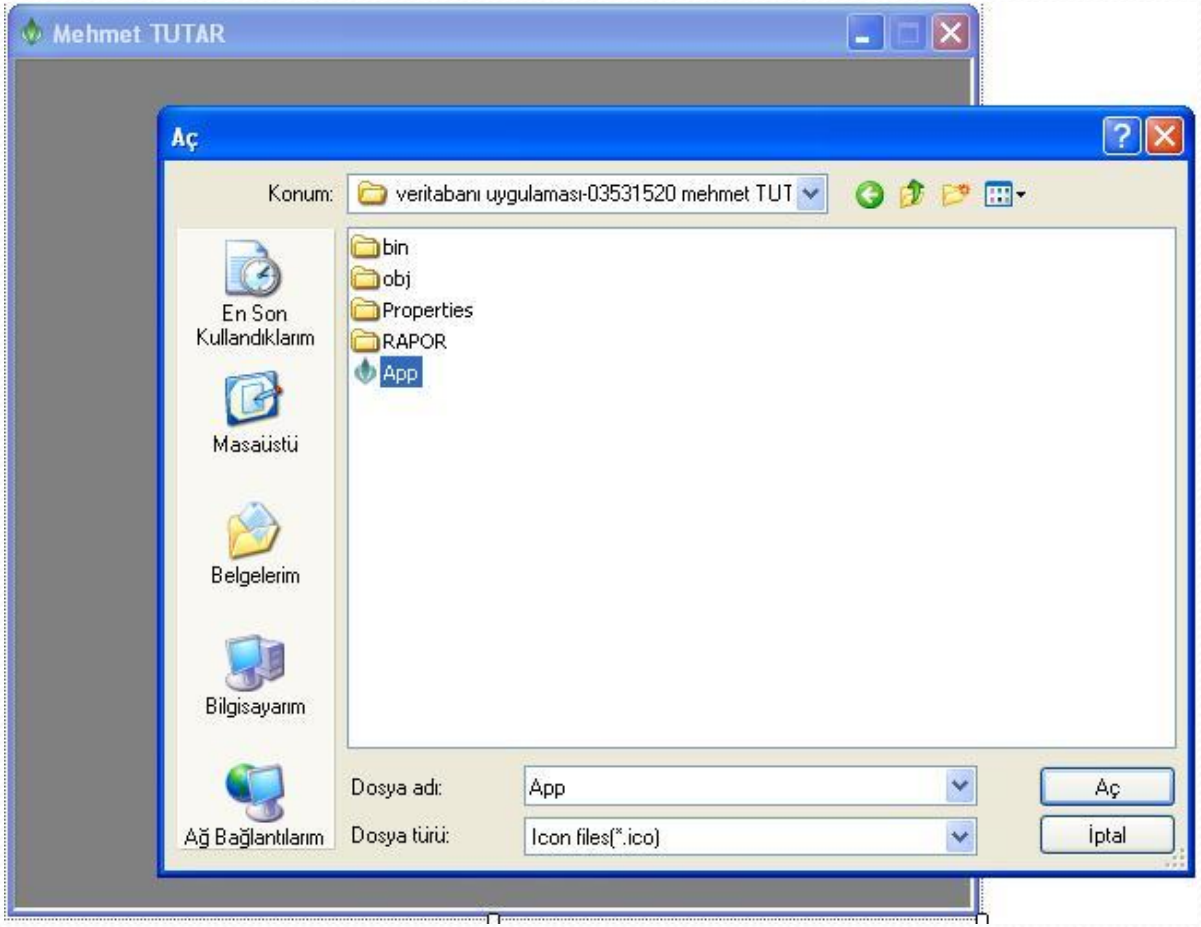

1520 Mehmet TUTAR ** <--> ** 0353
FIRAT ÜNİVERSİTESİ
ELEKTRONİK-BİLGİSAYAR
03531520 mehmet TUTAR
ÖĞRENCİ İŞLEMLERİ
Ögr No:
Adı:
Soyadı:
Bölüm:
KAYITLI ÖĞRENCİLER
ID Ogr Adi Soyadi Bolum
34 520 MEHMET TUTAR BİLGİSAYAR
35 514 MUSTAFA KARATAŞ ELEKTRONİ
MENÜ İŞLEMLERİ
Temizle
Bul
Kayıt
Sil
Çıkış

KULLANILAN ELEMANLAR:

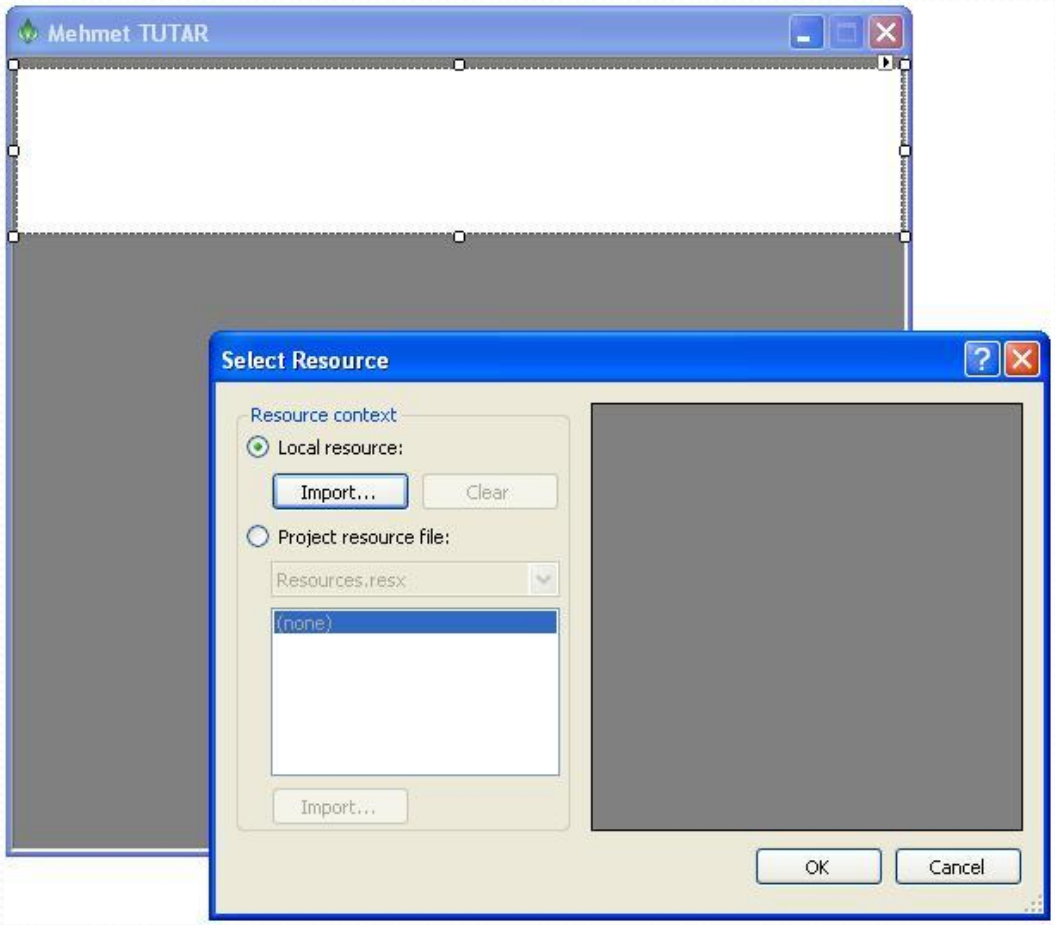
1.TextBox
2.Buton
3.Datagridview
4.Panel
5.Label
6.ComboBox
7.PictureBox
8.Timer

TASARIM KISMI:

- İlk önce formumuzun boyutunu ayarladıktan sonra, properties penceresinde bulunan icon sekmesine tıkladığımızda yandaki gibi bir görüntü ekrana gelecek. Buradan formumuzun sağ üst köşesinde bulunmasını istediğimiz ikonu seçerek Aç sekmesine tıkladığımız zaman formumuza ikonumuz eklenecektir.



- Daha sonra ToolBox penceresinde bulunan PictureBox nesnesine tıklayıp formumuzun üzerine bırakıyoruz. Daha sonra PictureBox seçili iken properties penceresinden İmage sekmesine tıklayarak karşımıza çıkan pencereden local resource tıklayıp görüntülenmesini istediğimiz resmi import diyerek uygulamamıza ekliyoruz.



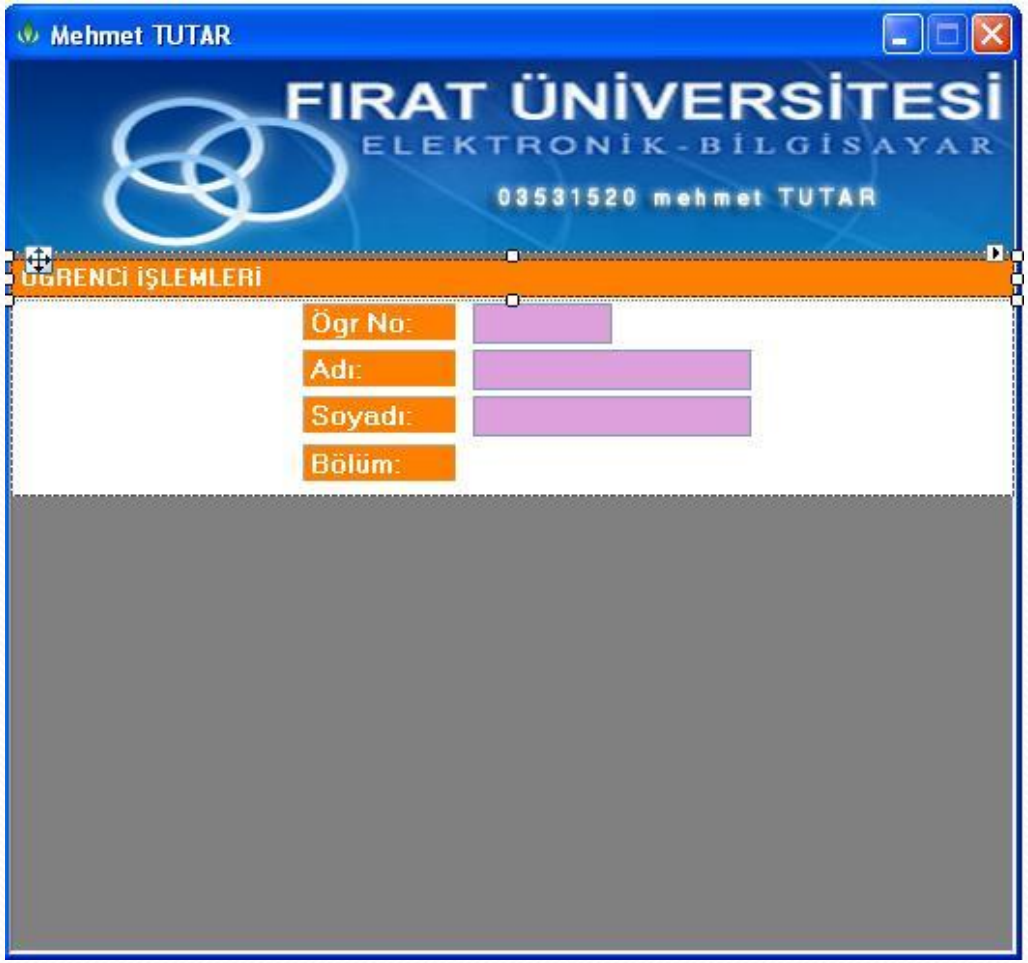
- Şekilde de görüldüğü gibi resmimiz uygulamamıza eklenmiştir.

Sonraki adımda ise Çalışmamızdaki elemanların kontrolünü kolaylaştırmak için uygulamamıza bir panel ekliyoruz.Panelin boyutlarını Mouse yardımı ile veya Properties penceresindeki size sekmesine tıklayarak ayarlayabiliriz.

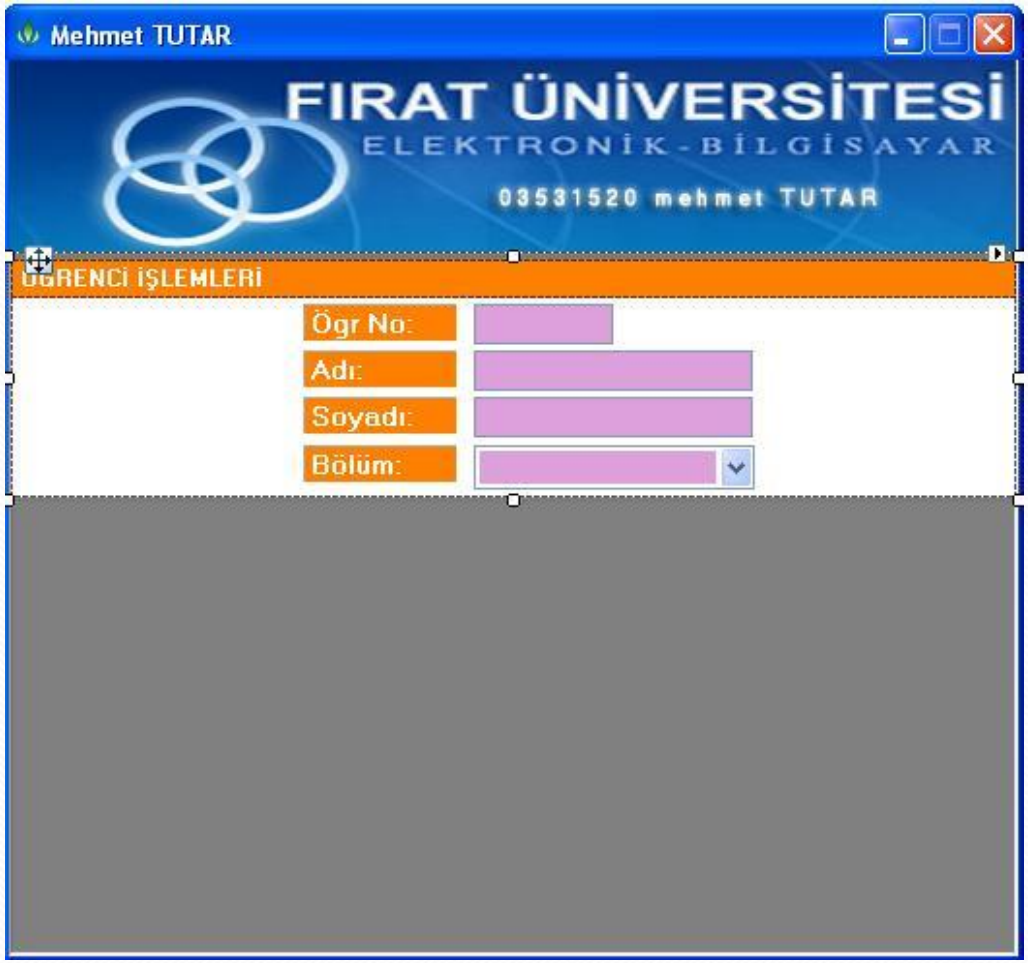
Bir sonraki adımda ise panelimizin üzerine bir panel daha ekleyerek görünüm açısından daha belirgin olmasını sağladık.Panelimizin rengini ise yine properties penceresinde bulunan Backcolor sekmesine tıklayarak seçtik. Daha sonra bu panelimizin üzerine bir label ekledik. Ve Text sekmesine tıklayarak görüntülenmesini istediğimiz yazıyı yazdık.Font sekmesine tıklayarak fontunu ve Forecolor sekmesine tıklayarak ise yazı rengini değiştirdik.

Bu adımda ise kullanacağımız elemanlarımızı panelimizin üzerine yerleştirdik. ToolBox penceresinden label ekledik.properties penceresinden Backcolor sekmesine tıklayarak arka zemin rengini değiştirdik.Font sekmesinden yazı fontunu ve Forecolor sekmesinden yazı rengini değiştirdik.

ToolBox penceresinden TextBox ekledik. Properties penceresinden Backcolor sekmesine tıklayarak arka zemin rengini değiştirdik.Font sekmesinden yazı fontunu ve Forecolor sekmesinden yazı rengini değiştirdik.

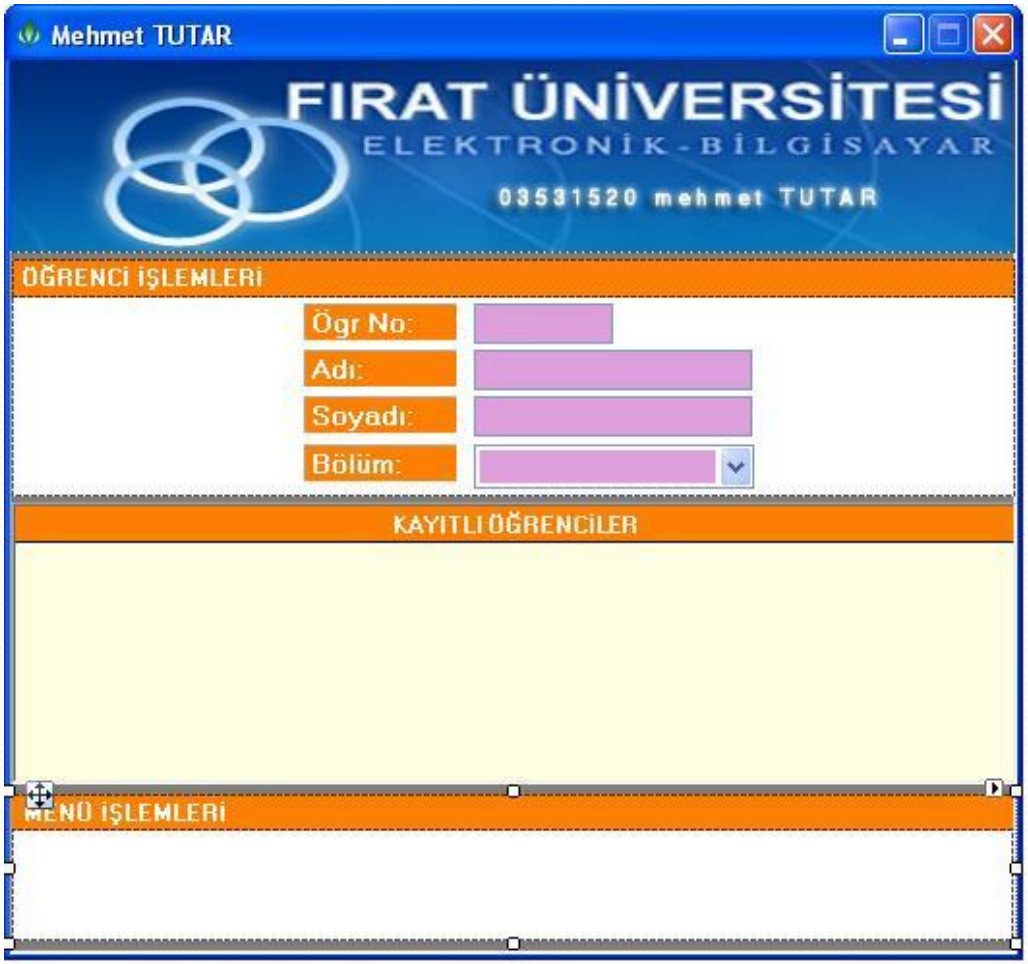


Bu adımda ToolBox penceresinden ComboBox ekledik. Properties penceresinden Backcolor sekmesine tıklayarak arka zemin rengini değiştirdik.Font sekmesinden yazı fontunu ve Forecolor sekmesinden yazı rengini değiştirdik.Görüntülen mesini istediğimiz değerleri ise properties penceresinden Items sekmesine tıklayarak değerlerimizi yazdık ve Ok seçeneğini tıkladık.

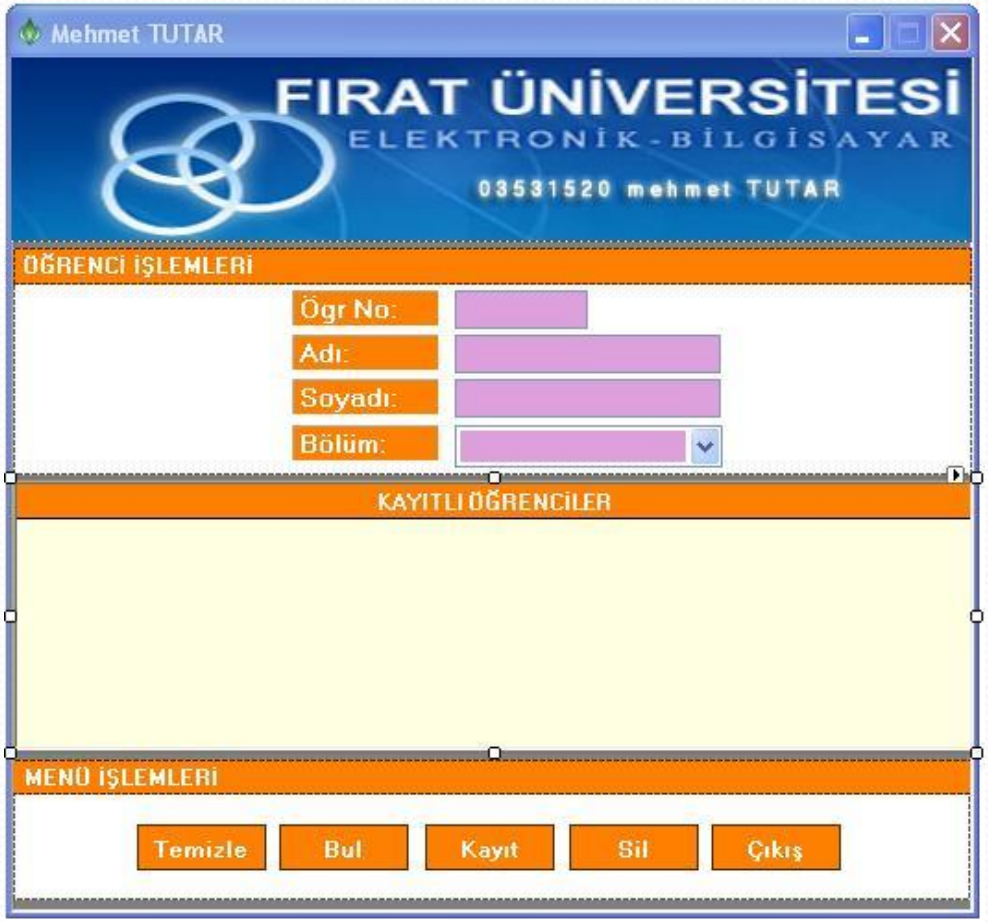


Bu adımda ise uygulamamıza bir DataGrid ekledik.Properties penceresinden CaptionText sekmesine tıklayarak Datagrid'in başlığını yazıyoruz. Daha sonra CaptionForecolor sekmesinden yazı rengini değiştiriyoruz. CaptionFont sekmesinden ise yazı fontunu değiştiriyoruz.

Sonraki adımda ise Çalışmamızdaki elemanların kontrolünü kolaylaştırmak için uygulamamıza bir panel daha ekliyoruz.Panelin boyutlarını Mouse yardımı ile veya Properties penceresindeki size sekmesine tıklayarak ayarlayabiliriz. Daha sonra ise panelimizin üzerine bir panel daha ekleyerek görünüm açısından daha belirgin olmasını sağladık.Panelimizin rengini ise yine properties penceresinde bulunan Backcolor sekmesine tıklayarak seçtik. Daha sonra bu panelimizin üzerine bir label ekledik. Ve Text sekmesine tıklayarak görüntülenmesini istediğimiz yazıyı yazdık.Font sekmesine tıklayarak fontunu ve Forecolor sekmesine tıklayarak ise yazı rengini değiştirdik.



- Bu adımda ise uygulamamızda kullanacağımız kontrolleri yerleştiriyoruz.ToolBox penceresinden Button sekmesine tıklayarak formumuzun üzerine bırakıyoruz.Daha sonra properties penceresinden Text sekmesine tıklayarak butonumuzun başlığını yazıyoruz.Font sekmesine tıklayarak yazı fontumuzu,Backcolor sekmesine tıklayarak arka planı belirliyoruz.Daha sonra properties penceresinden FlatStyle sekmesine tıklayarakbutonumuzun görüntüsünü seçiyoruz.



KOD KISMI:

- **1.BAĞLANTILAR:**

```
OleDbConnection odc;
OleDbDataAdapter adp;
DataSet dt;
OleDbCommandBuilder cb;
```

- **2.FORM_LOAD**:

```
this.Text = " ** 03531520 Mehmet TUTAR ** <--> ";
timer1.Interval = 100;
timer1.Enabled = true;

textBox2.Focus();
    string baglanti = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
  Application.StartupPath.ToString() + "\\mtutar.mdb;" + "Jet OLEDB:Database
  Password=03531520";
string sorgu="SELECT * FROM tutar";
odc = new OleDbConnection(baglanti);
adp = new OleDbDataAdapter(sorgu,odc);
dt=new DataSet();

adp.Fill(dt, "tutar");
dataGrid1.ReadOnly=true;
dataGrid1.CaptionText="KAYITLI ÖĞRENCİLER";
dataGrid1.DataSource = dt.Tables[0];
```

- **3.DATAGRİD_DOUBLECLİCK:**

```
textBox1.DataBindings.Clear();
textBox2.DataBindings.Clear();
textBox3.DataBindings.Clear();
textBox4.DataBindings.Clear();
comboBox1.DataBindings.Clear();
textBox1.DataBindings.Add("Text", dt.Tables[0],"ID");
textBox2.DataBindings.Add("Text", dt.Tables[0],"Ogr");
textBox3.DataBindings.Add("Text", dt.Tables[0],"Adi");
textBox4.DataBindings.Add("Text", dt.Tables[0],"Soyadi");
comboBox1.DataBindings.Add("Text", dt.Tables[0], "Bolum");
textBox2.Focus();
```

- **4.BUL_CLİCK:**

```
int kayit=0;
        CcurrencyManager yonet= (CurrencyManager)this.BindingContext[dt.Tables["tutar"]];
                foreach(DataRow satir in dt.Tables[0].Rows)
                {
                        if(satir["Ogr"].ToString()==textBox2.Text)
                        {
                                yonet.Position = kayit;
                                textBox1.DataBindings.Clear();
                                textBox2.DataBindings.Clear();
                                textBox3.DataBindings.Clear();
                                textBox4.DataBindings.Clear();
        comboBox1.DataBindings.Clear();

                textBox1.DataBindings.Add("Text", dt.Tables[0],"ID");
                textBox2.DataBindings.Add("Text", dt.Tables[0],"Ogr");
                textBox3.DataBindings.Add("Text", dt.Tables[0],"Adi");
                textBox4.DataBindings.Add("Text", dt.Tables[0],"Soyadi");
        comboBox1.DataBindings.Add("Text", dt.Tables[0], "Bolum");

                                textBox2.Focus();
                                return;
                        }
                        kayit++;
                        if(kayit>dt.Tables["tutar"].Rows.Count-1)
                        {
                                MessageBox.Show ("  Aradığınız Ogr No'lu Öğrenci
bulunamadı!");
                                textBox2.Focus();
                        }
                }
```

- **5.KAYIT_CLİCK:**

Kayıt

```
if (textBox2.Text == "")
        {
            MessageBox.Show("  Lütfen Alanları boş bırakmayın !!!");
            textBox2.Focus();
        }
        else
        {
            System.Data.DataRow ekle;
            cb = new OleDbCommandBuilder(adp); // OleDbDataAdapter'i silme, güncelleme ve kayıt'a hazır hale getirmek için

            ekle = dt.Tables["tutar"].NewRow();
            ekle["Ogr"] = textBox2.Text;
            ekle["Adi"] = textBox3.Text;
            ekle["Soyadi"] = textBox4.Text;
            ekle["Bolum"] = comboBox1.Text;

            dt.Tables["tutar"].Rows.Add(ekle);
            adp.Update(dt, "tutar");
            dt.Clear();
            adp.Fill(dt.Tables[0]);
            dataGrid1.DataSource = dt.Tables[0];
            MessageBox.Show(" Kayıt tamamlandı!", "İŞLEM TAMAM");
        }
```

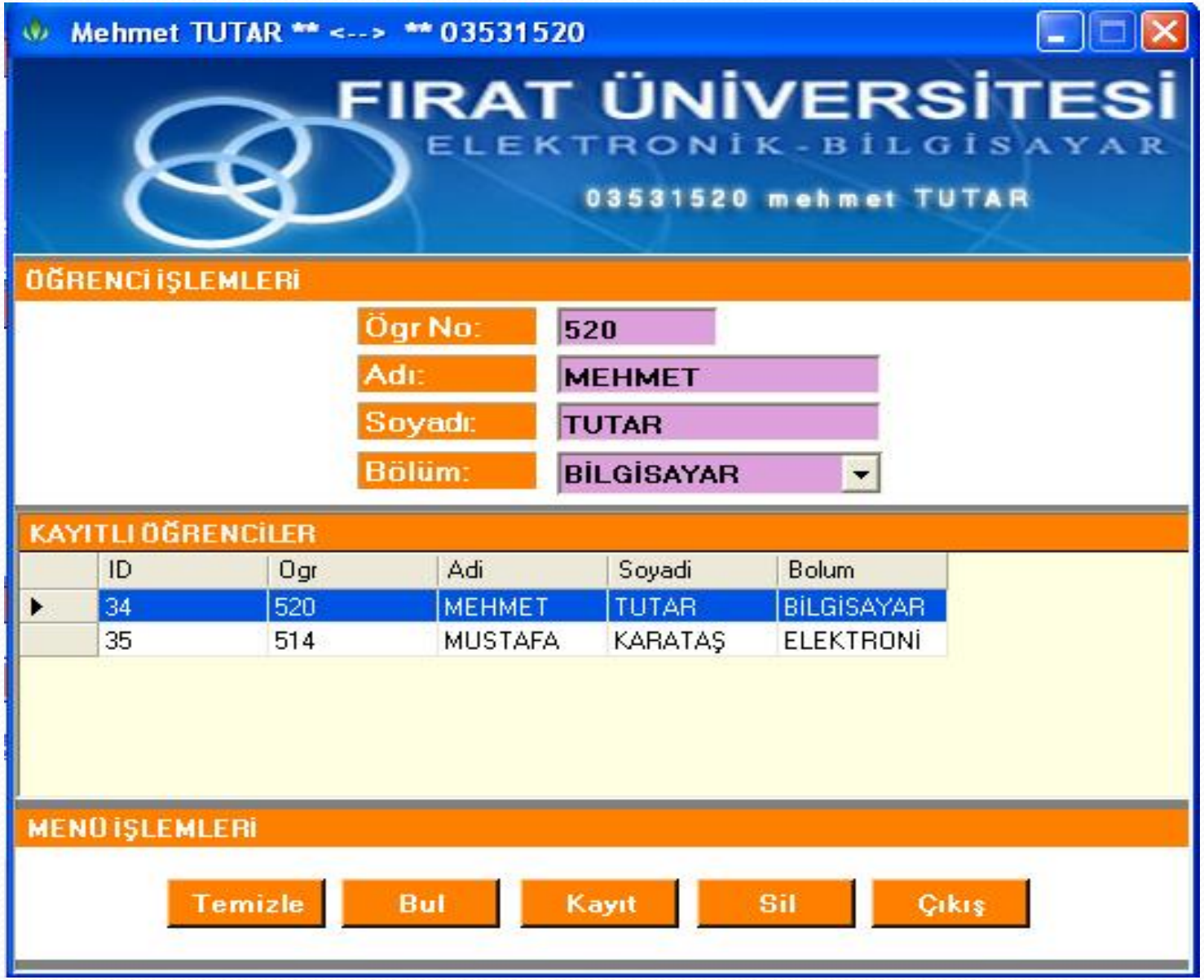
- **6.SİL_CLİCK:**

```
CurrencyManager sonraki=(CurrencyManager)this.BindingContext[dt.Tables["tutar"]];
                                        DialogResult uyar;
        uyar = MessageBox.Show(this, textBox2.Text + " Ogr nolu Öğrenci Kaydını Silmek istiyor musunuz?", "SİLME UYARISI", MessageBoxButtons.YesNo, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2);
                                        if(uyar==DialogResult.Yes)
                                        {
                                        cb=new OleDbCommandBuilder(adp); // OleDbDataAdapter'i silme, güncelleme ve kayıt'a hazır hale getirmek için
                                                sonraki.RemoveAt(sonraki.Position);
                                                adp.Update(dt, "tutar");
                                                dt.Clear();
                                                adp.Fill(dt.Tables[0]);
                                                dataGrid1.DataSource = dt.Tables[0];
                                                MessageBox.Show (" Kayıt silindi!", "İŞLEM TAMAM");
                                        }
                                        else
                                        {
                                                MessageBox.Show("  Silme işlemi İptal edildi!");
                                        }
```

- **7.TEMİZLE_CLİCK:**

```
textBox1.DataBindings.Clear();
textBox2.DataBindings.Clear();
textBox3.DataBindings.Clear();
textBox4.DataBindings.Clear();
comboBox1.DataBindings.Clear();

textBox1.Text="";
textBox2.Text="";
textBox3.Text="";
textBox4.Text="";
comboBox1.Text = "";


dt.Clear();
adp.Fill(dt.Tables[0]);
dataGrid1.DataSource = dt.Tables[0];

textBox2.Focus();
```

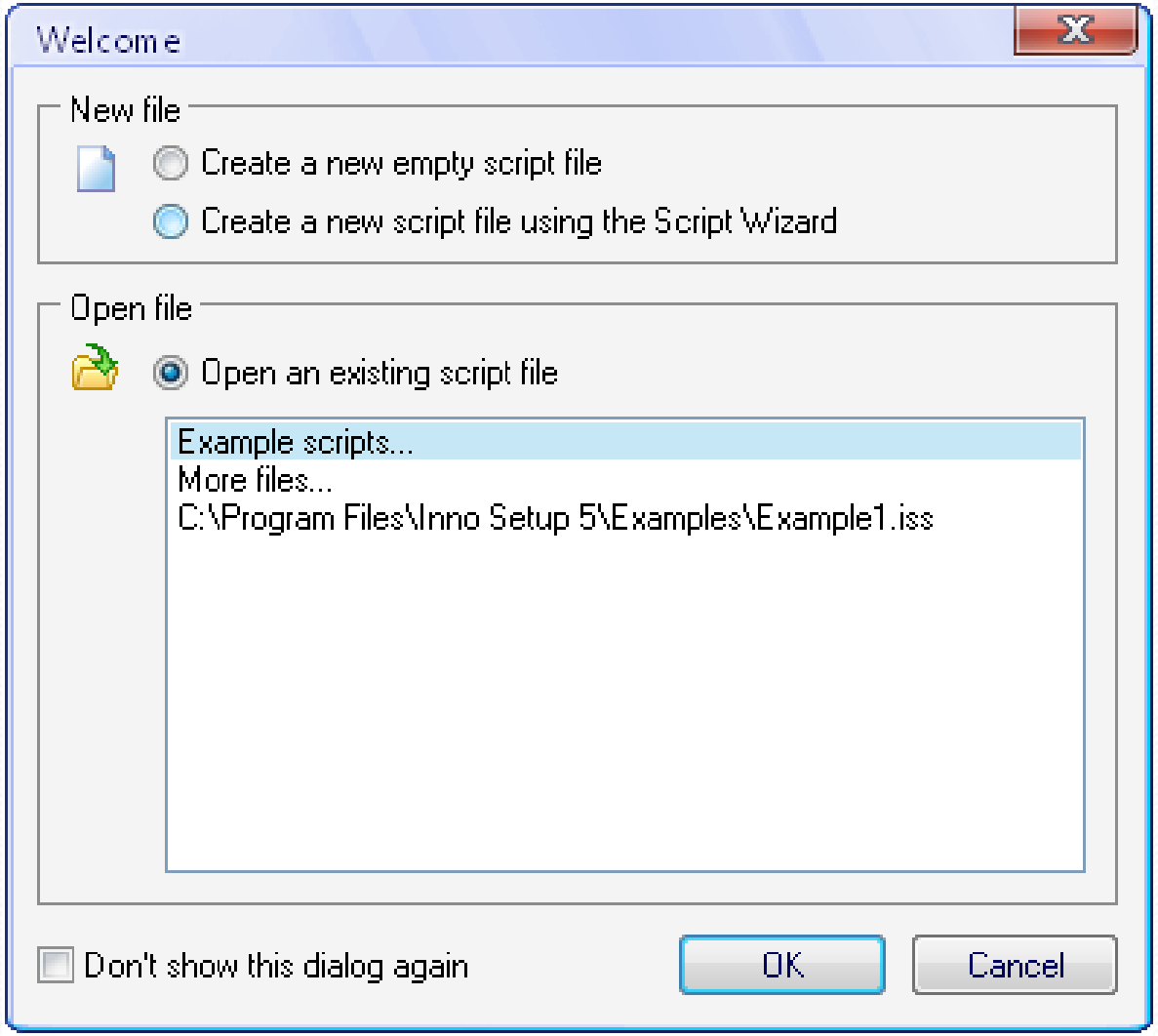
SON HALİ


Mehmet TUTAR ** <--> ** 03531520
FIRAT ÜNİVERSİTESİ
ELEKTRONİK-BİLGİSAYAR
03531520 mehmet TUTAR
ÖĞRENCİ İŞLEMLERİ
Ögr No:
520
Adı:
MEHMET
Soyadı:
TUTAR
Bölüm:
BİLGİSAYAR
KAYITLI ÖĞRENCİLER
ID
Ogr
Adi
Soyadi
Bolum
34
520
MEHMET
TUTAR
BİLGİSAYAR
35
514
MUSTAFA
KARATAŞ
ELEKTRONİ
MENÜ İŞLEMLERİ
Temizle
Bul
Kayıt
Sil
Çıkış

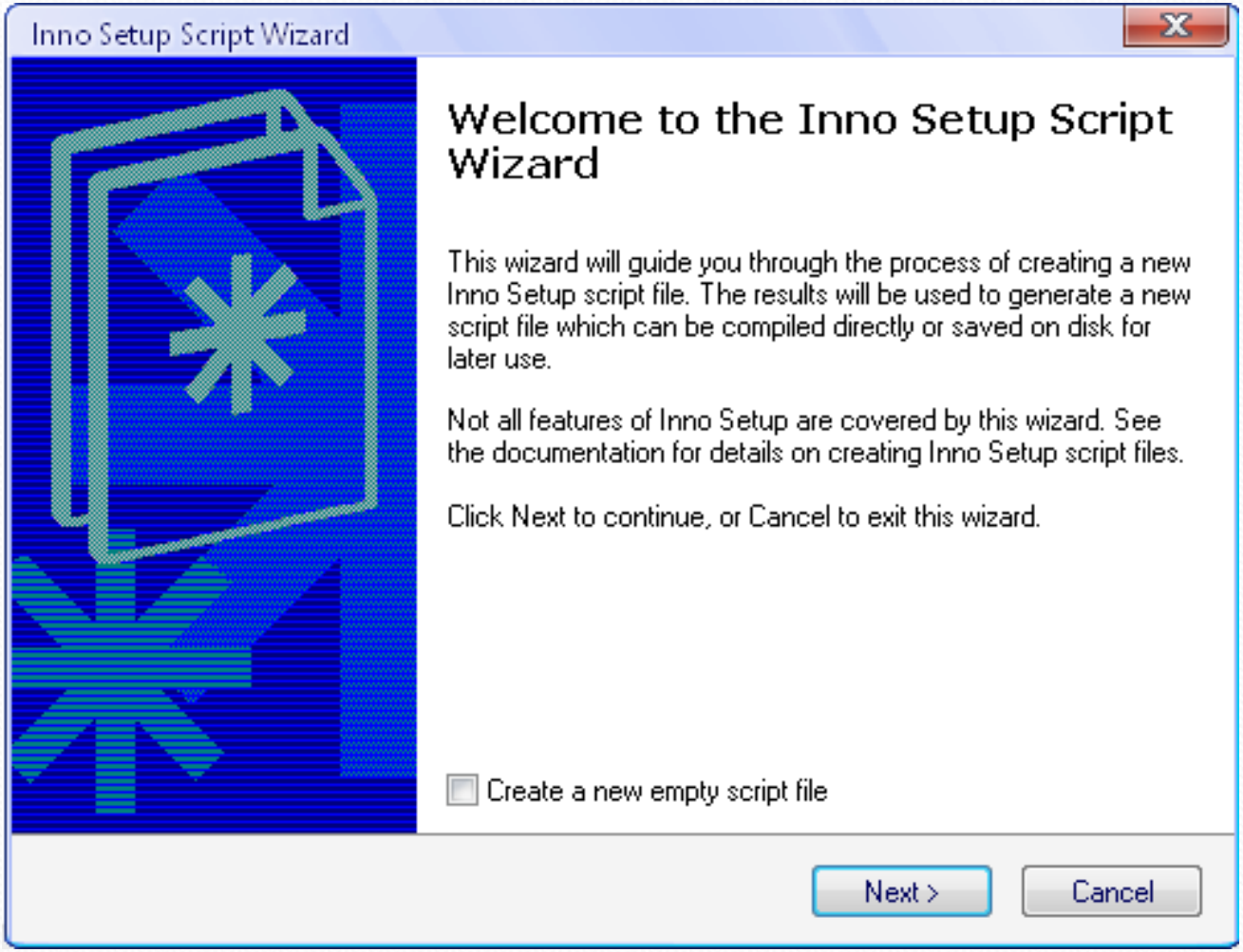
C# SETUP HAZIRLAMA

Inno Setup

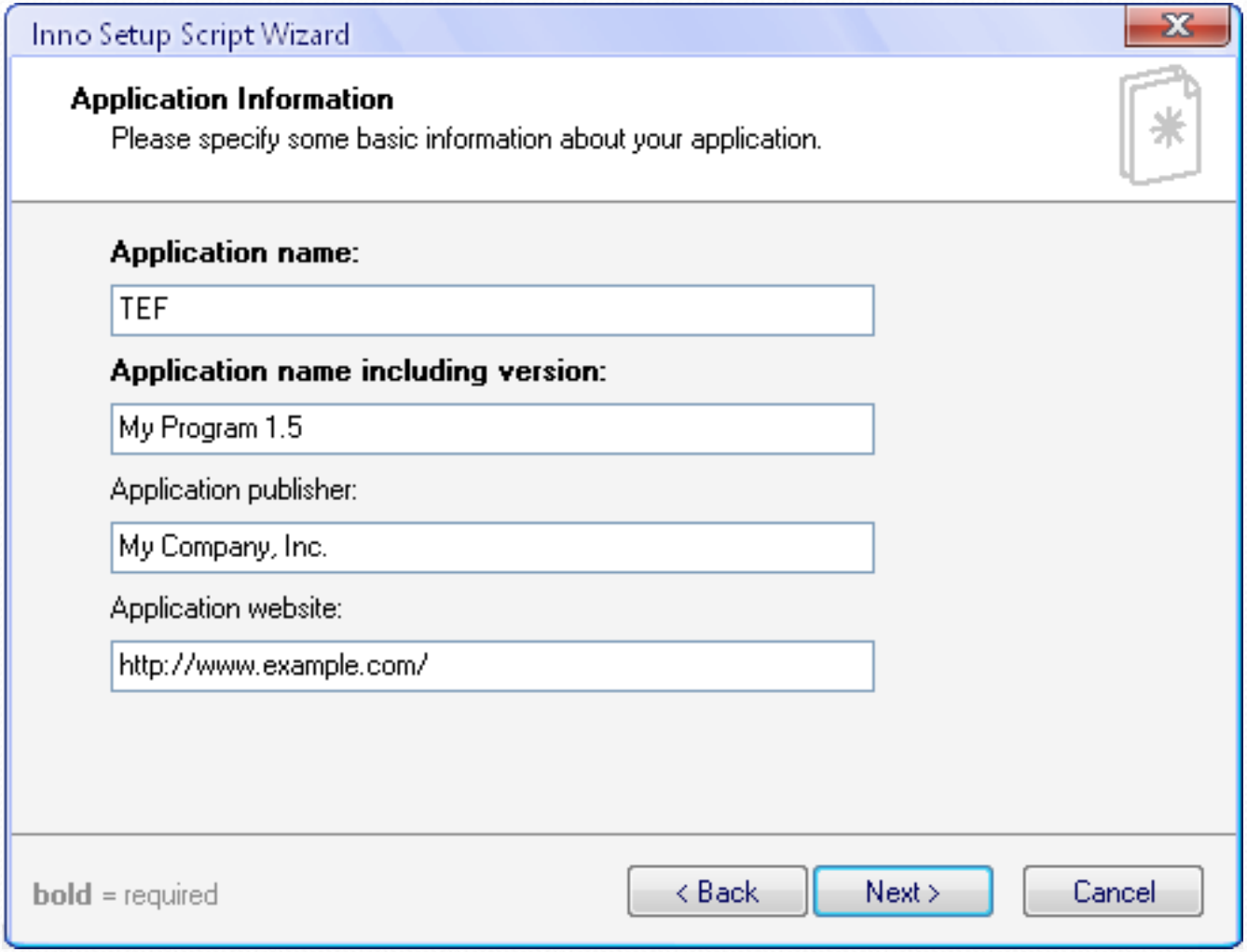
Inno Setup


Welcome
New file
Create a new empty script file
Create a new script file using the Script Wizard
Open file
Open an existing script file
Example scripts...
More files...
C:\Program Files\Inno Setup 5\Examples\Example1.iss
Don't show this dialog again
OK
Cancel

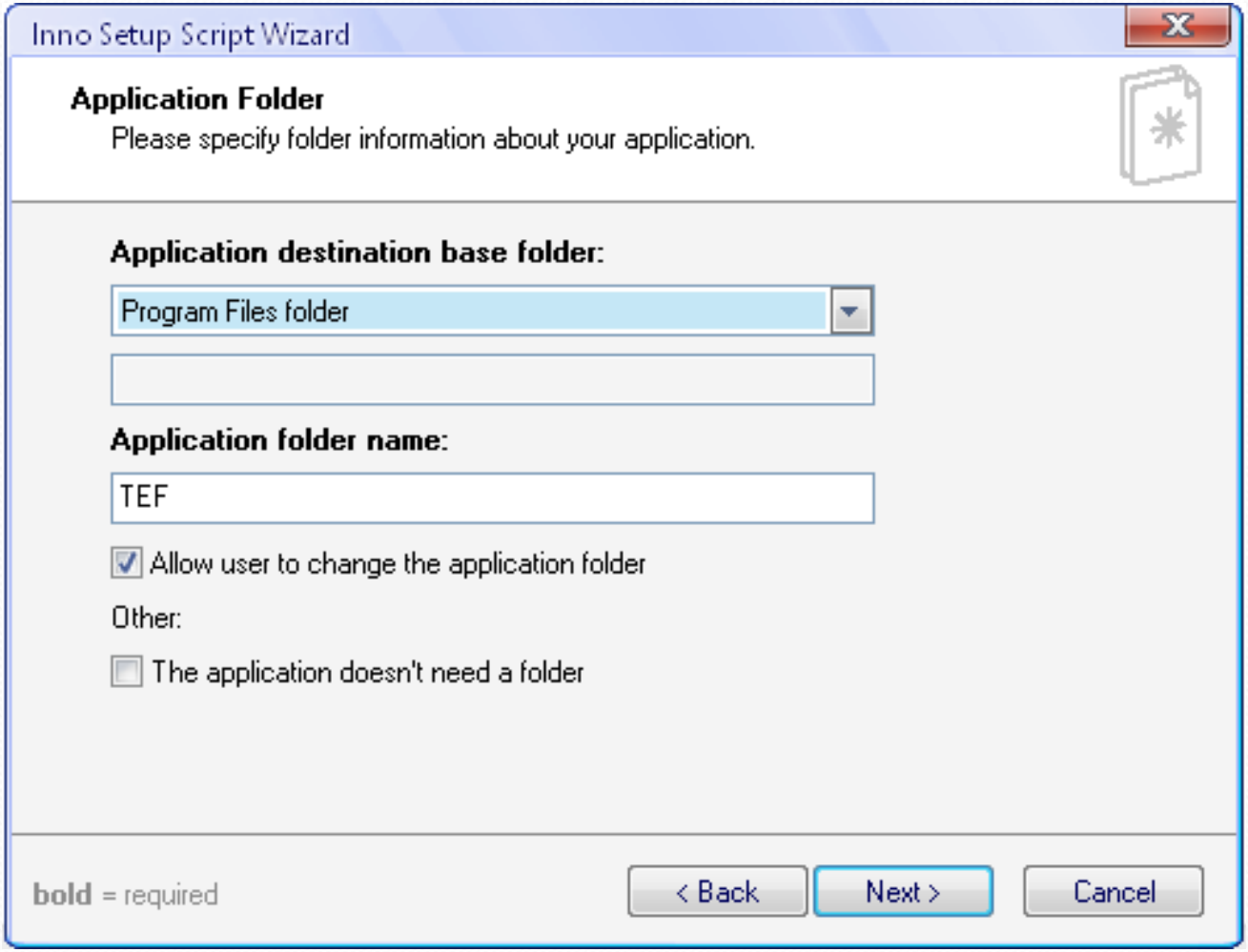
Inno Setup


Inno Setup Script Wizard
Welcome to the Inno Setup Script Wizard
This wizard will guide you through the process of creating a new Inno Setup script file. The results will be used to generate a new script file which can be compiled directly or saved on disk for later use.
Not all features of Inno Setup are covered by this wizard. See the documentation for details on creating Inno Setup script files.
Click Next to continue, or Cancel to exit this wizard.
Create a new empty script file
Next >
Cancel

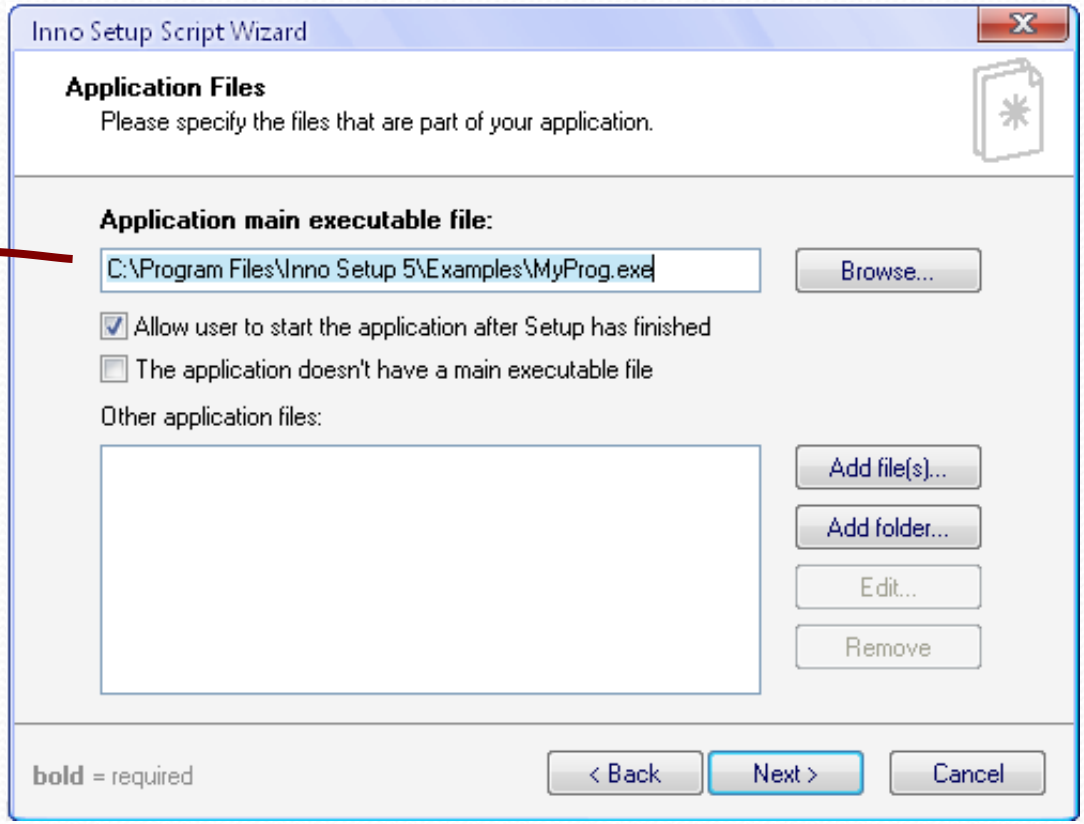
Inno Setup


Inno Setup Script Wizard
Application Information
Please specify some basic information about your application.
Application name:
TEF
Application name including version:
My Program 1.5
Application publisher:
My Company, Inc.
Application website:
http://www.example.com/
bold = required
< Back
Next >
Cancel

Inno Setup

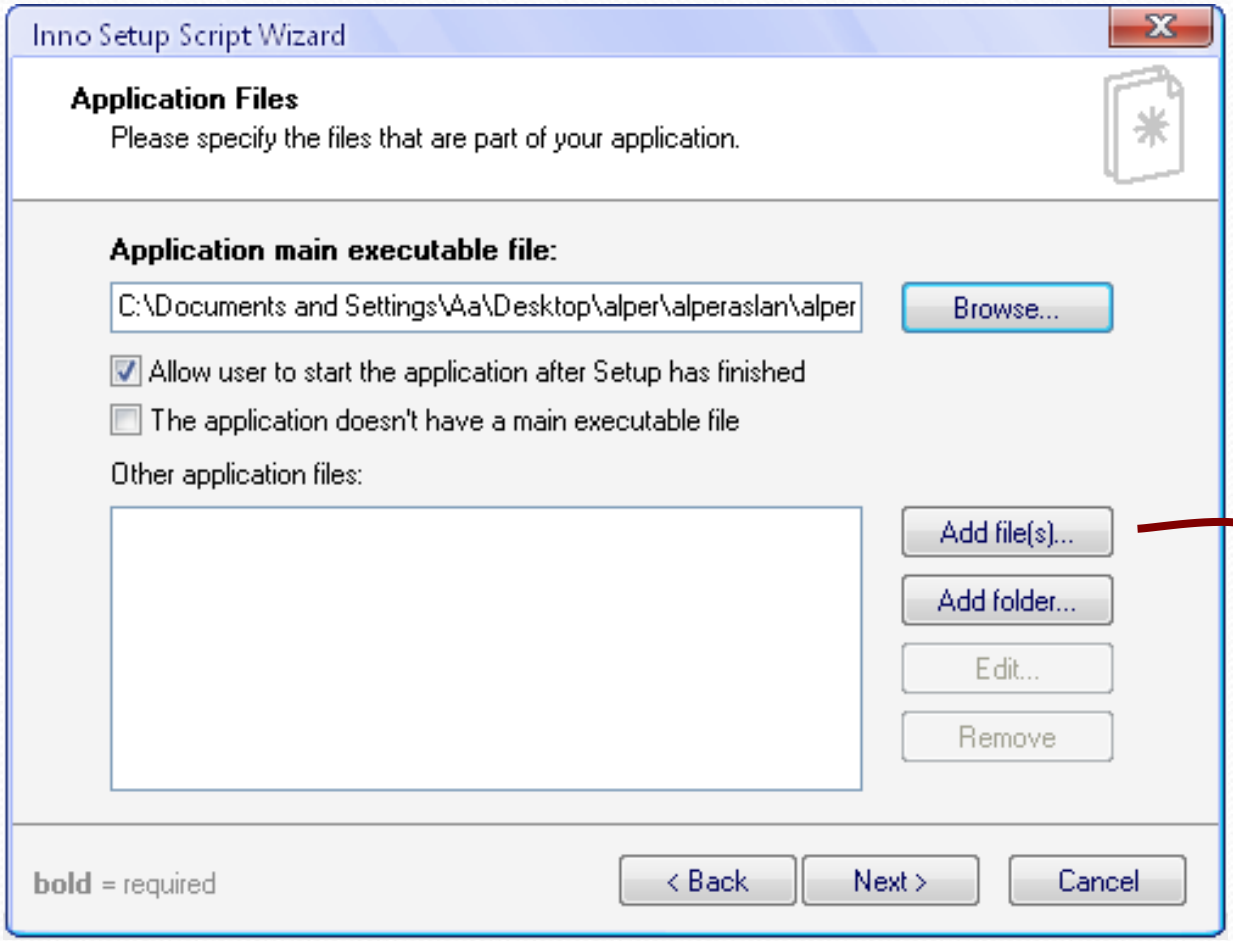

Inno Setup Script Wizard
Application Folder
Please specify folder information about your application.
Application destination base folder:
Program Files folder
Application folder name:
TEF
Allow user to change the application folder
Other:
The application doesn't need a folder
bold = required
< Back
Next >
Cancel

Inno Setup



Bu kısımda projemizin yolunu belirtiyoruz. (.exe)

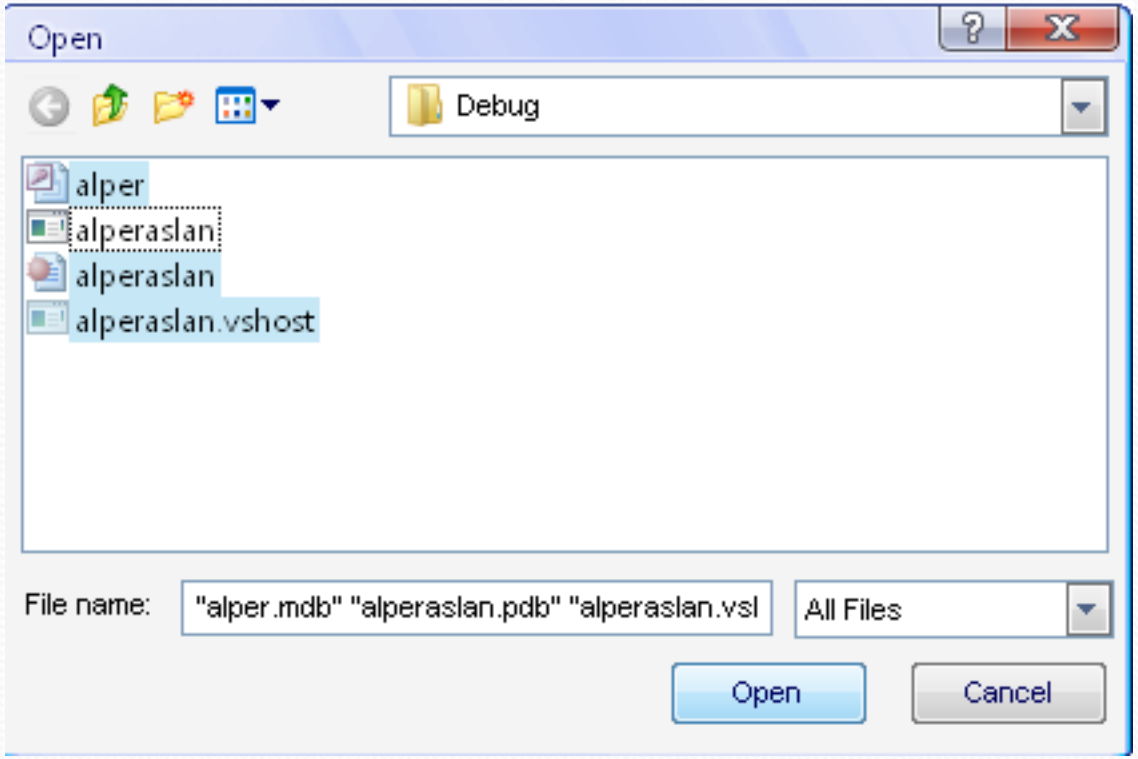
Inno Setup



Add file(s) kısmını kullanarak setup için gerekli diğer dosyaları ekliyoruz.

Inno Setup

.exe uzantılı dosyayı daha önce eklemiştik.



.NET

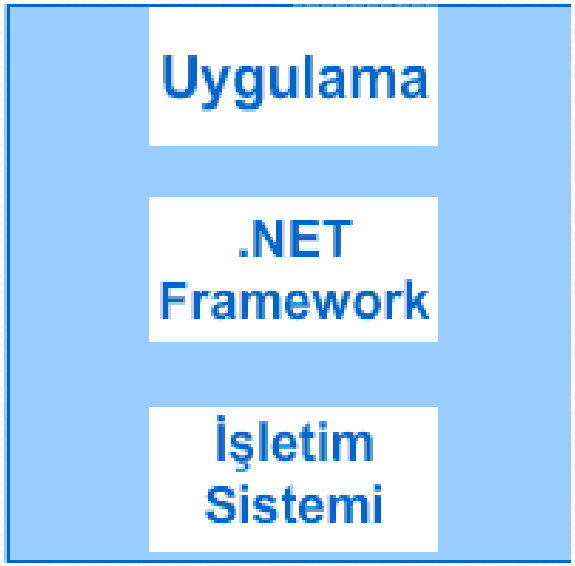
Microsoft tarafından geliştirilen .NET ileri düzey programcılığı tek bir çatı altında toplamayı hedefleyen platformdur.

.NET iki hedefi gerçekleştirmiş oldu;

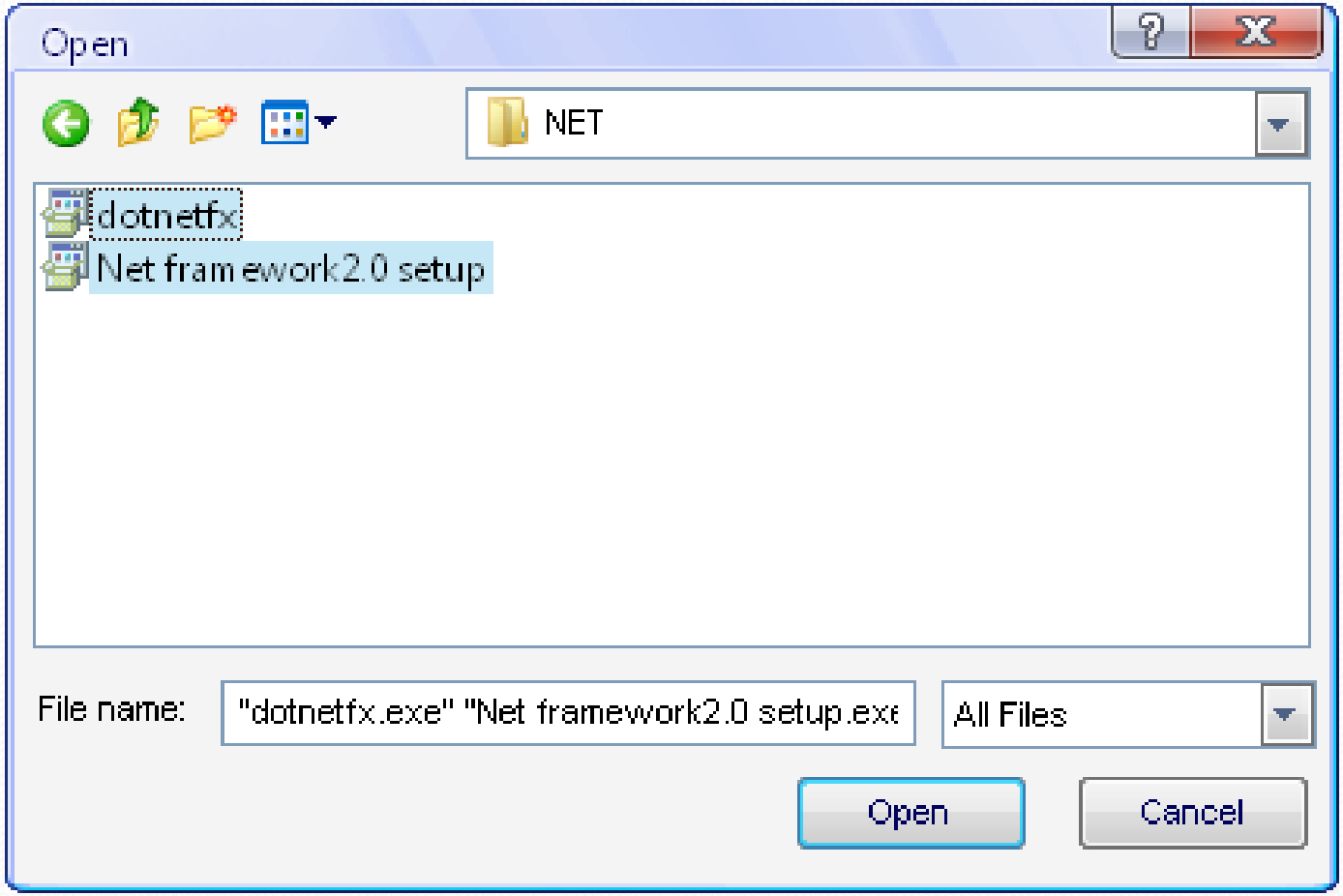
- Farklı programlama dillerinden standart ara kod oluşturmak
- Standart ara kodun hedef dile veya makine diline çevrilmesini tek bir noktadan gerçekleştirmek.

.NET Framework

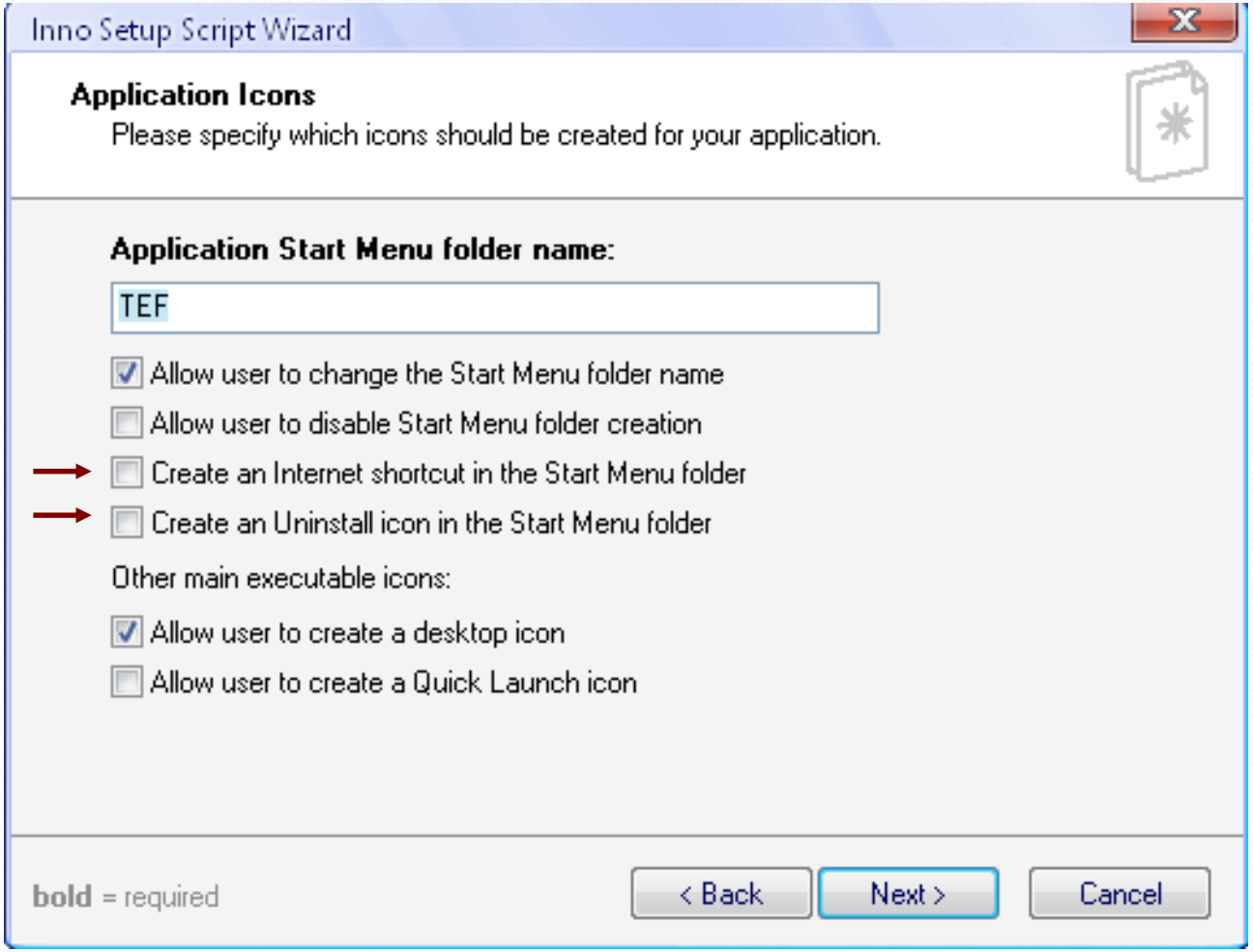
.NET Framework yaptığımız uygulamaların derleyip işletim sistemine uygun şekilde çalıştırılmasını amaçlar.



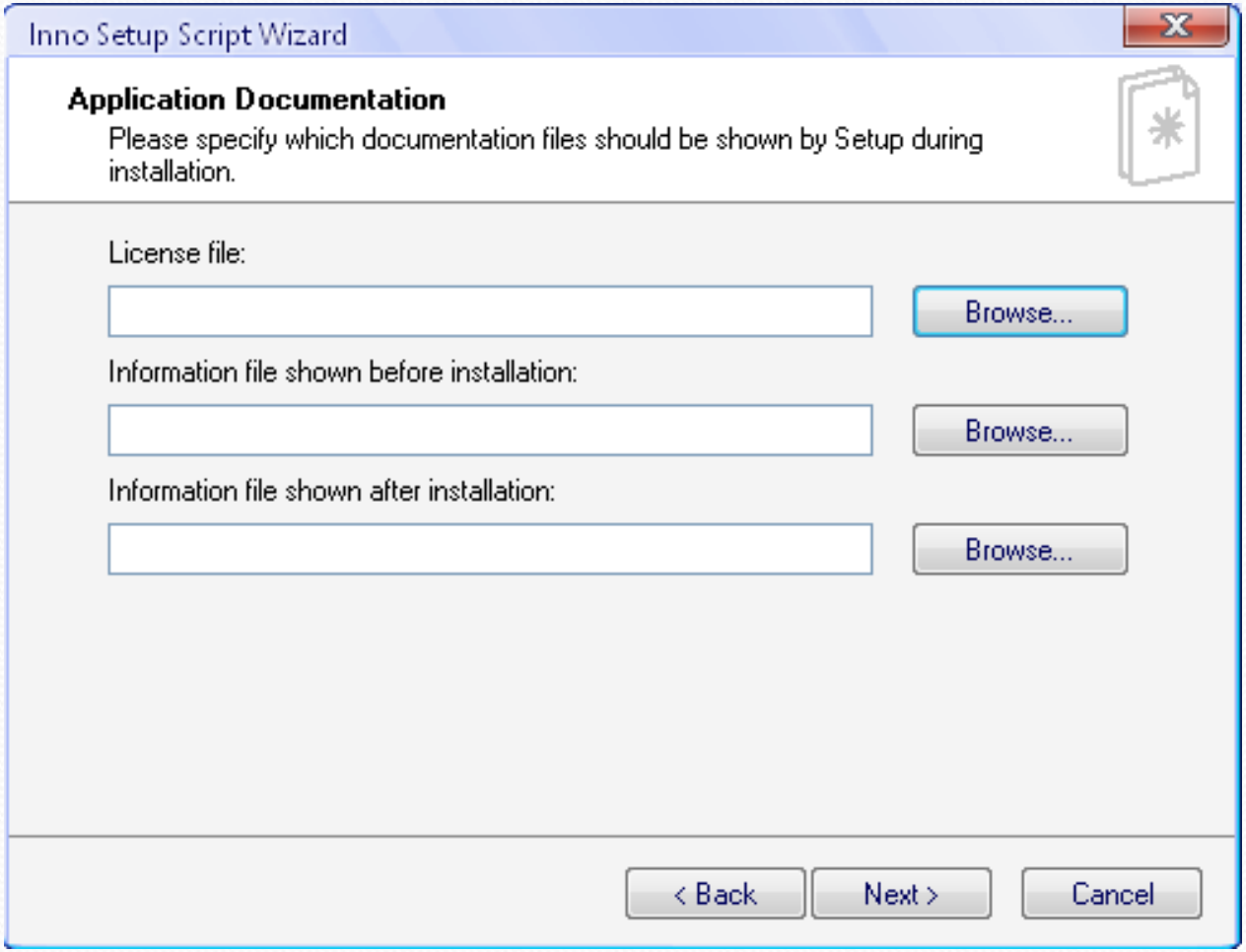
Inno Setup


Open
NET
dotnetfx
Net framework2.0 setup
File name:
"dotnetfx.exe" "Net framework2.0 setup.exe
All Files
Open
Cancel

Inno Setup

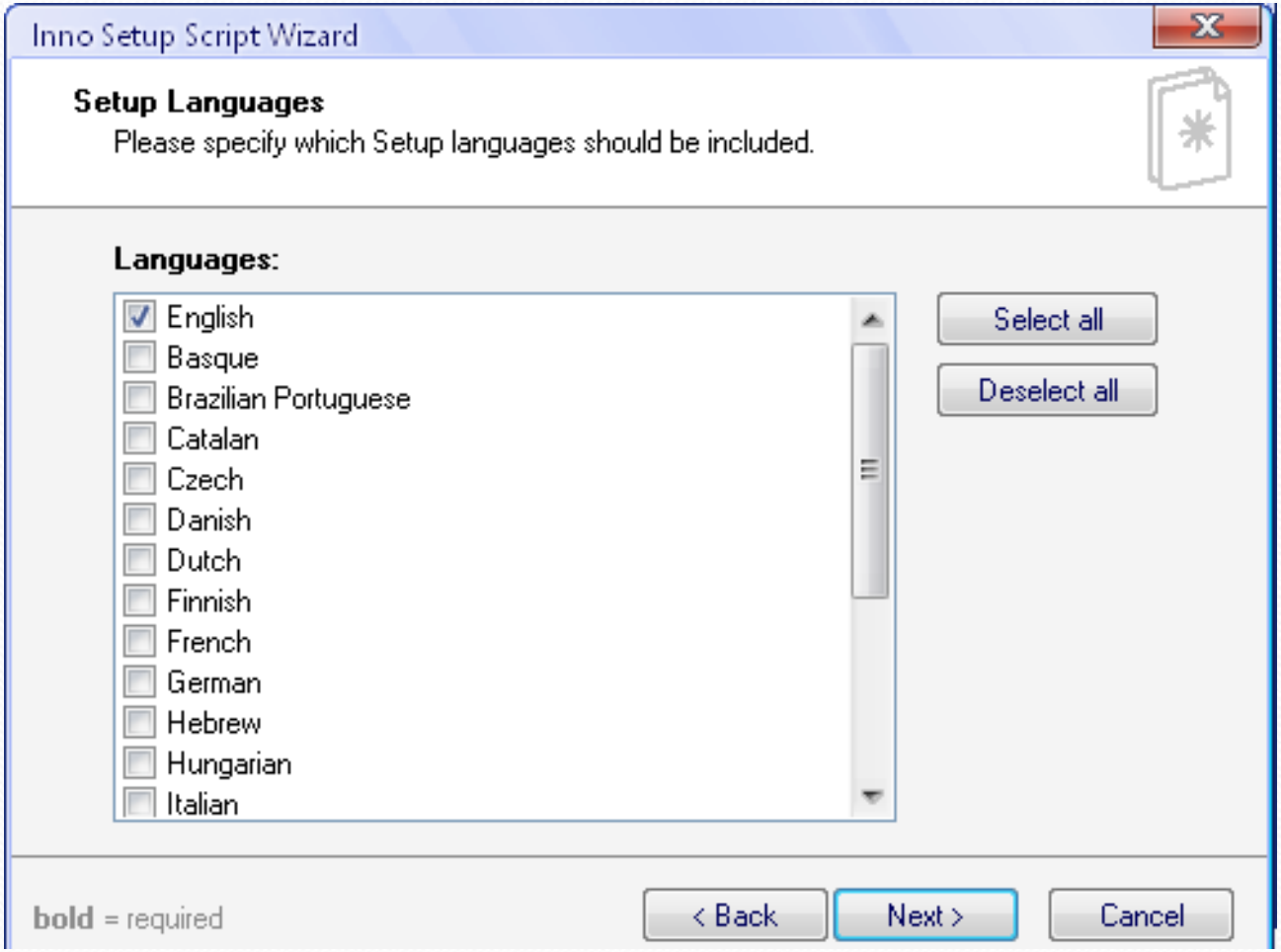

Inno Setup Script Wizard
Application Icons
Please specify which icons should be created for your application.
Application Start Menu folder name:
TEF
Allow user to change the Start Menu folder name
Allow user to disable Start Menu folder creation
Create an Internet shortcut in the Start Menu folder
Create an Uninstall icon in the Start Menu folder
Other main executable icons:
Allow user to create a desktop icon
Allow user to create a Quick Launch icon
bold = required
< Back
Next >
Cancel

Inno Setup



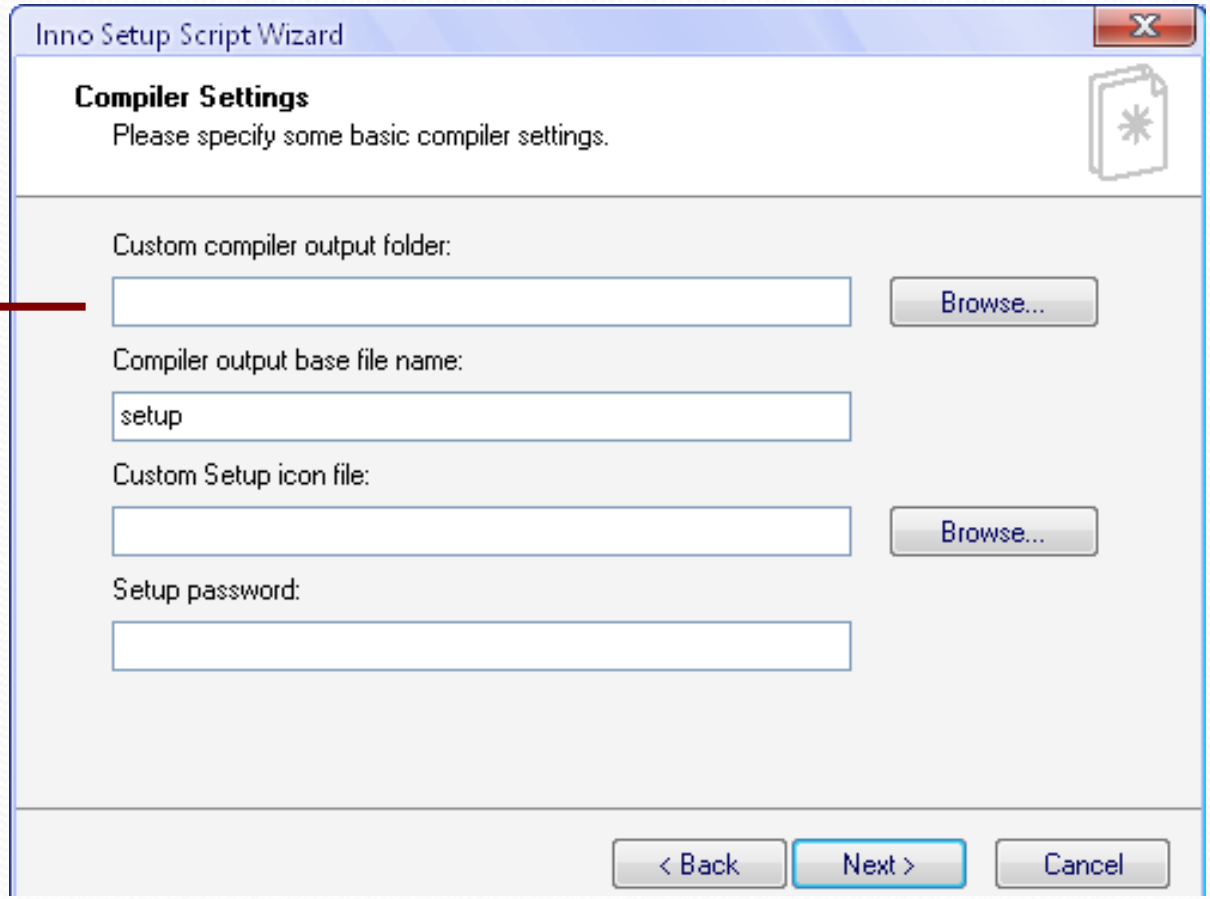
Bu kısımları geçiyoruz…

Inno Setup



Bu kısımları geçiyoruz…

Inno Setup



Bu kısımda setup dosyamızın oluşturulacağı klasörü seçiyoruz…

Untitled - Inno Setup Compiler 5.1.12

File Edit View Build Run Tools Help

```
; Script generated by the Inno Setup Script Wizard.
; SEE THE DOCUMENTATION FOR DETAILS ON CREATING INNO SETUP SCRIPT FILES!

[Setup]
AppName=TEF
AppVerName=My Program 1.5
AppPublisher=My Company, Inc.
AppPublisherURL=http://www.example.com/
AppSupportURL=http://www.example.com/
AppUpdatesURL=http://www.example.com/
DefaultDirName={pf}\TEF
DefaultGroupName=TEF
OutputDir=C:\Documents and Settings\Aa\Desktop\alper
OutputBaseFilename=setup
Compression=lzma
SolidCompression=yes

[Languages]
Name: "english"; MessagesFile: "compiler:Default.isl"

[Tasks]
Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}"; GroupDescription: "{cm:AdditionalIcons}"; Flags: unchecked

[Files]
Source: "C:\Documents and Settings\Aa\Desktop\alper\alperaslan\alperaslan\bin\Debug\alperaslan.exe"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Documents and Settings\Aa\Desktop\alper\alperaslan\alperaslan\bin\Debug\alper.mdb"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Documents and Settings\Aa\Desktop\alper\alperaslan\alperaslan\bin\Debug\alperaslan.pdb"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Documents and Settings\Aa\Desktop\alper\alperaslan\alperaslan\bin\Debug\alperaslan.vshost.exe"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Documents and Settings\Aa\Desktop\NET\dotnetfx.exe"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Documents and Settings\Aa\Desktop\NET\Net framework2.0 setup.exe"; DestDir: "{app}"; Flags: ignoreversion
; NOTE: Don't use "Flags: ignoreversion" on any shared system files

[Icons]
Name: "{group}\TEF"; Filename: "{app}\alperaslan.exe"
Name: "{group}\{cm:ProgramOnTheWeb,TEF}"; Filename: "http://www.example.com/"
Name: "{group}\{cm:UninstallProgram,TEF}"; Filename: "{uninstallexe}"
Name: "{commondesktop}\TEF"; Filename: "{app}\alperaslan.exe"; Tasks: desktopicon

[Run]
Filename: "{app}\alperaslan.exe"; Description: "{cm:LaunchProgram,TEF}"; Flags: nowait postinstall skipifsilent
```

1: 1 Modified Insert

Inno Setup

```
[Run]
Filename:Filename: "{app}\alperaslan.exe"; Description: "{cm:LaunchProgram,TEF}"; ogram,TEF}";
```

Setup sonra çalıştırılacak .exe leri RUN kısmında belirtiyoruz.
Bu kısma .Net Framework dosyasının .exe kısmınıda kopyalıyoruz.

Inno Setup

```
[Run]
Filename:Filename: "{app}\alperaslan.exe"; Description: "{cm:LaunchProgram,TEF}"; ogram,TEF}";
```

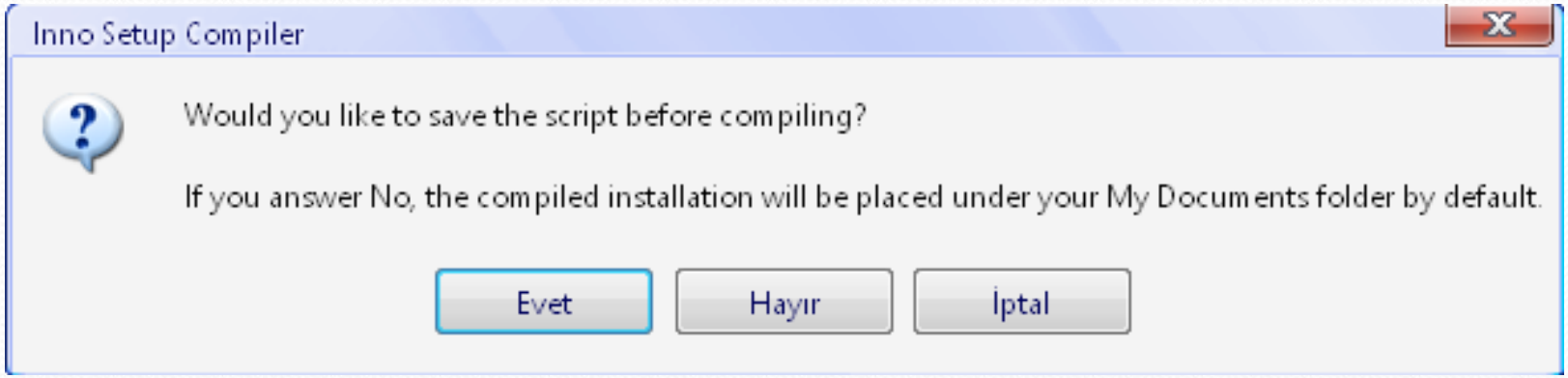
Setup sonra çalıştırılacak .exe leri RUN kısmında belirtiyoruz.

```
[Run]
Filename: "{app}\dotnetfx.exe"; Description: "{cm:LaunchProgram,dotnetfx.exe}"; Flags: nowait postinstall sk
Filename: "{app}\Net framework2.0 setup.exe"; Description: "{cm:LaunchProgram,Net framework2.0 setup.exe}";
```

Inno Setup

Compile seçiyoruz.

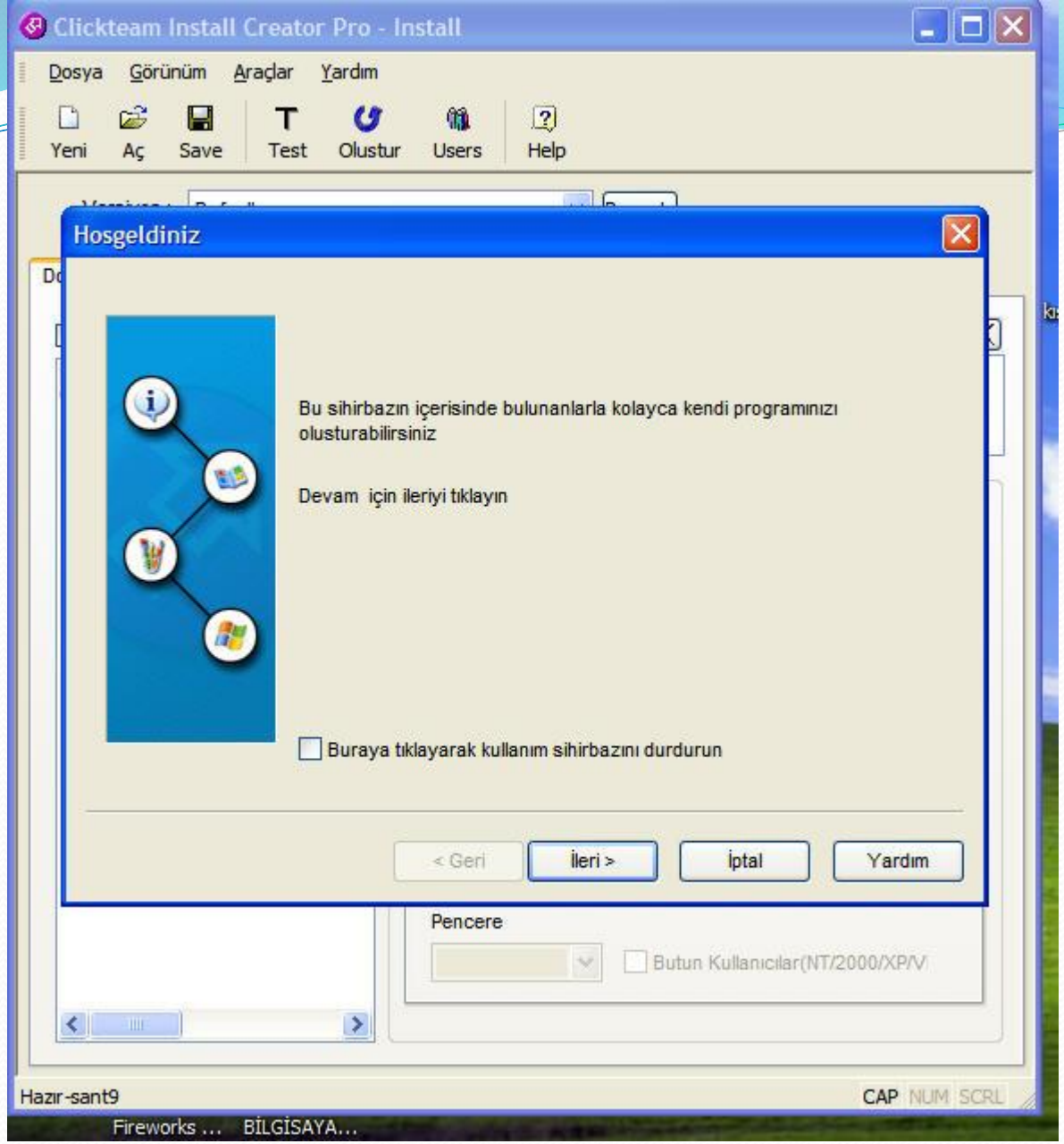
Inno Setup



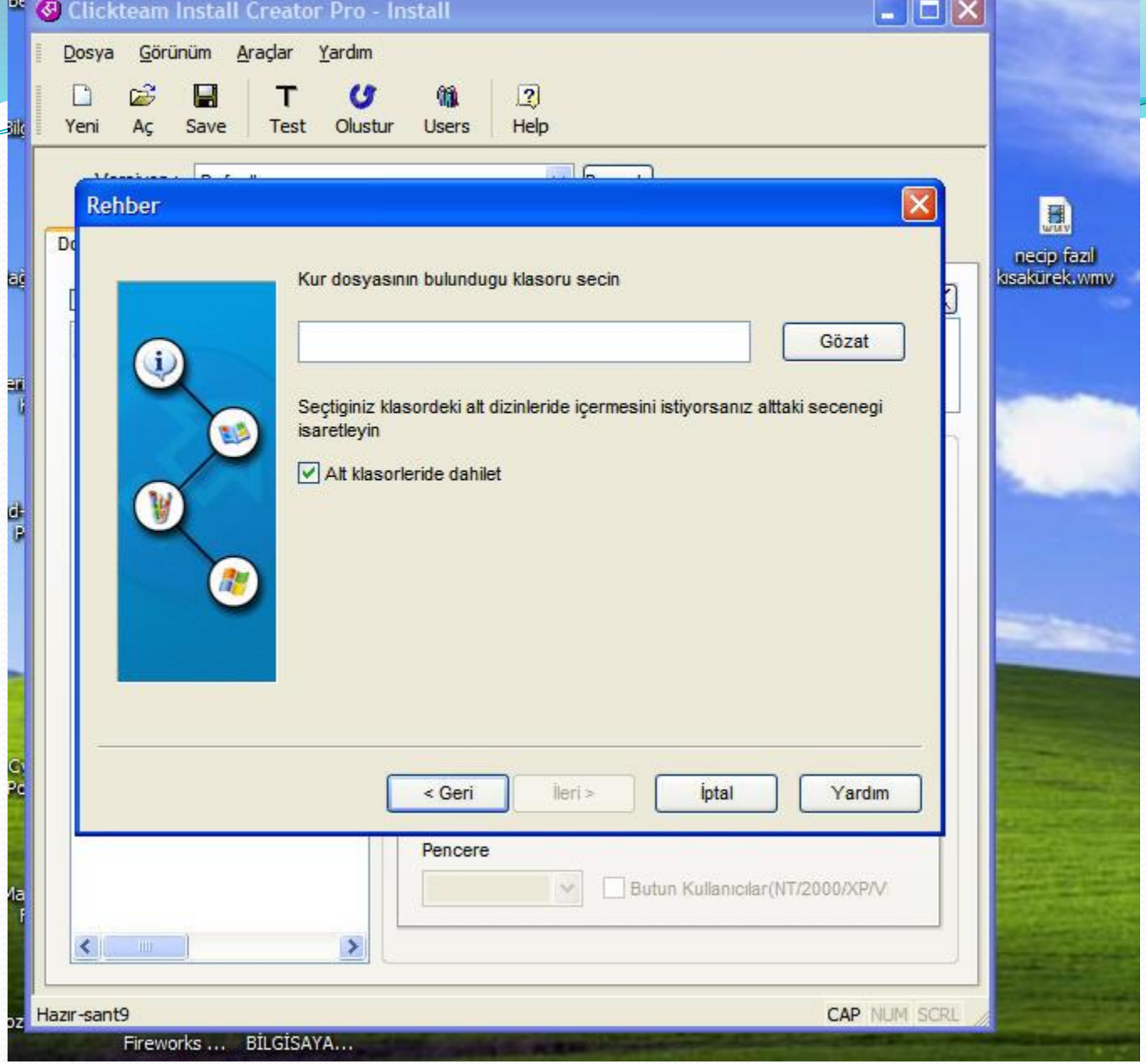
Hayır diyoruz ve setup oluşturulmaya başlıyor…

C# da Setup Hazırlama

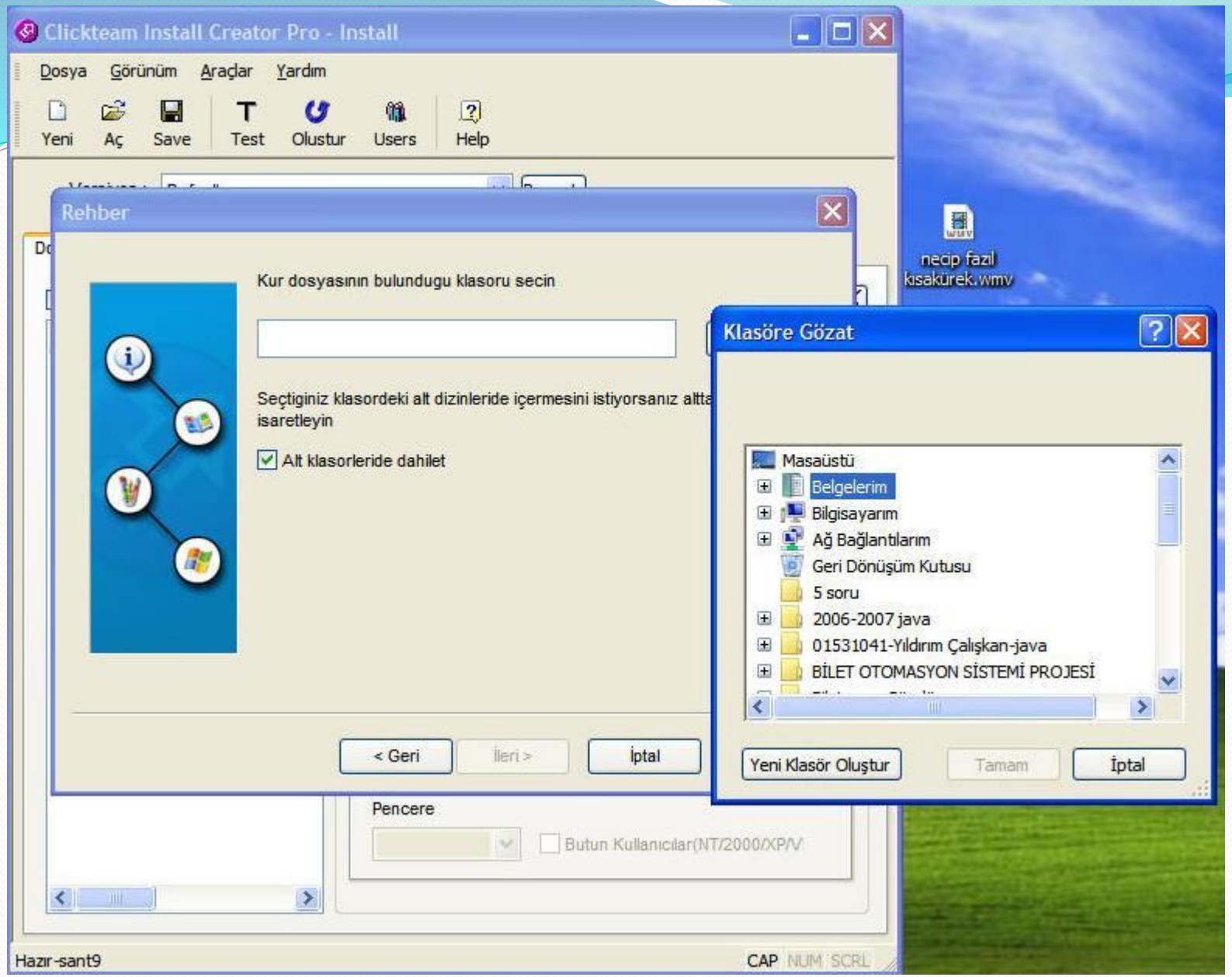
Install Creator



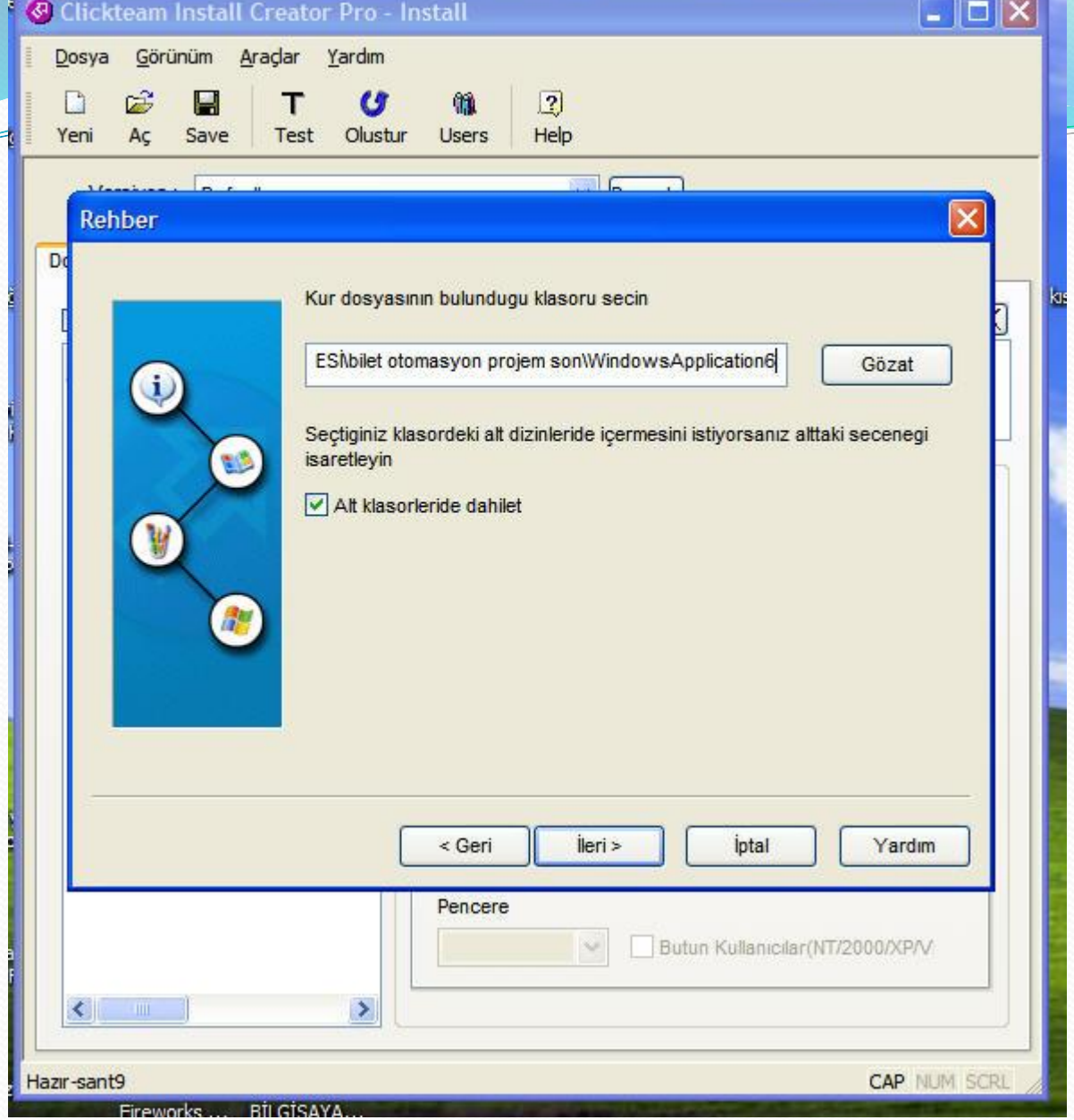
İleri deyip ilerliyoruz



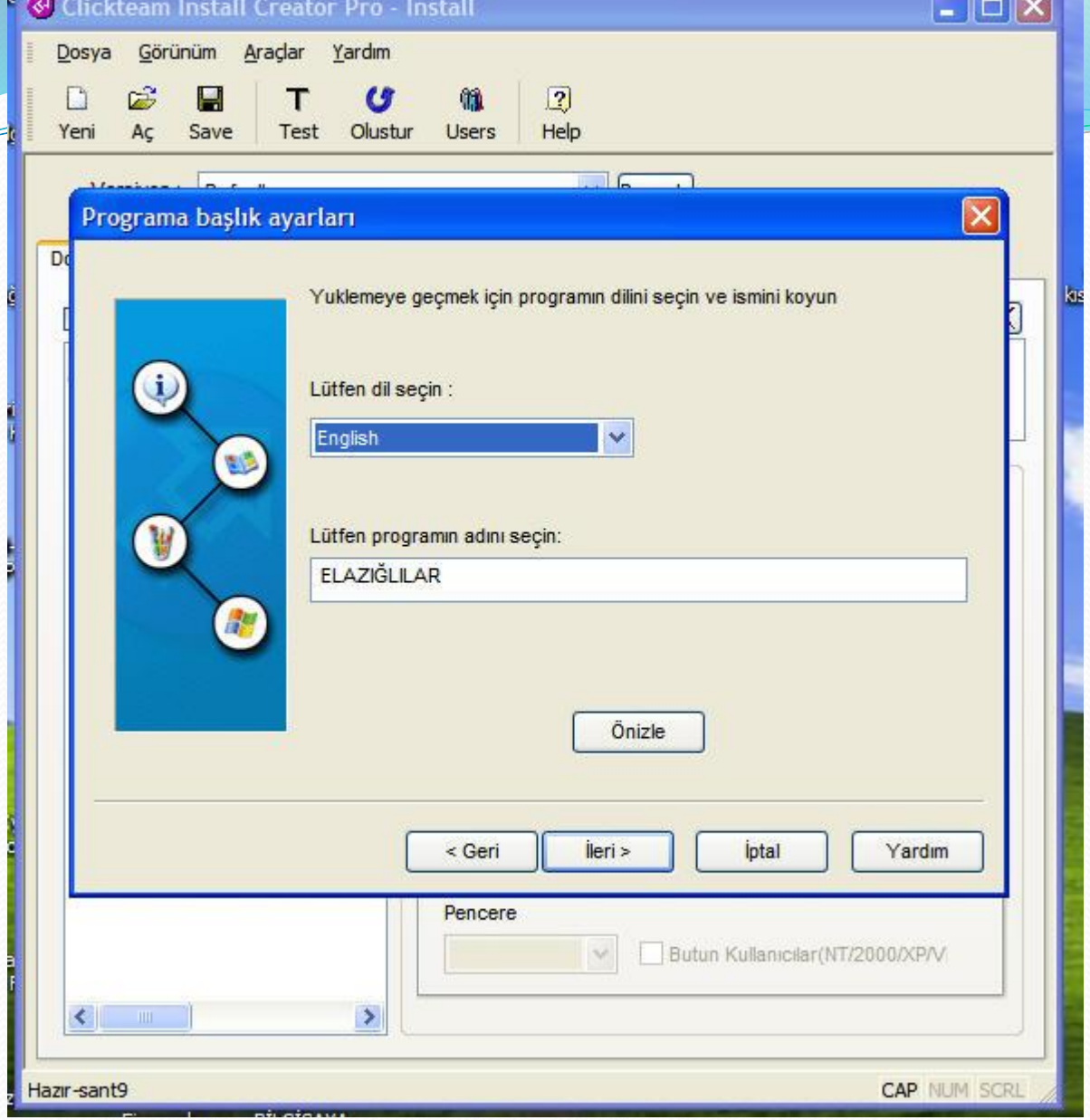
Programımızın Kur dosyalarının olduğu klasörü gösteriyoruz.



Bütün dosyalarımı klasörümün içine atmalıyım.



Net framework2.0 setup.exe ve dotnetfx.exe dosyalarımıda klasörüme atmalıyım.



Program dilini ve Programımın adını giriyorum.

Clickteam Install Creator Pro - Install

Dosya Görünüm Araçlar Yardım

Yeni Aç Save Test Olustur Users Help

Versiyon ismi

If you want the user to be able to choose between several different installation configurations, enter the name of one configuration here, more can be added later.

Default

Önizle

If you want to protect your install program with registration codes, please enter a product code (between 8 and 16 characters) that will be used to generate user registration codes:

Rasgele

Önizle

Check the option(s) below if you prefer short registration codes:

Kısa kayıt kodu üret

Sadece numara

< Geri

İleri >

İptal

Yardım

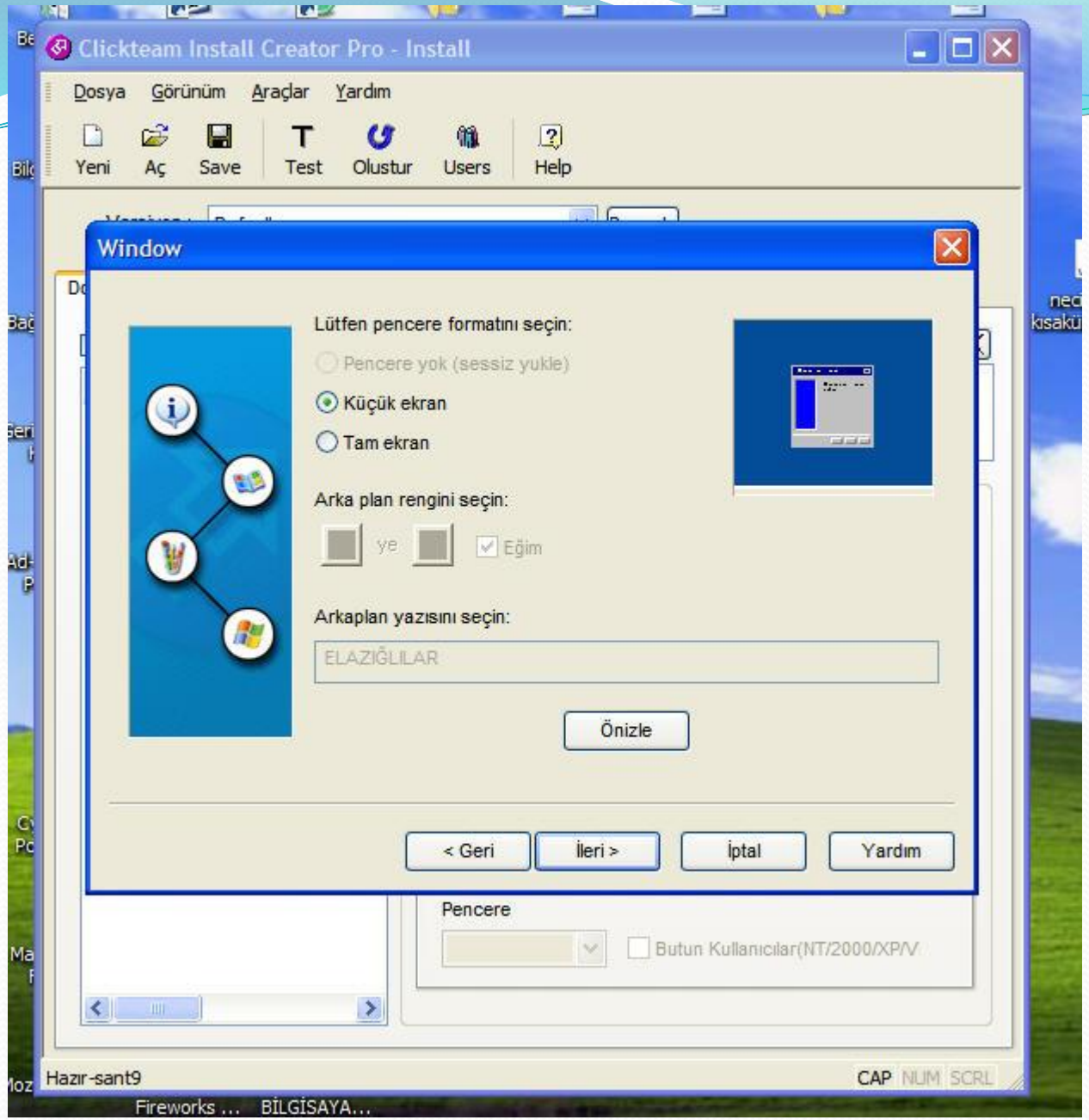
Pencere

Bu kısmı seçerek programımı bir serialle koruyabilirim.İstemiyorsak ileri diyelim

Hazır-sant9

CAP NUM SCRL

FIRAT ÜNİVERSİTESİ 1975



Kurulum ekranımızın boyutunu ayarlayalım

.NET

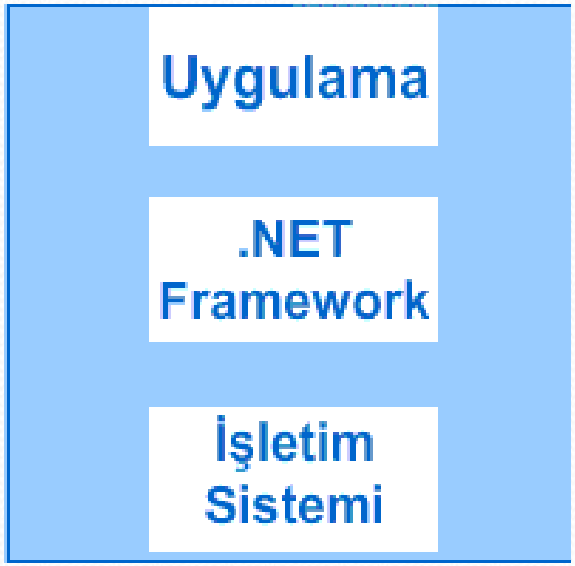
Microsoft tarafından geliştirilen .NET ileri düzey programcılığı tek bir çatı altında toplamayı hedefleyen platformdur.

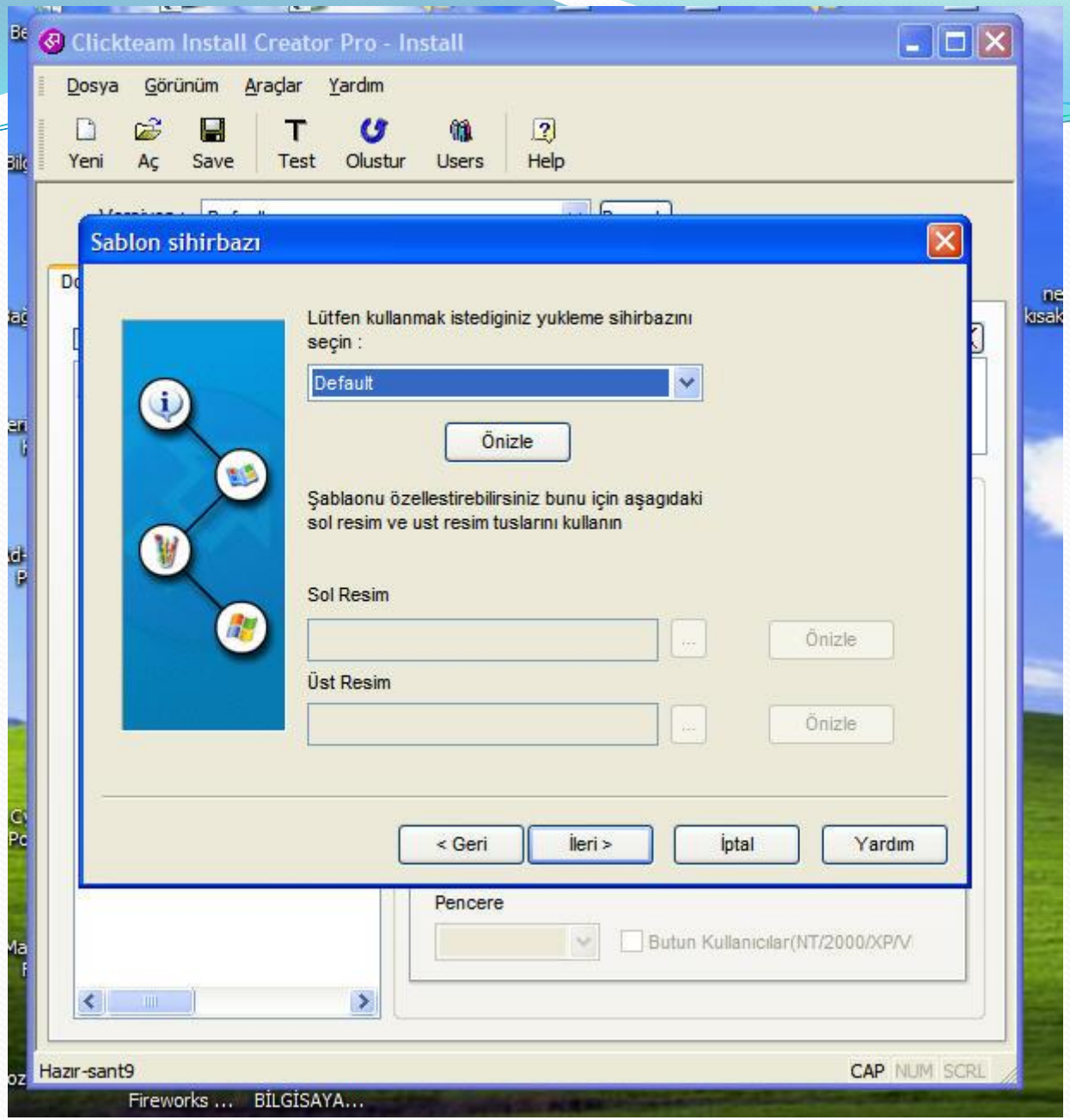
.NET iki hedefi gerçekleştirmiş oldu;

- Farklı programlama dillerinden standart ara kod oluşturmak
- Standart ara kodun hedef dile veya makine diline çevrilmesini tek bir noktadan gerçekleştirmek.

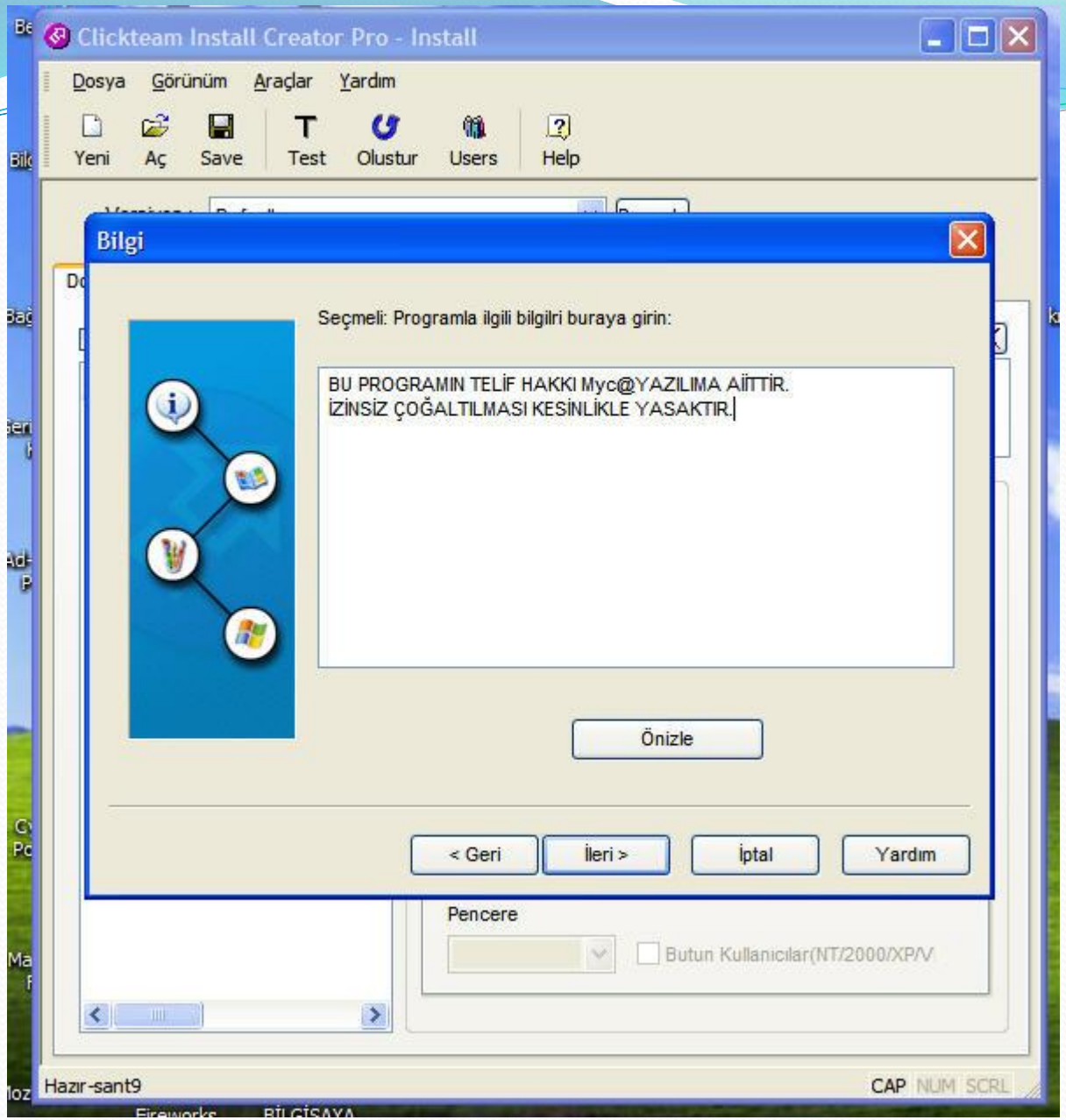
.NET Framework

.NET Framework yaptığımız uygulamaların derleyip işletim sistemine uygun şekilde çalıştırılmasını amaçlar.

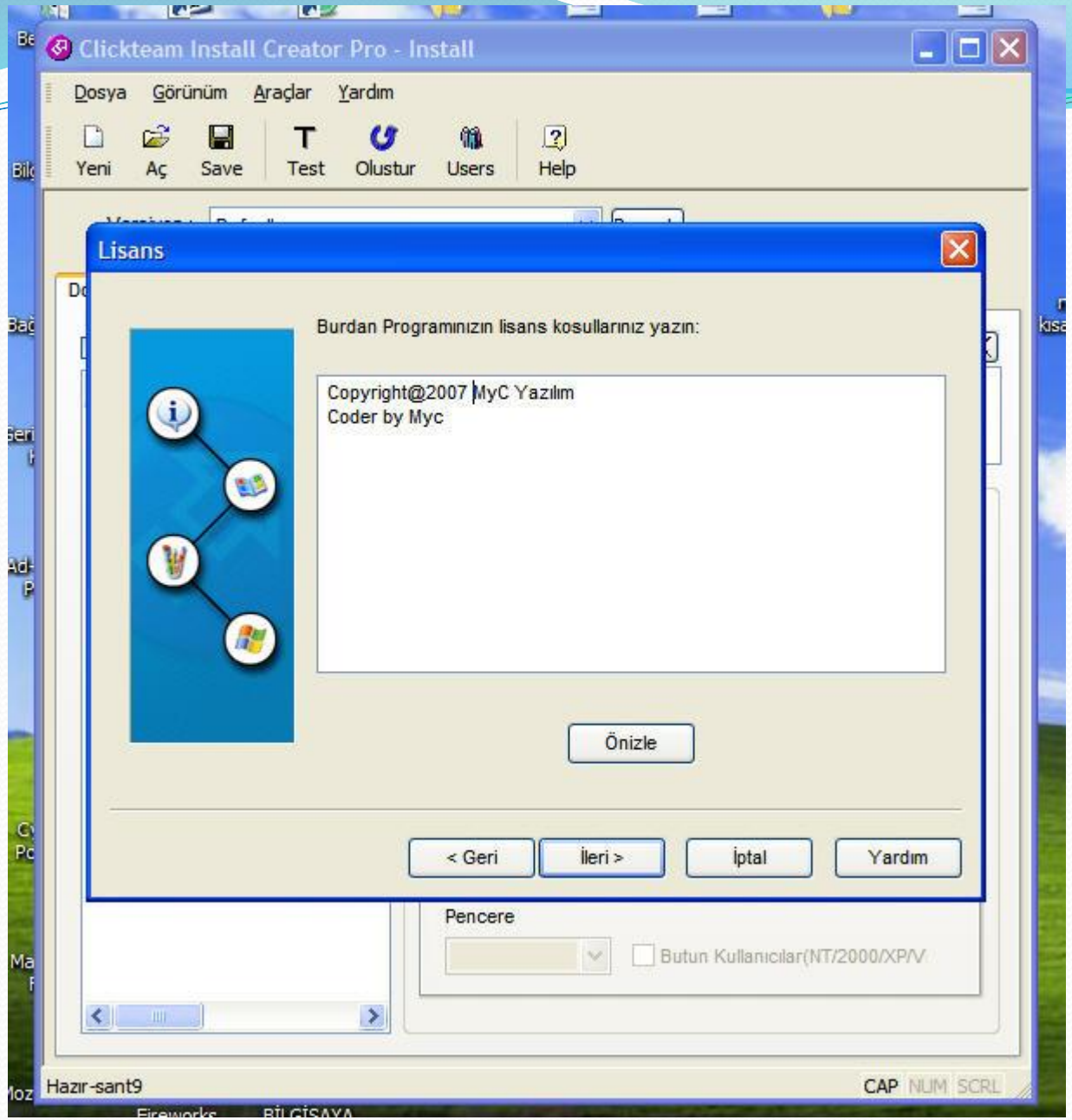




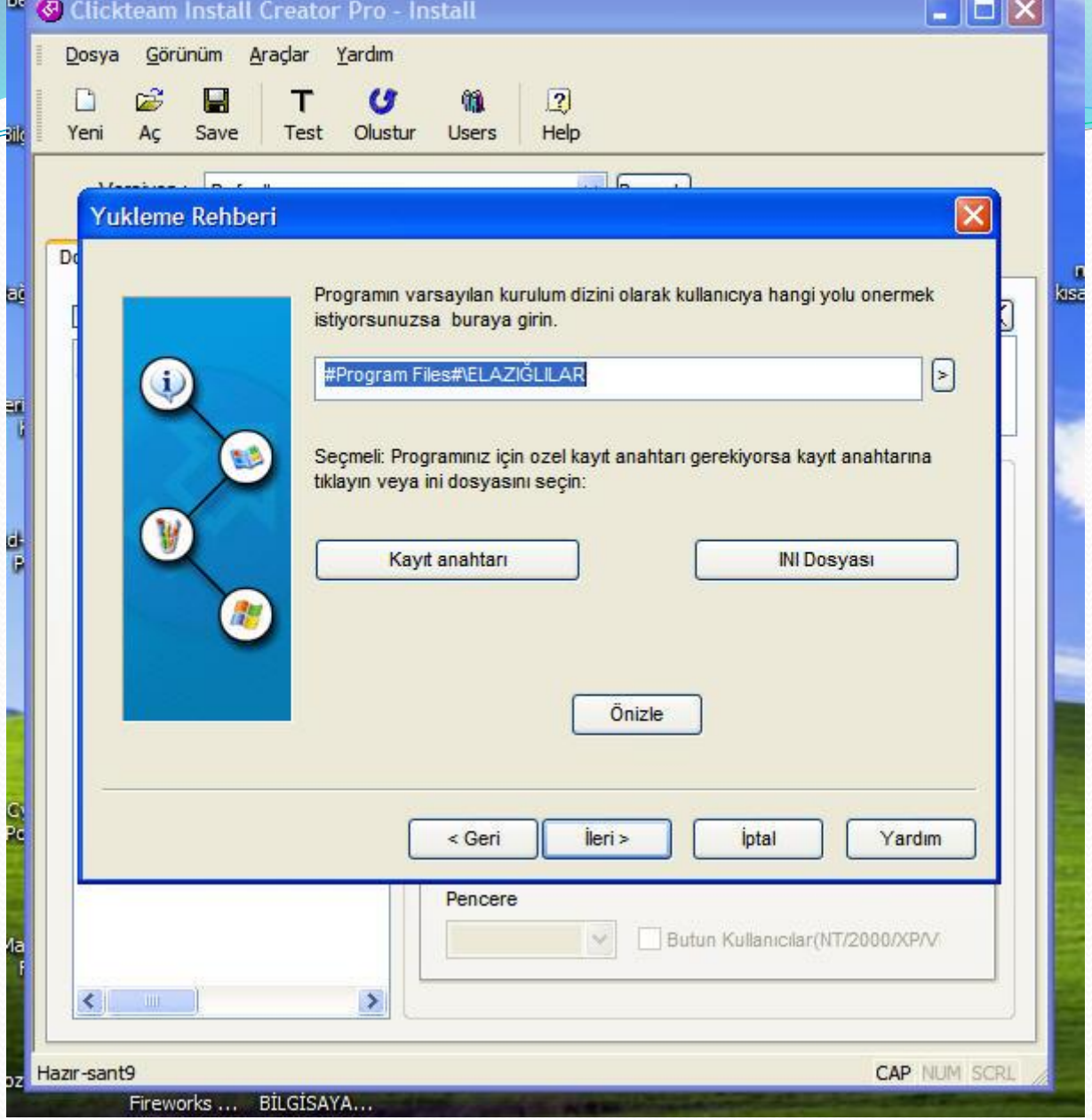
Yükleme şablonumu ayarlıyorum



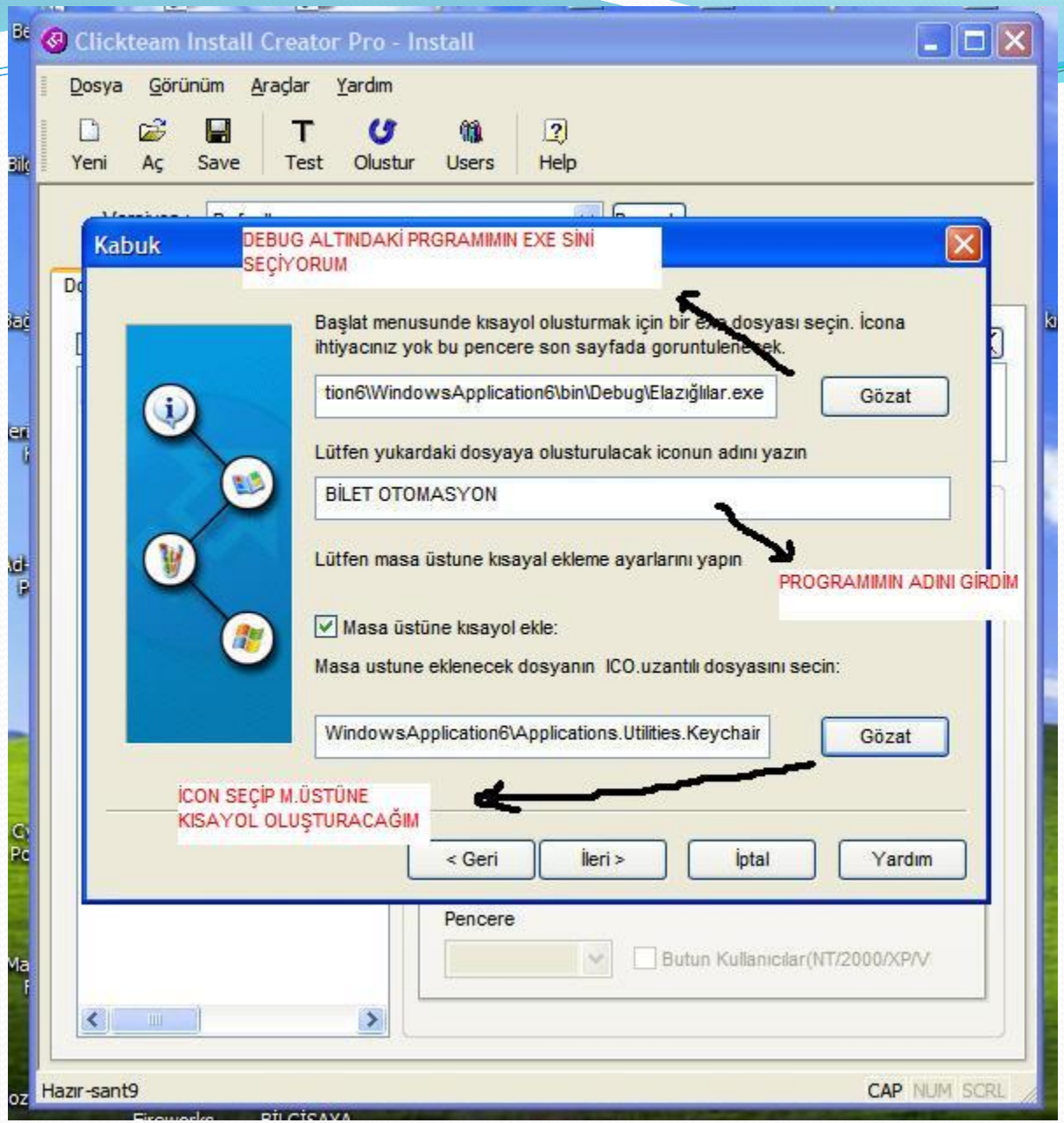
Programımla ilgili bilgi giriyorum



Programımın lisans koşullarını yazıyorum



Program kendisini hangi klasörün altına atacak onu belirliyorum


Clickteam Install Creator Pro - Install
Dosya Görünüm Araçlar Yardım
Yeni Aç Save Test Olustur Users Help
Kabuk
DEBUG ALTINDAKİ PRGRAMIMIN EXE SİNİ SEÇİYORUM
Başlat menusunde kısayol olusturmak için bir exe dosyası seçin. İcona ihtiyacınız yok bu pencere son sayfada goruntulenecek.
tion6\WindowsApplication6\bin\Debug\Elazığlılar.exe
Gözat
Lütfen yukardaki dosyaya olusturulacak iconun adını yazın
BİLET OTOMASYON
Lütfen masa üstune kısayal ekleme ayarlarını yapın
PROGRAMIMIN ADINI GİRDİM
Masa üstüne kısayol ekle:
Masa ustune eklenecek dosyanın ICO.uzantılı dosyasını secin:
WindowsApplication6\Applications.Utilities.Keychair
Gözat
İCON SEÇİP M.ÜSTÜNE KISAYOL OLUŞTURACAĞIM
< Geri
İleri >
İptal
Yardım
Pencere
Butun Kullanıcılar(NT/2000/XP/V
Hazır-sant9
CAP NUM SCRL

Clickteam Install Creator Pro - Install

Dosya Görünüm Araçlar Yardım

Yeni Aç Save Test Olustur Users Help

Son sayfa ayarları

Son sayfada istediginiz dosyayı calıstırmak için bir kısayol secin.Son sayfada menu kutusu içinde "goster belgesi "seklinde gosterilir:

DEBUG ALTINDAKİ PROGRAMIMIN EXE SİNİ SEÇİYORUM

ion6\WindowsApplication6\bin\Debug\Elazığlılar.exe

Gözat

Son sayfada çalıstırmak istediginiz exe.uzantılı dosyayı secebilirsiniz.Kısayol son sayfada işaret kutusunda >Göster dosya exe.<şeklinde gosterilir:

ion6\WindowsApplication6\bin\Debug\Elazığlılar.exe

Gözat

DEBUG ALTINDAKİ PROGRAMIMIN EXE SİNİ SEÇİYORUM

< Geri İleri > İptal Yardım

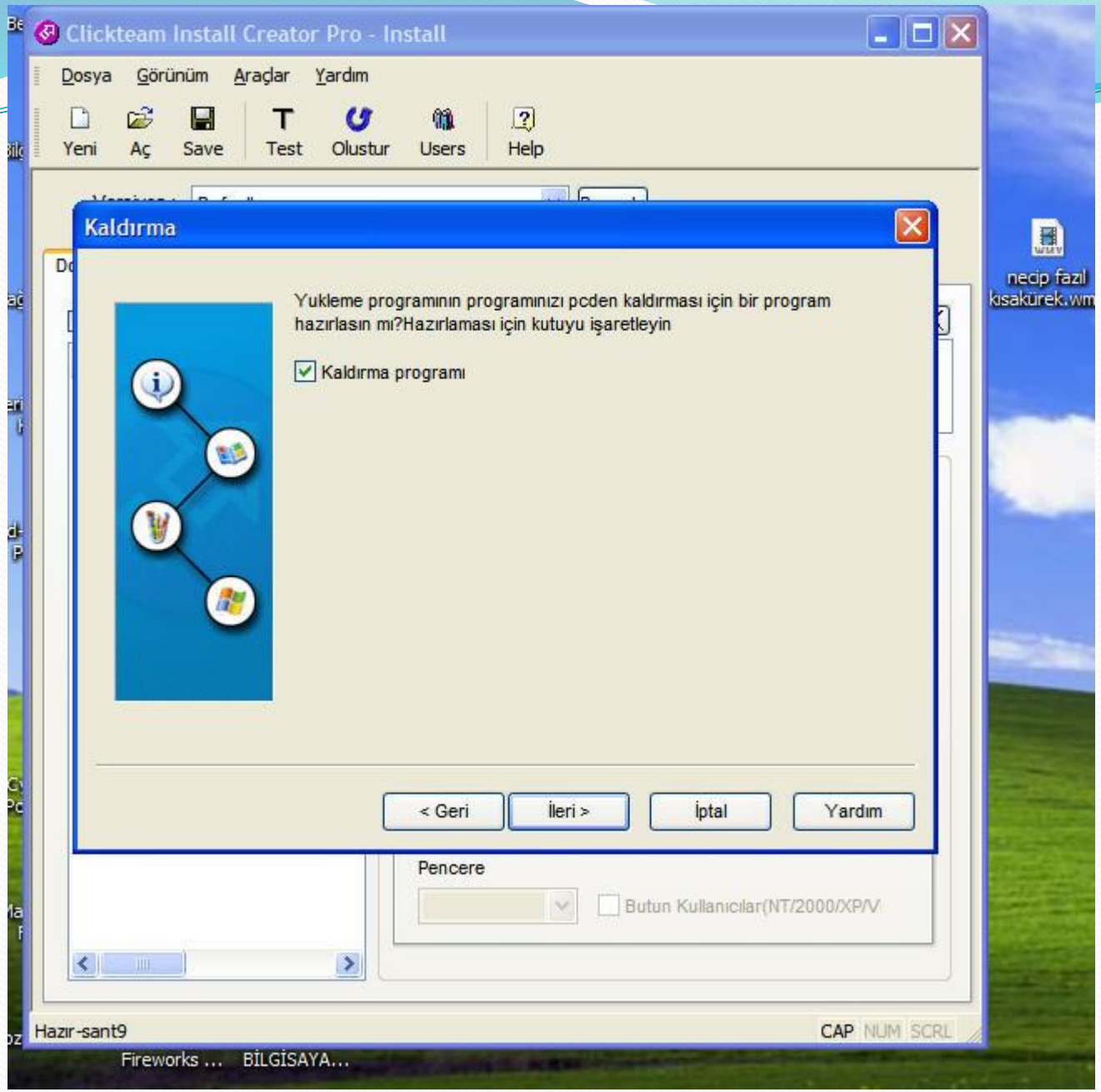
Pencere

Butun Kullanıcılar(NT/2000/XP/V

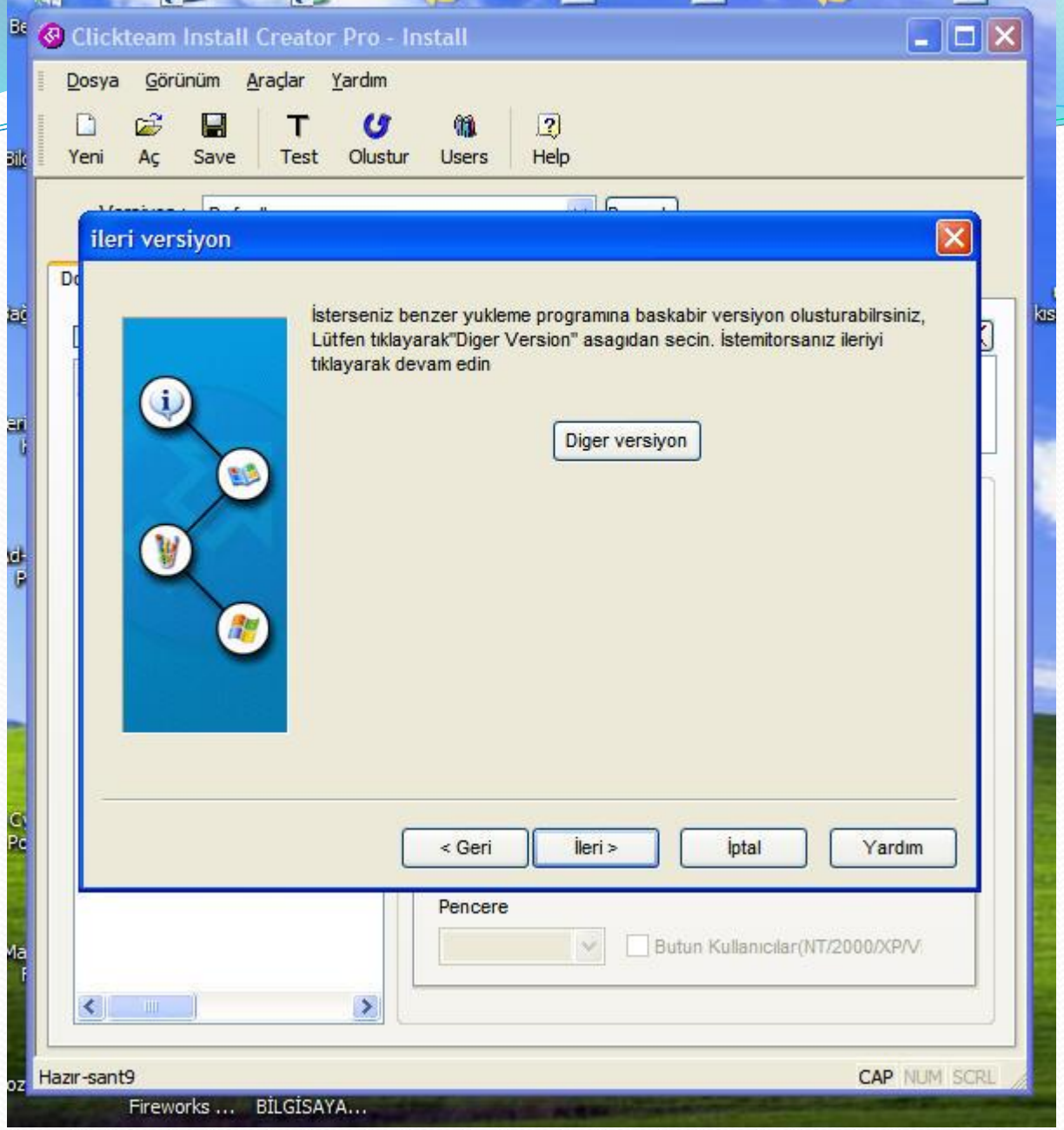
Hazır-sant9

CAP NUM SCRL

FIRAT ÜNİVERSİTESİ 1975

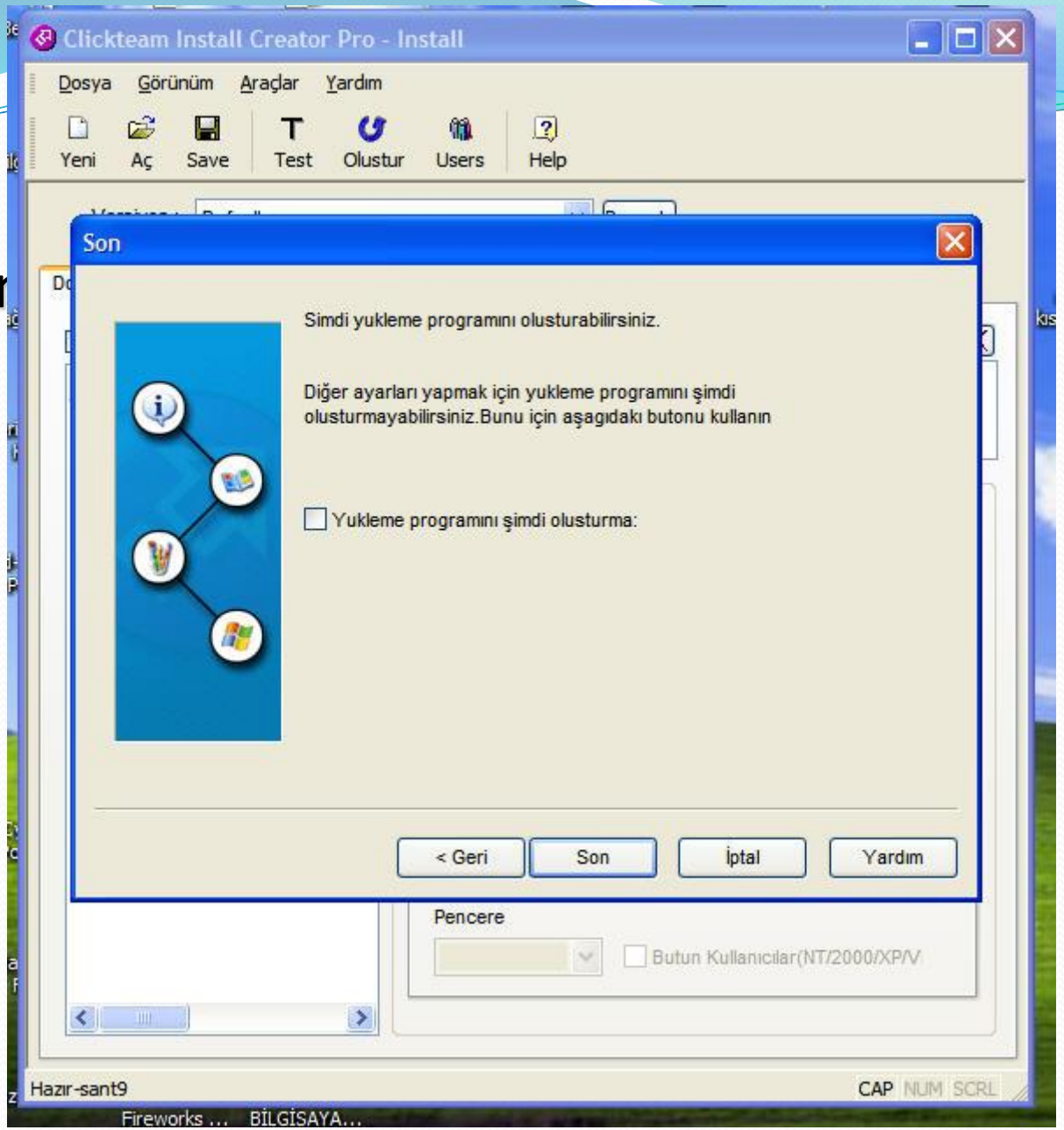


Uninstall oluşturulsun mu? EVET

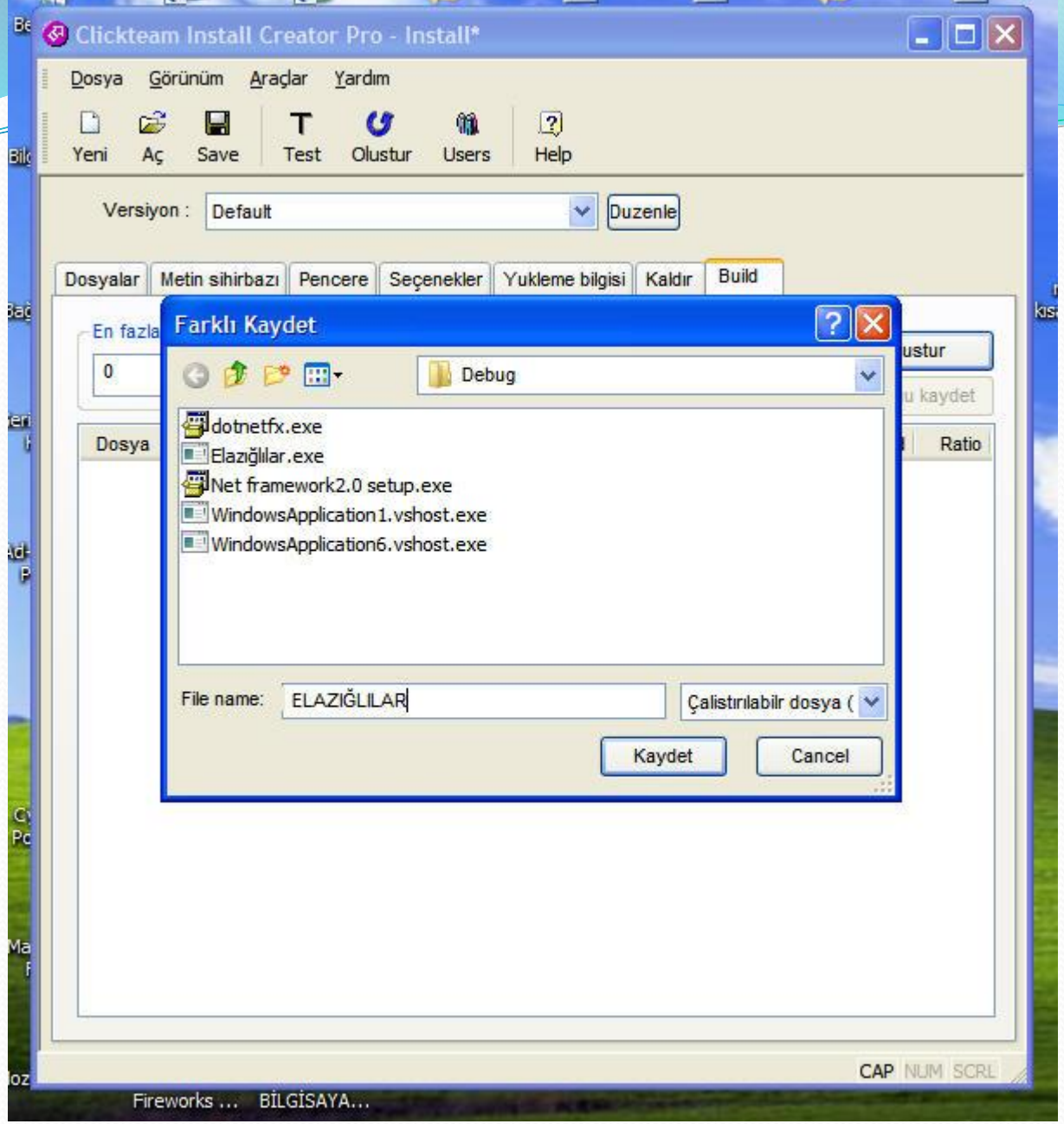


İleri diyoruz… Çok az kaldı

Inn


Clickteam Install Creator Pro - Install
Dosya
Görünüm
Araçlar
Yardım
Yeni
Aç
Save
Test
Olustur
Users
Help
Son
Simdi yukleme programını olusturabilirsiniz.
Diğer ayarları yapmak için yukleme programını şimdi olusturmayabilirsiniz.Bunu için aşagıdakı butonu kullanın
Yukleme programını şimdi olusturma:
< Geri
Son
İptal
Yardım
Pencere
Butun Kullanıcılar(NT/2000/XP/V
Hazır-sant9
CAP NUM SCRL
Fireworks ...
BİLGİSAYA...

SON…



Programı hangi isimle kayıt edecem

Clickteam Install Creator Pro - Install*

Dosya Görünüm Araçlar Yardım

Yeni Aç Save Test Olustur Users Help

Versiyon : Default Duzenle

Dosyalar | Metin sihirbazı | Pencere | Seçenekler | Yukleme bilgisi | Kaldır | Build

En fazla dosya boyutu: 0 KBytes

Sıkıstırma: Normal

İptalet

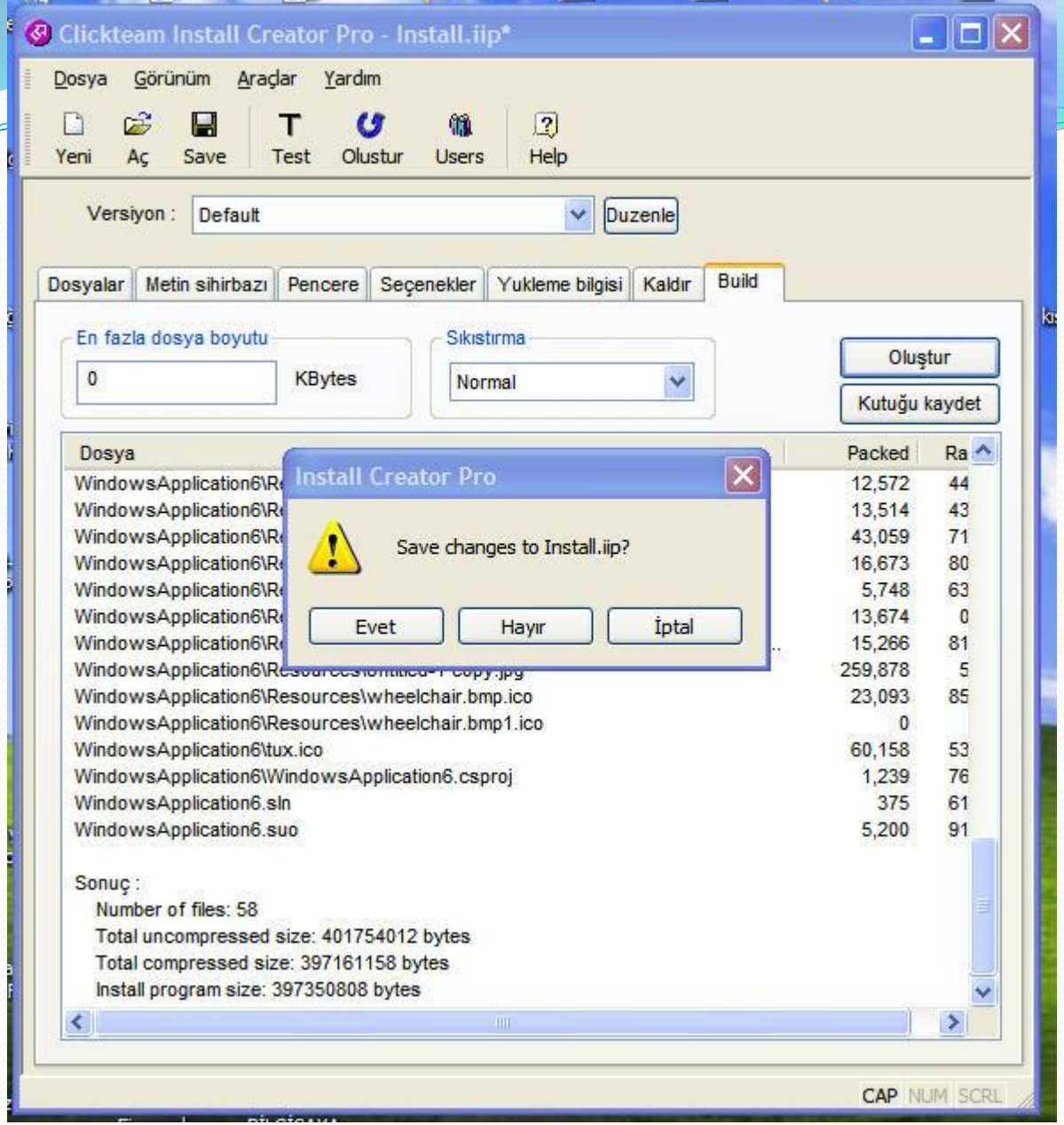
Kutuğu kaydet

Dosya	Packed	Ratio
Kaldır	29,480	61%
WindowsApplication6\Applications.Utilities.Keychain Access.app.Contents.Reso...	3,106	36%
WindowsApplication6\Bayrak.ico	148	81%
WindowsApplication6\bin\Debug\Consumer103_pcnet_t.ico	26,482	84%
WindowsApplication6\bin\Debug\dotnetfx.exe	23,430,052	1%
WindowsApplication6\bin\Debug\elazig.mdb	20,554	99%
WindowsApplication6\bin\Debug\Elazığlılar.exe	445,969	60%
WindowsApplication6\bin\Debug\Net framework2.0 setup.exe		

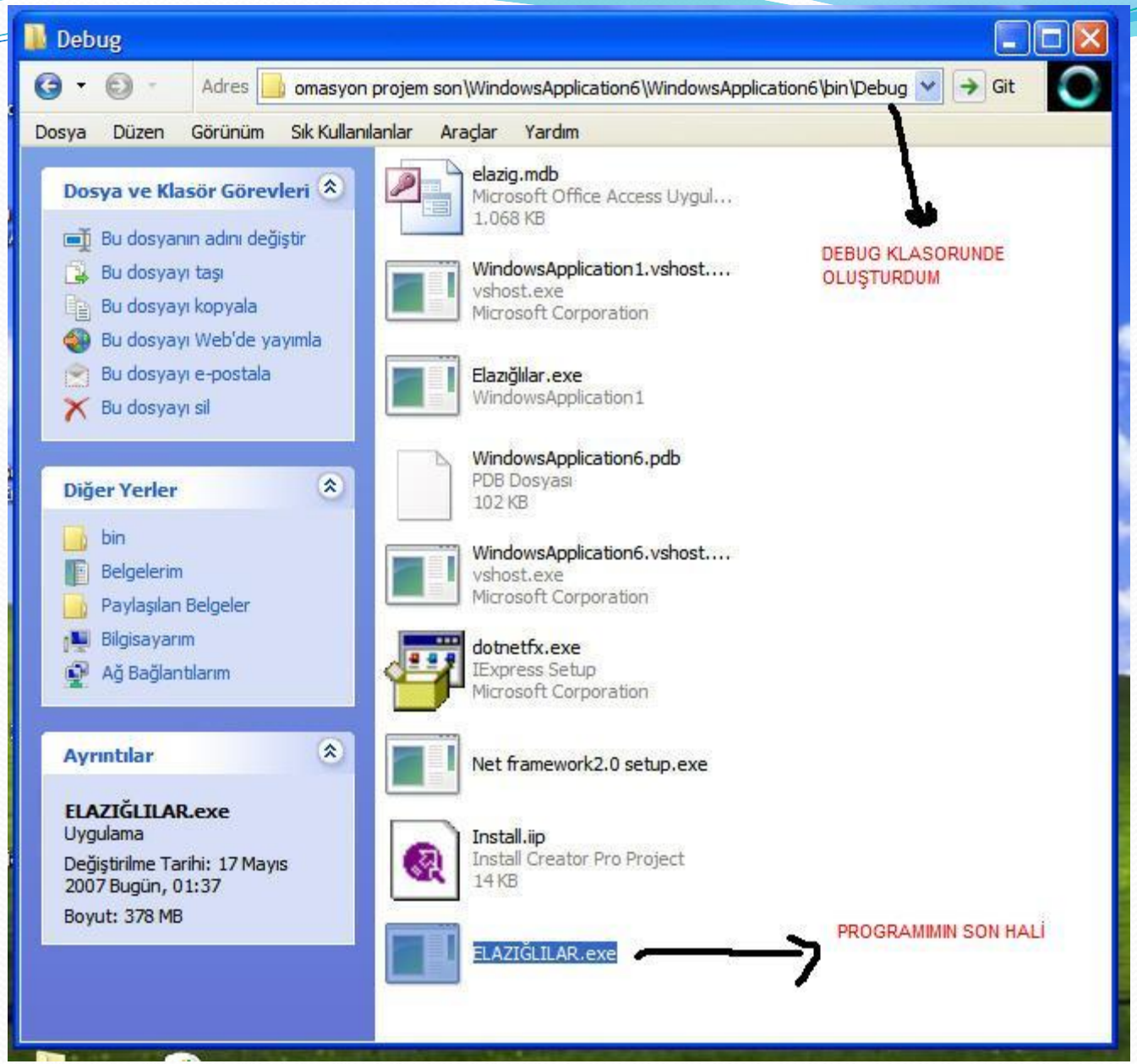
12.13 % CAP NUM SCRL

Fireworks ... BİLGİSAYA...

Programımım oluşuyor



İnstall.iip yi kaydederek sonradan tekrar programımızı oluşturabiliriz.


Debug
Adres
omasyon projem son\WindowsApplication6\WindowsApplication6\bin\Debug
Git
Dosya
Düzen
Görünüm
Sık Kullanılanlar
Araçlar
Yardım
Dosya ve Klasör Görevleri
Bu dosyanın adını değiştir
Bu dosyayı taşı
Bu dosyayı kopyala
Bu dosyayı Web'de yayımla
Bu dosyayı e-postala
Bu dosyayı sil
Diğer Yerler
bin
Belgelerim
Paylaşılan Belgeler
Bilgisayarım
Ağ Bağlantılarım
Ayrıntılar
ELAZIĞLILAR.exe
Uygulama
Değiştirilme Tarihi: 17 Mayıs 2007 Bugün, 01:37
Boyut: 378 MB
elazig.mdb
Microsoft Office Access Uygul...
1.068 KB
WindowsApplication1.vshost....
vshost.exe
Microsoft Corporation
Elazığlılar.exe
WindowsApplication1
WindowsApplication6.pdb
PDB Dosyası
102 KB
WindowsApplication6.vshost....
vshost.exe
Microsoft Corporation
dotnetfx.exe
IExpress Setup
Microsoft Corporation
Net framework2.0 setup.exe
Install.iip
Install Creator Pro Project
14 KB
ELAZIĞLILAR.exe
DEBUG KLASORUNDE OLUŞTURDUM
PROGRAMIMIN SON HALİ