

Visual Studio-C# Giriş

OOP

Programlama Dilleri

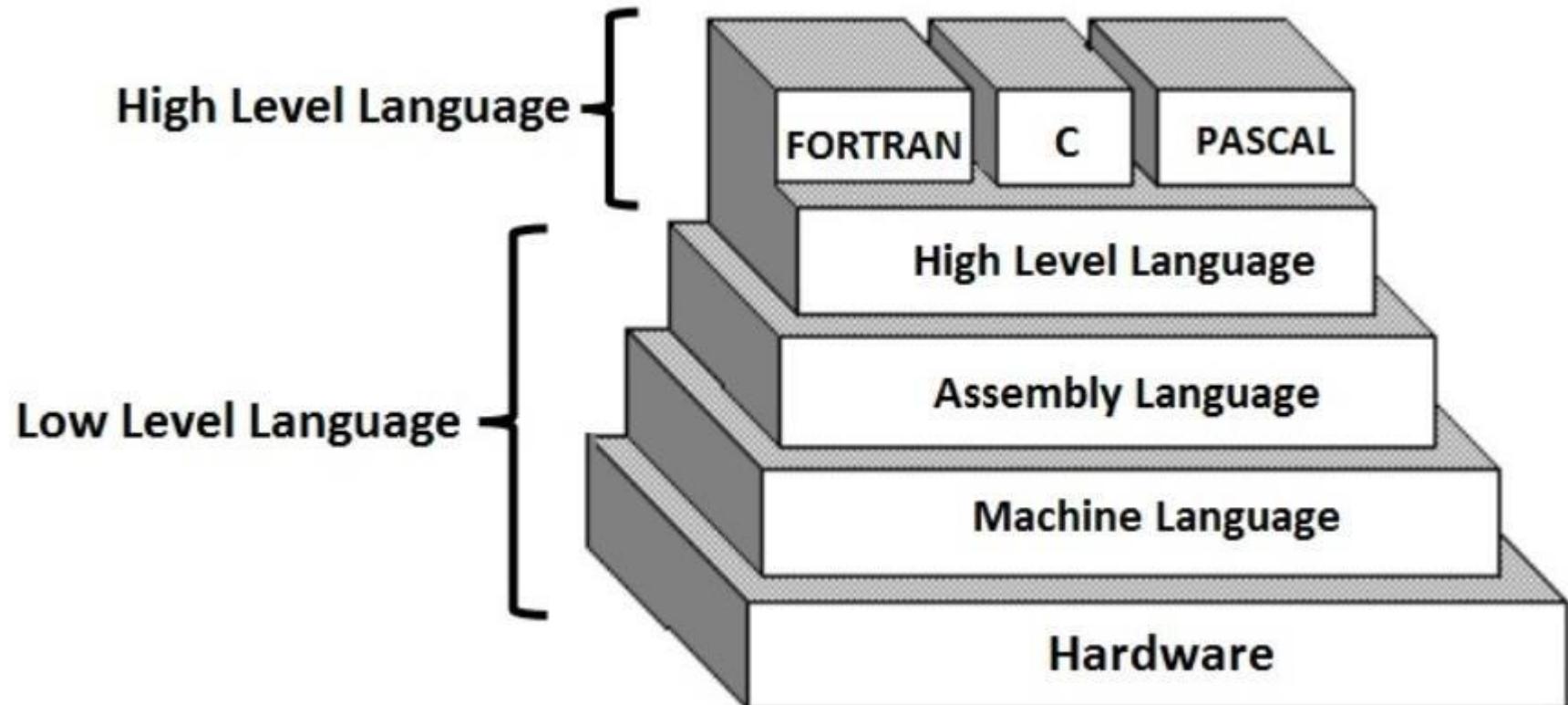
Programlama dilini gözünüzde büyütmeyin !



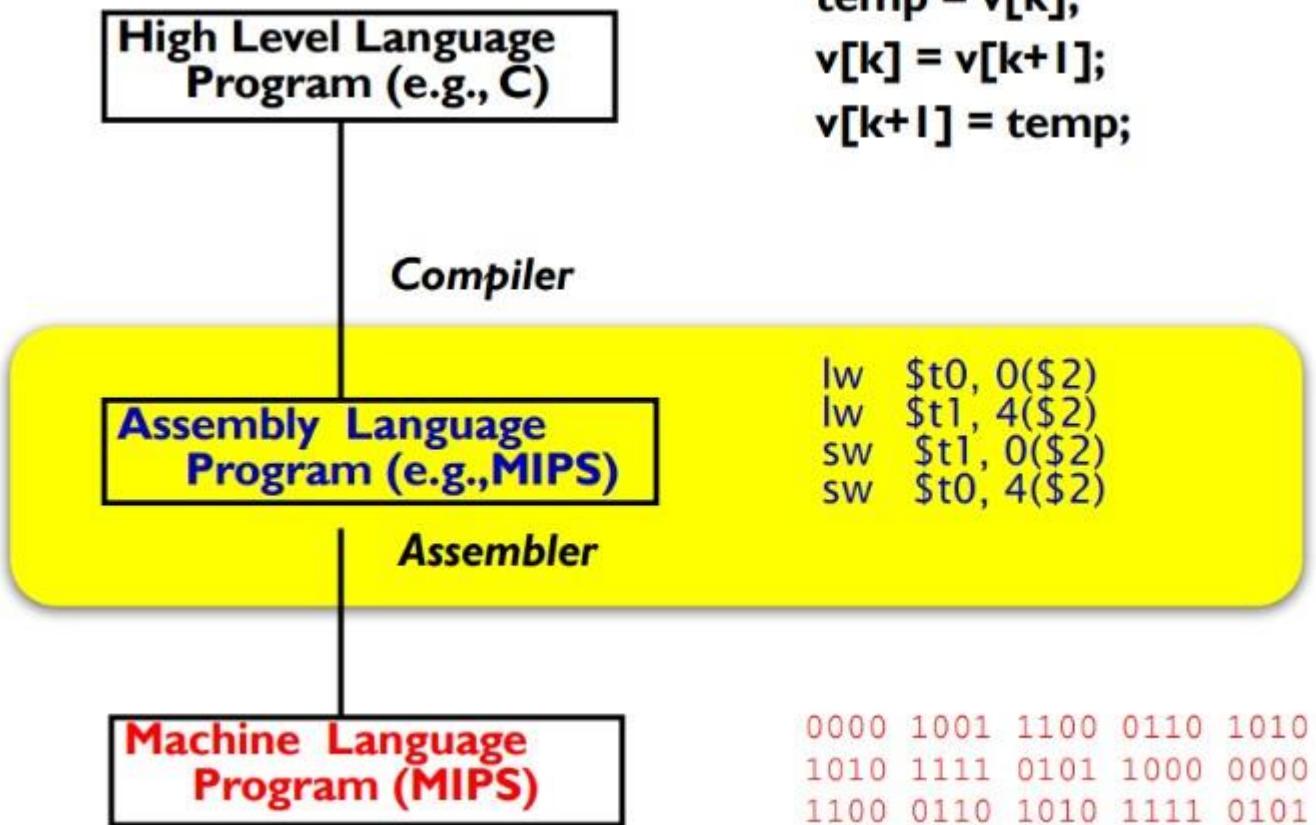
Programlama Dilleri

- Bir **programlama dili**, bir problemin çözümünün bilgisayardaki gerçekleştirimini ifade etmek amacıyla programlar oluşturulması için kullanılan bir dildir.
- **Programlama Dilleri Değerlendirme Kriterleri**
 - **Okunabilirlik:** Bir programlama dilinin değerlendirilmesinde en önemli kriterlerden birisi, programların okunabilme ve **anlaşılabilme kolaylığıdır**. Programlama dillerinin okunabilir olmaları, programlarda hata olasılığını azaltır ve programların bakımını kolaylaştırır. Okunabilirlik, bir programlama dilinin genelindeki **basitlik**, **sadelik** ve **ortogonalite** tarafından etkilenir. *Karmaşık bir sözdizim, bir programın yazımı sırasında kısa yollar sağlayabilir ancak programın daha sonra değiştirilmek amacıyla okunmasını ve anlaşılmasını zorlaştırır.*
 - **Yazılabilirlik:** Bir programlama dilinde program yazma kolaylığını belirleyen en önemli etkenlerden birincisi, programlama dilinin **sözdizimidir**. Buna ek olarak, programlama dilinin **soyutlama** yeteneği, dilin yazılabılırlığını önemli ölçüde etkilemektedir.
 - **Güvenilirlik:** Programlama dillerinde güvenilirlik, çeşitli faktörler tarafından belirlenir. Örneğin; dilde bulunan **tip denetimi** ve **istisnai durum işleme** verilebilir.
 -

Programlama Dilleri (devam...)



Programlama Dilleri (devam...)



Programlama Paradigmaları

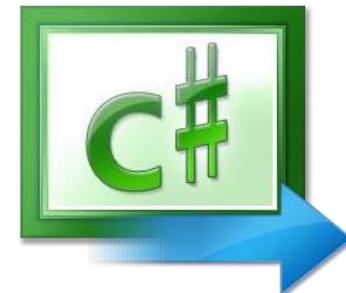
- Belirli bir paradigmayı destekleyen dillere paradigma-yönelik diller denir. Örneğin, Smalltalk ve Eiffel gibi diller nesneye yönelik programlama paradigmاسını desteklerler. Benzer şekilde, FORTRAN ve Pascal, imperative paradigmayı desteklerler. Bazı diller paradigma bağımsız olup, birden çok paradigmayı destekleyebilirler. Örneğin, C++ hem imperative hem de nesneye yönelik programların geliştirilmesini destekler.
 - Imperative Paradigma:** Bu dillerde bir program, **bir dizi deyimden oluşur** ve her deyimin çalışması, bellekteki bir veya daha fazla yerleşimin değerinin değiştirilmesine neden olur. Örneğin, iki değişkenin toplanması ve üçüncü bir değişkenin elde edilmesi, bu değerlerin birleştirilerek sonucun yeni bir yerleşimde saklanması olarak gösterilebilir (**C, Pascal, Ada**).
 - Nesneye Yönelik Paradigma:** Nesnelerin sınıf ve alt sınıflara gruplanması, nesneye yönelik programmanın temel noktasıdır (**C++, C#, Java**).
 - Fonksiyonel Paradigma:** Fonksiyonel dillerde, veriler ve sonucu elde etmek için veriye uygulanacak fonksiyonel dönüşümler, paradigmanın temelini oluşturur (**Lisp-AI**).
 - Mantık Paradigması:** Mantık programlama paradigmاسında programlama, bir işin nasıl yapılacağıının belirtilmesi yerine, ne yapılması istendiğinin belirtilmesi olarak görülür (**Prolog-AI**).

C# Programlama Dili

- **Microsoft** tarafından geliştirilmiş olan bir programlama dilidir.
- C++ ve Java dillerine **syntax** olarak oldukça benzer, ancak C#'ın bu dillere benzerliği yanında farkları da vardır.

Örneğin:

- C#, C++'dan farklı olarak % 100 **nesneye yönelik paradigmayı** sahiptir.
- Java'dan farklı olarak ise C#'ta **işaretçi (pointer)** kullanılabilir. Bu sayede eski yazılım bileşenleriyle uyumlu bir şekilde çalışılabilir.



C# ve Java Farkları

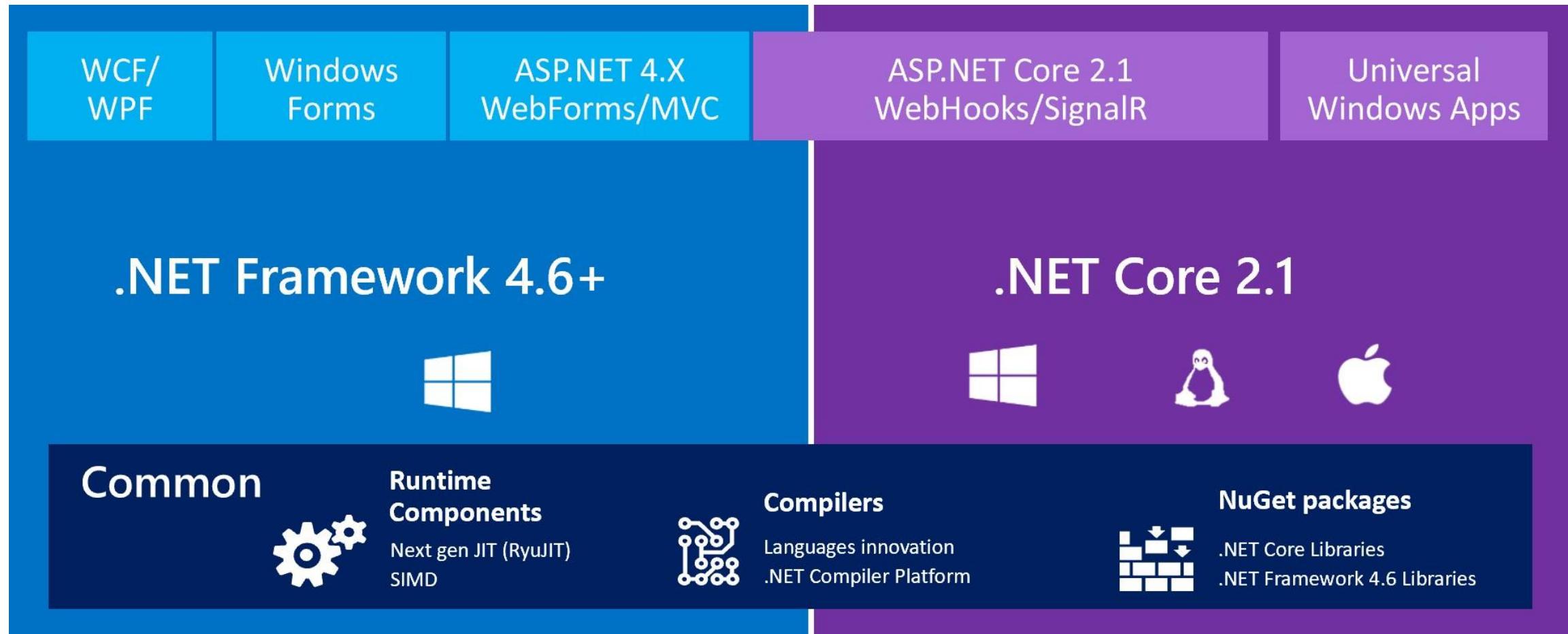
- C++ dilindeki **struct** yapısı C# da korunmuş, Java'da bulunmuyor.
- C# da **goto** kavramı var fakat buna karşın Java da continue ve label ifadeleri bulunuyor.
- C# da **delegate** kavramı ile fonksiyondan fonksiyona parametre aktarımı söz konusu.
- C# da yordamlara parametre传递 ref, out gibi özelliklerle belirtilebiliyor.
- C# da **attribute** kavramı var; Java'da ise derleyiciler için direktifler yok.
- C#' da **pointer** kavramından bahsedilebilir. ‘**unsafe**’ anahtar sözcüğü kullanılırsa bellek erişilebilir duruma geliyor.
- C# da **operatör overloading** varken, Java'da bu kavram yok.

MS.NET Uygulama Geliştirme Platformu

- .NET Framework, Microsoft tarafından geliştirilen, **açık Internet protokollerini ve standartları** üzerine kurulmuş bir **uygulama geliştirme** platformudur.
- **Masaüstü uygulamasından, web tabanlı uygulamaya, mobil uygulamadan, servis uygulamalarına** kadar her şey bu platform içinde düşünülmüştür ve desteklenmiştir.



MS.NET Uygulama Geliştirme Platformu

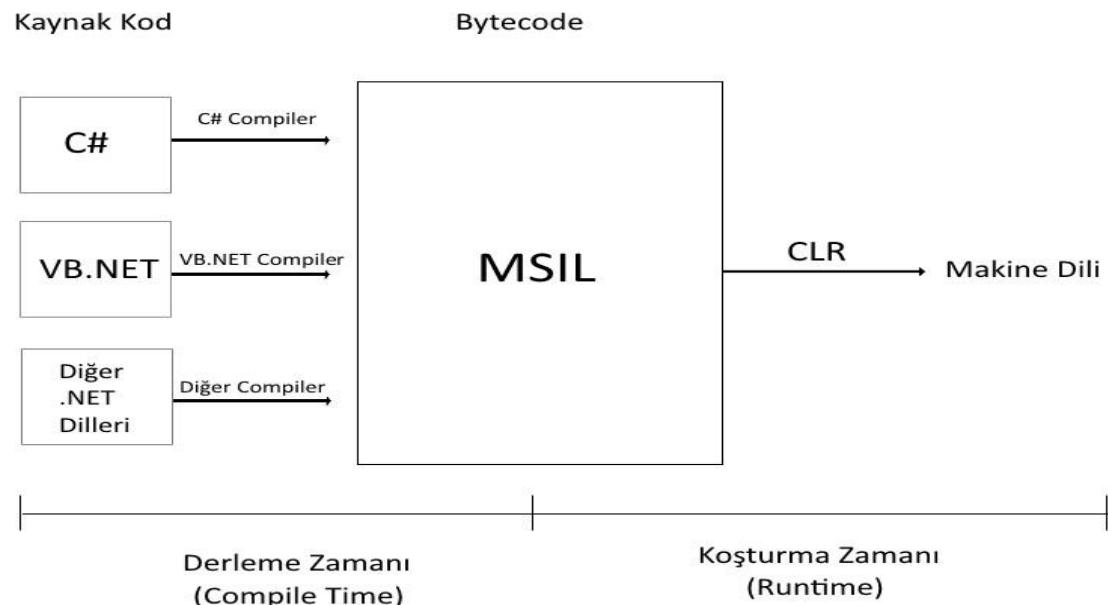


MS.NET Uygulama Geliştirme Platformu



Ortak Dil Çalışma Platformu (CLR)

- Ortak Dil Çalışma Platformu (Common Language Runtime), .NET altyapısında programların çalışmasını denetleyen, programın işletim sistemiyle haberleşmesini sağlayan birimdir.



Ortak Dil Çalışma Platformu (CLR)

- Bir C# kodu yazıp derlediğimizde bu kod Microsoft Intermediate Language (MSIL) 'a dönüştürülür.
- Bu kod "sözde kod" içeren bir dosyadır.
- CLR, MSIL'i sistem üzerinde çalıştırır. Yani CLR'ın olduğu her ortamda MSIL çalışacaktır.



Ortak Dil Çalışma Platformu (CLR)

C# Kaynak Kodu

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

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
            Console.ReadLine();
        }
    }
}
```

MSIL

```
.method private hidebysig static void Main(string[] args) cil managed
{
    .entrypoint
    // Code size     19 (0x13)
    .maxstack 8
    IL_0000: nop
    IL_0001: ldstr    "Hello World"
    IL_0006: call     void [mscorlib]System.Console::WriteLine(string)
    IL_000b: nop
    IL_000c: call     string [mscorlib]System.Console::ReadLine()
    IL_0011: pop
    IL_0012: ret
} // end of method Program::Main
```

İlk Windows Form Uygulaması

Senaryo: Bu form uygulamamızda kullanıcıdan alınan sayıları toplayan sonrasında ekrana yazdırın bir uygulama yazalım.

- Projeyi oluştur.
- Geliştirme işlemini yap.
- İsimlendirme standartlarına dikkat et.
- Validasyon kontrolleri yap.
- Toplama işleminin tekrar kullanılabilmesi için bir fonksiyon oluştur.
- Messagebox kullan.

Yeni Proje:

ilk Windows Form Uygulamasi

Create a new project

Recent project templates

- Console App (.NET Core) C#
- ASP.NET Web Application (.NET Framework) C#
- Blank Solution
- Windows Forms App (.NET Framework) C#
- ASP.NET Core Web Application C#
- Polls Django Web Project Python

windows forms x ▾ Language ▼ Platform ▼ Project type ▼

Windows Forms App (.NET Framework)
A project for creating an application with a Windows Forms (WinForms) user interface
C# Windows Desktop

ASP.NET Web Application (.NET Framework)
Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.
C# Windows Web

Windows Forms Control Library (.NET Framework)
A project for creating controls to use in Windows Forms (WinForms) applications
C# Windows Desktop Library

ASP.NET Web Application (.NET Framework)
Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.
Visual Basic Windows Web

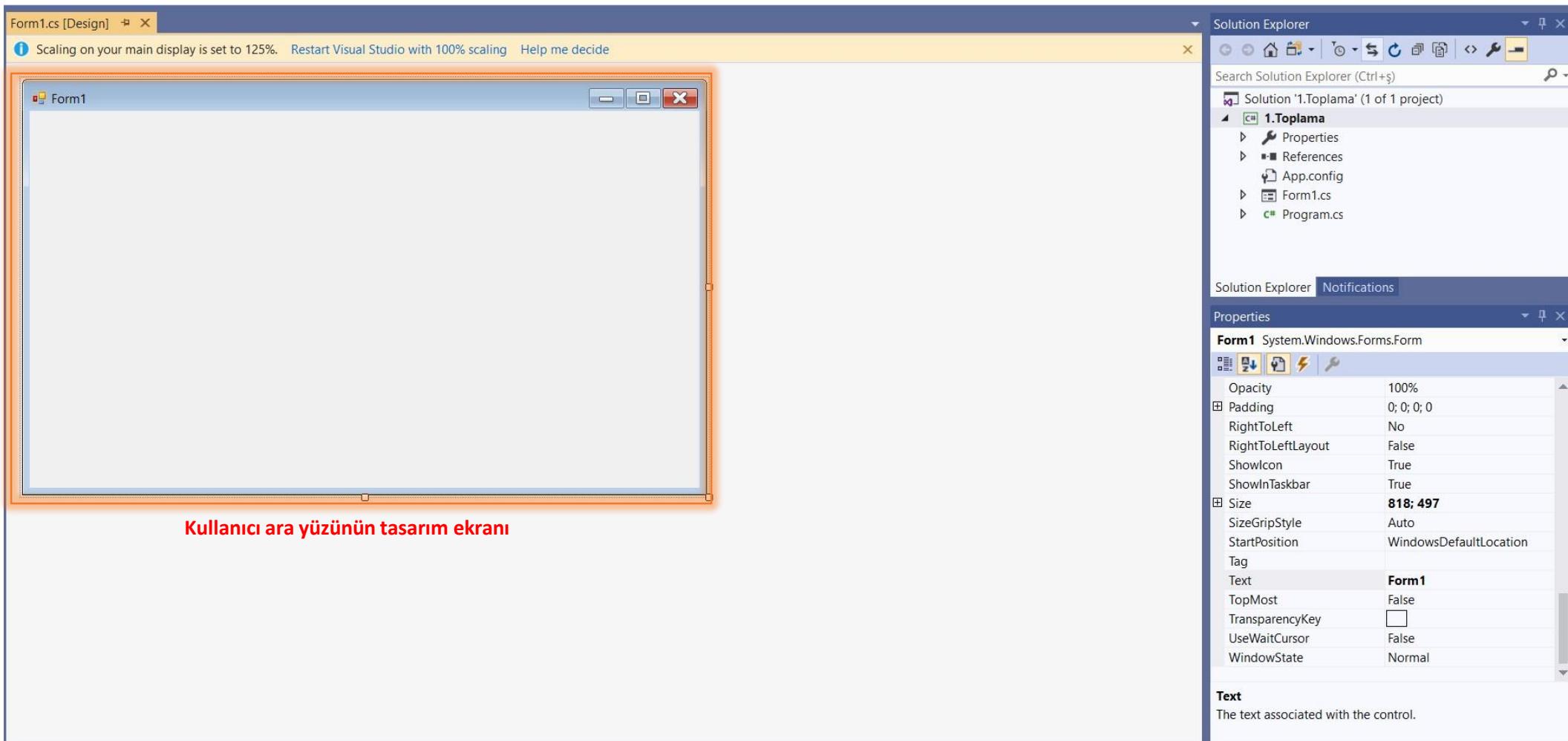
Windows Forms App (.NET Framework)
A project for creating an application with a Windows Forms (WinForms) user interface
Visual Basic Windows Desktop

Windows Forms Control Library (.NET Framework)
A project for creating controls to use in Windows Forms (WinForms) applications
Visual Basic Windows Desktop

Next Back

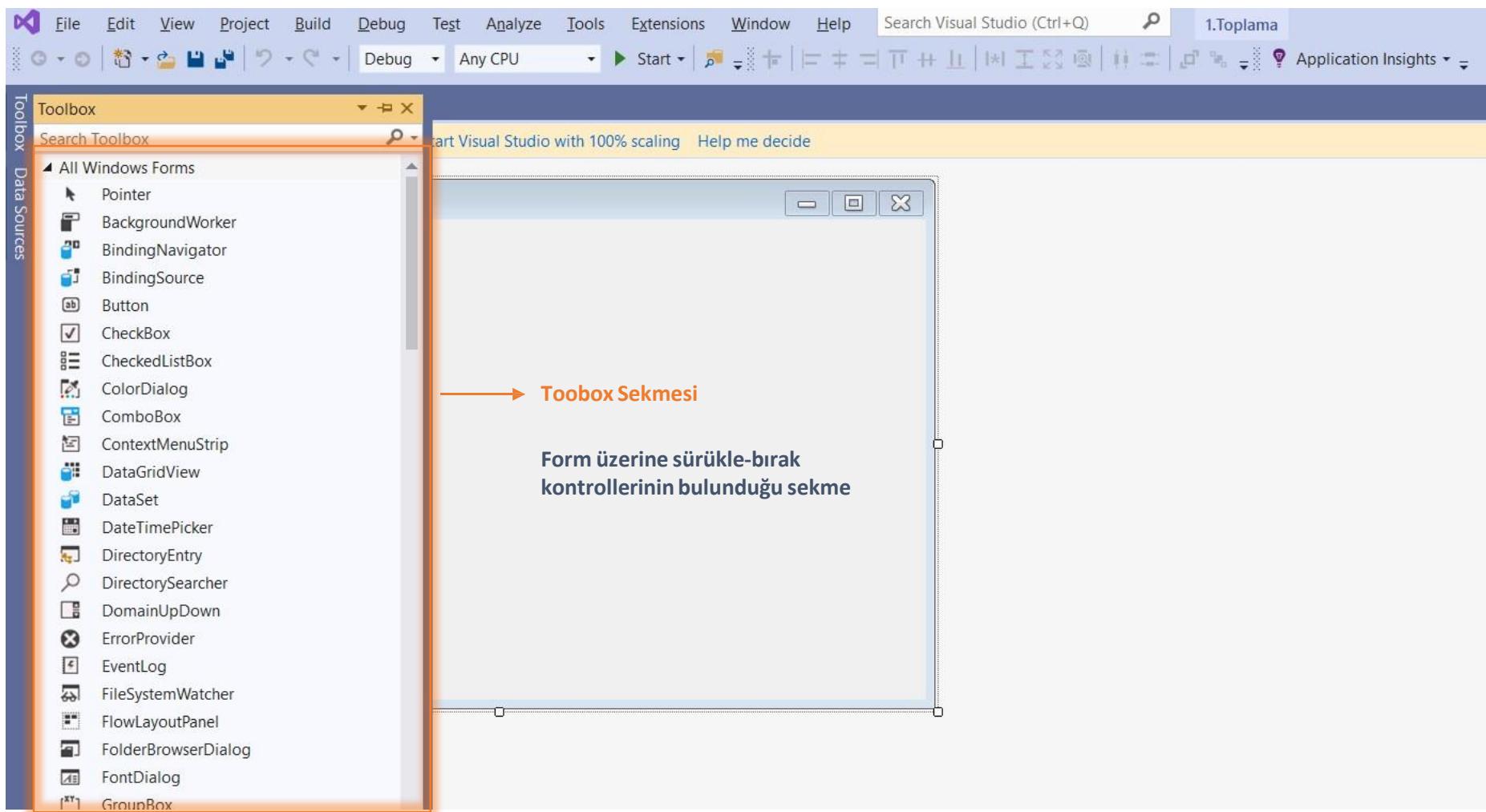
İlk Windows Form Uygulaması

Tasarım:



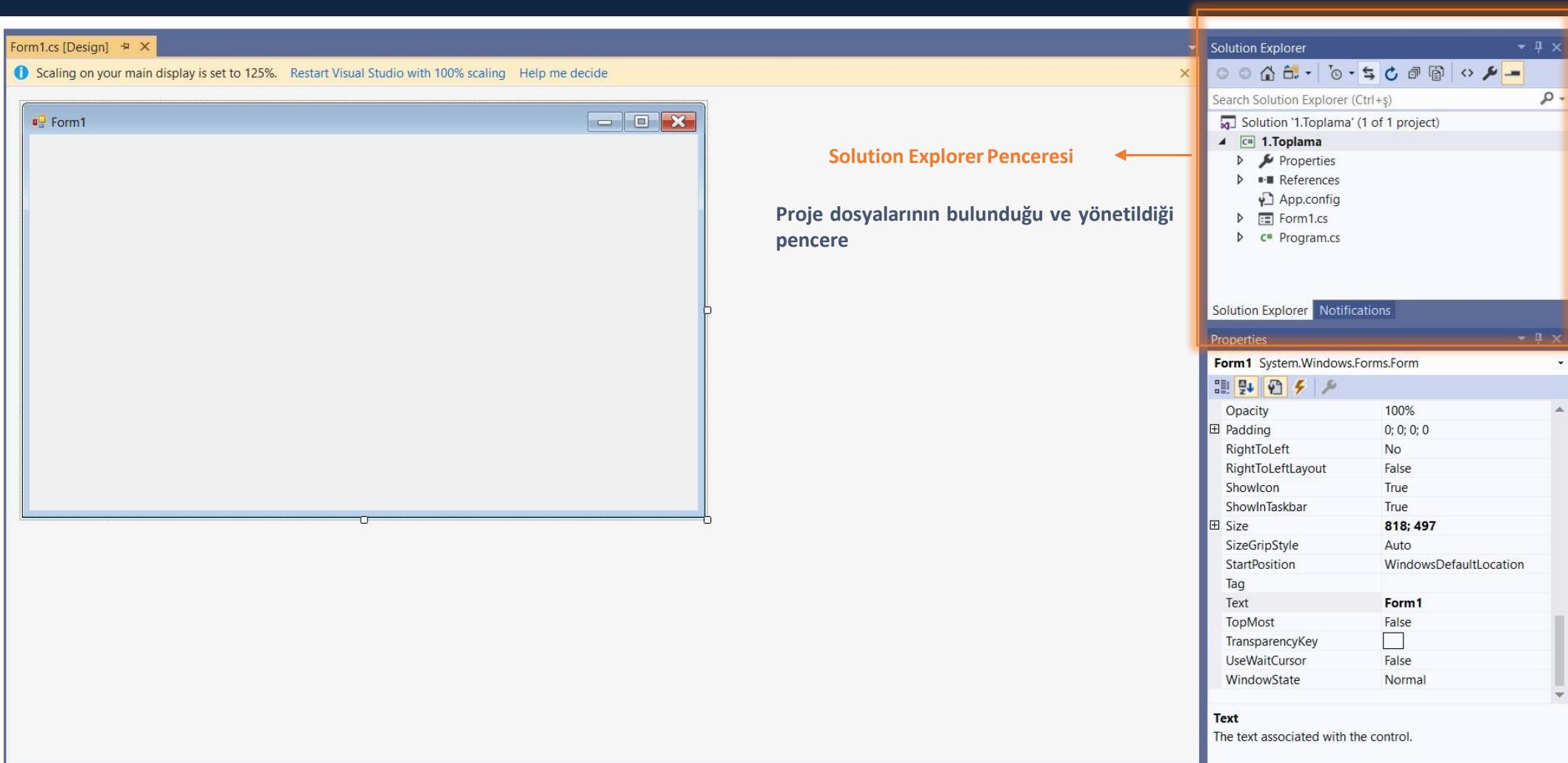
ilk Windows Form Uygulaması

Toolbox:



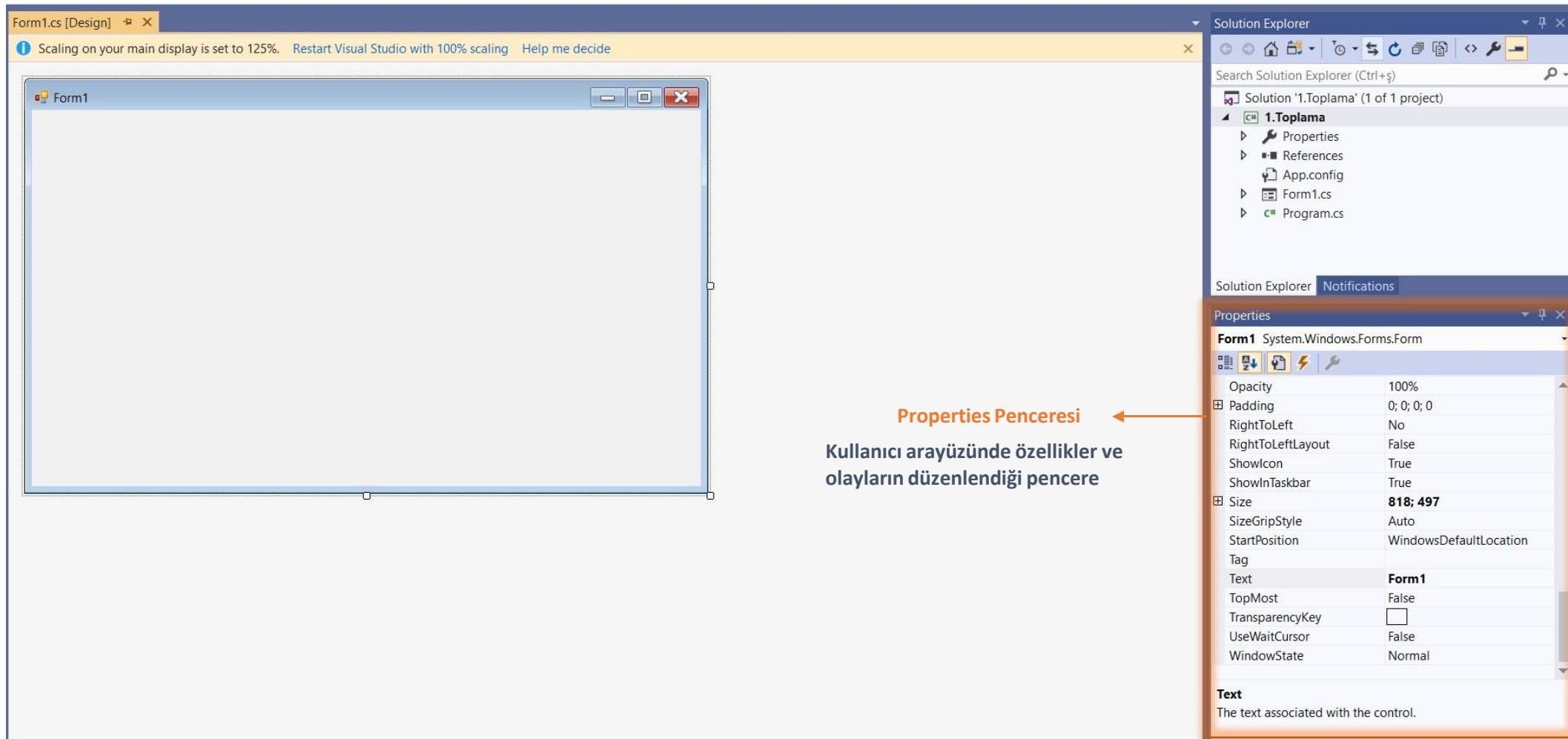
ilk Windows Form Uygulaması

Solution Explorer:



ilk Windows Form Uygulaması

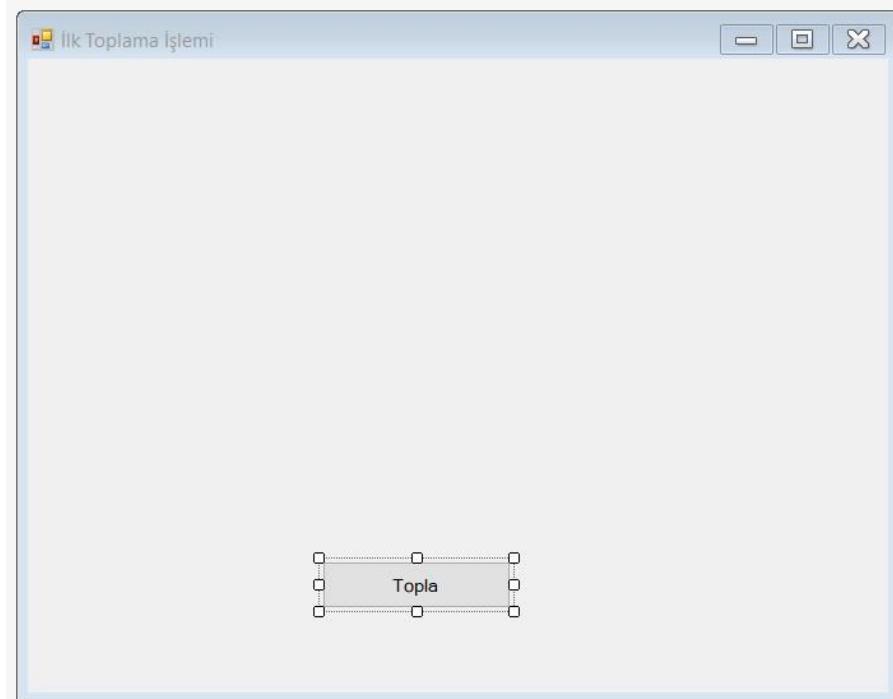
Properties:



İlk Windows Form Uygulaması

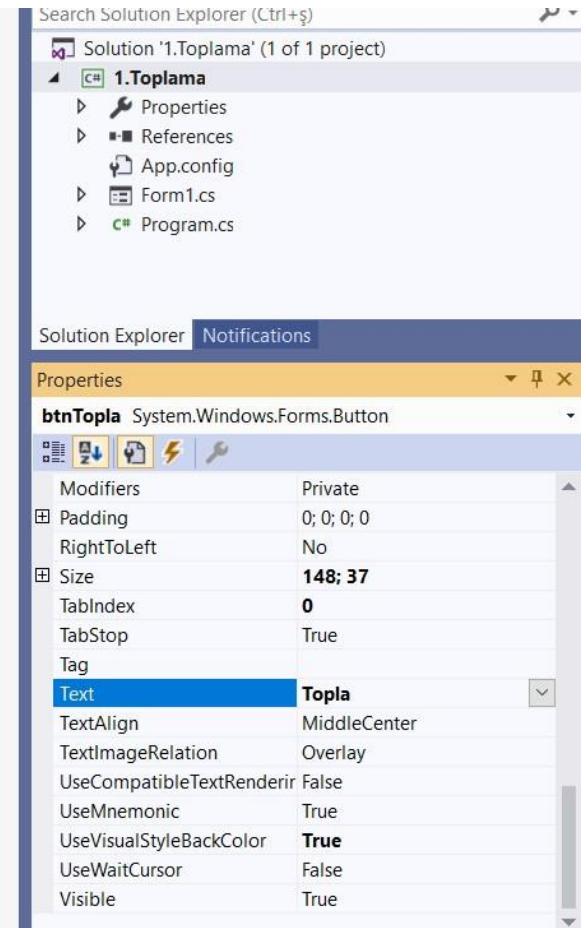
Property Adı	Açıklama
AcceptButton	Aktif formda Enter tuşuna basıldığı zaman hangi butonun çalışacağını belirtir.
BackColor	Aktif formun arka plan rengini düzenlememizi sağlar
CancelButton	Aktif formda Esc tuşuna basıldığı zaman hangi butonun çalışacağını belirtir.
ForeColor	Aktif formda kontroller üzerindeki yazıların renklerini düzenlememizi sağlar
Name	Formun Adını düzenlememizi sağlar
Size	Formun Boyutlarını düzenlememizi sağlar
Text	Formun başlığını düzenlememizi sağlar
Visible	Formumuzun görünürlüğünü düzenlememizi sağlar

İlk Windows Form Uygulaması



Toolbox sekmesinden
sürükleyip bırakarak
formumuza buton
ekliyoruz

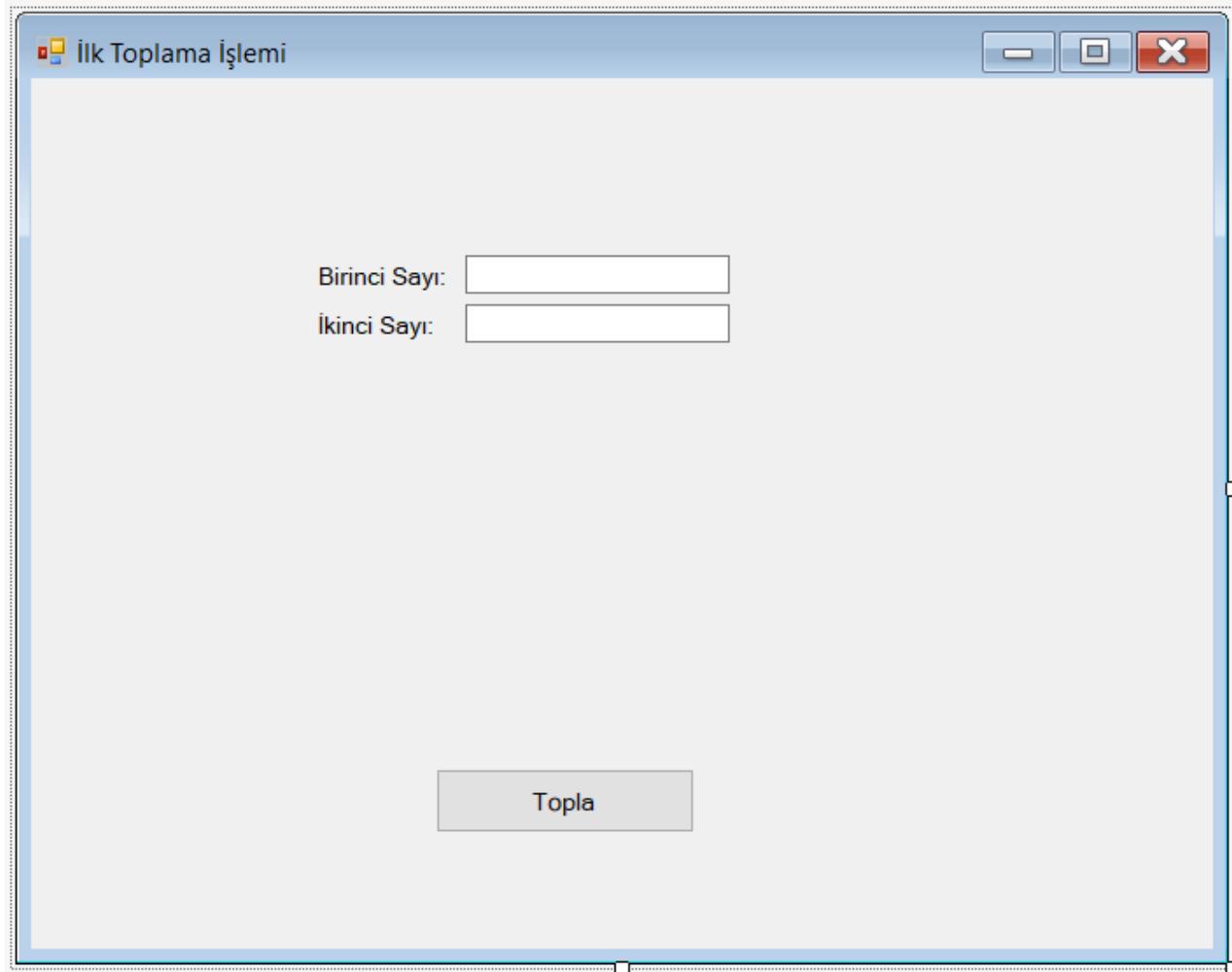
Properties penceresinden
oluşturduğumuz butonun
Text'ini değiştiriyoruz



İlk Windows Form Uygulaması

Property Adı	Açıklama
BackColor	Butonun arka plan rengini düzenlememizi sağlar
Enabled	Butonun Etkin/EtkinDeğil olarak düzenlememizi sağlar
Text	Butonun üzerindeki yazıyı düzenlememizi sağlar
Font	Butonun Text'inin fontunu düzenlememizi sağlar
Name	Butonun adını düzenlememizi sağlar
ForeColor	Butonun Text'inin rengini düzenlememizi sağlar
Size	Butonun boyutlarının düzenlememizi sağlar
Visible	Butonun görünürüğünü düzenlememizi sağlar

İlk Windows Form Uygulaması



- Form ismini değiştirdik.
- İki label ekledik.
- İki textbox ekledik.
- İsimlendirmeleri yaptık.
- Ekranda biraz küçük kaldılar ve işin doğrusu biraz şekilsizler.
- Yani bir yererde eksiklik var ama olsun şimdilik görmezden gelelim ve «**Functionality First**» diyelim ☺

İlk Windows Form Uygulaması

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar has 'File', 'Edit', 'View', 'Project', 'Toolbox', 'Help', and 'File' (dropdown). The title bar says 'Form1.cs* Form1.cs [Design]*'. The main window displays the code for 'Form1.cs'.

```
Form1.cs* Form1.cs [Design]*  
1.Toplama  
1.using System;  
2.using System.Collections.Generic;  
3.using System.ComponentModel;  
4.using System.Data;  
5.using System.Drawing;  
6.using System.Linq;  
7.using System.Text;  
8.using System.Threading.Tasks;  
9.using System.Windows.Forms;  
10.  
11. namespace _1.Toplama  
12. {  
13.     3 references  
14.     public partial class ToplamaFormu : Form  
15.     {  
16.         1 reference  
17.         public ToplamaFormu()  
18.         {  
19.             InitializeComponent();  
20.         }  
21.         1 reference  
22.         private void BtnTopla_Click(object sender, EventArgs e)  
23.         {  
24.         }  
25.     }  
26.  
110 % No issues found Error List  
Entire Solution 0 Errors 0 Warnings 0 Messages Build + IntelliSense Search Error List  
Code Description Project File Line Suppression St... Error List Output
```

Annotations on the code:

- An orange box surrounds the 'using' statements, with an arrow pointing to the text "Kullanılan Kütüphaneler".
- An orange box surrounds the entire class definition, with an arrow pointing to the text "Kodların Bulunduğu Bölüm".
- An orange box surrounds the 'BtnTopla_Click' event handler, with an arrow pointing to the text "Button'a tıklayınca çalışacak fonksiyon".

- Butona çift tıkladık ve **kodlama penceresi** açıldı.
- **BtnTopla_Click** isimli bir fonksiyon oluştı.
- Bunun adı **event** yani **olay**.
- Tıklama olayı, ama tek tıklama!
- Her bileşenin birden çok event'i var.
- **Event:** Kullanıcının uygulama bileşenleriyle etkileşime girdiğinde çalışan aksiyon ve kod parçası.

İlk Windows Form Uygulaması

```
1 reference
private void BtnTopla_Click(object sender, EventArgs e)
{
    int birinciSayi, ikinciSayi, toplam;
    birinciSayi = Convert.ToInt32(txtBirinciSayi.Text);
    ikinciSayi = Convert.ToInt32(txtIkinciSayi.Text);
    toplam = birinciSayi + ikinciSayi;
    MessageBox.Show("Toplam: " + toplam);
}
```

Versiyon 1.0

- Kodu yazdık
- İsimlendirme standartlarına uyduk. X, Y vs. kullanmadık.
- Toplam sonucunun ekranada görünmesi için de Mesaj olarak gösterdik.

Eksikler

- Event'lerin içerisinde bu şekilde direkt **iş fonksiyon kodu** yazılmaz, ayıptır.
- Validasyon yani giriş doğruluk kontrolü yok. Boşluk ya da metin girilebiliyor.
- Hata yönetimi yok.

İlk Windows Form Uygulaması

```
private int Topla (int sayı1, int sayı2)
{
    return sayı1 + sayı2;
}

1 reference
private Boolean Dogrula()
{
    try
    {
        Convert.ToInt32(txtBirinciSayı.Text);
        Convert.ToInt32(txtİkinciSayı.Text);
        return true;
    }
    catch (Exception)
    {
        return false;
    }
}

1 reference
private void BtnTopla_Click(object sender, EventArgs e)
{
    if (Dogrula())
    {
        int toplam = Topla(Convert.ToInt32(txtBirinciSayı.Text),
                           Convert.ToInt32(txtİkinciSayı.Text));
        MessageBox.Show("Toplam: " + toplam);
    }
    else
        MessageBox.Show("Girdiğiniz değerlerde hata var!");
}
```

Versiyon 1.1

- *Topla()* isimli bir fonksiyonumuz var.
- *Dogrula()* isimli bir metodumuz var.
- Doğrulama için hızlı bir trick yaptık.

Eksikler

- Doğrulama trick iyi değil. Hatanın hangi kontrolde olduğunu anlamıyoruz.
- Ekleme işlemi başarılıysa ekran temizlenmiyor.
- Bileşene focus yok.
- Ekran hazırlama gibi bir fonksiyona ihtiyaç olabilir.

İlk Windows Form Uygulaması

```
private void EkraniAyarla()
{
    txtBirinciSayi.Text = txtIkinciSayi.Text = "0";
    txtBirinciSayi.Focus();
}

private void BtnTopla_Click(object sender, EventArgs e)
{
    if (Dogrula())
    {
        int toplam = Topla(Convert.ToInt32(txtBirinciSayi.Text),
                           Convert.ToInt32(txtIkinciSayi.Text));
        MessageBox.Show("Toplam: " + toplam);
    }
    else
        MessageBox.Show("Girdiğiniz değerlerde hata var!");

    EkraniAyarla();
}

private void ToplamaFormu_Shown(object sender, EventArgs e)
{
    EkraniAyarla();
}
```

Versiyon 1.2

- *EkraniAyarla()* isimli bir fonksiyonumuz var. Fonksiyonu ihtiyacımız olduğunda çağrıyoruz.
- *_Shown()* isimli formumuz açıldığında otomatik olarak açılan bir metodumuz/event’imiz var.

Eksikler

- Doğrulama trick iyi değil. Hatanın hangi kontrolde olduğunu anlamıyoruz.
- Ekran UI (ön yüz) tarafını biraz daha iyileştirebiliriz.
- Sorularımız var/olmalı/neler?
 - *Ortamı ve oluşturduğumuz dosyaları biraz daha tanımadız lazımdır.*
 - Hata olursa nasıl bulacağız? Debug?

İlk Windows Form Uygulaması

Dosya Hiyerarşisi

Ad	Değiştirme tarihi	Tür	Boyut
1.Toplama	25.08.2019 11:25	Dosya klasörü	
1.Toplama.sln	24.08.2019 22:06	Visual Studio Solut...	2 KB

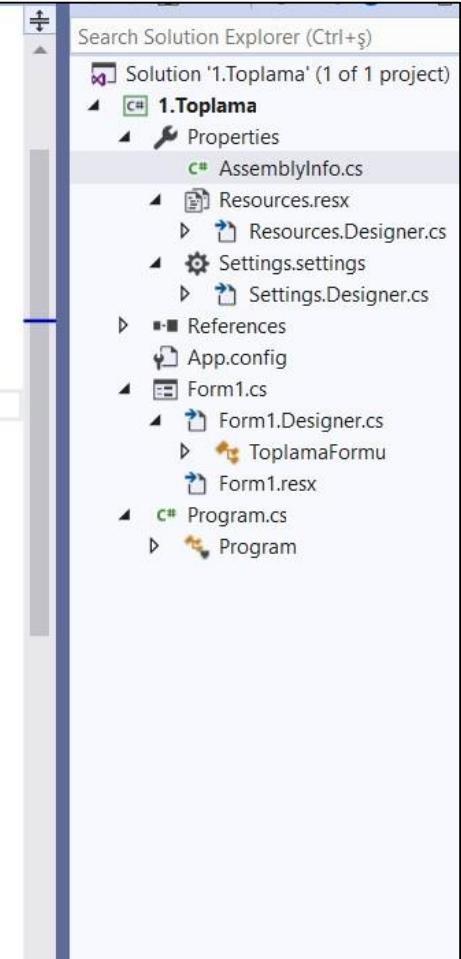
Ad	Değiştirme tarihi	Tür	Boyut
bin	24.08.2019 22:05	Dosya klasörü	
obj	24.08.2019 22:05	Dosya klasörü	
Properties	24.08.2019 22:05	Dosya klasörü	
1.Toplama.csproj	24.08.2019 22:27	Visual C# Project F...	
App.config	25.08.2019 11:25	XML Configuration...	
Form1.cs	25.08.2019 10:07	Visual C# Source F...	
Form1.Designer.cs	25.08.2019 10:02	Visual C# Source F...	
Form1.resx	25.08.2019 10:02	Microsoft .NET Ma...	
Program.cs	24.08.2019 22:23	Visual C# Source F...	

Ad	Değiştirme tarihi	Tür	Boyut
AssemblyInfo.cs	24.08.2019 22:05	Visual C# Source F...	2 KB
Resources.Designer.cs	24.08.2019 22:05	Visual C# Source F...	3 KB
Resources.resx	24.08.2019 22:05	Microsoft .NET Ma...	6 KB
Settings.Designer.cs	24.08.2019 22:05	Visual C# Source F...	2 KB
Settings.settings	24.08.2019 22:05	Settings-Designer ...	1 KB

ilk Windows Form Uygulaması

AssemblyInfo.cs – Assembly Ayarları

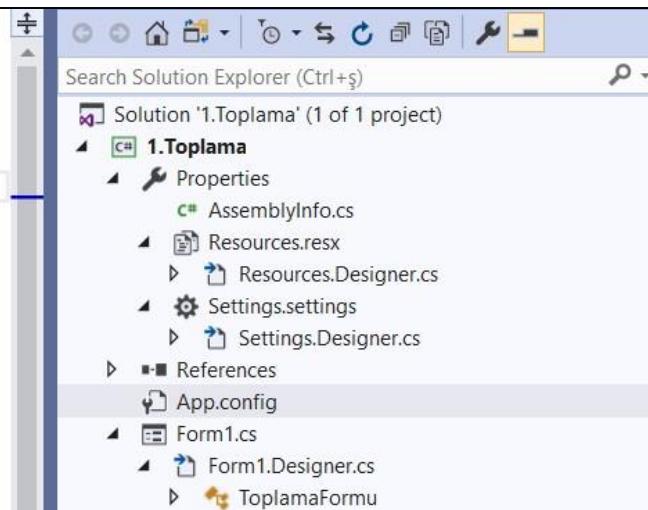
```
7 // associated with an assembly.
8 [assembly: AssemblyTitle("1.Toplama")]
9 [assembly: AssemblyDescription("")]
10 [assembly: AssemblyConfiguration("")]
11 [assembly: AssemblyCompany("")]
12 [assembly: AssemblyProduct("1.Toplama")]
13 [assembly: AssemblyCopyright("Copyright © 2019")]
14 [assembly: AssemblyTrademark("")]
15 [assembly: AssemblyCulture("")]
16
17 // Setting ComVisible to false makes the types in this assembly not visible
18 // to COM components. If you need to access a type in this assembly from
19 // COM, set the ComVisible attribute to true on that type.
20 [assembly: ComVisible(false)]
21
22 // The following GUID is for the ID of the typelib if this project is exposed to COM
23 [assembly: Guid("f6a7b134-95d8-47aa-9b65-58773f8e9ddb")]
24
25 // Version information for an assembly consists of the following four values:
26 //
27 //      Major Version
28 //      Minor Version
29 //      Build Number
30 //      Revision
31 //
32 // You can specify all the values or you can default the Build and Revision Numbers
33 // by using the '*' as shown below:
34 // [assembly: AssemblyVersion("1.0.*")]
35 [assembly: AssemblyVersion("1.0.0.0")]
36 [assembly: AssemblyFileVersion("1.0.0.0")]
```



ilk Windows Form Uygulaması

App.config – Uygulama Parametreleri

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <configuration>
3   <startup>
4     <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
5   </startup>
6 </configuration>
```



İlk Windows Form Uygulaması

Form1.Designer.cs – Form ve UI Bileşenlerinin Detayları

The screenshot shows the Microsoft Visual Studio interface. On the left is the Solution Explorer window, which lists the project '1.Toplama' with its files: AssemblyInfo.cs, Resources.resx, Resources.Designer.cs, Settings.settings, Settings.Designer.cs, References, App.config, Form.cs, Form1.Designer.cs (which is selected), Form1.resx, Program.cs, and Program. In the center is the code editor window displaying the contents of Form1.Designer.cs. The code defines a partial class ToplamaFormu within the namespace _1.Toplama. It includes standard boilerplate code for a Windows Form designer, followed by a region for Windows Form Designer generated code. This region contains the initialization logic for several UI components: btnTopla (a button), lblBirinciSayi (a label), txtBirinciSayi (a text box), txtikinciSayi (another text box), and lblIkinciSayi (another label). The code uses the System.Windows.Forms namespace for these components.

```
namespace _1.Toplama
{
    partial class ToplamaFormu
    {
        /// <summary> Required designer variable.
        private System.ComponentModel.IContainer components = null;

        /// <summary> Clean up any resources being used.
        protected override void Dispose(bool disposing)...

        #region Windows Form Designer generated code

        /// <summary> Required method for Designer support - do not modify the contents ... 
        private void InitializeComponent()
        {
            this.btnTopla = new System.Windows.Forms.Button();
            this.lblBirinciSayi = new System.Windows.Forms.Label();
            this.txtBirinciSayi = new System.Windows.Forms.TextBox();
            this.txtikinciSayi = new System.Windows.Forms.TextBox();
            this.lblIkinciSayi = new System.Windows.Forms.Label();
            this.SuspendLayout();

            //
            // btnTopla
            //
            this.btnTopla.Location = new System.Drawing.Point(231, 394);
            this.btnTopla.Name = "btnTopla";
            this.btnTopla.Size = new System.Drawing.Size(148, 37);
            this.btnTopla.TabIndex = 0;
            this.btnTopla.Text = "Topla";
            this.btnTopla.UseVisualStyleBackColor = true;
            this.btnTopla.Click += new System.EventHandler(this.BtnTopla_Click);
        }
    }
}
```

İlk Windows Form Uygulaması

Program.cs – İlk Çalışan Application C# dosyası (Main() fonksiyonu)

The screenshot shows the Visual Studio IDE interface. On the left, the Project Explorer displays the project structure:

- AssemblyInfo.cs
- Resources.resx
- Resources.Designer.cs
- Settings.settings
- Settings.Designer.cs
- References
- App.config
- Form1.cs
- Form1.Designer.cs
 - ToplamaFormu
- Form1.resx
- Program.cs
- Program

The code editor on the right contains the content of Program.cs:namespace _1.Toplama
{
 static class Program
 {
 /// <summary>
 /// The main entry point for the application.
 /// </summary>
 [STAThread]
 static void Main()
 {
 Application.EnableVisualStyles();
 Application.SetCompatibleTextRenderingDefault(false);
 Application.Run(new ToplamaFormu());
 }
 }
}

İlk Windows Form Uygulaması

A screenshot of the Visual Studio IDE. The title bar says "1.Toplama". The "Debug" menu is open, showing options like Start Debugging (F5), Start Without Debugging (Ctrl+F5), Performance Profiler..., Attach to Process..., Step Into (F11), Step Over (F10), Toggle Breakpoint (F9), New Breakpoint, Delete All Breakpoints (Ctrl+Shift+F9), Disable All Breakpoints, Clear All DataTips, Export DataTips..., Import DataTips..., Options..., and 1.Toplama Properties... . The code editor shows a C# file named "Form1.cs [Design]" with the following code:

```
1.Toplama
private void BtnTopla_Click(object sender, EventArgs e)
{
    try
    {
        private int Topla(string birinciSayi, string ikinciSayi)
        {
            int toplam = Convert.ToInt32(birinciSayi) +
                         Convert.ToInt32(ikinciSayi);
            return toplam;
        }

        if (Dogrula())
        {
            int toplam = Topla(Convert.ToInt32(txtBirinciSayi.Text),
                               Convert.ToInt32(txtIkinciSayi.Text));
            MessageBox.Show("Toplam: " + toplam);
        }
        else
            MessageBox.Show("Girdiğiniz değerlerde hata var!");

        EkraniAyarla();
    }
}
```

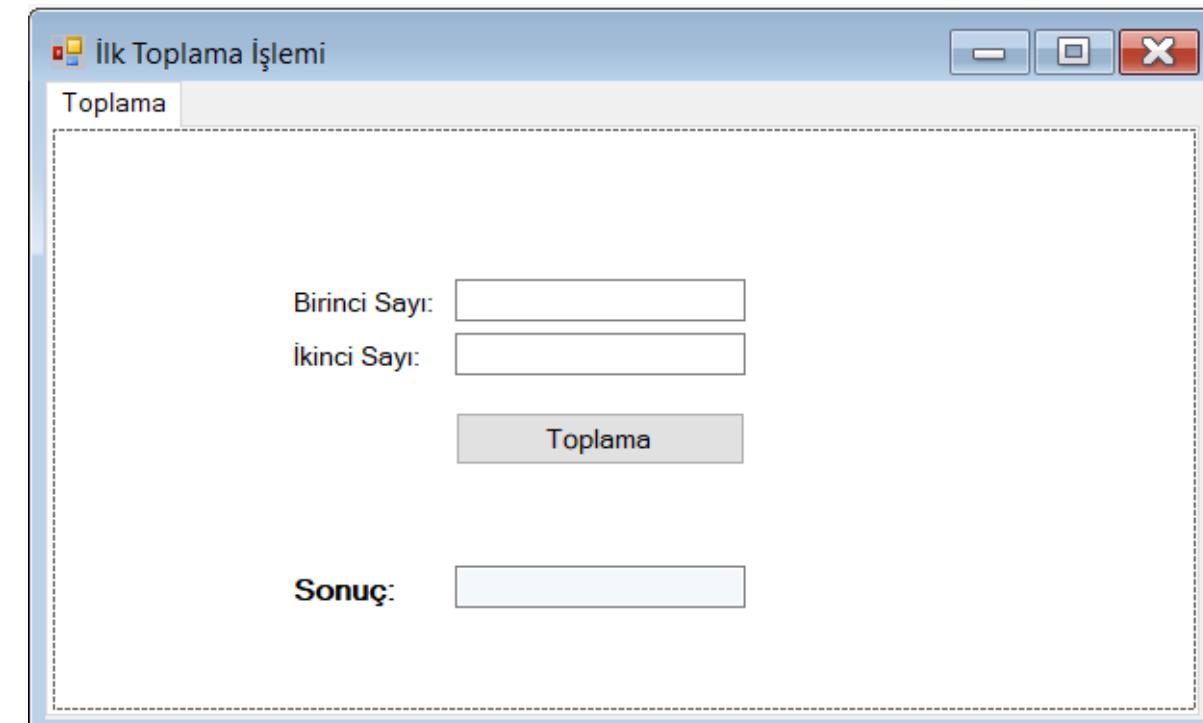
Debug !!!

- *Debug edebilme, uygulamayı adım adım izleyebilmemizi ve hataları bulmamızı sağlar.*
- *Breakpoint koyarak ilgili satırlardan ilerleyebiliriz.*
- *Step-Into ve Step-Over yapabilmek çok önemli.*
- *Breakpoint'ler disable edilebilmeli.*
- *Değişkenlerin değerleri izlenebilmeli.*

İlk Windows Form Uygulaması

Versiyon 1.3 - İhtiyaçlar

- TabControl bileşeni atalım ve bu bileşenin özelliklerini inceleyelim.
 - İki tane tabpage olacak, ikincisini silelim.
- Anchor özelliğini görelim (Maksimize edince nasıl görüntü bozuluyor).
- Toplam için yeni bir textbox ekleyelim ve Enabled false yapalım.
- Form açıldığında bilgisayar ekranını ortalaşın (StartPosition özelliği).
- Toplam değerini bu bileşene yazacak şekilde kodu revize edelim.



İlk Windows Form Uygulaması

Versiyon 1.3 - Çıktı

- TabControl işi tamamlandı.
- Anchor işi tamamlandı.
- Form açıldığında bilgisayar ekranını ortalaşın (StartPosition özelliği).
- Toplam değerini göstermek için yeni bileşen kullanıldı.

Eksikler

- Doğrulama trick iyi değil. Hatanın hangi kontrolde olduğunu anlamıyoruz.
- Yeni bileşen temizlenmiyor. Nasıl temizlenmeli? (Hata/Eksik?)

```
private void BtnTopla_Click(object sender, EventArgs e)
{
    if (Dogrula())
    {
        int toplam = Topla(Convert.ToInt32(txtBirinciSayi.Text),
                           Convert.ToInt32(txtIkinciSayi.Text));
        txtSonuc.Text = toplam.ToString();
    }
    else
        MessageBox.Show("Girdiğiniz değerlerde hata var!");
    EkraniAyarla();
}
```

Kodda değişen satır ...

İlk Windows Form Uygulaması

Versiyon 1.3.1 - Çıktı

- EkraniAyarla() fonksiyonu ve Toplama event'i değiştirildi.

Eksikler

- Doğrulama trick iyi değil ya da diğer bir deyişle yeterli değil. Hatanın hangi kontrolde olduğunu anlamıyoruz.
 - Bu eksik beklemeye devam ediyor.
 - Bazı işler böyledir hiç sıra gelmez 😊
 - Burada validasyon için daha köklü bir çözüm yapabiliriz/yapmalıyız.
 - Nedir? Tartışalım...

Varsayılan değere sahip parametre eklendi...

```
private void EkraniAyarla(bool toplamBasarili = false)
{
    if (!toplamBasarili)
        txtBirinciSayi.Text = txtIkinciSayi.Text = txtSonuc.Text = "0";
    txtBirinciSayi.Focus();
}
```

```
private void BtnTopla_Click(object sender, EventArgs e)
{
    bool dogrulamaSonucu = Dogrula();
    if (dogrulamaSonucu)
    {
        int toplam = Topla(Convert.ToInt32(txtBirinciSayi.Text),
                           Convert.ToInt32(txtIkinciSayi.Text));
        txtSonuc.Text = toplam.ToString();
    }
    else
        MessageBox.Show("Girdiğiniz değerlerde hata var!");
    EkraniAyarla(dogrulamaSonucu);
}
```

İlk Windows Form Uygulaması

Versiyon 1.3.2 - Çıktı

- Textbox bileşenlerine sadece sayı girilmesine izin verildi.
 - Bu özelliğin desteklenmesi için bileşenlerin KeyPress event'leri kullanıldı.

Eksikler

- Textbox bileşen girişinde bir sıkıntı var (mı)?

Yeni Talep

- Toplama butonuna her basıldığında bir loglama yapmak istiyoruz?

Soru?

- Neden sadece tam sayı toplayabiliyoruz?
Ondalıklı sayı toplayamaz mıyız?

```
private void TxtBirinciSayi_KeyPress(object sender, KeyPressEventArgs e)
{
    if (
        char.IsLetter(e.KeyChar) ||
        char.IsSymbol(e.KeyChar) ||
        char.IsWhiteSpace(e.KeyChar) ||
        char.IsPunctuation(e.KeyChar)
    )
        e.Handled = true;
}

1 reference
private void TxtIkinciSayi_KeyPress(object sender, KeyPressEventArgs e)
{
    if (
        char.IsLetter(e.KeyChar) ||
        char.IsSymbol(e.KeyChar) ||
        char.IsWhiteSpace(e.KeyChar) ||
        char.IsPunctuation(e.KeyChar)
    )
        e.Handled = true;
}
```

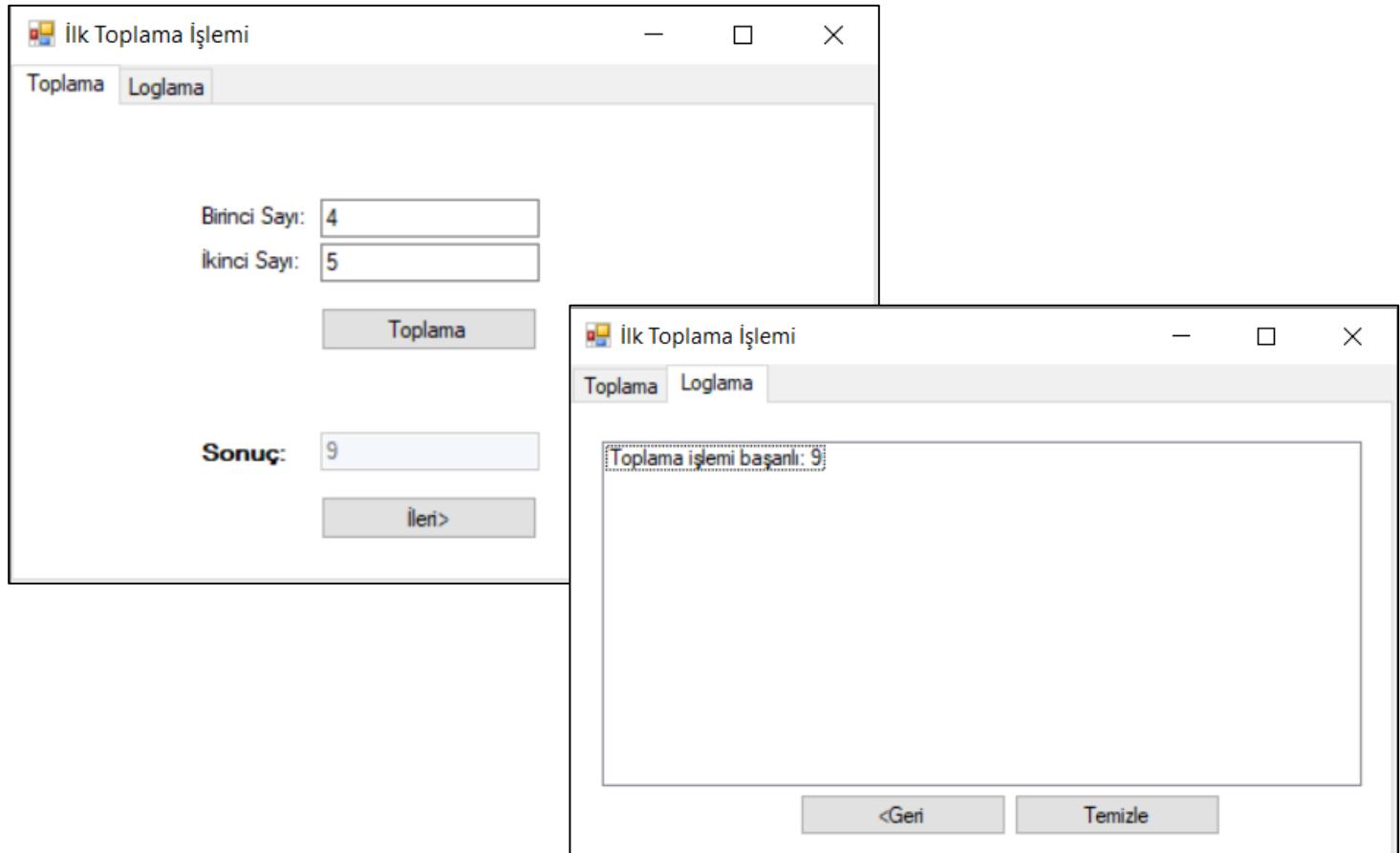
İlk Windows Form Uygulaması

Versiyon 1.4 - Çıktı

- Toplama butonuna her basıldığında bir loglama işlemi yaptık. Bunun için bir listview kullandık. Listview'i ikinci bir TabControl içerisinde yerleştirdik.

Soru?

- Neden sadece tam sayı toplayabiliyoruz? Ondalık sayı toplayamaz mıyız?



Sabitler

- Sabitler, içeriği sabit olan değer ve ifadelerin saklanması için kullanılan değişken türleridir.
- Başlangıçta sabit değişkene bir değer vererek, ihtiyaç duyulduğunda bu değişkeni kullanabiliriz.
- Değerini sonradan değiştirmek istediğimizde derleyici, **derleme zamanı** hatası verir.
- Sabit değişkenlerinin **tanımlandığı satırda değer verilmek zorundadır**.
- C# programlama dilinde **const** anahtar sözcüğü kullanılarak tanımlanır.

```
const double pi = 3.141592;  
const int sayı = 9;  
const char ilkKarakter = 'a';
```

Değişkenler

- Program içerisinde üretilen değerleri **bellekte geçici olarak** saklamak için kullanılırlar.
- İlk değerlerinin atanması **zorunlu değildir**. Program içerisinde de sonradan değer ataması yapılabilir.
- Aynı satırda aynı tipte birden fazla değişken tanımlaması ve değer ataması yapılabilir.
- Global olarak tanımlanmayan değişkenler **sadece tanımlandıkları metot içerisinde** kullanılabilirler.
- Değişkenlerin tipini öğrenmek için **GetType()** metodu kullanılır.

[değişkenin tipi] [değişkenin adı] = [ilk değer] ;

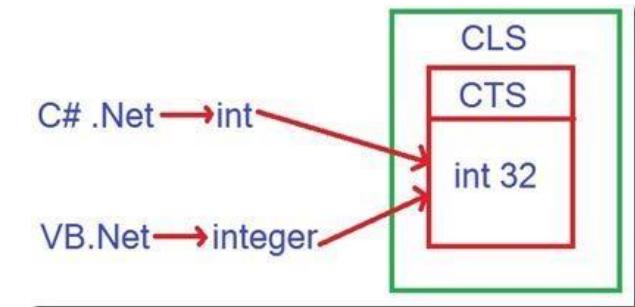
```
int yarıCap = 12;  
float alan;  
int yükseklik = 2, taban = 4;  
string ad = "Ada", soyad = "Kılınç";
```

Değişken Türleri

Değişken Türü	Boyutu	CTS	Değer Aralığı	Örnek
byte	1 bayt	System.Byte	0, ..., 255 (tam sayı)	byte a=5;
sbyte	1 bayt	System.Byte	-128, ..., 127 (tam sayı)	sbyte a=5;
short	2 bayt	System.Int16	-32768, ..., 32767 (tam sayı)	short a=5;
ushort	2 bayt	System.UInt16	0, ..., 65535 (tam sayı)	ushort a=5;
int	4 bayt	System.Int32	-2147483648, ..., 2147483647 (tam sayı)	int a=5;
uint	4 bayt	System.UInt32	0, ..., 4294967295 (tam sayı)	uint a=5;
long	8 bayt	System.Int64	-9223372036854775808, ..., 9223372036854775807 (tam sayı)	long a=5;
ulong	8 bayt	System.UInt64	0, ..., 18446744073709551615 (tam sayı)	ulong a=5;
float	4 bayt	System.Single	$\pm 1.5 \cdot 10^{-45}$, ..., $\pm 3.4 \cdot 10^{38}$ (reel sayı)	float a=5F; veya float a=5f;
double	8 bayt	System.Double	$\pm 5.0 \cdot 10^{-324}$, ..., $\pm 1.7 \cdot 10^{308}$ (reel sayı)	double a=5; veya double a=5d; veya double a=5D;
decimal	16 bayt	System.Decimal	$\pm 1.5 \cdot 10^{-28}$, ..., $\pm 7.9 \cdot 10^{28}$ (reel sayı)	decimal a=5M; veya decimal a=5m;
bool	1 bayt	System.Boolean	True ya da False	bool cevap = False;
char	1 bayt	System.Char	16 Bit Unicode Karakter	char harf = 'a';

Değişken Türleri (devam...)

- Değişken tanımlamaları istenirse programlama diline ait değişken tipi tanımlamasıyla, istenirse .NET uygulama geliştirme platformunun bir özelliği olan CTS (Common Type System) tanımlaması ile yapılabilir.
- .NET uygulama geliştirme platformunda **programlama dilleri arasındaki standardizasyonu** CTS dediğimiz sistem sağlar.
- C#.NET ile program yazarken tamsayı veri tipinde değişken tanımlamak için **int** ifadesini kullanırız.
- VB.NET'te ise tamsayı veri tipi **Integer** olarak geçer.
- Her iki dil farklı olsa da CTS dediğimiz sistem sayesinde **int** ve **Integer** tanımlamalar MSIL'e çevrilirken CTS sisteminde tanımlanan **System.Int32** olarak ara-dile çevrilecektir.



```
System.Int32 yarıCap = 4;  
System.Char ilkKarakter = 'a';
```

```
int yarıCap = 12;  
char ilkKarakter = 'a';
```

Değişken Tip Dönüşümleri (Type-Casting)

- Üstü Kapalı (**Implicit**) Tip Dönüşümü
- Açık (**Explicit**) Tip Dönüşümü
- **Tostring()** Metodu İle Tip Dönüşümü
- **Convert** Sınıfı İle Tip Dönüşümü

Üstü Kapalı (Implicit) Tip Dönüşümü

- Değişkenin tanımlandığı veri tipinden daha yüksek kapasiteli bir veri tipindeki değişkene atanabilir. Bu duruma **Üstü Kapalı (Implicit) Tür Dönüşümü** denir.

```
int sayı1 = 10;
float sayı2 = sayı1; //sayı1 değişkeni bulunduğu int veri tipinden daha kapasiteli float veri tipine dönüştü

char cinsiyet = 'K';
int Cinsiyet = cinsiyet; //cinsiyet değişkeni bulunduğu char veri tipinden daha kapasit. int veri tipine dönüş.

short sayı3 = 10882;
long sayı4 = sayı3; //sayı3 değişkeni bulunduğu short veri tipinden daha kapasiteli long veri tipine dönüştü
```

Düşük kapasiteden yüksek kapasiteye dönüsürken sorun yok.

Açık (Explicit) Tip Dönüşümü

- **Açık Tip Dönüşümü**, derleyicinin izin vermediği durumlarda kullanılır.
[Değişken] = (hedef tür) [değişken adı]

```
int a = 120;  
byte b;
```

```
//Aşağıdaki kodu çalıştırduğumızda aşağıdaki gibi bir hata alırız  
//Cannot implicitly convert type 'int' to 'byte'...  
b = a;
```

```
/*a değişkenini daha az kapasiteli byte'a tipine çeviriyoruz.  
Şanslıyız çünkü değer 0-255 arasında. Yani veri kaybımız olmadı.*/  
b = (byte)a;
```

```
/*a değişkenini yine daha az kapasiteli byte'a tipine çeviriyoruz.  
Değer 255'den büyük olduğu için veri kaybımız yaşadık.*/
```

```
a = 498;  
b = (byte)a;
```

ToString() Metodu İle Tip Dönüşümü

- **ToString()** metodu Bir değişken veya sabitin değerini **string** veri tipine dönüştürerek tutar.
- Kullanımı: **[değişken adı] . ToString();**

```
bool a = true;  
Console.WriteLine(a.ToString());
```

```
int sayı = 98;  
Console.WriteLine(sayı.ToString());
```

```
float sayı2 = 98.875f;  
Console.WriteLine(sayı2.ToString());
```



True

98

98,875

Convert Sınıfı İle Tip Dönüşümü

- **System** isim alanının altındaki **Convert** sınıfı tür dönüşümü yapılabilen metotları içeren bir sınıfır.
- Bu metodlar ile hemen hemen her türü, her türe CTS karşılıklarını kullanarak dönüştürülür.
- Kullanımı: *[değişken] = Convert . [hedef tür] ([değişken]);*

```
bool a = true;  
Console.WriteLine(Convert.ToString(a));
```

```
int sayı = 98;  
Console.WriteLine(Convert.ToString(sayı));
```

```
float sayı2 = 98.875f;  
Console.WriteLine(Convert.ToString(sayı2));
```

True
98
98,875

Operatörler

- Programlama dillerinde tek başlarına herhangi bir anlamı olmayan ancak programın işleyişine katkıda bulunan karakter ya da karakter topluluklarına **operatör** denir. Örneğin **a+b** ifadesinde **+** işaretinin bir operatördür.
- Operatörlerin etki ettikleri sabit ya da değişkenlere ise **operand** denir. Örneğin **a+b** ifadesinde **a** ve **b** birer operanddır.

Operatörler
()
! + - ++ --
* / %
+ -
< <= > >= is as
== !=
&
^
&&
: ?
= *= /= %= += -= &=
^= !=

En Yüksek

En Düşük

Operatörlerde
İşlem önceliği

Operatörler (devam...)

- a) Aritmetik Operatörler
- b) Atama Operatörü
- c) Mantıksal Operatörler
- d) Karşılaştırma Operatörleri
- e) «as» ve «is» Operatörleri
- f) typeof operatörü
- g) Bitsel Operatörler

Operatörler (devam...)

Operatör	Açıklama
+	Toplama
-	Çıkarma
*	Çarpma
/	Bölme
%	Mod Alma

Operatör	Açıklama
<	Küçüktür
>	Büyüktür
<=	Küçük Eşittir
>=	Büyük Eşittir
==	Eşittir
!=	Eşit Değildir
-->	Uzaklaşma Operatörü

Operatör	Açıklama
&&	Ve
	Veya
!	Değil
^	Özel Veya
??	Null coalescing
?:	Koşul

Operatör	Açıklama
+=	Eşitliğin sağındaki ve solundaki değeri toplar ve solundaki değişkene atar
-=	Eşitliğin sağındaki ve solundaki değeri çıkarır ve solundaki değişkene atar
*=	Eşitliğin sağındaki ve solundaki değeri çarpar ve solundaki değişkene atar
/=	Eşitliğin sağındaki değeri solundaki değerine bölerek solundaki değişkene atar

for Döngüsü

- Bir veya birden fazla kod satırının belirtilen koşulları sağladığı sürece artış ve ya azalım değeri kadar tekrarlanmasını sağlayan döngüdür.
- Başlangıç ve bitiş değerleri olmak zorundadır.
- Başlangıç değeri, koşul, artış veya azalım kısımları farklı bir yerde tanımlanmak istenirse boş bırakılabilir. Fakat noktalı virgüler kesinlikle olmalıdır.
- Genel yazım biçimi şu şekildedir:

```
for ( [başlangıç değeri] ; [koşul] ; [artış veya azalım] )
{
    . . .
    [isLeme alınacak komutlar];
    . . .
}
```

foreach Döngüsü

- foreach döngüsü bir dizinin ve ya koleksiyonun **her elemanı için bulundurduğu komutları çalıştırın** döngüdür.
- Dizinin her elemanın değerini geçici bir değişkene atayarak bu değişken üzerinde işlemler yapmamızı sağlar.
- **foreach** döngüsü ile dizi veya koleksiyondaki elemanların değerleri değiştirilemez. Sadece «**read – only**» işlemler yapılabilir.
- Dizi veya koleksiyonların eleman sayıları bilinmediğinde kullanılır.
- Genel yazım biçimi şu şekildedir:

```
foreach ( [değişkenVeriTipi] [değişkenAdı] in [dizi/koleksiyon]
{
    . . .
    [işleme alınacak komutlar];
    . . .
}
```

foreach Döngüsü (devam...)

Örnek: Bir dizinin tüm elemanlarının ekrana yazdırılması.

```
int[] sayilar = { 1, 3, 5, 7 };
foreach (var sayi in sayilar)
{
    Console.WriteLine("Okunan sayı:" + sayi.ToString());
}
```

Cıktı:

```
Okunan sayı:1
Okunan sayı:3
Okunan sayı:5
Okunan sayı:7
```

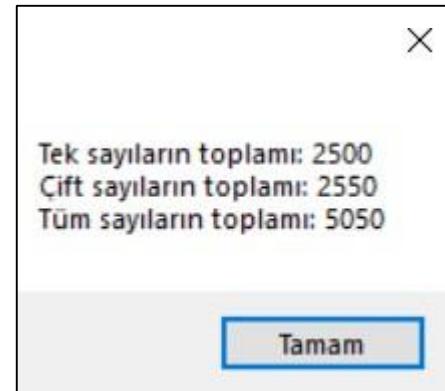
ÖRNEKLER

Örnekler

Örnek 1: 1-100 arasındaki Tek ve Çift sayıların Toplamı

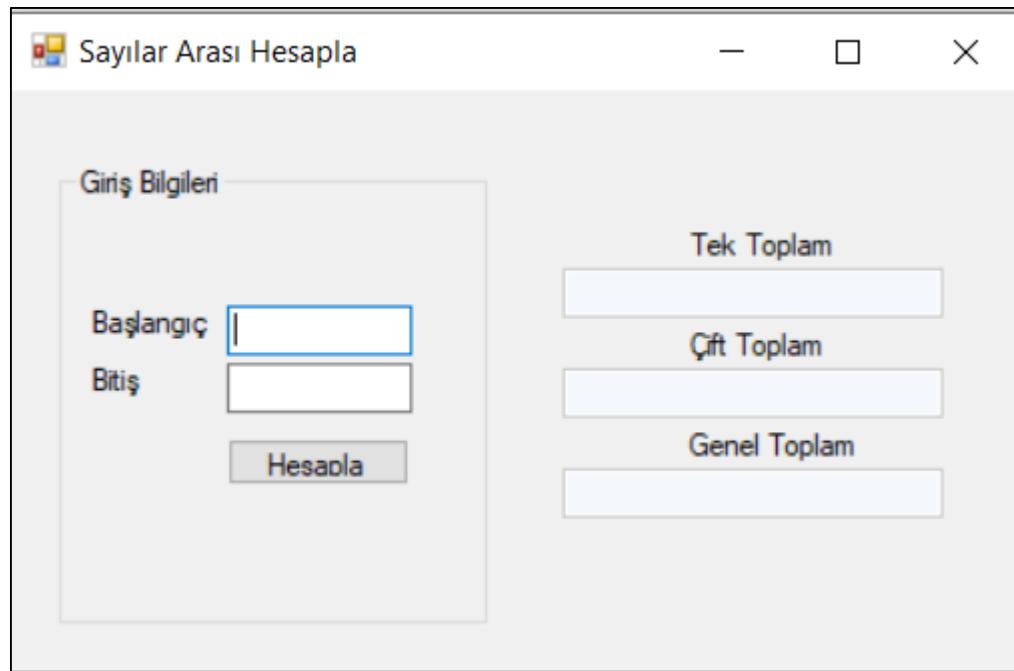
```
int tekToplam = 0, ciftToplam = 0;  
for (int i = 1; i <= 100; i++)  
{  
    if ((i % 2) == 0)  
        ciftToplam += i;  
    else  
        tekToplam += i;  
}
```

```
MessageBox.Show("Tek sayıların toplamı: " + tekToplam +  
               "\nÇift sayıların toplamı: " + ciftToplam +  
               "\nTüm sayıların toplamı: " + (tekToplam + ciftToplam));
```



Örnekler

Örnek 1: 1-100 arasındaki Tek ve Çift sayıların Toplamı,



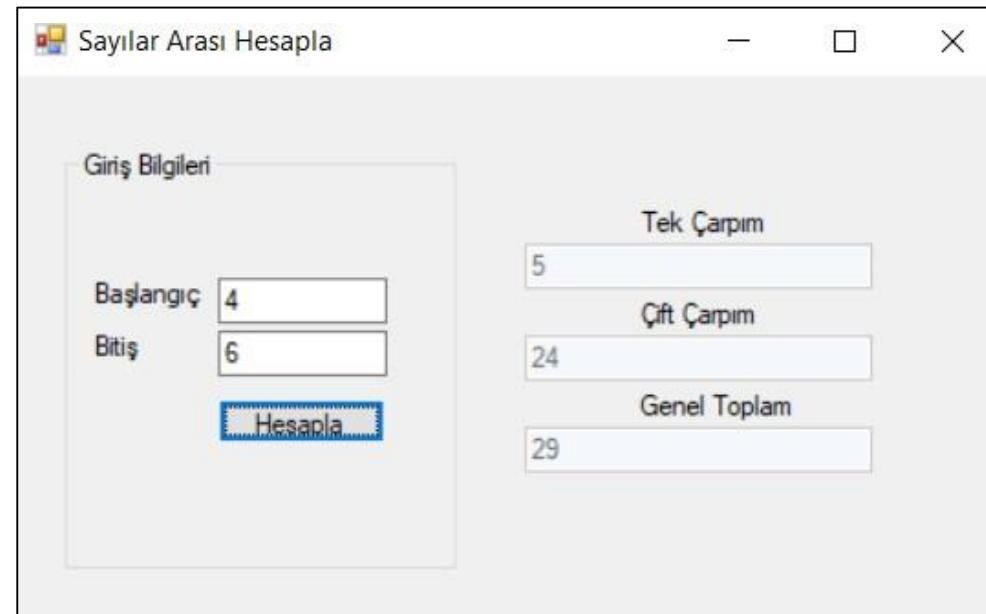
Yeni Versiyon

- Yandaki gibi bir grafik arayüze (GUI) sahip olmalı.
- Sadece sayısal giriş olmalı.
- Başlangıç bitiş değerinden küçük olmalı
- Hesaplama aşağıdaki prototipe sahip bir fonksiyonda yapılmalı.

```
int SonucDon(enToplamTur tur, int bas, int bit)
```

Örnekler

Örnek 2: 1-100 arasındaki Tek ve Çift sayıların Çarpımı (Diğer Formu Kopyalayarak Yapalım)



Nesne Yönelimli Programlamaya Giriş

- Nesne yönelimli programlama (OOP), (Object Oriented Programming) nesneyi merkezine alan bir bilgisayar programlama yaklaşımıdır.
- Nesne yönelimli programlama terminolojisindeki “object/nesne” ve “oriented/yönelimli/yönelik” kavramları ilk olarak 1966 veya 1967 yılında NYP'nin babası olarak bilinen **Alan Kay** tarafından kullanılmıştır.
- NYP paradigmisinin bazı özelliklerini destekleyen ilk programlama dili olan **Simula'dan** sonra, Alan Kay, Dan Ingalls, Adele Goldberg ve arkadaşları tarafından NYP paradigmisini tam destekleyen programlama dili **Smalltalk'ın** ilk versiyonu 1969-1972 yılları arasında **Xerox PARC'ta** geliştirilmiştir.



Nesne Yönelimli Programlamaya Giriş

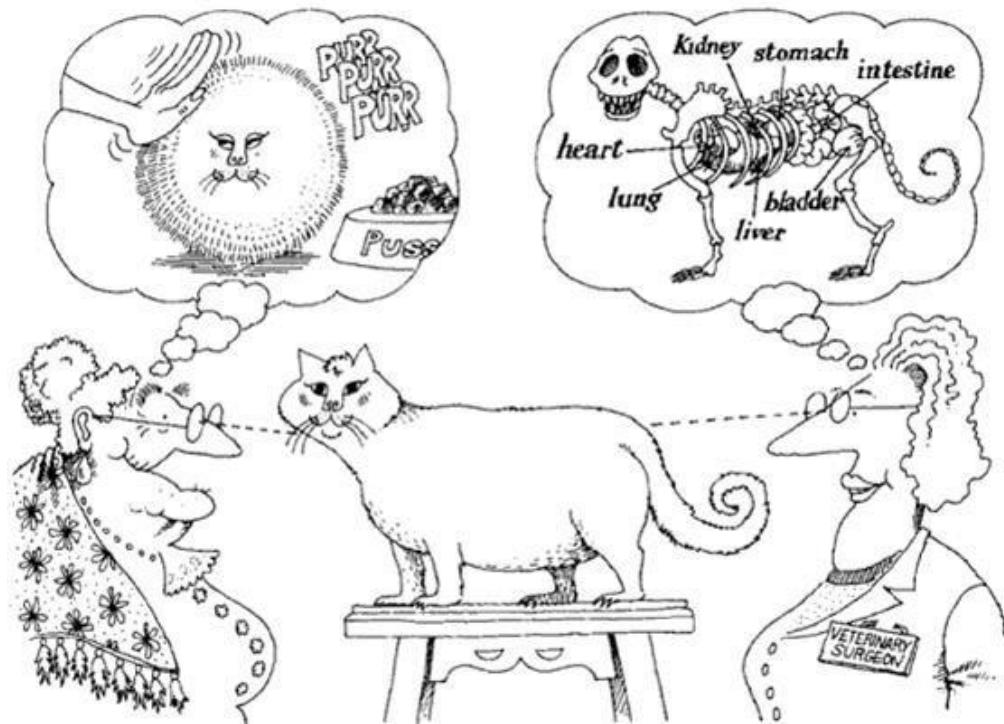
- 90'lı yıllarda programlama dillerine olan destek artınca *nesneye yönelik programlama etkin* ve **yaygın** olarak kullanılmaya başlandı.
- Nesneye yönelik programlamanın temel kavramları;
 - Büyük yazılımlar geliştirmeyi kolaylaştıran **Soyutlama (Abstraction)**,
 - Yazılımları değiştirmeyi ve korumayı kolaylaştıran **Saklama/Sarmalama (Encapsulation)**,
 - Yazılımları genişletilebilir kıyan **Sınıf Hiyerarşisi** ve **Kalıtım (Inheritance)**,
 - Kalıtım hiyerarşisi içerisinde farklı işlemler yapan (fakat) aynı isimdeki özellik veya metodların kullanımına imkan sağlayan **Çok Biçimlilik (Polymorphism)**

Nesne Yönelimli Programlamaya Giriş



Soyutlama

Önemli özelliklere **odaklanabilmek** için ayrintıları
göz ardı etme sürecidir.



Soyutlama (devam...)

- Soyutlama temel olarak **veri** ve **kontrol** soyutlaması olarak yapılır.
 1. Bir veri tipinin nasıl yapılandığının ayrıntılarını göz ardı etmemize izin veren soyutlama tarzına “**veri soyutlaması**” denir.
 2. **Kontrol soyutlaması** ise yapısal programlama ile gelen altprogram, fonksiyon gibi kavramlar üzerinde yapılan soyutlamadır.

Örnek: Bir **Kisi** nesnesinde, kişinin yaşıni tutan bir **tamsayı değişkeni** olan **yas** değişkenini ele alalım.

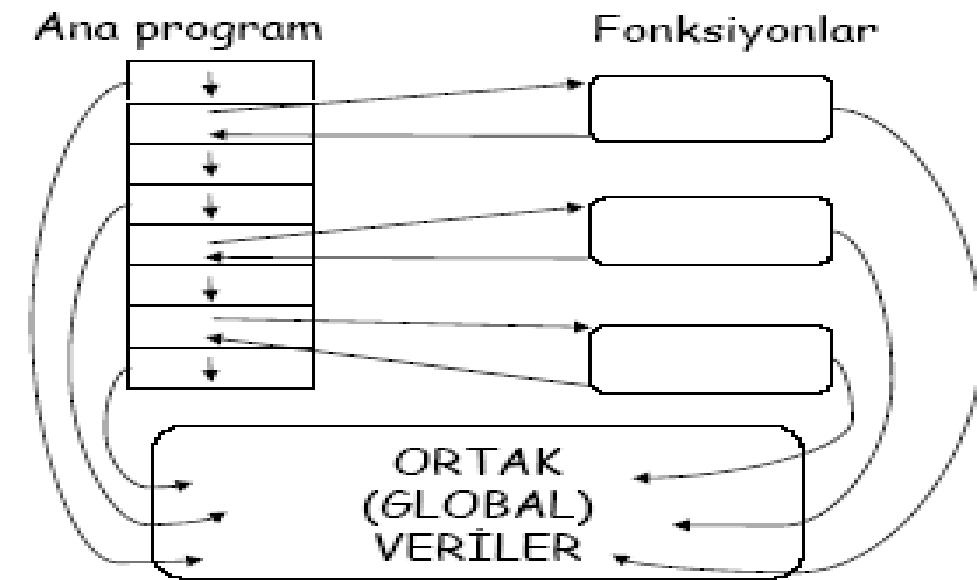
- Bu **yas** değişkenine, programın çalışma anında, "-10" değerinin atanmasını kimse engelleyemez. İşte burada nesne yönelimli programlamanın getirdiği görünürlük ve özellik tanımlama gibi yetenekler kullanarak "yas" değerine gerçek bir değerin girilmesi garanti edilebilir. Buna **veri soyutlaması** adı verilir.

Saklama/Sarmalama (Encapsulation)

- Gerçek hayatı **information hiding** yöntemiyle sıkça karşılaşırız.
- Örneğin arabanın deposuna ne kadar yakıt kaldığını öğrenmek için depoya baktığımızda göremeyiz.
- Bunun yerine gerekli bilgiyi bize aracın gösterge panelindeki yakıt göstergesi vermektedir.
- Saklama işlemi Sınıflar kullanılarak gerçekleştirilir.

Yapısal Programlama vs. Nesne Yönelimli Programlama

- Yapısal Programlamada sadece **bir soruna odaklı farklı fonksiyonlar yazılır** ve **sadece o sorun** için fonksiyonlar bulunur.
- Dolayısıyla yapısal programlama yaklaşımıyla oluşturulmuş bir program **binlerce ayrı isimde tanımlanmış değişken** ve **yüzlerce farklı fonksiyona** sahiptir.
- En ufak işlem için bile **ayrı fonksiyon oluşturulması gereklidir** ve bu programın **karmaşıklığını** artırır.



Gerçek dünyadaki sistemler sadece fonksiyonlardan oluşmaz. Sistemin gerçeğe yakın bir modelini bilgisayarda oluşturmak zordur.

Yapısal Programlama vs. Nesne Yönelimli Programlama

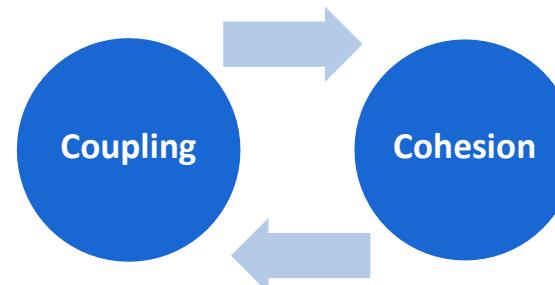
- Nesne yönelimli programlama yaklaşımı **doğaldır**.
- Nesnelerden oluşan bir dünya düşünmek **basittir**.
- Nesne Yönelimli Programlama yaklaşımı yapısal programlama yaklaşımının daha genişletilmiş ve gerçeğe yakın versiyonudur.
 - Değişkenler ve metodlar burada da **kullanılır** fakat burada **nesne** üzerine **odaklanılır**.
- Yapısal programlamadaki **büyük ve karmaşık** sistemleri tasarlamak yerine, birbiriyle iletişim ve etkileşim halinde nesnelerin olduğu bir dünya kolayca tasarlabilir.
 - Bu nesnelerin çalışması için ayrıca bir mantık oluşturmaya gerek kalmaz.
- Nesne yönelimli programlama yaklaşımında, projenin erken safhalarında projeyi tamamlamak için **gerekli** olan **tüm nesnelerin** tanımlanması, nesneleri tanımlayan **sınıfların** **yaratılması** yatar. Böylece yaratılan **her nesne** kendi verisini kendinde barındırır ve **diğer nesnelerin kendisinden** istediği görevleri **yerine getirir**.

Yapısal Programlama vs. Nesne Yönelimli Programlama

Dikkat: Nesne Yönelimli Programlama yanlış bir tasarımla **KARMAŞIKLIĞI** artırabilir.

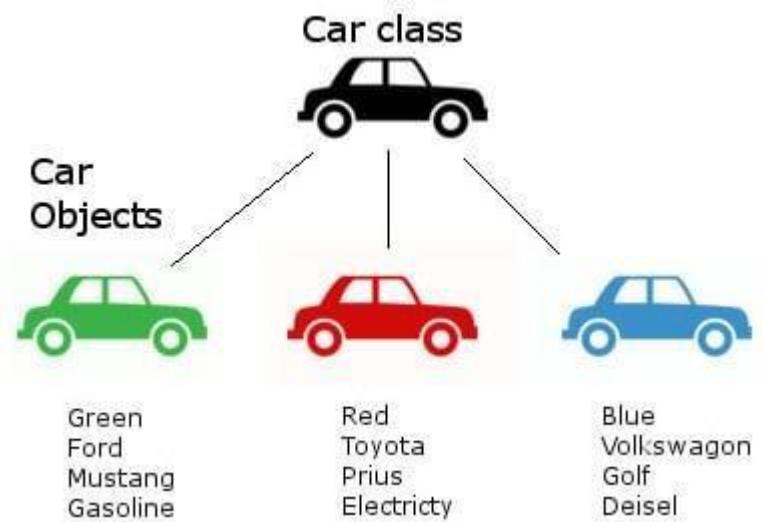
Reçete:

1. Her zaman **Temiz Kodlama** (Clean Code) tarafında kalmalıyız.
2. **Tasarım Prensiplerinden** (Design Principles) ve **Tasarım Kalıplarından** (Design Patterns) uzaklaşmamalıyız.



Nesne Kavramı ilk Tanım

- Nesne, **niteliklere/özelliklere** ve **davranışlara** sahip somut bir varlıktır.
- Bir nesnenin **niteliği/özelliği** onun sahip olduğu **özelliklerdir.**
 - Nesnenin bir **niteliğinin değeri** o nesnenin durumunu oluşturur.
- Bir nesnenin **davranışları** onun yapabildiği / onunla yapılabilen işlemlerdir / operasyonlardır.



Örnek 1: Sipariş verme yazılımı

Yapısal Programlama Çözümü

- Müşteri ve sipariş bilgileri ayrı değişkenlerde tutulurlar. Diziler kullanılır.
- Bu değişkenler üzerinden işlemler yapılır (sipariş ekle, tutar hesaplama, ödeme gerçekleştir ...).

Nesne Yönelimli Programlama Çözümü

- Programın içereceği nesnelerden birisi “**Siparis**” nesnesi olur.

Örnek 1: Sipariş verme yazılımı (devam..)

Siparis nesnesi şunları içerir:

Nitelik/özellik olarak;

- Sipariş Tarihi
- Müşteri
- Durumu

Davranış olarak;

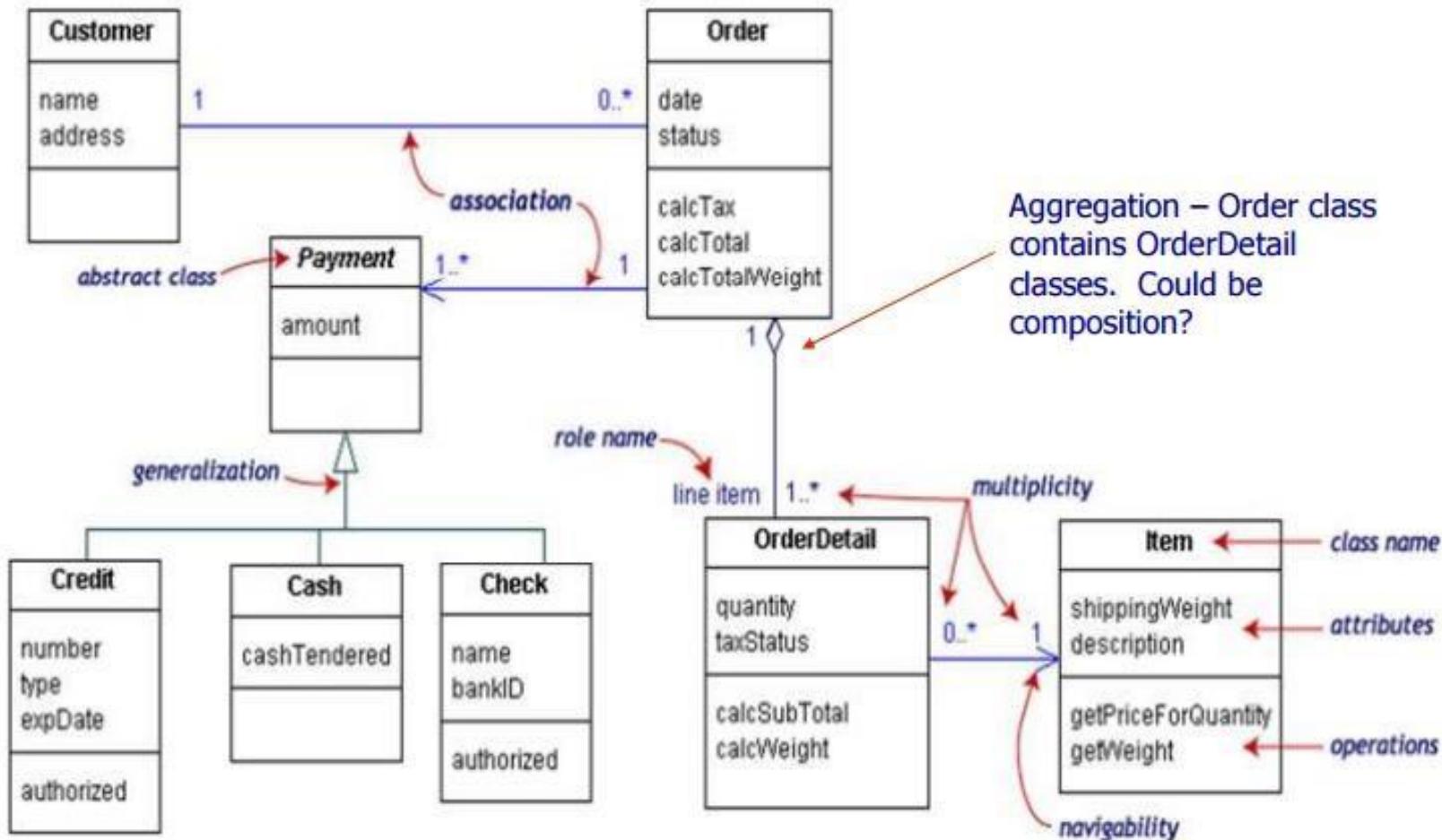
- Oluştur
- Öde
- Kalem (Ürün) Ekle / Çıkart
- Sevk Et

Örnek 1: Sipariş verme yazılımı (devam..)

- Sipariş verme yazılımının tasarımını için gereken tüm nesneler **düşünülmeli** ve nesnelerin **nitelikleri / özelliklerini** ve davranışları tanımlanmalıdır.
- Başlıca nesneleri şunlar olmalıdır :
 - Müsteri Nesnesi,
 - Ödeme Nesnesi,
 - Ürün Nesnesi,
 - Sipariş Detayı Nesnesi

Önemli: Aslında bu nesnelerin hemen hepsinin karşılığı olan bir veritabanı nesnesi (tablo) olmalı değil mi?

Örnek 1: Sipariş verme yazılımı (devam...)



Örnek 1: Sipariş verme yazılımı (devam...)

- Her iki programlama yaklaşımıyla (yapısal ve nesne yönelimli) da **doğru bir yazılım** üretilebilir?
- Aynı zamanda iki programlama yaklaşımıyla da **yeniden kullanılabilir** (kısmen) modüller üretilebilir.
- İki programlama yaklaşımındaki önemli farklılıklar:
 - Yazılımcının projenin erken safhalarındaki **planlama ve düşünme farklılığı**
 - Üretilen programların **karmaşıklıkları**
 - Yeni ihtiyaçların ve ***düzenlemelerin sisteme adapte edilmesindeki kolaylık***

Örnek 2: Trafik Simülasyon

Amaç: Bir şehrin trafik karışıklığını gidermek için gerçek zamanlı trafiği simüle eden bir simülasyonun yapılması gerekmektedir.

Gerekli olan nesneler :

- Arac Nesnesi,
- Yaya Nesnesi,
- Trafik İşaretçileri Nesnesi
- Bu nesnelerin her biri kendi bilgilerini saklar ve kendi davranışlarına sahip olur.
- Böylelikle nesnelerin yardımcılığıyla bir şehrin trafiği gerçek zamanlı olarak simüle edilebilir.

Örnek 2: Trafik Simülasyon (devam...)

Arac nesnesi

Yaya nesnesi

Nitelikler

- Plaka Bilgisi
- Kimlik Numarası Sürücü bilgisi
- Anlık Hızı • Anlık Hızı Bilgisi

Davranışlar

- Anlık Hız Değiştirme Metodu
- Trafik Kuralı Kontrol Metodu

Davranışlar

Nitelikler

- Plaka Bilgisi
- Kimlik Numarası Sürücü bilgisi
- Anlık Hızı • Anlık Hızı Bilgisi

Sınıf ve Nesne Kavramı

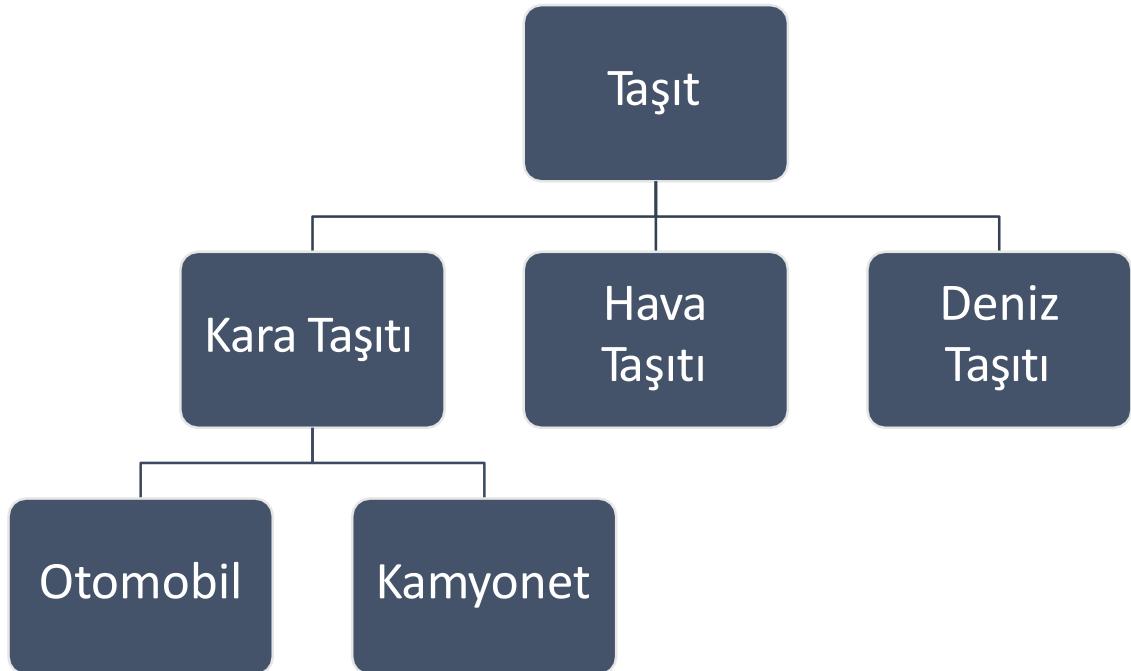
*Dünyayı anlayabilme kabiliyetimizin çoğu
nesneleri ve olayları*

sınıflar halinde kategorilendirebilmemizden gelmektedir.

- Örneğin; Çocukken “**hayvan**” kavramını, “**hayvan**” kelimesini öğrenmeden çok öncesinde biliyordunuz. Hayvanlarla ilk karşılaşmanız köpeğinizle veya sokak kedisi ile veya bir çiftlikteki keçi ile olmuş olabilir. **Konuşmayı öğrendikçe, tüm hayvanlar için aynı terimleri kullanıyordu.**
- Tecrübelendikçe, **hayvanlar** arasındaki farkları anlayabildik.
- Mesela bir **köpek** ile **koyunun** arasındaki farkı, sonrasında ise yavruları ile yetişkinleri arasındaki fark gibi...

Sınıf ve Nesne Kavramı (devam...)

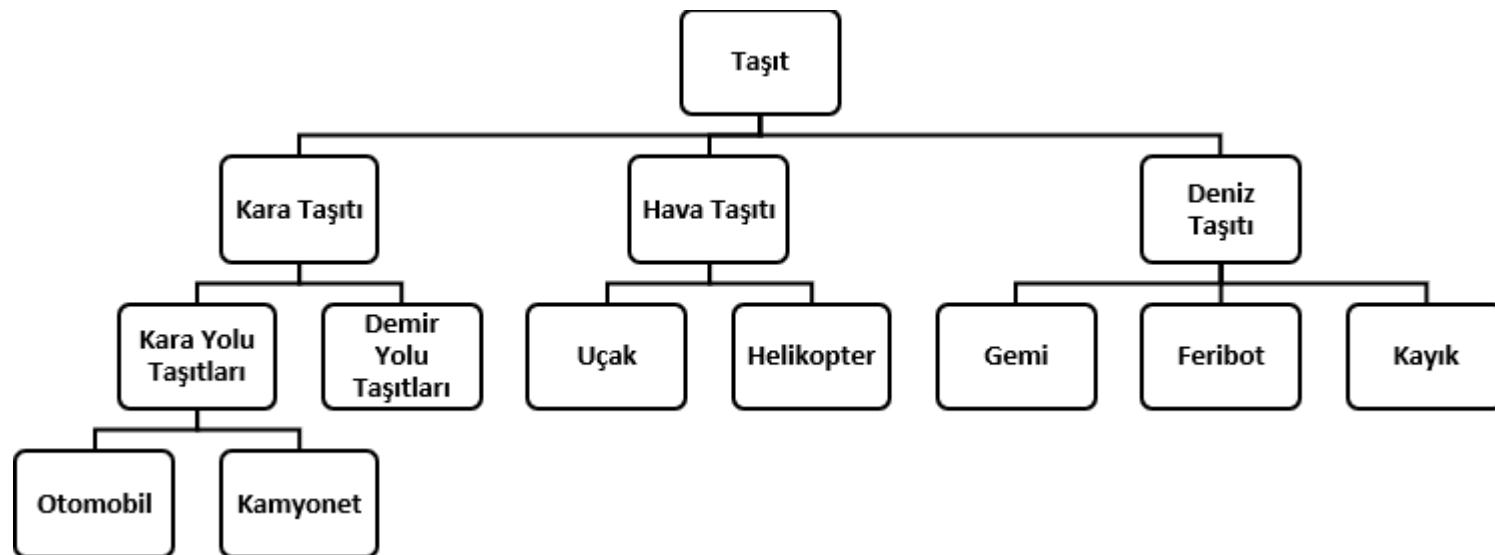
- Aynı şekilde Taşıtlar sınıfını anlamamız, Kara taşıtları ile Deniz ve Hava taşıtları arasındaki benzerlikleri görmemize yardımcı oldu.
- *Kara taşıtları sınıfını öğrendikten* sonra Otomobil ile Kamyonet arasındaki benzerlikleri daha rahat kavrayabildik.



Yukarıdan aşağı incelediğimizde taşitlar genel olarak kara, hava ve deniz olmak üzere 3 alt sınıfa ayrılırlar. Bir kara taşıtı ise genel bir taşıtin tüm özelliklerini barındırır.

Sınıf ve Nesne Kavramı (devam...)

- **Sınıf:** NYP yaklaşımında sınıflar varlıkların ortak özellik ve davranışlarını (operasyonları) tanımlamamızı ve onları *soyutlamamızı* sağlayan özel veri türleridir (*kullanıcı tanımlı veri türü – user defined data type*)



Sınıf ve Nesne Kavramı (devam...)

- **Nesne** daha genel bir sınıfın **somutlaşmış** bir örneğidir.
- **Nesneler** gerçek dünyadaki sınıfların birer örneğidir (**instance**).
- **Örneğin:** Sizin **arabanız** daha genel bir sınıf olan **Araba** sınıfının **somutlaşmış** bir örneğidir. Bir nesne, bulunduğu sınıfın somutlaşmış bir örneğidir.
- **Canlı cisimleri** de “**nesne**” olarak düşünebilirsiniz:
 - **Ev bitkiniz,**
 - **Balığınız,**
 - **Aile bireyleri** birer nesnedir.

Sınıf ve Nesne Kavramı (devam...)

Örnek: Bir **araba** sınıfı ile ilgili hangi özellikleri ve davranışları biliyoruz?

Düşünelim...

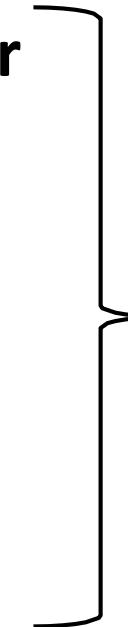
Araba Sınıfı Özellikler ve Metotlar

- **Özellikler**

- Marka: string
- Model: int
- Renk: string
- YakitTuru: eYakitTur
- Fiyat: Decimal
- Motor: motor_sinifi

- **Metotlar / Davranışlar**

- SatinAl()
- Sat()
- Calistir(): boolean
- Sur()
- Durdur(): boolean



Dikkat

- Farklı bu kadar çok metodу tek sınıfı barındırmak mantıklı değil.
- Bir sınıfı fazla sorumluluk yüklemiş oluyoruz.
- Her sınıfın tek sorumluluğu olmalı.
- **Single Responsibility Principle**

Araba Sınıfı UML Sınıf Diyagramı Gösterimi

Araba	Sınıf, Interface Adı
Marka: string Model: short Renk: string YakitTuru: eYakitTur Fiyat: Decimal Motor: motor_sinifi	Üyeler ve Özellikler
SatinAl() Sat() Calistir(): boolean Sur() Durdur(): boolean	Metotlar / Davranışlar

Sınıf ve Nesne Kavramı (devam...)

- Nesnelerin üyeleri, sınıf içerisinde kullanılan diğer değişkenler ile karışmaması için **field** (üye değişkenler) olarak adlandırılır.
- Nesnelerin **özellikleri (property)** arka planda üye değişkenleri ile ilişkilendirilirler.
- Nesneler üzerinden üye değişkenlerine erişilemez (private) ancak özelliklere erişilebilir (public).
- Genellikle; üyeleri küçük harfle, özellikler BÜYÜK harfle başlar.
- Nesnenin örnek değişkenlerinin icerikleri, o nesnenin **durumunu** belirler.

Sınıf ve Nesne Kavramı (devam...)

Araba sınıfına ait 2 nesnenin durumu (**state**)

Araba1	Araba2
Marka: Renault Megane	Marka: Citroen C5
Model: 2009	Model: 2012
Renk: Beyaz	Renk: Füme
YakitTuru: eDizel	YakitTuru: eBenzin
Fiyat: 28.500 TL	Fiyat: 41.500 TL
Motor: motor_nesnesi1	Motor: motor_nesnesi2
SatinAl()	SatinAl()
Sat()	Sat()
Calistir()	Calistir()
Sur()	Sur()
Durdur()	Durdur()

Sınıf Oluşturmak

- Sınıfları tanımlarken öncelikle **bir sınıf başlığı/tanımı** yaratılmalıdır. Sınıf başlığı üç parçadan oluşur:

1. İsteğe bağlı bir **erişim belirleyici** (access modifier),
2. **class** anahtar kelimesi,
3. **Sınıfin adı**

[erişim belirleyici] **class** [sınıfAdı]

Örneğin; **public class** Araba

i Sınıf adları nesnelerin tipini tanımladığı için genellikle tek bir addan oluşur.

Sınıf Oluşturmak (devam...)

- Araba sınıfını tanımlarken kullanılan `public` anahtar sözcüğü sınıf erişim belirleyicisidir.
- Oluşturulan sınıfın amacına uygun olan erişim belirleyicileri kullanılır.

Sınıf Erişim Belirleyicisi	Açıklama
Public (UML → +)	Sınıfa erişim sınırsızdır
Protected (UML → #)	Sınıfa erişim bulunduğu sınıf ve <i>bu sınıfından türetilen sınıflar</i> ile sınırlıdır.
Internal – Varsayılan (UML → ~)	Sınıfa erişim ait olduğu assembly/exe ile sınırlıdır.
Private (UML → -)	Sınıfa erişim ait olduğu sınıf ile sınırlıdır.

Sınıf Oluşturmak (devam...)

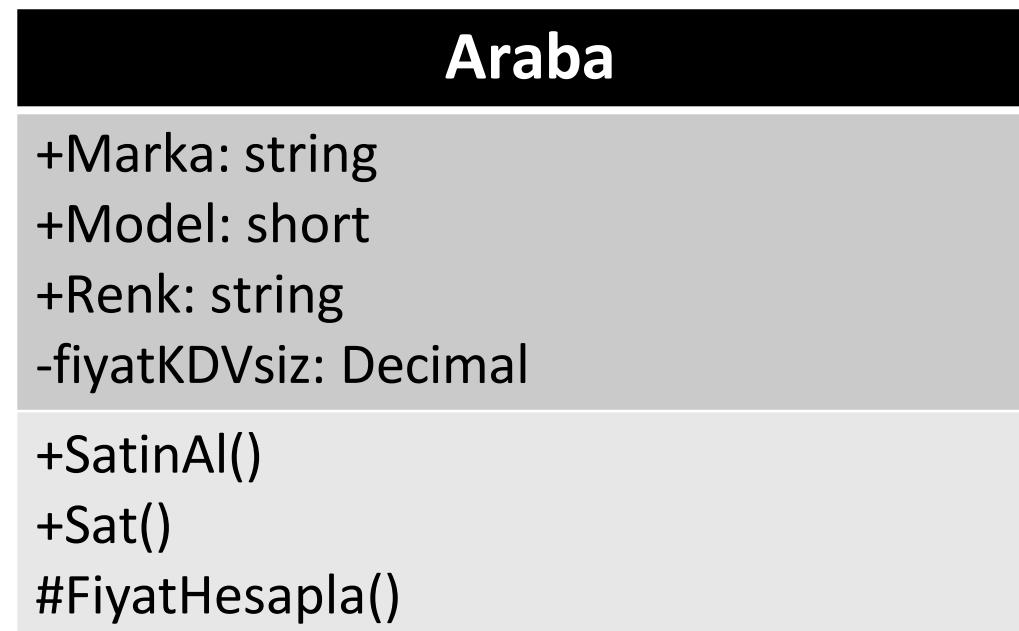
- Sınıf başlıklarları tanımlanırken kıvırcık parantezler (“{}”) içerisinde sınıf gövdeleri tanımlanmalıdır.

Örneğin:

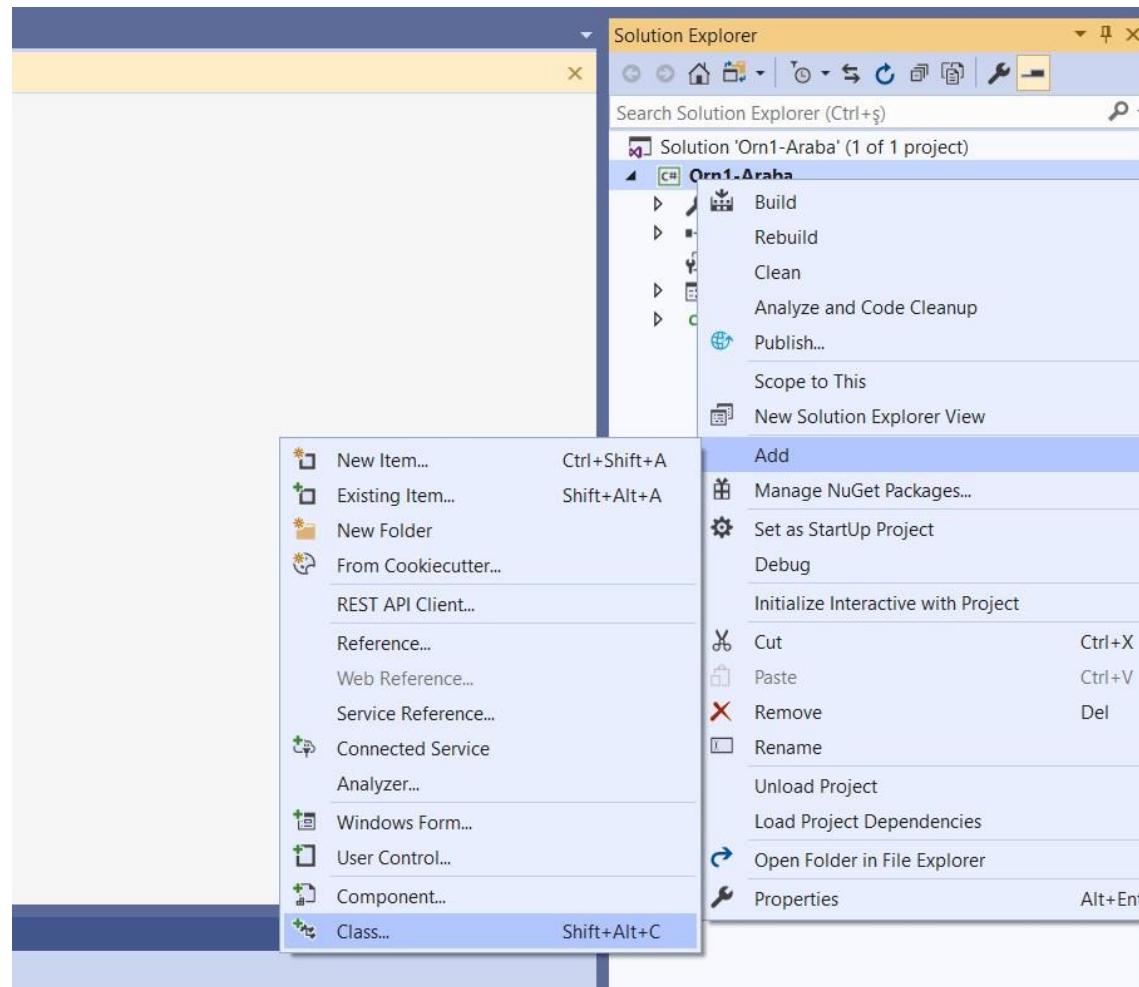
```
public class Araba  
{  
    // üye değişkenleri, özellikleri ve metodları buraya yazılır  
}
```

Örnek 1: Araba Sınıfının Oluşturulması

- Araba sınıfını oluşturarak, sınıfın araba isimli bir nesne oluşturalım.
- Erişim belirleyicilere dikkat edelim.



Örnek 1: Araba Sınıfının Oluşturulması (devam...)



Örnek 1: Araba Sınıfının Oluşturulması (devam...)

```
public class Araba
{
    //Üyeler
    private decimal fiyatKDVsiz;
    //Özellikler
    public string Marka { get; set; }
    public short Model { get; set; }
    public string Renk { get; set; }
    //Davranışlar
    public void SatinAl()
    {   //Todo: Satın alma kodları
    }
    public void Sat()
    {   //Todo: Satış kodları
    }
    protected void FiyatHesapla()
    {   //Todo: Fiyat hesaplama kodları
    }
}
```

```
private void BtnTest_Click(object sender, EventArgs e)
{
    Araba araba = new Araba();
    araba.Marka = "Citroen C5";
    araba.Model = 2017;
    araba.Renk = "Kırmızı";
    araba.Sat();
    araba.SatinAl();
    MessageBox.Show("Araba markası: " +
                    araba.Marka + "\n" +
                    "Modeli: " + araba.Model +
                    "\n" +
                    "Rengi: " + araba.Renk);
}
```

fiyatKDVsiz üye değişkenine ve **FiyatHesapla** metoduna neden erişemedik?

Örnek 2: DortgenPrizma Sınıfının Oluşturulması

DortgenPrizma

-en: float

-boy: float

-yukseklik: float

+HacimHesapla(float _en, float _boy, float _yukseklik): float

Örnek 2: DortgenPrizma Sınıfının Oluşturulması (devam...)

```
public class DortgenPrizma
{
    private float en;
    private float boy;
    private float yukselik;
    public float HacimHesapla(float pEn,
                               float pBoy,
                               float pYukseklik)
    {
        en = pEn;
        boy = pBoy;
        yukselik = pYukseklik;
        return (en * boy * yukselik);
    }
}
```

Sınıfin Kullanımı

```
private void BtnTest_Click(object sender,
EventArgs e)
{
    DortgenPrizma dp = new DortgenPrizma();
    float hacim = dp.HacimHesapla(4, 5, 2);
    MessageBox.Show("Hesaplanan hacim: " +
                    hacim.ToString());
}
```

Örnek 3: KimlikBilgileri Sınıfının Oluşturulması

KimlikBilgileri

```
+TCKimlikNo: long  
+Ad: string  
+Soyad: string  
#DogumYeri: string  
+DogumTarihi: DateTime  
+KimlikBilgisiOlustur():string
```

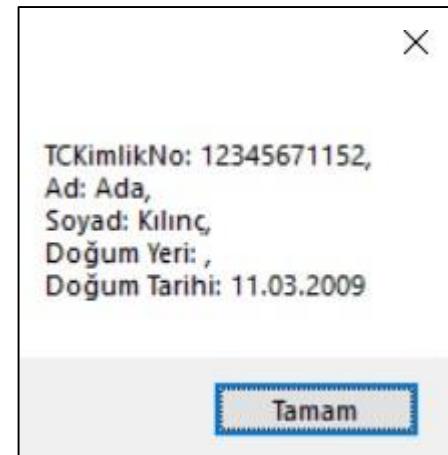
Örnek 3: KimlikBilgileri Sınıfının Oluşturulması (devam...)

```
public class KimlikBilgileri
{
    public ulong TCKimlikNo;
    public string Ad;
    public string Soyad;
    protected string DogumYeri;
    public DateTime DogumTarihi;
    public string KimlikBilgisiOlustur()
    {
        return "TCKimlikNo: " + TCKimlikNo.ToString() + ", " +
               "\nAd: " + Ad + ", " +
               "\nSoyad: " + Soyad + ", " +
               "\nDoğum Yeri: " + DogumYeri + ", " +
               "\nDoğum Tarihi: " + DogumTarihi.ToShortDateString();
    }
}
```

Örnek 3: KimlikBilgileri Sınıfının Oluşturulması (devam...)

Sınıfın Kullanımı

```
private void BtnTest_Click(object sender, EventArgs e)
{
    KimlikBilgileri kb = new KimlikBilgileri();
    kb.Ad = "Ada";
    kb.Soyad = "Kılınç";
    kb.TCKimlikNo = 12345671152;
    kb.DogumTarihi = Convert.ToDateTime("11.03.2009");
    MessageBox.Show(kb.KimlikBilgisiOlustur());
}
```



Doğum yerı ?
Sadece aile içi ☺

Örnek 4: Ogrenci Sınıfının Oluşturulması

Ogrenci
+OgrNo: ulong
+ProgramTuru: EProgramTur (Lisans, YL, Doktora)
+Birim: string
+Bolum: string
+KimlikBilgisi: KimlikBilgileri
+OgrenciBilgisiOlustur():string

- Bir sınıf düzen sınıflardan yaratılmış bir **nesne üye barındırabilir**.
- Örneğin, ad, soyad, adres vb. gibi üye değişkenlerine sahip **KimlikBilgileri** sınıfından yaratılan bir nesne **Ogrenci** sınıfının üyesi olabilir.

Örnek 4: Ogrenci Sınıfının Oluşturulması (devam...)

```
public class Ogrenci
{
    public ulong OgrNo;
    public EProgramTur ProgramTuru;
    public string Birim;
    public string Bolum;
    public KimlikBilgileri KimlikBilgisi;
    public string
    OgrenciBilgisiOlustur()
    {
        string bilgi;
        bilgi = "Öğrenci No: " + OgrNo +
            "\nProgram Türü: " + ProgramTuru +
            "\nBirim: " + Birim +
            "\nBölüm: " + Bolum +
            "\nKimlik No: " + KimlikBilgisi.TCKimlikNo + "\nAd: " +
            KimlikBilgisi.Ad +
            "\nSoyad: " + KimlikBilgisi.Soyad +
            "\nDoğum Yeri: " +
            "\nDoğum Tarihi: " +
            KimlikBilgisi.DogumTarihi.ToShortDateString(); return bilgi;
    }
}
```

```
public enum EProgramTur
{
    Lisans,
    YuksekLisans,
    Doktora
}
```

Örnek 4: Ogrenci Sınıfının Oluşturulması (devam...)

```
private void BtnOgrenciTess_Click(object sender, EventArgs e)
{
    Ogrenci ogrenci = new Ogrenci();
    ogrenci.OgrNo = 1892322;
    ogrenci.ProgramTuru = EProgramTur.Lisans;
    ogrenci.Birim = "Mühendislik ve Mimarlık Fakültesi";
    ogrenci.Bolum = "Bilgisayar Mühendisliği";
    /*
    ogrenci.KimlikBilgisi = new KimlikBilgileri();
    ogrenci.KimlikBilgisi.Ad = "Ada";
    ogrenci.KimlikBilgisi.Soyad = "Kılınç";
    ogrenci.KimlikBilgisi.TCKimlikNo = 12345671152;
    ogrenci.KimlikBilgisi.DogumTarihi = Convert.ToDateTime("11.03.2009");
    */
    KimlikBilgileri kb = new KimlikBilgileri();
    kb.Ad = "Ada";
    kb.Soyad = "Kılınç";
    kb.TCKimlikNo = 12345671152;
    kb.DogumTarihi = Convert.ToDateTime("11.03.2009");
    ogrenci.KimlikBilgisi = kb;
    MessageBox.Show(ogrenci.OgrenciBilgisiOlustur());
}
```

UML Sınıflar Arası Birliktelik Türleri

Association

- İlişkinin en genel hali.
- Nesneler tamamen bağımsız yaşamaya sahiptirler.
- Sahiplik yok (**ownership**).
- Ok ile gösterilir.

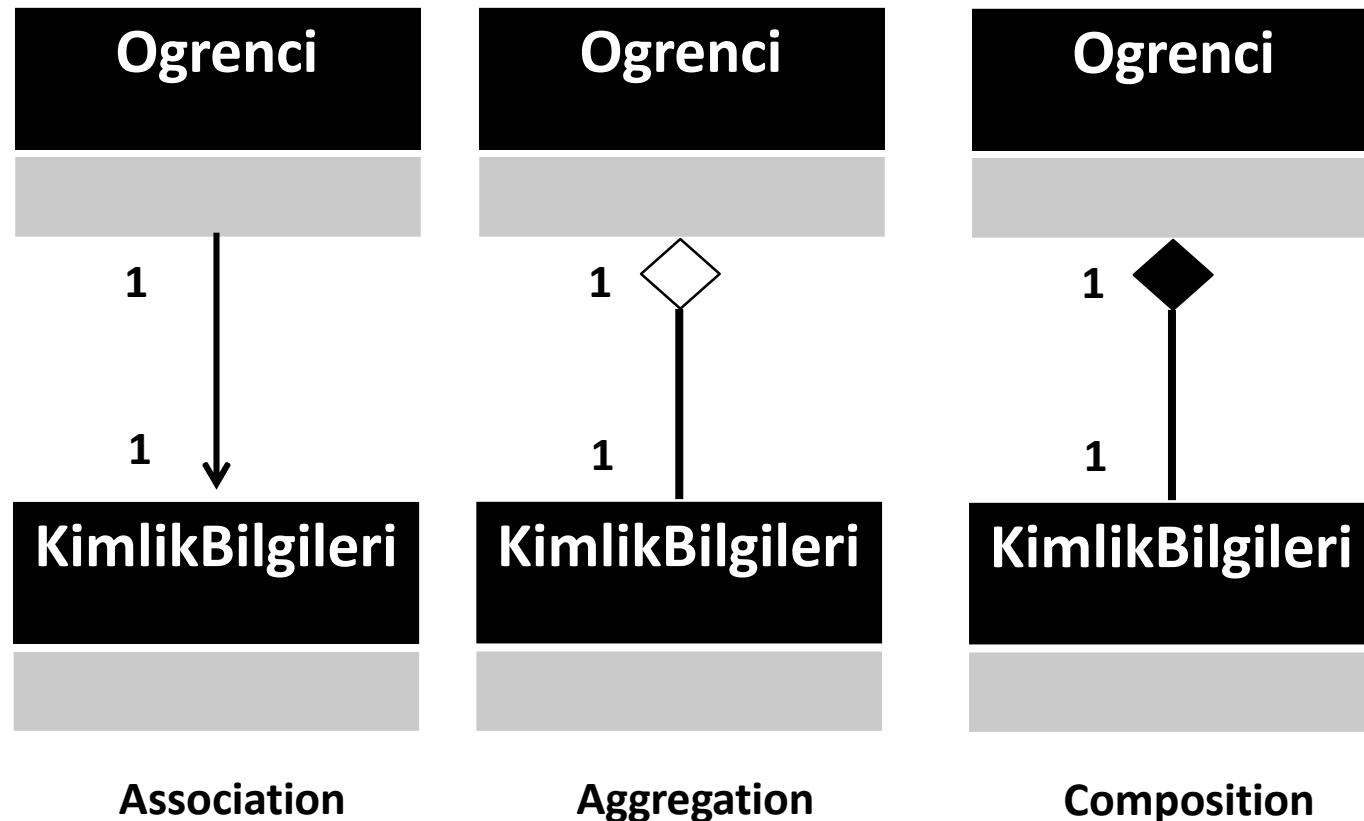
Aggregation

- **Is-part-of:** Bir parçası olmak.
- Nesneler bağımsız yaşamaya sahiptirler.
- Yine de bir sahiplik söz konusudur (**ownership**).
- İçi boş elmas ile gösterilir.

Composition

- **Is-made-of:** Daha güçlü bir parçası olmak.
- Nesneler birlikte oluşturulur ve birlikte yok edilirler.
- Sahiplik işin temelidir.
- İçi dolu siyah elmas ile gösterilir.

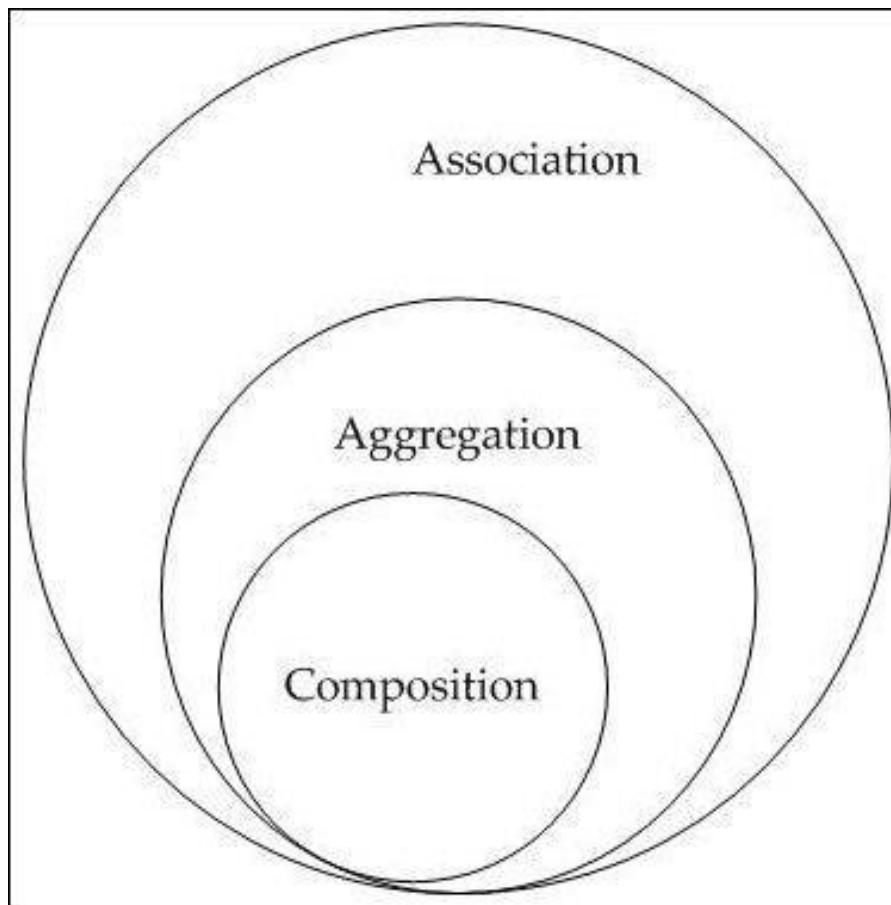
UML Sınıflar Arası Birliktelik Türleri



Dikkat

- Coupling ve cohesion'a dikkat ediyoruz.
- **Coupling:** Nesneler arası bağımlılık.
- **Cohesion:** Sınıfın üyeleri arasındaki benzerlik. Tek iş yapma.
- Düşük *coupling* olmalı
- Yüksek *cohesion* olmalı.
- Çok fazla composition ve aggregation coupling'i (genelde) düşürür.

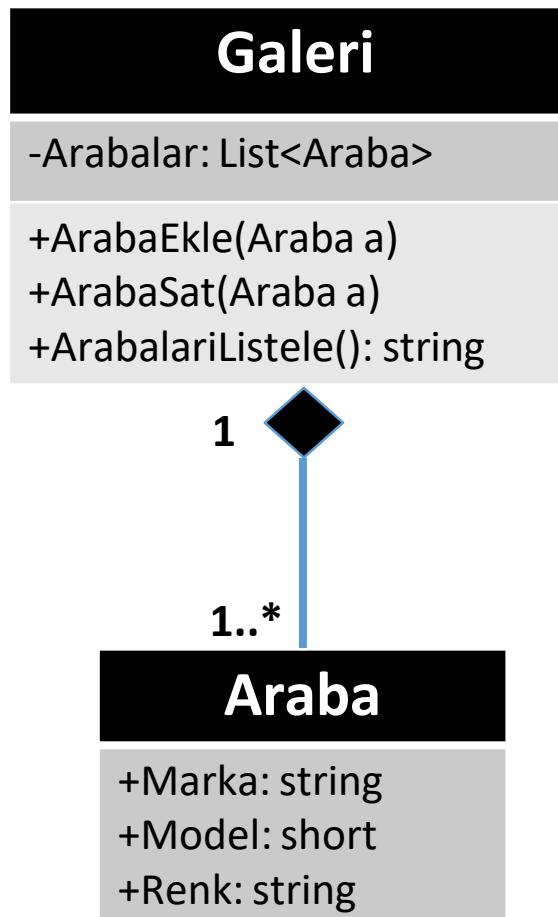
UML Sınıflar Arası Birliktelik Türleri



Uygulama: Araba Galerisi

- **Gereksinim:** Bir araba galerisi uygulaması yapmak istiyoruz. Galeriye arabalar eklenip, çıkartılabilir (satılabilir), listeleme yapılabilir. Arabaların marka, model ve renk bilgisi olması yeterli. Galeriye ait farklı bilgiler (firma bilgileri vb.) isteğe bağlı eklenebilir.
- **Yaklaşım:** Nesne yönelimli mantıkla, soyut düşünelim.
- **Sorular soralım:**
 - Soru 1:** Gereksinimler tam mı? (**Requirements Gathering and Analysis**)
 - Soru 2:** Hangi sınıflara (varlıklara) ihtiyacımız var? (**Design-Dev**)
 - Soru 3:** Sınıfların birbirleri ile ilişkisi nasıl kurulmalı? (**Design-Dev**)
 - Soru 4:** Nasıl bir ekran ara yüzü tasarlayalım? (**Design-UI**)

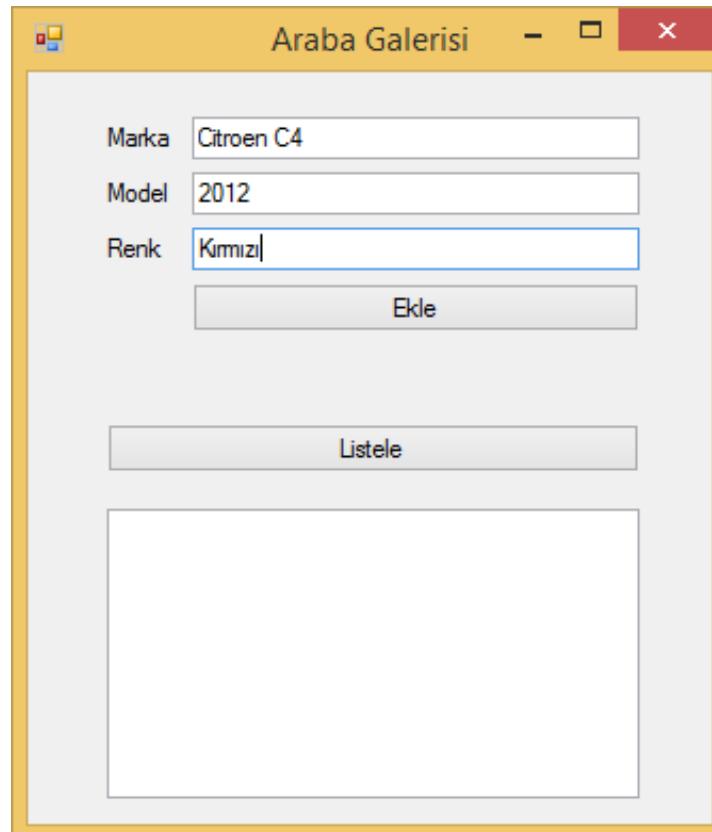
Uygulama: Araba Galerisi (devam...)



Soru 2: Hangi sınıflara ihtiyacımız var?
Soru 3: Sınıfların birbirleri ile ilişkisi nasıl kurulmalı?

Uygulama: Araba Galerisi (devam...)

Soru 4: Nasıl bir ekran ara yüzü (UI) tasarlayalım?



Notlar

- Listelenen kayıtları görmek için önce bir **textbox (Multiline)** kullanalım.
- Daha sonra bir **DatagridView** kullanalım.

Nesne Yaratma

- Bir sınıfı tanımlamak bir nesne yaratmak anlamına gelmemektedir.
- Sınıf sadece kendisinden yaratılacak bir nesnenin özelliklerini ve davranışlarını belirleyen soyut açıklamasıdır.
- Oluşturmak istediğiniz bir **varlığın** tüm özelliklerini önceden belirlediğinizde, nesne yaratmadan önce bir sınıfın, üyelerini ve metotlarını yaratmalısınız.
- Sınıf tanımlamasını, yeni bir ev inşa etmek için evin ayrıntılı bir planı olarak ya da kek yapmak için gerekli olan kek tarifi olarak da düşünebilirsiniz.

*Sınıflar oluşturulacak olan herhangi bir nesnenin **ayrıntılı bir planıdır.***

Nesne Yaratma (devam...)

- Bir nesneyi yaratmak **iki aşamadan** oluşur.
- İlk olarak bir **değişken tanımlar** gibi bir **tip** ve bir tanımlayıcı atamalısınız.
- Sonrasında **nesneyi yaratabilirsiniz**.
- Örneğin; **int** veri tipinde **x** değişkenini **int x;** şeklinde tanımlayabildiğimiz gibi **Personel** nesnesini de, **Personel per;** şeklinde tanımlayabiliriz.
- **Personel** sınıfından **per** adında bir nesne oluşturduğumuzda, **derleyiciye per** adını kullanacağımız **bildirilir**.
- Fakat henüz **per** nesnesinin **barındırdığı değerler için** **bellekte bir yer tutulmamıştır**.
- Gereken bellek yerini «**rezerve/allocate**» etmek nesneyi **new** operatörünü kullanarak **yaratmamız gereklidir**.

Nesne Yaratma (devam...)

- **per** adındaki **Personel** nesnesini iki aşamada yaratırız. Öncelikle nesne için bir referans tanımlamalı, sonrasında new operatörü ile derleyicinin **per** nesnesi için yeterli alanı tahsis etmesini sağlarız.

```
Personel per;  
per = new Personel();
```

- Aynı satırda iki işlemi de tanımlayabiliriz:

```
Personel per = new Personel();  
ifadesindeki Personel() bir metot gibi gözükmektedir.
```

- Aslında bu **Personel** nesnesinin **constructor**'ı başka bir deyişle kurucu metodudur.
- Sınıfınız için bir constructor oluşturmazsanız, C# sizin için adı sınıfın adı ile aynı olan bir tane kurucu metot yaratır.

Nesne Yaratma (devam...)

- Nesnelerin **tanımlayıcıları** (Örn: per nesnesi), nesnelerin bellekteki adreslerinin yerlerini gösterirler.
- Herhangi bir sınıf **referans tipi** (ing.: reference type) olarak **kullanılır**.
- Diğer bir deyişle **bellekteki özel bir adresi** gösteren bir **tip** denebilir.
 - **int**, **double** ve **float** gibi **veri tipleri**, değişkelerin değerlerini tutar;
 - Bu *veri tiplerinin aksine* **referans tipleri** bellekteki adresleri tutarlar.

Metotlara Parametre olarak Nesne Aktarımı

- Bir metoda nesne aktardığımızda aslında o metoda bir referans aktarmış oluruz (pass-by-reference).

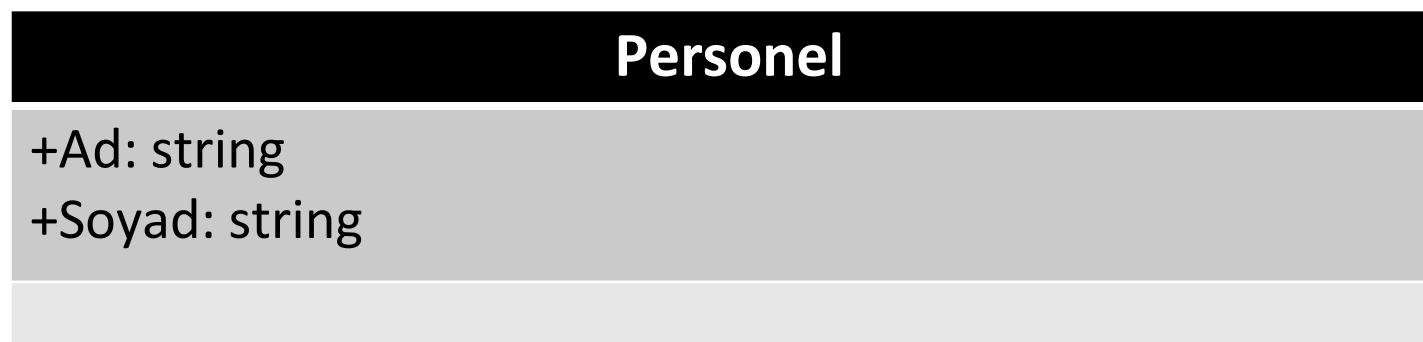
Böylece nesne parametresinde yapılan herhangi bir değişiklik, çağrııldığı metottaki nesneyi de etkiler.

- Gösterim biçimi aşağıdaki gibidir:

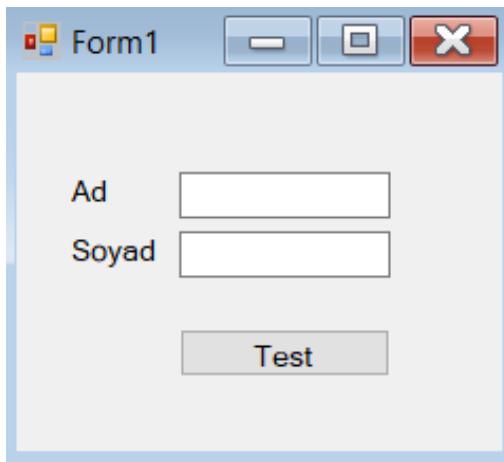
```
void BilgiGoster(Personel p);
```

Örnek1: Metoda Nesne Aktarımı

- Form üzerinde **Personel** sınıfından, **personel** adında bir nesne oluşturunuz.
- Bu nesneyi form üzerindeki PersonelBilgiDegistir metoduna aktarınız.
- Metot içerisinde **personel bilgilerini değiştirip (BÜYÜK harfe dönüştürüp)**, metot dışındaki nesneler üzerinde, *personel bilgisinin değişip değişmediğini kontrol ediniz.*



Örnek1: Metoda Nesne Aktarımı (devam...)



```
public class Personel
{
    public string Ad;
    public string Soyad;
}
```

```
private void PersonelBilgiDegistir(Personel p)
{
    p.Ad = p.Ad.ToUpper();
    p.Soyad = p.Soyad.ToUpper();
}

private void BtnTest_Click(object sender, EventArgs e)
{
    Personel personel = new Personel();
    personel.Ad = txtAd.Text;
    personel.Soyad = txtSoyad.Text;

    //metoda parametre olarak nesne yolladık
    PersonelBilgiDegistir(personel);

    //nesne metot dışına çıktığında da değeri değişti
    txtAd.Text = personel.Ad;
    txtSoyad.Text = personel.Soyad;
}
```

Properties (Özellikler)

Properties Yaratmak

- **Property**, sınıfın bir **üyesine (değişkenine)** erişim sağlayan bir **sınıf üyesidir**. Property'ler, **üyelerin nasıl ayarlanacağını ve erişimin nasıl olacağını** tanımlar.
- Propertyler, **private üyeler ile public metotlar** arasındaki **en iyi özellikleri bir arada barındırır**.
- **Public** metotlar gibi private verileri dışarıdan müdahalelere karşı korur.
- Property yarattığınızda, **kullandığınız sözdizimi** daha kolay **anlaşılabilir** ve **doğal olur**.

Properties Yaratmak (devam...)

- C# programcılar property'leri **smart fields** olarak adlandırırlar.
- Propertyler, sınıfın üyesine erişildiğinde işleme alınacak olan ifadeleri barındıran **erişimcilere** sahiptir.
- Özellikle propertyler nesnenin üyesinin değerini **ayarlamaya yarayan set erişimcilerine** ve saklanan verilerin **değerlerine erişimini sağlayan get erişimcilerini** içerirler.
- Bir property, set erişimcisine sahipse **yazılabilir**, get erişimcisine sahipse **okunabilir** olmaktadır. Bir property sadece get erişimcisine sahipse bu property **sadece okunabilir (read-only)** olur.
- C#'ta get ve set erişimcilerini genellikle **getter** ve **setter** olarak adlandırırlar.

Örnek2: Isci Sınıfı

- **kimlikNo** üye değişkenine erişimi sağlayan **KimlikNo** property'sini tanımlayınız.
- Tanımladığınız **KimlikNo** özelliğini kullanan KarsilamaMesaji() metodunu tanımlayınız.

```
Iisci  
-kimlikNo: int  
+KimlikNo: int  
+KarsilamaMesaji(): string
```

Örnek2: Isci Sınıfı

```
public class Isci
{
    private int kimlikNo;
    0 references
    public int KimlikNo
    {
        get
        {
            return kimlikNo;
        }
        set
        {
            kimlikNo = value;
        }
    }
    0 references
    public string KarsilamaMesaji()
    {
        return "Hoş geldiniz " + kimlikNo + " numaralı çalışanımız...";
    }
}
```

Properties Yaratmak (devam...)

- Property tanımlanmasında değişken tanımlanmasına benzer bir şekilde;
 - Erişim belirleyici, veri tipi ve tanımlayıcı(ad) içermektedir.
 - Ayrıca **kıvırcık parantezler** içerisinde tanımlanan ifadeleri barındıran bir metot içerir.
- Property ismi kullandığı üyenin isminin baş harfi büyük hali olarak tanımlanır.
 - Örn.: **kimlikNo** üye değişkeni için **KimlikNo**
- Property tanımlayıcısından sonra, *küme parantezi içerisinde erişimciler (get ve set)* tanımlanır.
- Erişimciler (**get** ve **set**) **metot gibi görünebilir** fakat mettlardaki gibi tanımlayıcılarının yanında **parantez ‘()’ bulunmaz**.

Properties Yaratmak (devam...)

- **set** erişimcisi, **parametre alan** ve bunu **değişkene atayan** bir metot gibi davranır.
- Fakat **set erişimcisi** bir **metot değildir** ve beraberinde **bir parantez** ile kullanılmaz.
- **get** erişimcisi ise property ile **ilişkilendirilmiş** üyenin değerini *geriye döndürür*.
- Bir property'nin set ve get erişimcilerinden faydalananırken **set ve get anahtar kelimelerini** ayrıca **kullanmamıza gerek yoktur**.
- İşlem yapacağımız property'e **eşittir** operatörü '=' ile *değer atadığımızda set bloğu işleme alınmaktadır*.
- Property'nin adını kullanarak **eristiğimizde** ise **get bloğu işleme alınır**.

Properties Yaratmak (devam...)

- Örneğin; **usta** adındaki **Iisci** sınıfından türetilmiş bir nesne tanımladığımızda, **KimlikNo** propertiesinin değerini aşağıdaki gibi atarız:

```
Iisci usta = ... Iisci();
usta.KimlikNo = 888191012;
```

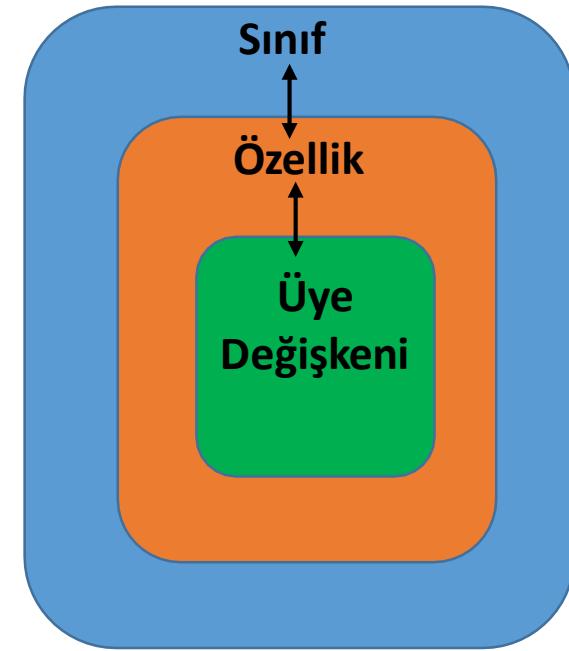
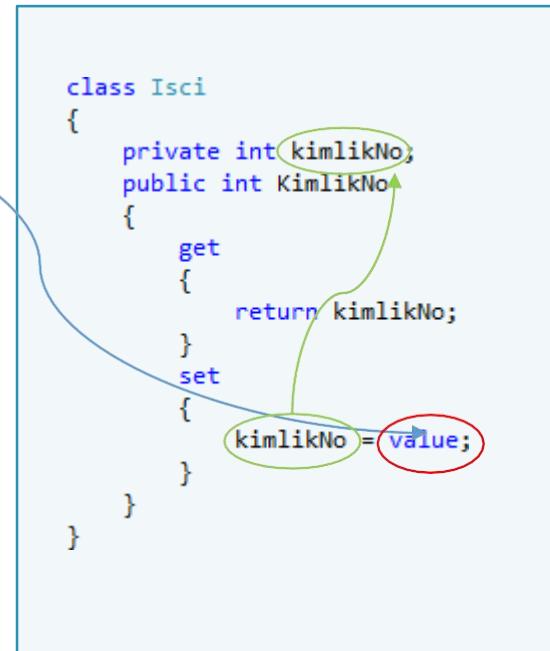
- İkinci ifadede, **KimlikNo** property'sinin değeri **888191012** olarak ayarlanmış olur. Eşitliğin sağındaki değer property'nin set erişimcisine **implicit parametre** olarak gönderilmiştir.

DEBUG EDEREK GÖRELİM...

Properties Yaratmak (devam...)

```
Isci usta = new Isci();
usta.KimlikNo = 888191012;
```

- 888191012 değeri set bloğuna aktarılmış ve set erişimcisinde **value** adını almıştır.
- Set erişimcisinin içerisinde de value değeri sınıf field'ı olan **kimlikNo** değişkenine atanmıştır.



Properties Yaratmak (devam...)

- Propertyler **get** erişimcisi sayesinde, basit bir değişken gibi kullanılabilmektedir.
- Örnek: Önceden tanımlanmış olan işçinin kimlik numarası aşağıdaki gibi yazdırılabilir:

```
Isci usta = ...  
//set erişimcisi çalışıyor  
usta.KimlikNo = 888191012;  
  
//get erişimcisi çalışıyor  
MessageBox.Show("Kimlik No: " + usta.KimlikNo);
```

- Fakat **usta.kimlikNo** üye **private** erişim belirleyicisine sahip olduğu için erişilemez ve ekrana yazdırılamaz.

Properties Yaratmak (devam...)

- Önceki örneklerde **get** ve **set** erişimcileri, üyenin değerini döndürme veya değeri üyeeye atama dışında bir işlem yapmamaktadır.
- Eğer kullanılan üye **private** yerine **public** olarak tanımlansaydı **property** kullanmamıza gerek kalmazdı.
- Fakat bu yöntem (Sınıfın üyelerini private, onlara erişmeyi sağlayan metodları public tanımlanması), nesneye yönelik programmanın **tutarlı olması** için **genel olarak kullanılmaktadır**.

Properties Yaratmak (devam...)

- Veriyi saklı tutmak ve aynı zamanda
 - verinin değerlerinin **nasıl ayarlandığını** ve
 - kullanıldığını **kontrol edebilmek**,

Nesne Yönelimli Programlamanın çok **önemli bir özelliği**dir.

- Erişimciler, sınıfın bazı üyelerinin **değerlerinin nasıl ayarlanarak döndürüleceğine** veya **üyelere nasıl erişileceğine** dair kısıtlamalar koyarak düzenlenebilir.

Properties Yaratmak (devam...)

- Örnek: Kimlik numarasının **değer aralığını belirleyen** bir set erişimcisi yandaki gibi yazılmaktadır.
- Bu kod bloğunda **set** erişimcisi **Isci** sınıfının **kimlikNo** üyesinin değerinin 1000000'den daha büyük olamayacağını kesinleştiren bir **kısıtlama** koyar.
- Eğer **kimlikNo** değişkenine direkt olarak erisip, değer ataması yapılmasına izin verilseydi, atanan değerin kontrolü yapılamazdı.
- Tanımlanan sınıf için bir **set** erişimcisi kullandığınızda, izin verilen verilerin değerleri üzerinde tam kontrolü elde edilmiş olunur.

```
get  
{  
    return kimlikNo;  
}  
set  
{  
    if (value < 1000000)  
        kimlikNo = value;  
    else  
        kimlikNo = -1;  
}
```

DEBUG EDEREK GÖRELİM...

Auto Implemented Properties

```
public int KimlikNo{get; set;}
```

- Üstteki formatta oluşturulmuş propertylere **auto-implemented property** denir.
- İçerisinde **özellikleştirmek** **istediğiniz** Propertyler, auto-implemented property olarak yaratılamaz.
- Ayrıca get ve set erişimcilerinden herhangi birisi kullanılmayacaksá auto-implemented property olarak yaratılamaz.

Auto Implemented Properties (devam...)

- Sınıfin içerisinde bulunan propertynin kullanacağı üyelere **backing field** denir.
- Auto-implemented property kullandığınızda backing field yaratmaya gerek yoktur.
- Propertynin kullanacağı üye **derleyici tarafından oluşturulur**.
- Örnekteki **KimlikNo** propertysi için **kimlikNo** üyesinin oluşturulmasına gerek yoktur.
- Derleyici **kimlikNo** üyesini otomatik olarak yaratır ve kullanır.

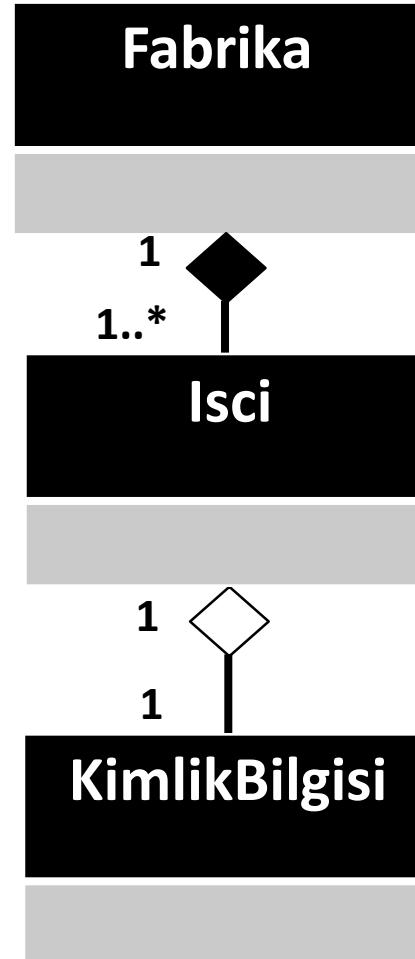
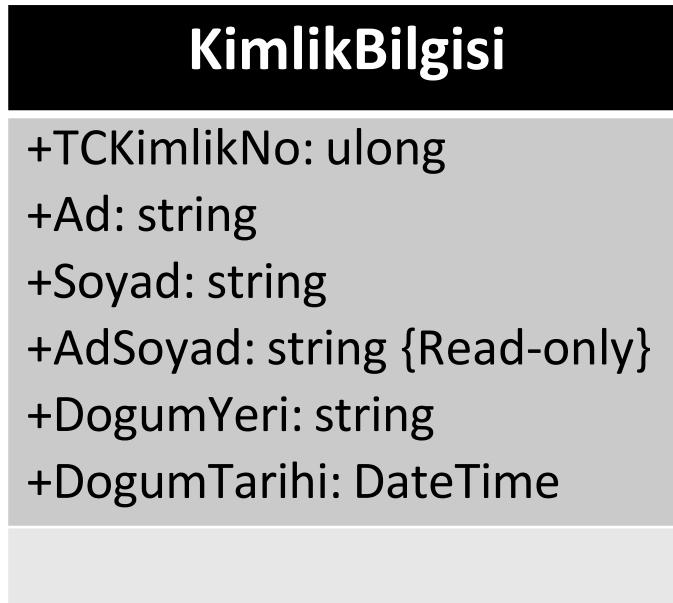
Örnek3: Isci ve KimlikBilgisi

Versiyon 1.0 (Senaryo ve Gereksinimler)

- Seyrek fabrikasında çalışan işçilerin hepsi şirket kimlik numarası, maaş ve diğer nüfus kimlik bilgilerine (TCKimlikNo, Ad, Soyad, Doğum Yeri, Doğum Tarihi) sahiptirler. Fabrikaya çalışan ekleme ve çalışan listeleme işlemleri yapılabilir.
- **Özel Not:** Her çalışanın «adını ve soyadını» birleştirerek **büyük harfe otomatik çeviren** [AdSoyad](#) özelliği bulunmaktadır.

Örnek3: Isci ve KimlikBilgisi (devam...)

Tasarım: UML Sınıf Diyagramı



Örnek3: Isci ve KimlikBilgisi (devam...)

```
public class KimlikBilgisi
{
    1 reference
    public ulong TCKimlikNo { get; set; }
    2 references
    public string Ad { get; set; }
    2 references
    public string Soyad { get; set; }
    1 reference
    public string AdSoyad
    {
        get
        {
            string adSoyad = (this.Ad + " " + this.Soyad).ToUpper();
            return adSoyad;
        }
    }
    1 reference
    public string DogumYeri { get; set; }
    0 references
    public DateTime DogumTarihi { get; set; }
}
```

```
public class Isci
{
    1 reference
    public int SirketKimlikNo { get; set; }
    1 reference
    public decimal Maas { get; set; }
    2 references
    public KimlikBilgisi Kimlik { get; set; }
}
```

```
public class Fabrika
{
    0 references
    public string Ad { get; set; }
    2 references
    private List<Iisci> Calisanlar { get; set; } = new List<Iisci>();
    0 references
    public void CalisanEkle(Iisci isci)
    {
        Calisanlar.Add(isci);
    }
    0 references
    public List<Iisci> CalisanlariListele()
    {
        return Calisanlar;
    }
}
```

Örnek3: Isci ve KimlikBilgisi (devam...)

```
private void BtnTest_Click(object sender, EventArgs e)
{
    Fabrika fabrika = new Fabrika();
    fabrika.Ad = "Seyrek Fabrikası";

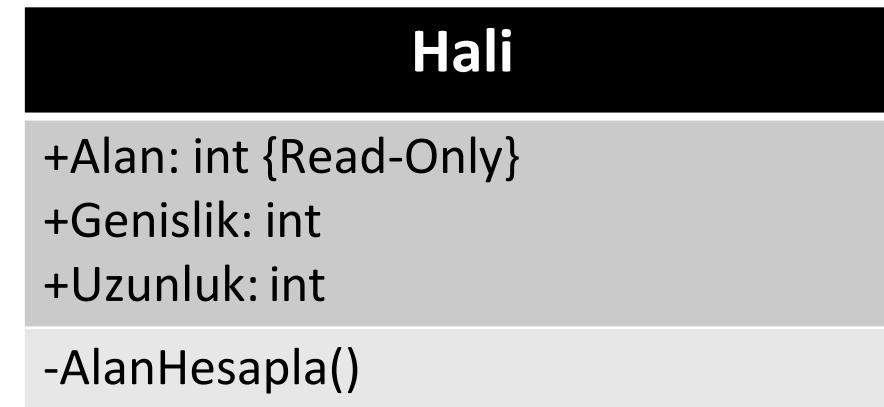
    KimlikBilgisi kimlik = new KimlikBilgisi();
    kimlik.TCKimlikNo = 1221122313;
    kimlik.Ad = "Ahmet";
    kimlik.Soyad = "Demir";
    kimlik.DogumYeri = "İzmir";
    //AdSoyad read-only yani sadece okunabilir
    //kimlik.AdSoyad = "...";

    Isciisci = new Isci();
   isci.Kimlik = kimlik;
   isci.SirketKimlikNo = 9812;
   isci.Maas = 4000;
    MessageBox.Show(isci.Kimlik.AdSoyad);

    fabrika.CalisanEkle(isci);
}
```

Örnek4: Hali Sınıfının Yaratılması

- **Hali** sınıfında, **Alan** propertyi set erişimcisi içermemektedir (read-only).
- **AlanHesapla()** metodu private tanımlanmıştır.
- **Soru:** Nasıl Alan hesaplayacağız?



Örnek4: Hali Sınıfının Yaratılması (devam...)

```
public class Hali
{
    private int alan;
    1 reference
    public int Alan
    {
        get
        {
            AlanHesapla();
            return alan;
        }
    }
    2 references
    public int Genislik { get; set; }
    2 references
    public int Uzunluk { get; set; }
    1 reference
    private void AlanHesapla()
    {
        alan = Uzunluk * Genislik;
    }
}
```

```
private void BtnTest_Click(object sender, EventArgs e)
{
    Hali hali = new Hali();
    hali.Genislik = 4;
    hali.Uzunluk = 3;
    MessageBox.Show("Alan: " + hali.Alan);
}
```

EK: Auto-Implemented Prop. Nasıl Read-Only Olur?

- Eğer sadece **okunabilir (read-only)** auto implemented propertyler yaratmak istiyorsak, set erişimcisini **private** olarak tanımlamalıyız.

Örnek:

```
public int KimlikNo { get; private set; }
```

- Böylece **KimlikNo** property'sine değer ataması yapılamaz.
- Fakat kendi sınıfındaki metodlar değer ataması yapabilirler.

this Referansı

- Bir sınıf yaratıldığında, sınıfın kaynak kodunun bir **kopyası** **bilgisayarın hafızasında** (memory) saklanmaktadır.
- Bununla birlikte, er ya da geç bu sınıftan defalarca **nesne** oluşturulacaktır.
- Her yeni nesne oluşturduğunuzda, nesnenin her üye değişkeni için **bellekte saklama alanı sağlanır**.
- **this** referansı ile *sınıfin o anki nesnesine ulaşabilirsiniz*.
- **this** referansı kodda **karışıklığı engellemek** için de kullanılabilir.
- **Örneğin:** Sınıfın bir *üyesi* veya *ozelliği* ile sınıfın metoduna aktarılan parametrenin aynı isimde olması durumunda karışıklığı önlemek için **this** referansı kullanılabilir.

Örnek 1: Kitap Sınıfında this Referansının Kullanılması

- Sizden **Kitap** isimli bir sınıf oluşturmanız istenmektedir. Bu sınıf decimal veri türünde, read-only ve public olan **Fiyat** ve **VergiTutari** özelliklerine sahiptir.
- Yine decimal türündeki **VERGIORANI** özelliği ise private olup aynı zamanda sabittir ve değeri 0.18'dir.
- Sınıfin **VergiTutariHesapla()** metodu bulunmakta ve bu metot **Fiyat** isimli değişkeni parametre olarak almaktadır.
- Metot ilk olarak; Fiyat özelliğine parametreyi atayacak daha sonra VergiTutarini hesaplayacaktır.

Kitap
+Fiyat: decimal {Read-Only}
+VergiTutari: decimal {Read-Only}
-VERGIORANI: decimal
+VergiTutariHesapla(decimal Fiyat)

Örnek 1: Kitap Sınıfında this Referansının Kullanılması

- Direk olarak **Fiyat = Fiyat** eşitliğini kullanırsa derleyici uyarı verir. Bunun gibi ve buna benzer **karışıklıkları önlemek** için **this** referansı kullanılır.
- **this.Fiyat = Fiyat** eşitliğinin sol tarafındaki fiyat değişkeni sınıfın özelliği olduğunu belirtmektedir. Sağındaki Fiyat değişkeni ise bu metoda aktarılan fiyat parametresidir.

```
public class Kitap
{
    private static decimal VERGIORANI = 0.18M;
    2 references
    public decimal Fiyat { get; private set; }
    1 reference
    public decimal VergiTutari { get; private set; }

    0 references
    public void VergiTutariHesapla(decimal Fiyat)
    {
        this.Fiyat = Fiyat;
        VergiTutari = this.Fiyat * VERGIORANI;
    }
}
```

Kurucular / Kurucu Metotlar / Yapıcılar (Constructors)

- Herhangi bir sınıfından bir nesne yarattığınızda aşağıdaki gibi bir ifade kullanılır:

```
Iisci isc = new Iisci();
```

- Aslında bu ifadede **Iisci()** adında C#'ın sağladığı bir *metod çağrılmaktadır*.
- **Kurucu metotlar (Constructor)** bir nesnenin oluşmasını sağlayan metotlardır.
- Bir sınıf için **yazmamış olsanız bile**, yarattığınız sınıflar için *otomatik olarak parametre almayan public* bir kurucu metot sağlanır.

Kurucular (Constructors) (devam...)

- Parametresi olmayan kurucu metotlara sınıfın varsayılan kurucu metodu (**default constructor**) denir.
- Otomatik olarak yaratılan **Iisci()** kurucu metodu, **isc** adında **Iisci** sınıfından bir nesne oluşturur ve devamında **Iisci**'nin **başlangıç değerlerinin atanmasını** sağlar.
- Sınıfın üye değişkenleri varsayılan olarak:
 - **Sayısal** fieldlar için **0** değeri,
 - **Karakter** fieldları için '**\0**' değeri,
 - **Boolean** field'lar için **false** değeri,
 - **String** ya da diğer referans tipli nesne fieldlarına **NULL** değeri atanmaktadır.

Kurucular (Constructors) (devam...)

- Isci nesnesinin üyelerinin ve özelliklerinin başlangıç değerlerinin varsayılan değerler olması istenmiyorsa veya
 - Isci sınıfından bir *nesne* *yaratılırken* ek işlemler yapmak istediğinizde varsayılan kurucu metot yerine **kendi kurucu metodunuzu** oluşturabilirsiniz.
- **Sınıfin kurucu metotları sınıf ile aynı isme sahip olmak zorundadır.**
- Kurucu metotlarının geri dönüş türleri yoktur. (int , float veya void olarak tanımlanmazlar)

Örnek 2: Isci Sınıfının Kurucu Metodunu Yaratmak

- **Maas** özelliğine sahip bir **Iisci** sınıfı yaratınız.
- Bu sınıfın kurucu metodunu yaratıp, sınıfından yaratılan **Iisci** nesnelerinin **Maas** özelliğinin varsayılan değerinin 1800.00 ve **CalismaDurumu** özelliğinin **Calisiyor** olması sağlayınız.
- Form üzerinde **Iisci** sınıfından bir nesne yaratarak Maas ve CalismaDurumu özelliklerinin değerini ekranда gösteriniz.
DEBUG işlemi gerçekleştiriniz.

EcalismaDurumu Değerleri
Emekli,
Calisiyor,
Ayrildi

Iisci

+CalismaDurumu: ECalismaDurumu
+Maas: decimal
<<Constructor>>+Iisci()

Örnek 2: Isci Sınıfının Kurucu Metodunu Yaratmak (devam...)

```
public class Isci
{
    2 references
    public enum ECalismaDurumu
    {
        Emekli,
        Calisiyor,
        Ayrildi
    }
    2 references
    public decimal Maas { get; set; }
    2 references
    public ECalismaDurumu CalismaDurumu { get; set; }
    1 reference
    public Isci()
    {
        this.Maas = 1800;
        this.CalismaDurumu = ECalismaDurumu.Calisiyor;
    }
}
```

Kurucu Metotlara Parametre Geçirmek

- Kurucu metotlar, **parametre** de alabilirler.
- Aktarılan parametrelerin değerlerini kullanarak;
 - Yaratılan nesnenin özelliklerinin ya da üyelerinin değerlerini her bir nesne için **ayarlamak** mümkündür.

Örnek 3: Isci Sınıfının Kurucu Metoduna Parametre Geçirmek

- Örnek 2'deki Isci sınıfının kurucu metoduna geçirilen **cocuk** parametresine bağlı olarak **AsgariGecimIndirimi** ücretini aşağıdaki tabloya göre hesaplayınız. **DEBUG** işlemi gerçekleştiriniz.

Çocuk Sayısı	Asgari Geçim İndirimi
Çocuk Yok	80,33 ₺
1 ve Üzeri Çocuk	104,42 ₺

Isci

+CalismaDurumu: ECalismaDurumu
+Maas: decimal
+AsgariGecimIndirimi: decimal {Read-Only}
+CocukSayisi: short {Read-Only}

<<Constructor>>+Isci(short cocuk)
-AsgariGecimIndirimiHesapla()

Örnek 3: Isci Sınıfının Kurucu Metoduna Parametre Geçirmek (devam...)

```
public decimal Maas { get; set; }
2 references
public ECalismaDurumu CalismaDurumu { get; set; }
3 references
public decimal AsgariGecimIndirimi { get; private set; }
3 references
public short CocukSayisi { get; private set; }
1 reference
private void AsgariGecimIndirimiHesapla()
{
    if (this.CocukSayisi == 0)
        AsgariGecimIndirimi = 80.33M;
    else if (this.CocukSayisi >= 1)
        AsgariGecimIndirimi = 104.42M;
}
1 reference
public Isci(short cocukSayisi)
{
    this.Maas = 1800;
    this.CalismaDurumu = ECalismaDurumu.Calisiyor;
    this.CocukSayisi = cocukSayisi;
    AsgariGecimIndirimiHesapla();
}
```

```
private void BtnTest_Click(object sender, EventArgs e)
{
    Isci calisan = new Isci(1);
    MessageBox.Show("Çalışma durumu: " + calisan.CalismaDurumu +
                   "\nMaaş: " + calisan.Maas.ToString() +
                   "\nAsg.Çeç.İnd: " + calisan.AsgariGecimIndirimi);
}
```

Kurucu Metotların Aşırı Yüklenmesi

- C#'da yaratılan sınıflar için **otomatik olarak bir kurucu metot oluşturulmaktadır.**
- Sınıf için yeni bir kurucu metot yarattığınızda C# 'ın **otomatik olarak** yarattığı kurucu metot erişilemez olur.
- Fakat otomatik yaratılan kurucu metodun **aynısı tanımlanabilir.**
- **Hatta:** bir sınıf için **farklı parametrelerle** istedığınız kadar (anlam kargaşasına neden olmayacak şekilde) **kurucu metot** olşturabilirsiniz.
- C# 'ın diğer metotları gibi, kurucu metotları da **aşırı yüklenebilir.**

Örnek 4: Dortgen Sınıfının Kurucu Metotlarının Aşırı Yüklenmesi

- UML diagramında görüldüğü gibi bir **Dortgen** sınıfı tanımlamanız istenmektedir.
- Dortgen sınıfının farklı parametreler alan iki adet kurucu metodу bulunmaktadır.

```
classDiagram
    class Dortgen {
        +Uzunluk: int {Read-Only}
        +Genislik: int {Read-Only}
        +Alan: int {Read-Only}

        <<Constructor>>+Dortgen(int uzunluk, int genislik)
        <<Constructor>>+Dortgen(int tekuzunluk)
        +AlanHesapla()
    }
```

Örnek 4: Dörtgen Sınıfının Kurucu Metotlarının Aşırı Yüklenmesi

```
public class Dörtgen
{
    3 references
    public int Uzunluk { get; private set; }
    3 references
    public int Genislik { get; private set; }
    1 reference
    public int Alan { get; private set; }
    1 reference
    public Dörtgen(int uzunluk, int genislik)
    {
        this.Uzunluk = uzunluk;
        this.Genislik = genislik;
    }
    1 reference
    public Dörtgen(int tekUzunuk)
    {
        this.Uzunluk = this.Genislik = tekUzunuk;
    }
    0 references
    public void AlanHesapla()
    {
        Alan = this.Uzunluk * this.Genislik;
    }
}
```

iki parametreli kurucu metot çalışır

```
private void BtnTest_Click(object sender, EventArgs e)
{
    Dörtgen dikdörtgen = new Dörtgen(4, 5);
    dikdörtgen.AlanHesapla();
    MessageBox.Show(dikdörtgen.Alan.ToString());

    Dörtgen kare = new Dörtgen(4);
    kare.AlanHesapla();
    MessageBox.Show(kare.Alan.ToString());
}
```



Tek parametreli kurucu metot çalışır

Nesne Başlatıcıları (Object Initializers)

- **Nesne başlatıcısı**, bir sınıfın nesne yaratırken o nesnenin oluşturulması sırasında erişilebilir üyelerine veya özelliklerine değer atanmasına izin verir.
- Örnek: Parametresiz bir kurucu metot içeren ve **KimlikNumarası** adında **public** bir property içeren bir sınıfta, aşağıdaki ifadede olduğu gibi nesne başlatıcısı (object initializers) kullanılabilir:

```
Isci isc = newisci { KimlikNo = 104 };
```

Nesne Başlatıcıları (Object Initializers) (dvam...)

- Örnekteki ifadede **104** değeri **Isci** sınıfından yaratılmış olan **isc** nesnesinin **KimlikNo** property'sine değer olarak atanmıştır.
- Değer atama küme parantezleri içerisinde yapılmıştır.
- Bu ifade çalıştırıldığında,
 - **İlk olarak** sınıfın varsayılan kurucu metodu çalışır, sonra
 - Nesne başlatıcısı ile **KimlikNo** property'sine değer ataması yapılır.

Örnek 5: Nesne Başlatıcısı Kullanma

- KimlikNo ve Maas özelliklerine sahip, Maas özelliğine 1500 ve KimlikNo özelliğine -1 başlangıç değerini atayan bir varsayılan kurucu metodlu içeren bir Isci yaratınız.
- Öncelikle kurucu metottaki ifadelerin mi yoksa nesne başlatıcısındaki ifadelerin mi uygulandığını tespit ediniz.

Isci
+KimlikNo: int
+Maas: decimal
+Mesaj: string
<<Constructor>>+Isci()

Örnek 5: Nesne Başlatıcısı Kullanma (devam...)

```
public class Isci
{
    4 references
    public int KimlikNo { get; set; }
    4 references
    public decimal Maas { get; set; }
    2 references
    public string Mesaj { get; set; }
    1 reference
    public Isci()
    {
        this.KimlikNo = -1;
        this.Maas = 1500;
        this.Mesaj = "KimlikNo: " + this.KimlikNo +
                    "Maaş: " + this.Maas;
    }
}
```

```
private void BtnTest_Click(object sender, EventArgs e)
{
    Isci calisan = new Isci
    {
        KimlikNo = 222,
        Maas = 3000
    };
    MessageBox.Show(calisan.Mesaj);
    MessageBox.Show("KimlikNo: " + calisan.KimlikNo +
                  "Maaş: " + calisan.Maas);
}
```

Örnek 5: Nesne Başlatıcısı Kullanma (devam...)

```
Iisci calisan = new Iisci {KimlikNo = 222, Maas = 3000};
```

- Bu ifade aşağıdaki birer birer değer ataması yapılan ifadeler ile aynı işlemi yapmaktadır:

```
Iisci calisan = new Iisci();
```

```
calisan.KimlikNo = 222;
```

```
calisan.Maas = 3000;
```

Yıkıcı Metotlar

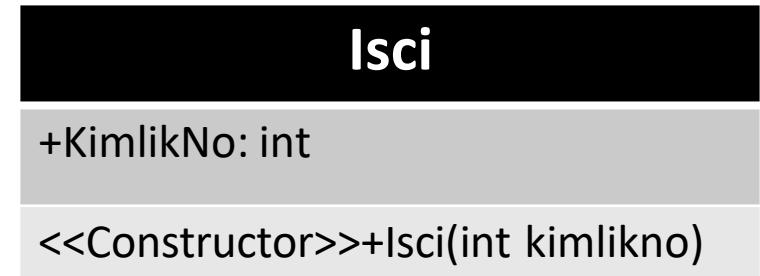
- **Yıkıcı metot**, o sınıfından yaratılmış bir nesne yok edildiğinde yapılacak olan eylemleri içeren metotlara denir.
- Genellikle sınıfından yaratılmış olan nesne **kapsam dışında kaldığında yok edilmektedir**.
- Kurucu metotlarda olduğu gibi, sınıf için bir yıkıcı metot tanımlanmadığında C# sizin için otomatik olarak bir **yıkıcı metot** sağlamaktadır.

Yıkıcı Metotlar (devam...)

- Yıkıcı metot tanımlamak için, “ ~ ” (Tilda) işaretini ile sınıfın adı olarak tanımlanır.
 ~ [sınıfınAdı] ()
 {
 }
- Yıkıcı metotlara herhangi bir parametre geçirilemez.
- Yıkıcı metotlar aşırı yüklenemezler(overload).
- Geri dönüş değerine sahip olmazlar.

Örnek 6: Isci Sınıfında Yıkıcı Metotların Kullanımı

- KimlikNo property'sine sahip olan Isci sınıfını yıkıcı metodunu oluşturunuz.
- Oluşturduğunuz Isci sınıfından farklı kimlik numaralarına sahip **iki adet nesne oluşturup** çalıştırınız.



Örnek 6: Isci Sınıfında Yıkıcı Metotların Kullanımı

```
public class Isci
{
    3 references
    public int KimlikNo { get; set; }
    1 reference
    public Isci(int kimlikNo)
    {
        this.KimlikNo = kimlikNo;
        Console.WriteLine(this.KimlikNo + " kimlik numaralı nesne yaratıldı");
    }
    0 references
    ~Isci()
    {
        Console.WriteLine(this.KimlikNo + " kimlik numaralı nesne yok edildi");
    }
}
```

Örnek 6: Isci Sınıfında Yıkıcı Metotlarının Kullanımı

```
private void BtnTest_Click(object sender, EventArgs e)
{
    Isci calisan = new Isci(2);
    calisan = null;
    GC.Collect();
}
```

```
'Orn6-Destruc.exe' (CLR v4.0.30319: Orn6-Destruc.exe): Loaded 'C:\Windows\Mi
'Orn6-Destruc.exe' (CLR v4.0.30319: Orn6-Destruc.exe): Loaded 'C:\Windows\Mi
2 kimlik numaralı nesne yaratıldı
2 kimlik numaralı nesne yok edildi
'Orn6-Destruc.exe' (CLR v4.0.30319: Orn6-Destruc.exe): Loaded 'C:\Windows\Mi
The program '[6404] Orn6-Destruc.exe' has exited with code 0 (0x0).
|
```

Garbage Collector

- GC **otomatik hafıza yönetimi** ile uygulama yazımını kolaylaştırmakta, kodlama süresini kısaltmaktadır.
- Yine GC, makine koduna derlenen dillere göre kodlama hatalarından dolayı karşılaşılan hafıza alanlarının leak edilmesi problemi ile de bir seviyeye kadar baş edebilmektedir.



Yıkıcı Metotlar (devam...)

- Yıkıcı metotlar çağrılmaya gerek kalmadan otomatik olarak uygulanırlar.
- En son yaratılan nesne ilk olarak yok edilir.
- Sınıftan yaratılmış bir nesne, kodda **onunla yapılacak bir işi olmadığından yok edilmeye uygun hale gelir.**
- Yıkıcı metotlar **program kapatılırken** yapmak istediklerimizi (bağlantı kapatma, geçici dosya silme vb.) yapmamızı sağlarlar.

Kalıtım, Miras (Inheritance) Kavramı

- **Sınıfları anlamak** gerçek hayatı nesneleri düzenlemenize yardımcı olur.
- **Kalıtımı anlamak** onları daha net bir şekilde organize etmenizi sağlar.
- Eğer **Braford'u** hiç duymadıysanız zihninizde canlandırmanız **mükün değildir**.

? Bradford ?

Hayvan
Memeli

? Braford ?

inek



Kalıtım, Miras (Inheritance) Kavramı (devam...)

- Bu fikir onun memeli olduğunu öğrenince daha da büyür ve onun bir inek olduğunu öğrenince bu fikir zihninizde net bir hal alır.
- Braford'un bir inek olduğunu öğrendiğinizde, onun birçok inekte ortak olan özelliklere sahip olduğunu anlarsınız.
- Bir Braford'u ayırt edebilmek için sadece ona ilişkin renki, büyülüğu, işaretleri gibi ufak detayları öğrenmeniz gereklidir.

Kalıtım, Miras (Inheritance) Kavramı (devam...)

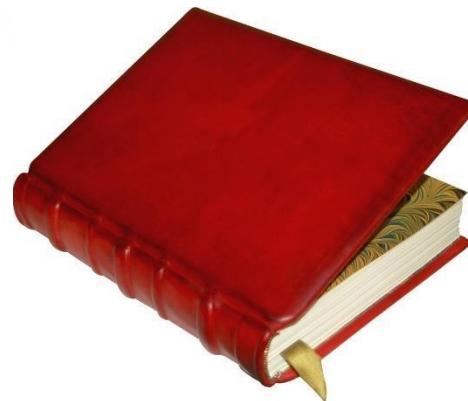
- Halbuki Braford'un özelliklerinin çoğu, şu sınıfların **hiyerarşik yapısından** gelir:
Hayvan → Memeli → İnek
- Tüm "**object-oriented**" programlama dilleri kalıtım özelliğinden *aynı sebepler* için *faydalananmaktadır*:
 - Programlarda kullanılan nesneleri düzenlemek,
 - Kalıtımıla bildiklerinizi kullanarak yeni nesneleri yaratmak
 - Kod parçalarının **reusability'sini** (tekrar kullanılabilirlik) artırmak.

Kalıtım, Miras (Inheritance) Kavramı (devam...)

- Kalıtım, sizin genel bir kategori hakkındaki bildiklerinizi daha **spesifik bir kategoriye uygulamanıza** olanak sağlayan **prensiptir**.
- Kalıtım terimi kullanıldığında, **genetik kalıtımı** düşünebilirsiniz.
 - **Kan grubu** veya **göz rengi** kalıtılmış genlerin ürünüdür.
 - Yürüyüşünüzün babaannenizle aynı olması, ki bu yürüyüş size **babanızdan kalıtılmış** denebilir.

Kalıtım, Miras (Inheritance) Kavramı (devam...)

- Farklı tipte *ürünler* satan **Ürün Satış** uygulaması geliştirmek istediğimizi varsayılmı.



Bu ürünlerin **sınıflarını oluşturabilir miyiz?**

Kalıtım, Miras (Inheritance) Kavramı (devam...)

- Telefon ve Kitap ürünlerinin özellikleri nelerdir?

Telefon	Kitap
+No: int	+No: int
+Adi: string	+ISBN: int
+Marka: string	+Adi: string
+Model: string	+Yazar: string
+Aciklama: string	+Aciklama: string
+Fiyat: decimal	+Fiyat: decimal

Kalıtım, Miras (Inheritance) Kavramı (devam...)

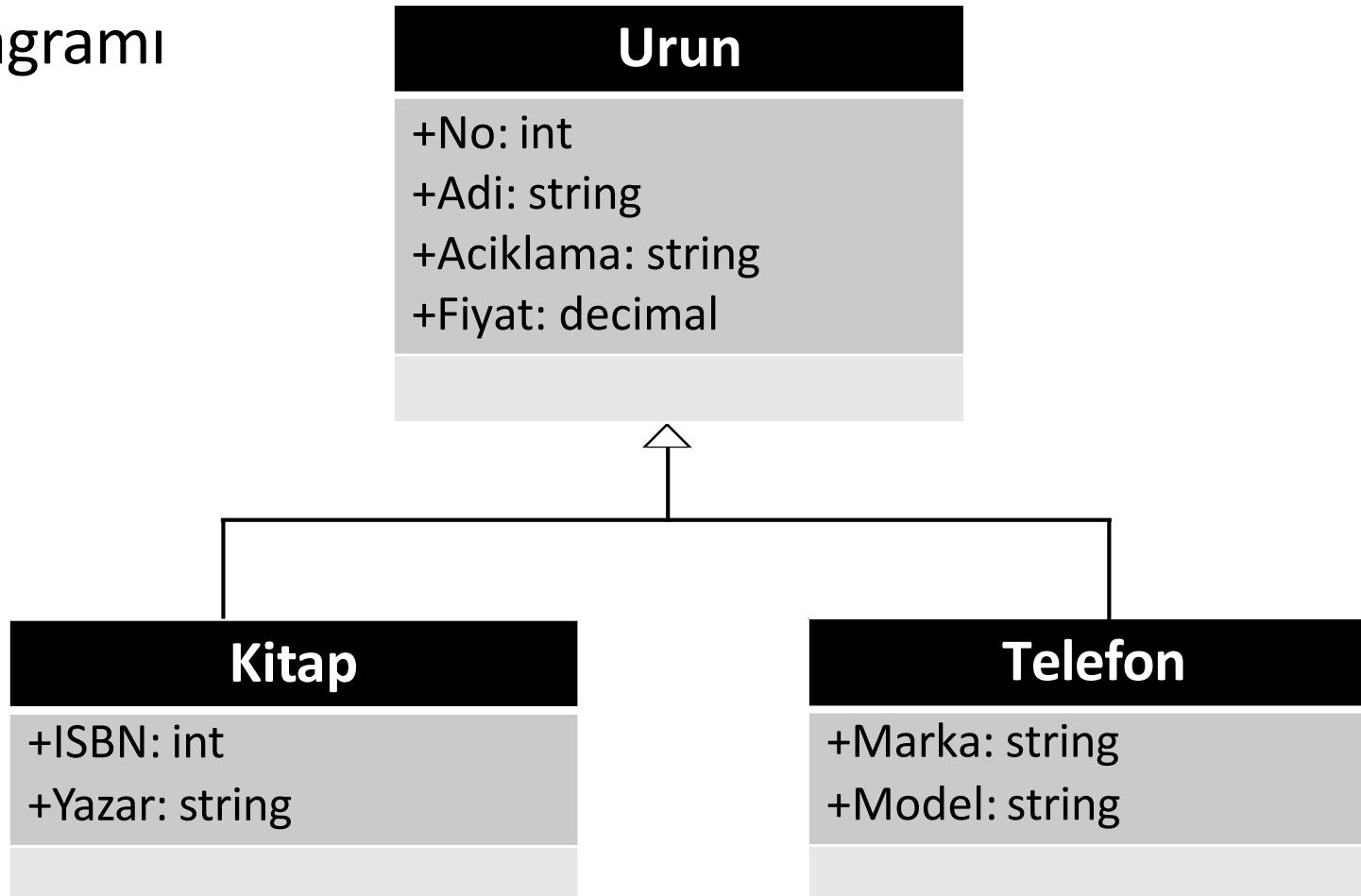
- Telefon ve Kitap ürünlerinin **ortak özellikler** nelerdir?

UrunOrtakOzellikler

+No: int
+Adi: string
+Aciklama: string
+Fiyat: decimal

Örnek1: İlk Kalıtım Örneği

- UML Sınıf Diyagramı



Kalıtımda Kullanılan Terimler

- Kalıtım için temel alınan sınıflara, **Urun sınıfı gibi, temel sınıflar** (ing.: **base classes**) denir.
- Temel sınıfından kalıtılıarak oluşturulmuş sınıfa, **Kitap gibi,**
 - ✓ **kalıtılmış sınıf** (ing.: **derived class**) veya
 - ✓ **genişletilmiş sınıf** (ing.: **extended class**) denir.
 - ✓ **miras alınmış sınıf** (ing.: **inherited class**) denir.

Kalıtımda Kullanılan Terimler (devam...)

- Ayrıca **superclass** ve **subclass** terimleri de temel sınıf ve kalıtılmış sınıflar için kullanılmaktadır.
 - ✓ **Kitap** sınıfı **Urun** superclass'ının subclass'ıdır.
- Buna benzer bir kullanım ayrıca **ana** (ing.: **parent**) ve **yavru** (ing.: **child**) sınıf kavramları da kullanılmaktadır.
 - ✓ **Kitap** sınıfı **Urun** **ana** sınıfının **yavru** sınıfıdır.

Sınıfların Genişletilmesi

- Başka bir sınıfın *kalıtım ile yeni genişletilmiş bir sınıf yaratmak için (O sınıfın bir yavru sınıfını oluşturmak için)* sınıf başlığında yavru sınıfın adı, iki nokta üst üste, ana sınıf adı yazılarak tanımlanır.

```
class [yavruSinifAdi] :  
[anaSinifAdi]
```

```
{  
}  
{  
class Kitap : Urun  
}
```



- Kitap bir Üründür.
- Book **is a** Product.

is a
relation

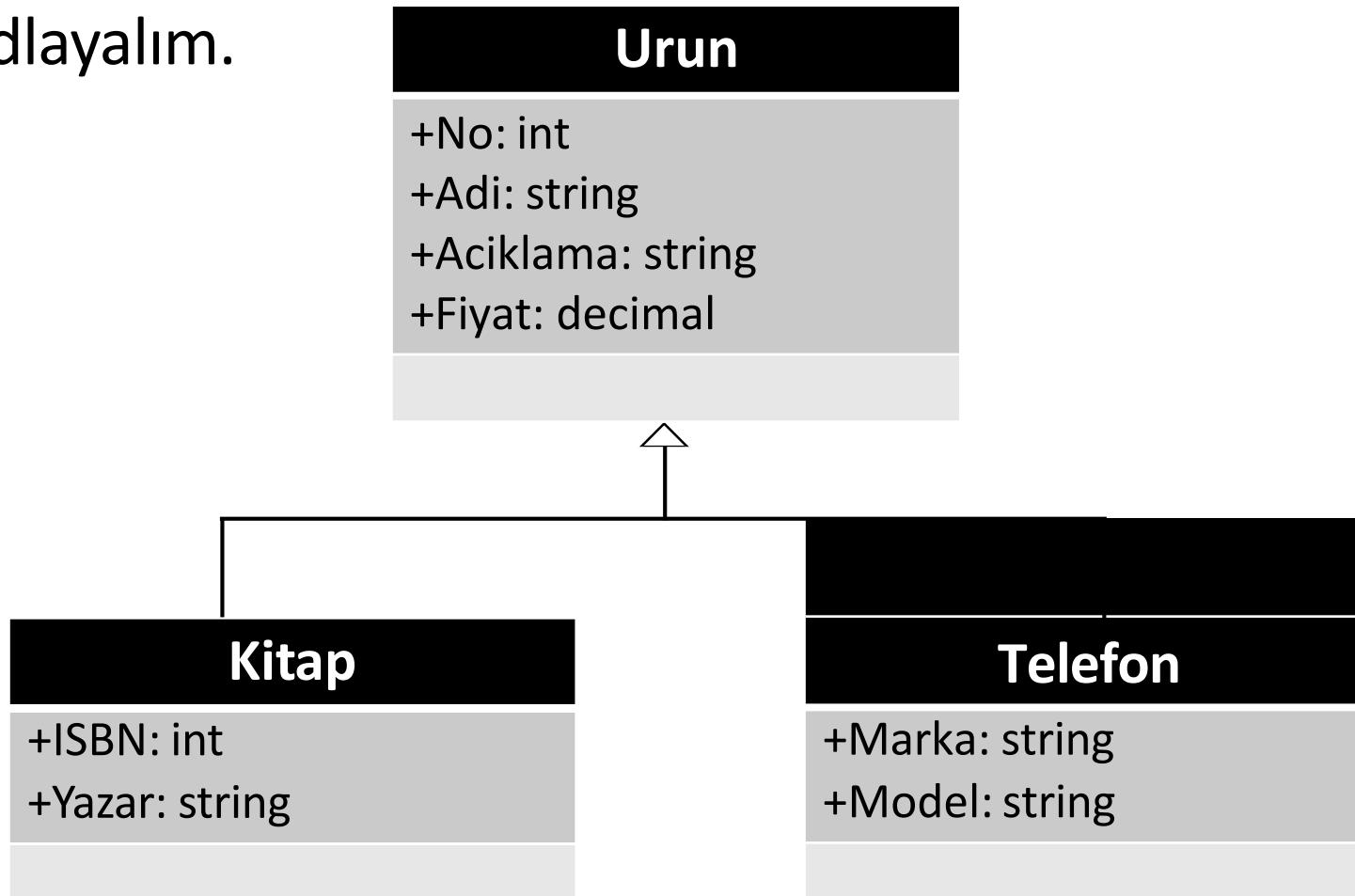
Örnek1: İlk Kalıtım Örneği

Senaryo:

- Sizden **kitap** ve **telefon** satan bir e-ticaret sistemi geliştirmeniz isteniyor. Sistemdeki her **kitap**; ad, açıklama, fiyat, numara, ISBN ve yazar adı bilgisine sahip olmalıdır. Sistemde kayıtlı olacak her **telefon**; ad, açıklama, fiyat, numara, marka ve model bilgilerini barındırmalıdır.

Örnek1: İlk Kalıtım Örneği (devam...)

- Adım adım kodlayalım.



Örnek1: İlk Kalıtım Örneği (devam...)

Adım 1

- Üç sınıfı da ayrı ayrı yaratalım (**Kalıtım bilmiyoruz**).
 - ✓ Urun
 - ✓ Kitap
 - ✓ Telefon
- Form üzerinde üç sınıftan birer tane **nesne** oluşturalım.
- Her nesne kaç özelliğe sahip?

Örnek1: İlk Kalıtım Örneği (devam...)

Adım 1

```
public class Urun
{
    0 references
    public int No { get; set; }
    0 references
    public string Adi { get; set; }
    0 references
    public string Aciklama { get; set; }
    0 references
    public decimal Fiyat { get; set; }
}
```

```
public class Kitap
{
    0 references
    public int ISBN { get; set; }
    0 references
    public string Yazar { get; set; }
}
```

```
public class Telefon
{
    0 references
    public string Marka { get; set; }
    0 references
    public string Model { get; set; }
}
```

Örnek1: İlk Kalıtım Örneği (devam...)

Adım 1

```
private void BtnTest_Click(object sender, EventArgs e)
{
    Urun urun = new Urun();
    urun.No = 1;
    urun.Adi = "Ürün 1";

    Kitap kitap = new Kitap();
    kitap.ISBN = 8728323;
    kitap.Yazar = "Yazar1";

    Telefon telefon = new Telefon();
    telefon.Marka = "Samsung";
    telefon.Model = "Galaxy xxx";|
}
```

Örnek1: İlk Kalıtım Örneği (devam...)

Adım 2

- Kalıtım / Miras işlemini gerçekleştirelim.
 - ✓ **Kitap** ve **Telefon** sınıflarını **Urun** sınıfından miras alalım.
- Form üzerinde oluşturulan nesnelerin özelliklerini gözlemleyin.
 - ✓ Şimdi her nesne kaç özelliğe sahip?

Örnek1: İlk Kalıtım Örneği (devam...)

Adım 2

```
private void BtnTest_Click(object sender, EventArgs e)
{
    Urun urun = new Urun();
    urun.No = 1;
    urun.Adi = "Ürün 1";

    Kitap kitap = new Kitap();
    kitap.ISBN = 8728323;
    kitap.Yazar = "Yazar1";
    kitap.Adi = "Sindrella";
    kitap.Fiyat = 23;

    Telefon telefon = new Telefon();
    telefon.Marka = "Samsung";
    telefon.Model = "Galaxy xxx";
    telefon.Adi = "S6 Edge";
    telefon.Fiyat = 900;
}
```

Örnek1: İlk Kalıtım Örneği (devam...)

Adım 2 Açıklama

- **Kitap** ve **Telefon** sınıflarından oluşturulan her bir nesne otomatik olarak **Urun** sınıfının erişim belirleyicisi **public** olan tüm *özelliklerini* içermektedir.
- Kalıtım tek yönlü çalışmaktadır:
 - ✓ Yavru sınıf, ana sınıfтан kalıtlara rak oluşturulur, ters yönde **oluşturulamaz**.
 - ✓ Program içerisinde bir **Urun nesnesi** oluşturduğunuzda Kitap sınıfının özelliklerine veya metodlarına erişemez.

Örnek1: İlk Kalıtım Örneği (devam...)

Adım 3

- **Urun** sınıfında **No** özelliğini **read-only** yapalım.
- **Urun** sınıfına bir tane **Constructor** ekleyelim ve burada **No** özelliğinin **random** olarak dolmasını sağlayalım.
- Form'da her nesneye ait **No** özellik değerini gösterelim.

Urun
+No: int {Read-Only}
+Adi: string
+Aciklama: string
+Fiyat: decimal
<<Constructor>>+Urun()

Örnek1: İlk Kalıtım Örneği (devam...)

Adım 3

```
public class Urun
{
    4 references
    public int No { get; private set; }
    6 references
    public string Adi { get; set; }
    0 references
    public string Aciklama { get; set; }
    2 references
    public decimal Fiyat { get; set; }
    1 reference
    public Urun()
    {
        Random random = new Random();
        int sayi = random.Next(1, 10000);
        this.No = sayi;
    }
}
```

```
private void BtnTest_Click(object sender, EventArgs e)
{
    Urun urun = ... Urun();
    urun.Adi = "Ürün 1";
    MessageBox.Show(urun.Adi + ": " + urun.No);

    Kitap kitap = ... Kitap();
    kitap.ISBN = 8728323;
    kitap.Yazar = "Yazar1";
    kitap.Adi = "Sindrella";
    kitap.Fiyat = 23 ;
    MessageBox.Show(kitap.Adi + ": " + kitap.No);

    Telefon telefon = ... Telefon();
    telefon.Marka = "Samsung";
    telefon.Model = "Galaxy xxx";
    telefon.Adi = "S6 Edge";
    telefon.Fiyat = 900;
    MessageBox.Show(telefon.Adi + ": " + telefon.No);
}
```

Örnek1: İlk Kalıtım Örneği (devam...)

Adım 3 Açıklama

- *Ana sınıfıta No özelliği read-only yapılınca yavru sınıflarda da bu özellik read-only oldu.*
- Sadece **Urun** sınıfından oluşturulan nesne mi No özellik **değeri aldı?** (Cevap: **Hayır**)
- **Kitap** ve **Telefon** sınıflarından oluşturulan nesneler de birer No özellik **değeri aldılar.**
- **Kalıtım** sadece özelliklerin değil aynı zamanda **metotlar ve kurucuların** da *ana sınıfından (Urun) miras alınarak yavru sınıflara (Kitap, Telefon) aktarılmasını sağlar.*

Örnek1: İlk Kalıtım Örneği (devam...)

Adım 4

- **Kitap** ve **Telefon** sınıflarına da **Constructor** ekleyelim.
- Nesneler form üzerinde oluşturulduğunda **SIRAYLA** hangi **Constructlar**ın çalıştığını **DEBUG işlemi yaparak gözlemleyelim.**

Kitap
+ISBN: int
+Yazar: string
<<Constructor>>+Kitap()

Telefon
+Marka: string
+Model: string
<<Constructor>>+Telefon()

Örnek1: İlk Kalıtım Örneği (devam...)

Adım 4

```
Kitap kitap = new Kitap();    ≤ 3.210ms elapsed
kitap.ISBN = 8728323;
kitap.Yazar = "Yazar1";
kitap.Adi = "Sindrella";
kitap.Fiyat = 23 ;
MessageBox.Show(kitap.Adi + ":" + kitap.No);

1 reference
public Kitap()    ≤ 1ms elapsed
{
}

public Urun()
{
    Random random = new Random();    ≤ 1ms elapsed
    int sayı = random.Next(1, 10000);
    this.No = sayı;
}

1 reference
public Kitap()
{
}

}    ≤ 1ms elapsed
```

Kitap nesnesi yaratılma aşaması.

Kitap Constructor'ın ilk satırına düşer ancak tamamlamadan **Urun** sınıfının **Constructor'ına** gider.

Urun sınıfının **Constructor'ı** tamamlanır.

Kitap sınıfının **Constructor'ı** tamamlanır.

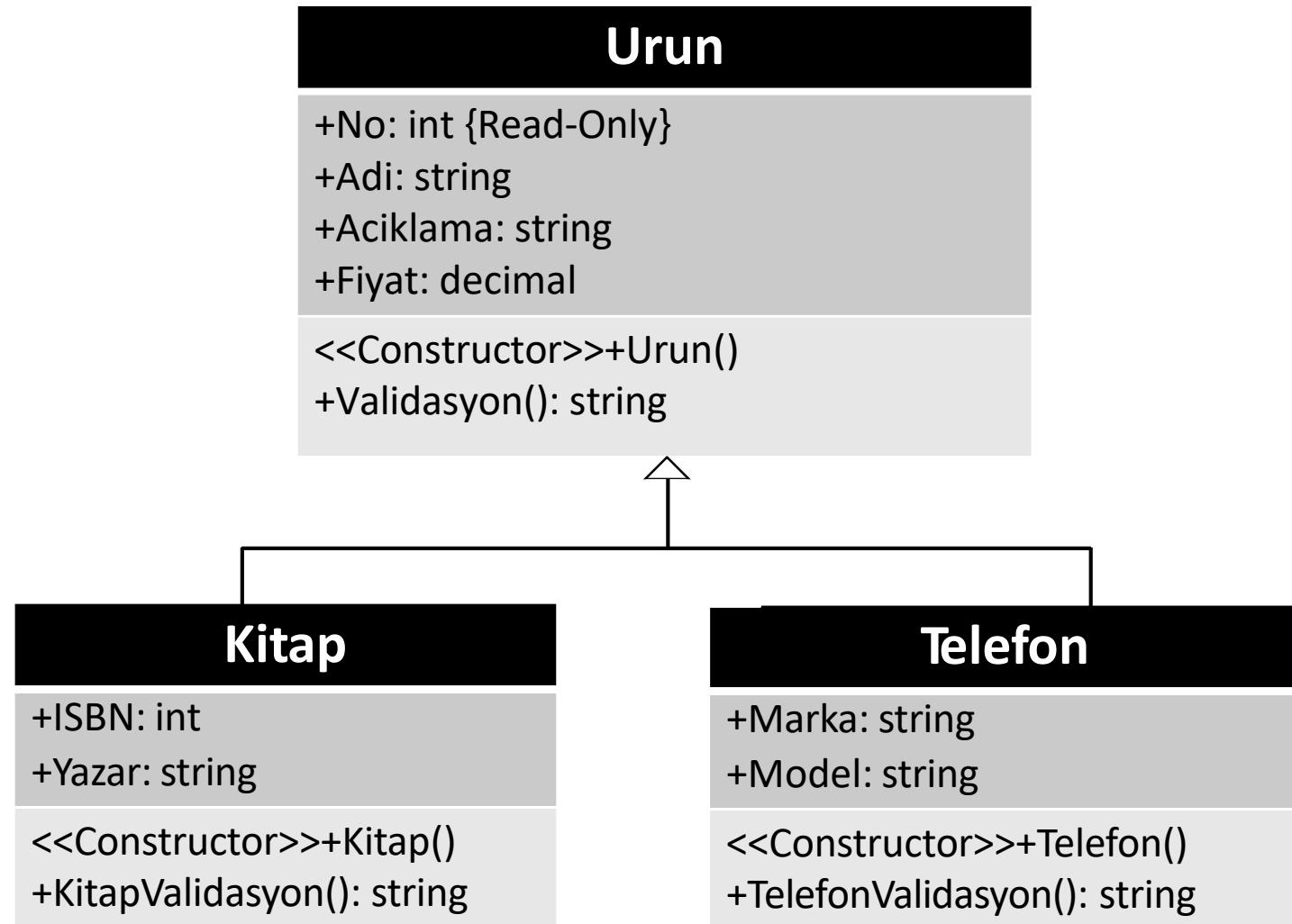
Örnek1: İlk Kalıtım Örneği (devam...)

Adım 5

- Bir ürünün **Adı** ve **Fiyatı** boş geçilemez.
 - ✓ Bu ürün **Kitapsa** **ISBN** ve **Yazar** adı,
 - ✓ **Telefon** ise **Model** ve **Marka** özelliklerini de ayrıca boş olamaz.
- **Ürün** sınıfına *Validasyon()* isimli bir metot ekleyelim.
- **Kitap** sınıfına *KitapValidasyon()* isimli bir metot ekleyelim.
 - ✓ Temel sınıfından **Validasyon()** metodunu da çağırırsın.
- **Telefon** sınıfına *TelefonValidasyon()* isimli bir metot ekleyelim.
 - ✓ Temel sınıfından **Validasyon()** metodunu da çağırırsın.

Örnek1: İlk Kalıtım Örneği (devam...)

Adım 5



Örnek1: İlk Kalıtım Örneği (devam...)

Adım 5.1 Çözüm

Ürün sınıfındaki
Validasyon() metodu

```
public string Validasyon()
{
    string hataMesaji = "";
    if (this.Adi == "")
        hataMesaji += "Ad özelliği boş olamaz.";
    if (this.Fiyat == 0)
        hataMesaji += "Fiyat özelliği 0 olamaz.";
    return hataMesaji;
}
```

Telefon sınıfındaki
TelefonValidasyon() metodu

```
public string TelefonValidasyon()
{
    return this.Validasyon();
}
```

Kitap sınıfındaki
KitapValidasyon() metodu

```
public string KitapValidasyon()
{
    return this.Validasyon();
}
```

```
private void BtnTest_Click(object sender, EventArgs e)
{
    Urun urun = new Urun();
    urun.Adi = "Ürün 1";
    //MessageBox.Show(urun.Adi + ": " + urun.No);
    MessageBox.Show("Ürün: " + urun.Validasyon());

    Kitap kitap = new Kitap();
    kitap.ISBN = 8728323;
    kitap.Yazar = "Yazar1";
    kitap.Adi = "";
    kitap.Fiyat = 23 ;
    //MessageBox.Show(kitap.Adi + ": " + kitap.No);
    MessageBox.Show("Kitap: " + kitap.Validasyon());

    Telefon telefon = new Telefon();
    telefon.Marka = "Samsung";
    telefon.Model = "Galaxy xxx";
    telefon.Adi = "S6 Edge";
    telefon.Fiyat = 0;
    //MessageBox.Show(telefon.Adi + ": " + telefon.No);
    MessageBox.Show("Telefon: " + telefon.Validasyon());
}
```

Örnek1: İlk Kalıtım Örneği (devam...)

Adım 5.2 Çözüm

```
public string KitapValidasyon()
{
    string hataMesaji = "";
    if (this.ISBN == 0)
        hataMesaji += "ISBN özelliği boş olamaz.";
    if (this.Yazar == "")
        hataMesaji += "Yazar özelliği boş olamaz.";
    return this.Validasyon() + " / " + hataMesaji;
}
```

```
public string TelefonValidasyon()
{
    string hataMesaji = "";
    if (this.Marka == "")
        hataMesaji += "Marka özelliği boş olamaz.";
    if (this.Model == "")
        hataMesaji += "Model özelliği boş olamaz.";
    return this.Validasyon() + " / " + hataMesaji;
}
```

```
private void BtnTest_Click(object sender, EventArgs e)
{
    Urun urun = ...;
    urun.Adi = "Ürün 1";
    //MessageBox.Show(urun.Adi + ": " + urun.No);
    MessageBox.Show("Ürün: " + urun.Validasyon());

    Kitap kitap = ...;
    kitap.ISBN = 8728323;
    kitap.Yazar = "Yazar1";
    kitap.Adi = "";
    kitap.Fiyat = 23 ;
    //MessageBox.Show(kitap.Adi + ": " + kitap.No);
    MessageBox.Show("Kitap: " + kitap.Validasyon());

    Telefon telefon = ...;
    telefon.Marka = "Samsung";
    telefon.Model = "Galaxy xxx";
    telefon.Adi = "S6 Edge";
    telefon.Fiyat = 0;
    //MessageBox.Show(telefon.Adi + ": " + telefon.No);
    MessageBox.Show("Telefon: " + telefon.Validasyon());
}
```

Örnek1: İlk Kalıtım Örneği (devam...)

Adım 6

- Ürün sınıfındaki Validasyon() metodunun erişim belirleyicisini **public'ten protected'a** çekersek ne olur?
 - ✓ **Soru1:** Validasyon() metodu hala Form üzerinden çağrılabılır mı?
 - ✓ **Soru2:** KitapValidasyon() bu metodu çağrılabılır mı?
 - ✓ **Soru3:** TelefonValidasyon() bu metodu çağrılabılır mı?

Değiştirip Görelim !!!

Örnek1: İlk Kalıtım Örneği (devam...)

Adım 6

- **Soru1:** Hayır
- **Soru2:** Evet
- **Soru3:** Evet

```
protected string Validasyon()
{
    string hataMesaji = "";
    if (this.Adi == "")
        hataMesaji += "Ad özelliği boş olamaz.";
    if (this.Fiyat == 0)
        hataMesaji += "Fiyat özelliği 0 olamaz.";
    return hataMesaji;
}
```

Örnek1: İlk Kalıtım Örneği (devam...)

Adım 6 Açıklama

- **protected** erişim belirleyicisine sahip olan özellikler veya metotlar,
 - Tanımlandıkları sınıfın içerisinde ya da
 - Tanımlı oldukları sınıflardan kalıtımıla oluşturulan sınıflar içerisinde erişilebilirler.
- Bu sınıfların **dışında** **erişilemezler**.
- Diğer bir deyişle, **protected** erişim belirleyicisine sahip **üyeler** ailenin **icerisinde** (ana - yavru) erişilebilirler.

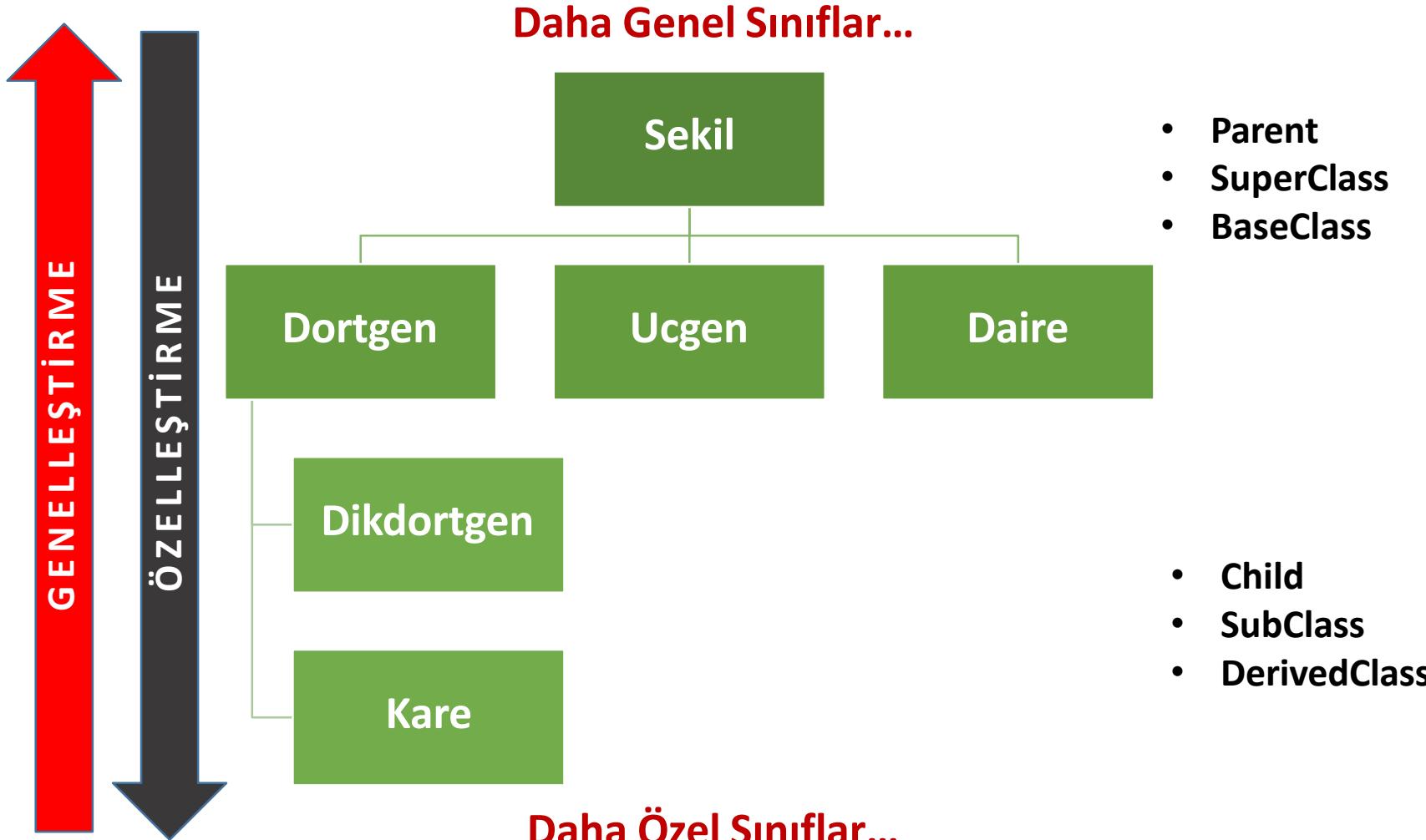
Sınıfların Genişletilmesi

- Sınıftan türetilerek yeni bir sınıf oluşturulmasını engellemek için sınıf başlığı **sealed** anahtar sözcüğüyle tanımlanır. Hazır olarak gelen **String** sınıfı sealed sınıflara örnek olarak verilebilir.

```
sealed class sinif
```

```
{  
}
```

Sınıfların Genişletilmesi (devam...)

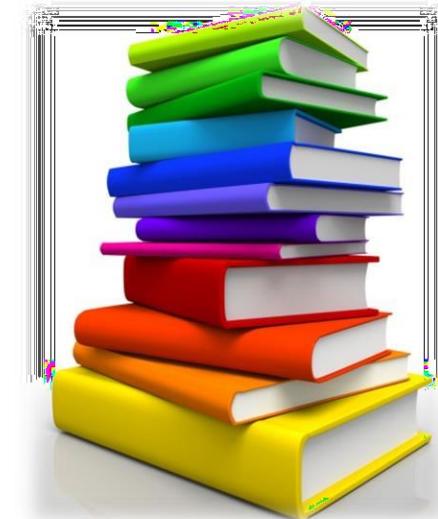


Kalıtım Avantajları

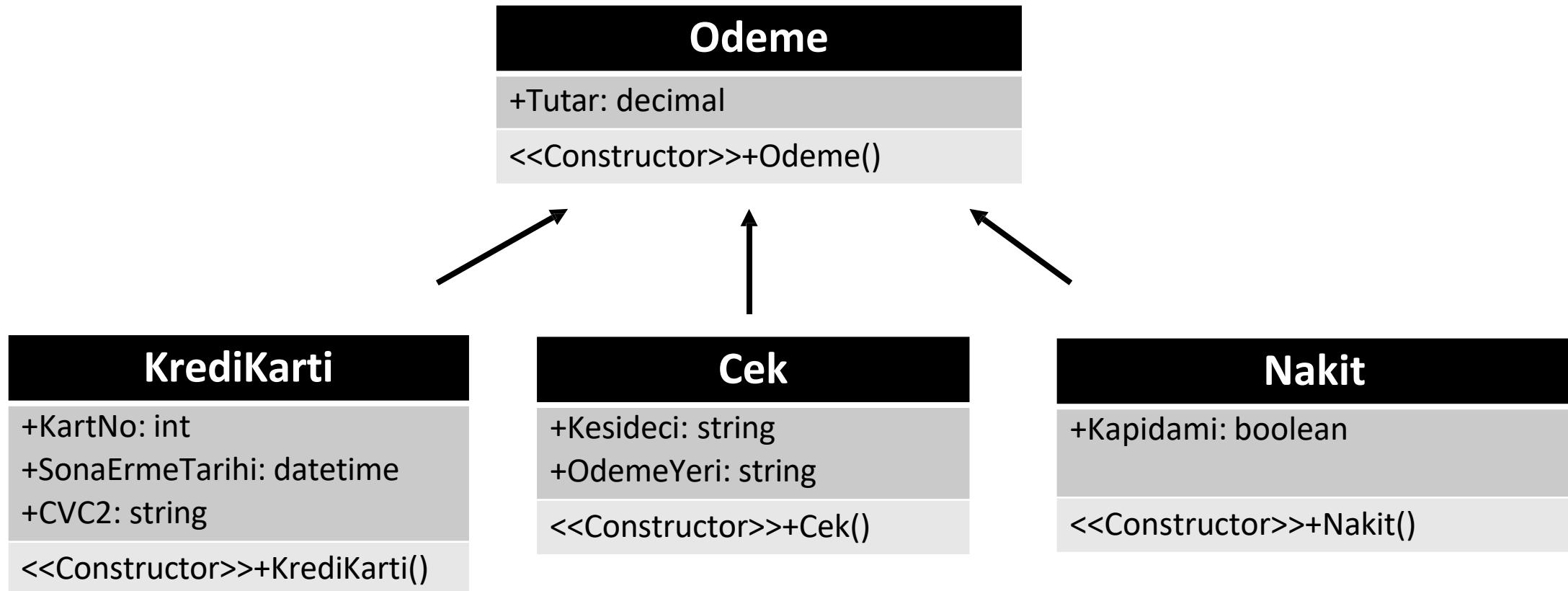
- Kalıtımı kullanabilme kabiliyeti, programı
 - ✓ daha kolay ve az kod yazma,
 - ✓ daha kolay anlama ve
 - ✓ daha az hata ile karşılaşmayı sağlamaktadır.
- Kalıtımı kullanarak, düzgün bir şekilde, hızlıca yeni sınıflar oluşturulabilir.

BÖLÜM 7 – Kalıtım Uygulamaları

- Örnekler
 - Uygulama 1 (Basit → Ödeme)
 - Uygulama 2 (Basit → Çok Seviyeli Kalıtım)
 - Uygulama 3 (Basit-Orta → Firma Çalışanları)
 - Uygulama 4 (Basit-Orta → Sipariş)



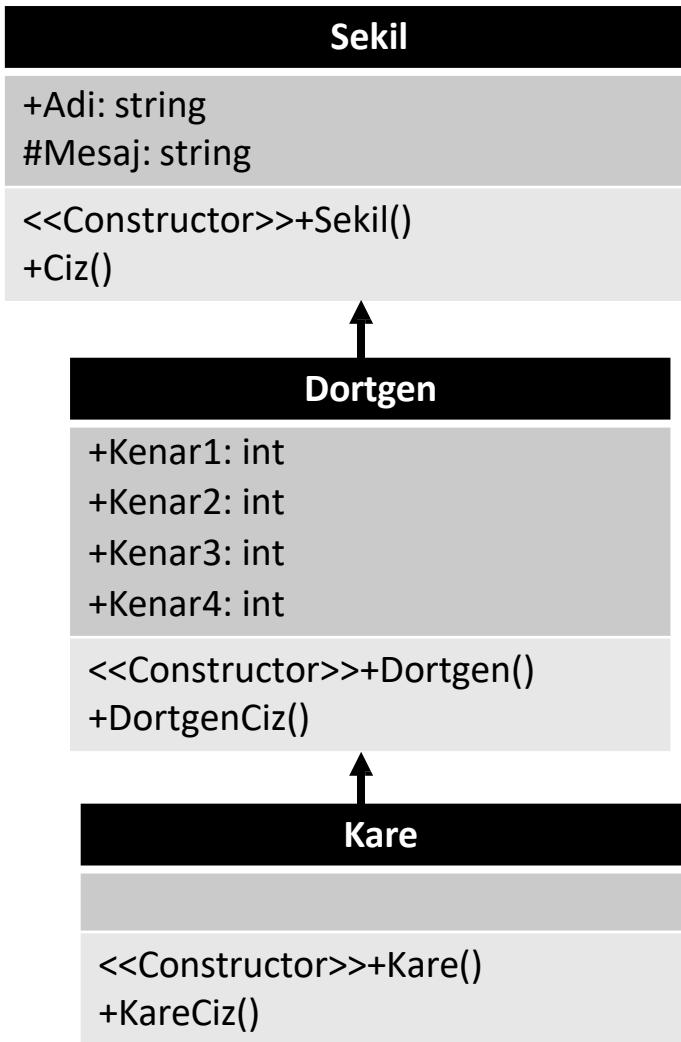
Uygulama 1: Ödeme



Uygulama 2: Çok Seviyeli Kalıtım

Notlar:

- Gerçekleştirmde soyut sınıf ve interface kullanılmayacaktır.
- Bir tane Kare nesnesi oluşturarak, oluşum sürecini **Debug** edip gözlemleyin.
- Mesaj özelliğinin girilen değerini üst sınıflardan alt sınıfa taşıyın ve nesne üzerindeki kullanımına dikkat edin.
- Adı özelliği **public** ancak sadece hiyerarşideki sınıflar tarafından yazılabilir olsun. Nesne üzerinden kullanımda bu özellik **read-only** olmalıdır.



Uygulama 3: Firma Çalışanları

Senaryo:

Firmadaki çalışanları Nesne Yönelimli Programlama (NYP) teknikleri ile modellemeye çalıştığımızı hayal edelim. Kişiler; TC Kimlik No, İsim ve Adres bilgilerine sahip olup, Adres bilgisi de kendi içerisinde İl, İlçe, Mahalle ve Açık Adres bilgilerini içermektedir. Adres ile Kişi varlığı arasında bir ilişki olsa da Adres bilgisi tek başına da kullanılabilir (Kişi ile Adres varlıkları, birbirlerinin oluşumlarından bağımsız değerlendirilmelidir). Firmadaki bir Çalışan aynı zamanda bir Kişi'dir. Çalışanlar; SGK Numarası, Firma, Görev ve Departman bilgilerini içermekte olup, Firma varlığı da kendi içerisinde üç bilgiyi barındırmaktadır: (a) Firma Adı (b) Firma Adresi ve (c) Firma Çalışanları. Firma ve Çalışan arasında 1-n şeklinde bir ilişki vardır.

Notlar:

1. Gerçekleştirimde soyut sınıf ve interface kullanılmayacaktır.
2. Firma çalışanlarına erişim dışarıya kısıtlanmalıdır.

Temel Sınıfın Metotlarını Ezme

- Önceden var olan bir sınıfından **miras alınarak**, genişletilmiş yeni bir sınıf oluşturduğunuzda, yeni sınıf içerisinde temel sınıfın *tüm özellik ve metotları tanımlanmış* olur.
- Bazen parent (ebeveyn) sınıfın üyeleri, özellikleri ve metotları **tam olarak** child (yavru) sınıfından yaratılan nesneler için **uygun olmayıabilir**.

Farklı işlemler yapan fakat aynı isimdeki özellik veya metodların kullanımına çok biçimlilik (polymorphism) denmektedir.

Temel Sınıfın Metotlarını Ezme (devam...)

Çok biçimlilik, “birçok form içeren” anlamına gelmektedir.

Aynı isimde olmalarına rağmen farklı işlemlerin yer aldığı metotlar için kullanılır.

Temel Sınıfın Metotlarını Ezme (devam...)

- **Günlük hayatı** çok biçimliliğe **örnek** gösterebilecek çeşitli olaylar bulunmaktadır:
 - Tüm **müzik aletleri** için “**çalmak**” eylemi kullanılmasına rağmen, *bir gitarın bir davuldan* farklı bir biçimde çalınıyor olması. (*Cal()* metotları aynı)
 - Tüm **araçlar** için “**sürmek**” eylemi kullanılmasına rağmen, *bir otomobilin* kullanışının *bir bisiklet* kullanımından farklı olması,
 - Tüm **okulların** “**mezun olma** koşulları”na sahip olması fakat bir *lise mezuniyeti* ile *ilkokul mezuniyeti* koşulları arasında farklılık olması çok biçimliliğe örnek gösterilebilir.

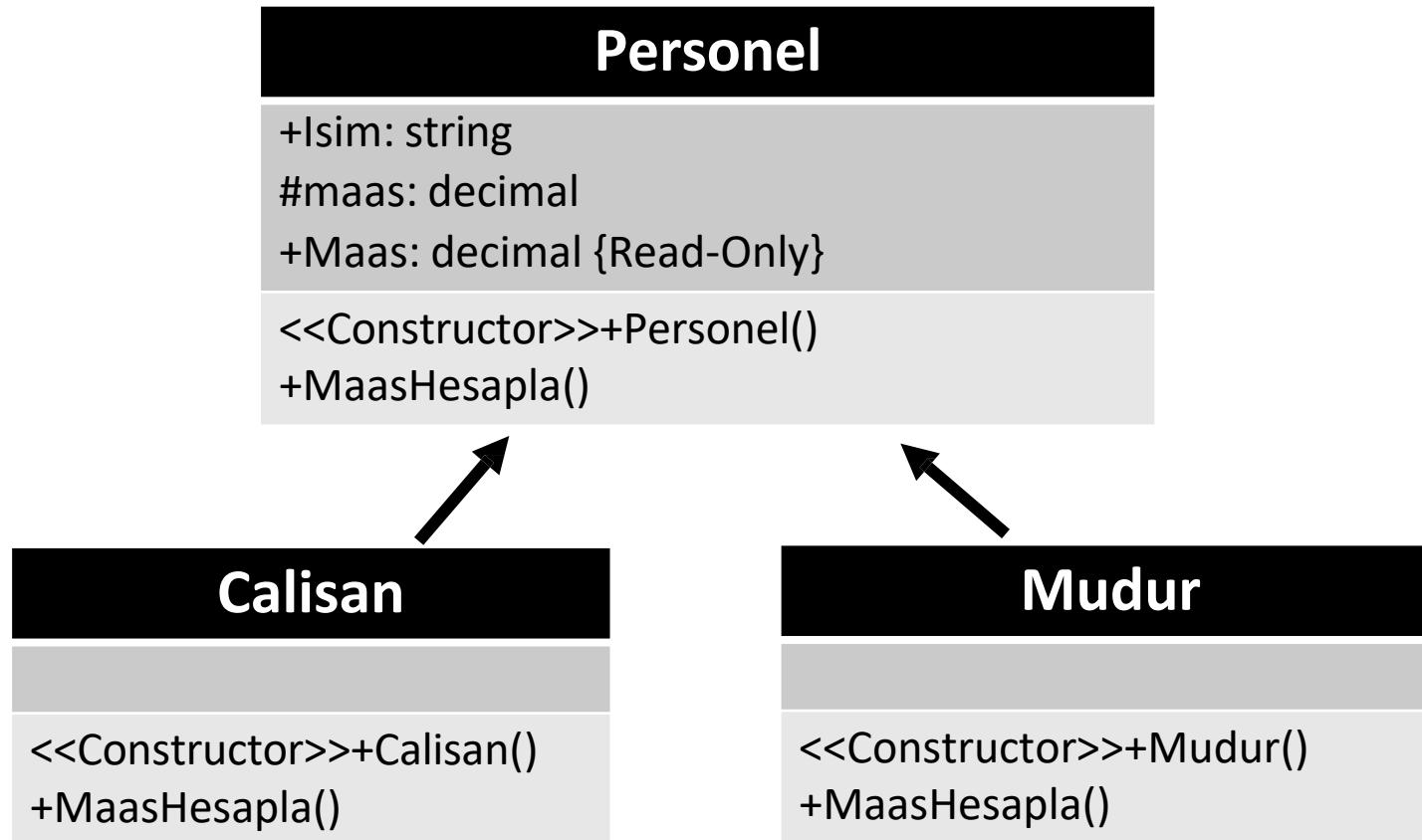
Temel Sınıfın Metotlarını Ezme (devam...)

- Bir **virtual** metot (ya da property) child sınıfındaki aynı isme sahip bir metot tarafından başına **override** anahtar kelimesi eklenerek ezilmesine olanak sağlar.

```
public class Parent
{
    public virtual void Metot1()
    {
        ...
    }
}

public class Child: Parent
{
    public override void Metot1()
    {
        ...
    }
}
```

Örnek 1: Maaş Hesapla



Örnek 1: Maaş Hesapla

Senaryo 1:

- Personel maaş hesaplama ile ilgili **Personel Temel sınıfında** herhangi özel bir hesaplama yapılmıyor.
- Çalışan maaşları 4000 TL'dir.
- Müdür maaşları 7000 TL'dir.
- Çok biçimlilik kullanarak bu durumu gerçekleştirelim.
- Form üzerinde nesneleri oluşturarak test ve **DEBUG** işlemi gerçekleştirelim.

Örnek 1: Maaş Hesapla

```
public class Personel
{
    0 references
    public string Isim { get; set; }
    protected Decimal maas;
    3 references
    public Decimal Maas {
        get {
            return maas;
        }
    }
    1 reference
    public Personel()
    {
        //Todo
    }
    5 references
    public virtual void MaasHesapla()
    {
        //Todo
    }
}
```

Senaryo 1

```
public class Calisan : Personel
{
    1 reference
    public Calisan()
    {
        //Todo
    }
    5 references
    public override void MaasHesapla()
    {
        maas = 4000;
    }
}
```

```
public class Mudur : Personel
{
    1 reference
    public Mudur()
    {
        //Todo
    }
    5 references
    public override void MaasHesapla()
    {
        maas = 7000;
    }
}
```

Örnek 1: Maaş Hesapla

Senaryo 1:

- **Calisan** MaasHesapla() metodу çağırıldığında Temel Sınıf olan **Personel**'in MaasHesapla() metodу çağrılıyor mu?
- **Cevap: Hayır**
- **İhtiyaç olsaydı?**

```
Personel personel = new Personel();
personel.MaasHesapla();
MessageBox.Show(personel.Maas.ToString());  
  
Calisan calisan = new Calisan();
calisan.MaasHesapla();
MessageBox.Show(calisan.Maas.ToString());  
  
Mudur mudur = new Mudur();
mudur.MaasHesapla();
MessageBox.Show(mudur.Maas.ToString());
```

i Yavru sınıf, temel sınıfın metodlarına **base** anahtar sözcüğü ile erişebilir.

Örnek 1: Maaş Hesapla

Senaryo 2:

- Personel maaş hesaplama ile ilgili ***Personel Temel sınıfında özel bir hesaplama yapılıyor.***
 - Maaş = ASGARIUCRET (2000)+ AILEGECIMINDIRIMI (500);
- Çalışan maaşları 1.5 kat.
- Müdür maaşları 3.5 kat.
- Çok biçimlilik kullanarak bu durumu implemente edelim.
- Form üzerinde nesneleri oluşturarak test ve **DEBUG** işlemi gerçekleştirelim.

Örnek 1: Maaş Hesapla

```
public class Personel
{
    private const Decimal ASGARIUCRET = 2000;
    private const Decimal AILEGECIMINDIRIMI = 500;

    0 references
    public string Isim { get; set; }
    protected Decimal maas;

    3 references
    public Decimal Maas {
        get {
            return maas;
        }
    }

    1 reference
    public Personel()
    {
        //Todo
    }

    7 references
    public virtual void MaasHesapla()
    {
        maas = ASGARIUCRET + AILEGECIMINDIRIMI;
    }
}
```

Senaryo 2

```
public class Calisan : Personel
{
    1 reference
    public Calisan()
    {
        //Todo
    }

    7 references
    public override void MaasHesapla()
    {
        base.MaasHesapla();
        maas *= 1.5M;
    }
}
```

```
public class Mudur : Personel
{
    1 reference
    public Mudur()
    {
        //Todo
    }

    7 references
    public override void MaasHesapla()
    {
        base.MaasHesapla();
        maas *= 4M;
    }
}
```

Örnek 2: Öğrenci ve Burslu Öğrenci

Senaryo:

- Öğrenciler; *isim*, *kredi sayısı* ve *toplam ders ücreti* bilgilerine sahiptirler.
- Öğrencilerin toplam ders ücretleri *read-only* olup, *birim ders ücreti* ile aldığı kredi sayısı çarpılarak *DersUcretiHesapla()* isimli bir fonksiyon aracılığı ile hesaplanmaktadır.

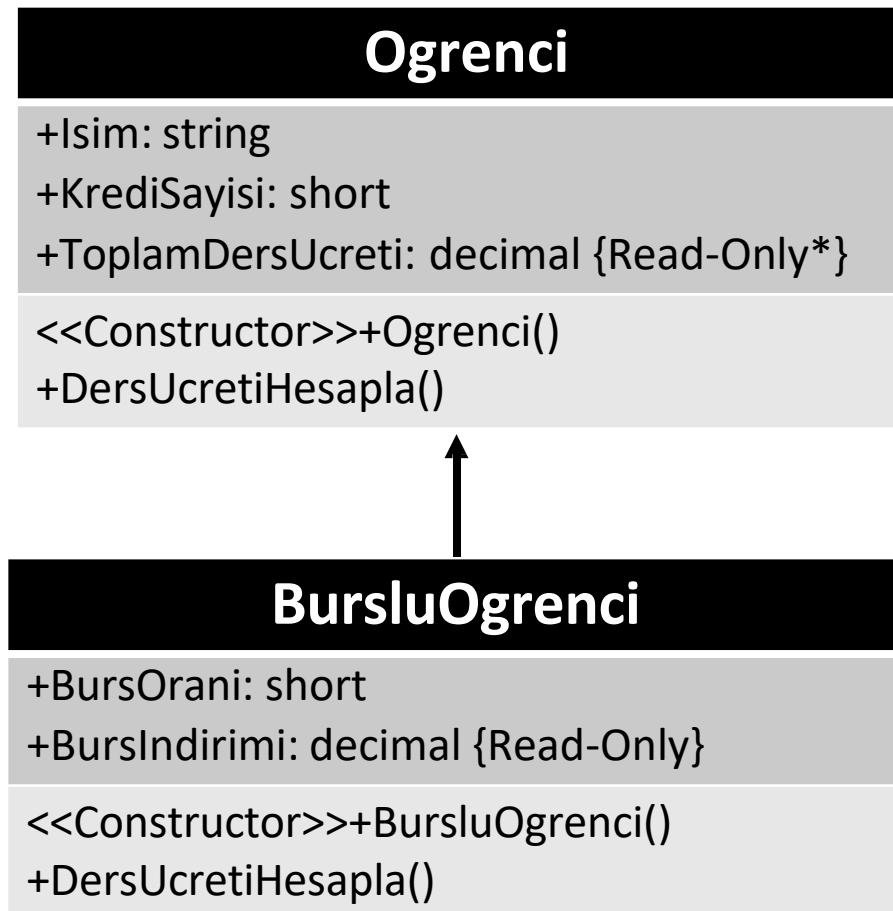
$$\text{BIRIMDERSUCRETI} = 60$$

- Burslu öğrenci de bir öğrencidir.
- Burslu öğrenciler, toplam ders ücreti üzerinden burs oranı kadar burs indirimi alırlar. Burs indirimi *read-only* olup aşağıdaki gibi hesaplanır.

$$\text{BursIndirimi} = (\text{ToplamDersUcreti} * \text{BursOrani}) / 100$$

- Çok biçimlilik kullanarak bu durumu implemente edelim.

Örnek 2: Öğrenci ve Burslu Öğrenci



Örnek 2: Öğrenci ve Burslu Öğrenci

```
public class Ogrenci
{
    private const Decimal BIRIMDERSUCRETI = 60;
    0 references
    public string Isim { get; set; }
    3 references
    public short KrediSayisi { get; set; }
    5 references
    public Decimal ToplamDersUcreti { get; protected set; }
    1 reference
    public Ogrenci()
    {
        //Todo
    }
    4 references
    public virtual void DersUcretiHesapla()
    {
        ToplamDersUcreti = BIRIMDERSUCRETI * KrediSayisi;
    }
}
```

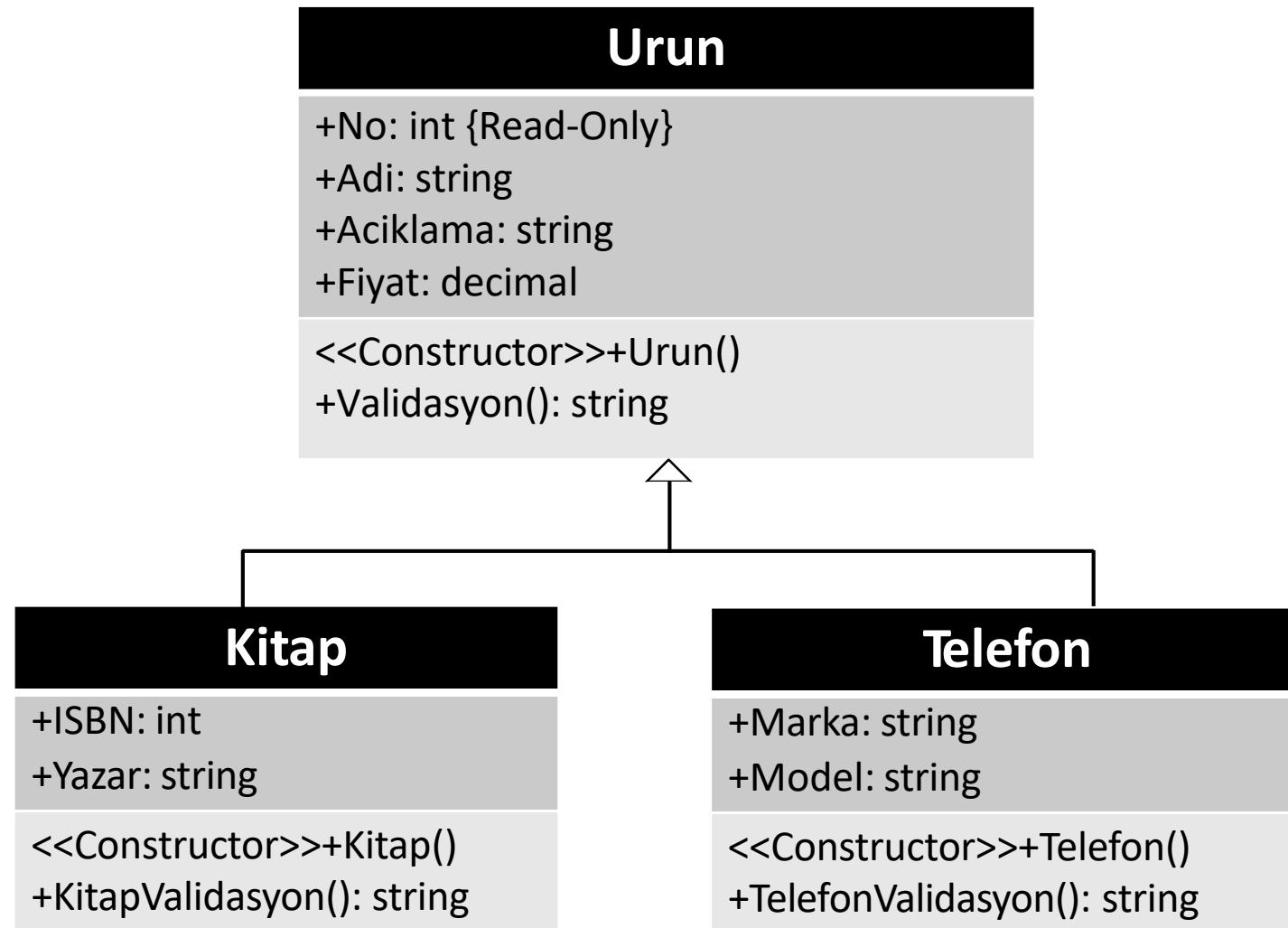
```
public class BursluOgrenci : Ogrenci
{
    2 references
    public short BursOrani { get; set; }
    3 references
    public decimal BursIndirimi { get; private set; }
    1 reference
    public BursluOgrenci()
    {
        //Todo
    }
    4 references
    public override void DersUcretiHesapla()
    {
        base.DersUcretiHesapla();
        BursIndirimi = (ToplamDersUcreti * BursOrani)/100;
        ToplamDersUcreti -= BursIndirimi;
    }
}
```

Örnek 2: Öğrenci ve Burslu Öğrenci

```
private void Form1_Load(object sender, EventArgs e)
{
    Ogrenci ogrenci = new Ogrenci();
    ogrenci.KrediSayisi = 23;
    ogrenci.DersUcretiHesapla();
    MessageBox.Show("Öğrenci toplam Ders ücreti: " + ogrenci.ToplamDersUcreti);

    BursluOgrenci bursluOgrenci = new BursluOgrenci
    {
        KrediSayisi = 23,
        BursOrani = 20
    };
    bursluOgrenci.DersUcretiHesapla();
    MessageBox.Show("Burslu Öğrenci" + Environment.NewLine +
                   "Burs İndirimi: " + bursluOgrenci.BursIndirimi + Environment.NewLine +
                   "ToplamDers Ücreti: " + bursluOgrenci.ToplamDersUcreti);
}
```

Örnek 3: İlk Kalıtım Örneği - Hatırla !



Örnek 3: İlk Kalıtım Örneği - Hatırla !

```
public string Validasyon()
{
    string hataMesaji = "";
    if (this.Adi == "")
        hataMesaji += "Ad özelliği boş olamaz.";
    if (this.Fiyat == 0)
        hataMesaji += "Fiyat özelliği 0 olamaz.";
    return hataMesaji;
}
```

```
public string TelefonValidasyon()
{
    string hataMesaji = "";
    if (this.Marka == "")
        hataMesaji += "Marka özelliği boş olamaz.";
    if (this.Model == "")
        hataMesaji += "Model özelliği boş olamaz.";
    return this.Validasyon() + " / " + hataMesaji;
}
```

```
public string KitapValidasyon()
{
    string hataMesaji = "";
    if (this.ISBN == 0)
        hataMesaji += "ISBN özelliği boş olamaz.";
    if (this.Yazar == "")
        hataMesaji += "Yazar özelliği boş olamaz.";
    return this.Validasyon() + " / " + hataMesaji;
}
```

Örnek 3: İlk Kalıtım Örneği - Hatırla !

- Eski tasarım *tam olarak doğru değil*.
- **KitapValidasyon()** ve **TelefonValidasyon()** **isimli** metotları *oluşturmayaya artık gerek yok*.
- Bunun yerine, **Temel Sınıftaki Validasyon()** metodu **virtual** yapılarak **yavru sınıflarda ezilebilir** ve NYP (OOP) çerçevesinde daha **doğru bir tasarım gerçekleştirilir**.

Soyut Sınıflar

- Kalıtım kavramını anladıktan sonra, sınıfların yaratılması daha da kolaylaşır.
- **Yavru (Child)** bir sınıf oluşturduğunuzda, genel özellikleri **miras alınır** ve sonrasında sınıfa gereken yeni, **spesifik özellikler eklenir/kazandırılır**.

Örneğin:

Ressam ile Sair sınıfları Sanatçı sınıflarının daha da özelleşmiş halidir.

Bu sınıflar, Sanatçı sınıfının **erişilebilen tüm özelliklerini ve metodlarını barındırmakla beraber **spesifik** özellikleri de barındırırlar.**

Soyut Sınıflar (devam...)

- Temel sınıfları düşünmenin bir yolu, **yavru sınıfların** ortak özelliklerinin **tümünü düşünmek** olabilir.
- Genişletilmiş sınıf incelendiğinde, ebeveyn sınıfının daha genel olduğu **gözlenir**.
- Bazen yavru sınıfların daha genel bir örneğini oluşturmak için **ebeveyn sınıf** oluşturulur.
 - Örneğin, önceden bir **Sanatçı** yaratmamış olabilirsiniz; herbir **Sanatçı** bir **Ressam'ın**, **Sair'in** ya da **Muzisyen'in** daha genel halidir.

Soyut Sınıflar (devam...)

*Sonradan genişletilmek üzere yaratılan fakat kendisinden **nesne** oluşturulmayan sınıflara soyut sınıf (abstract class) denir.*

- Soyut sınıfları tanımlarken **abstract** anahtar kelimesi kullanılır.
- Soyut sınıflar da normal sınıflar gibi özellikler ve metodlar içermektedir. Normal sınıflardan farklı olarak bu sınıflardan **new** operatörünü kullanarak **nesne** yaratılamaz.
- Bunun yerine, soyut sınıflar bir **ana sınıf sağlar**.
- Soyut sınıflar genellikle soyut metodlar (abstract methods) içerirler.
- Soyut bir metod **hiçbir metod ifadesi içermez**; bu sınıfın türetilen sınıflar, bu metodları ezmelidir.

Soyut Sınıflar (devam...)

- Soyut bir metodun başlığında, isteğe bağlı erişim belirleyicisi, **abstract** anahtar kelimesi, istenilen **metodun tipi** ve **adı** bulunur:
public abstract void Hesapla();
- Soyut metodlarda **kod gövdesi bulunmaz**. Bu metodların gövdeleri **kalıtılan sınıfta tanımlanır**.
- Soyut bir sınıfın miras alınarak, yeni bir sınıf yaratıldığında, yeni sınıfın **override** anahtar kelimesini kullanarak **soyut metodların gövdeleri** oluşturulmalıdır.

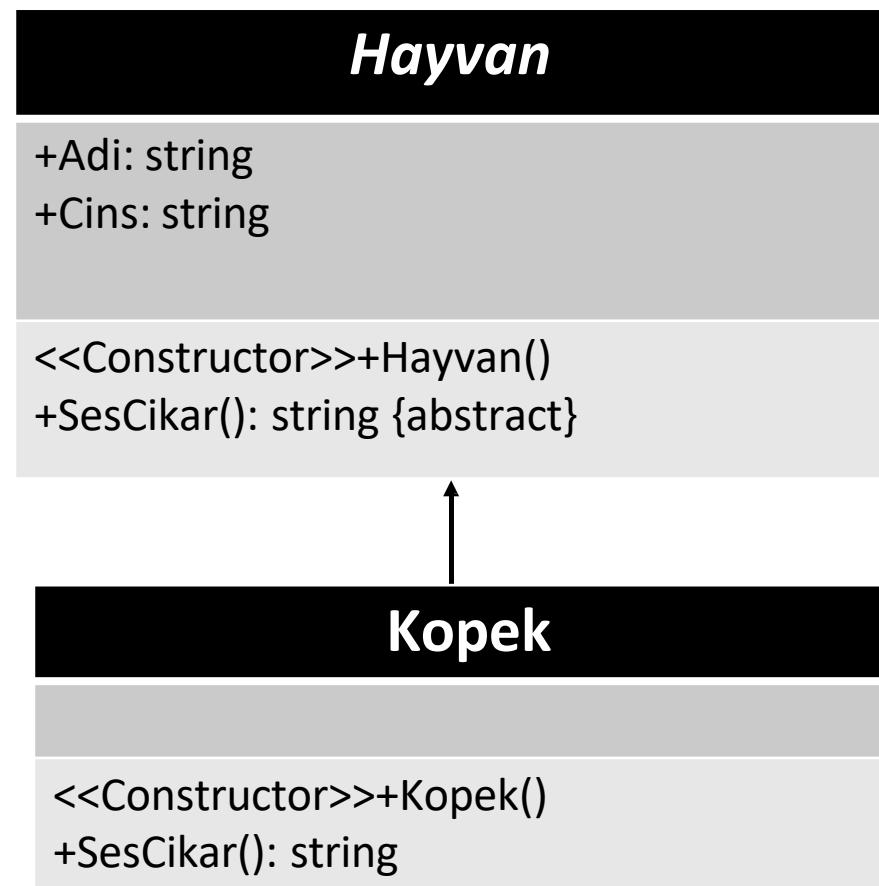
Soyut Sınıflar (devam...)

virtual anahtar kelimesi ile tanımlanan metodun **yavru sınıfı** **ezilmesi** zorunlu değildir.

Fakat **abstract** tanımlanmış **metot ezilmelidir**.

Aslında esas **çok biçimlilik (polymorphism)** soyut metotlar ve ara yüzler (**interface**) aracılığı ile gerçekleştirilmektedir.

Örnek1: Hayvan Soyut Sınıfının Yaratılması



Örnek1: Hayvan Soyut Sınıfının Yaratılması

```
public abstract class Hayvan
{
    1 reference
    public string Adi { get; set; }
    0 references
    public string Cins { get; set; }
    0 references
    public Hayvan()
    {
        //Todo
    }
    2 references
    public abstract string SesCikar();
}
```

```
2 references
public class Kopek : Hayvan
{
    2 references
    public override string SesCikar()
    {
        return "Havhav";
    }
}
```

Örnek1: Hayvan Soyut Sınıfının Yaratılması

- Form üzerinde bir **Kopek nesnesi** yaratalım.
- Kopek bilgilerini **ekranda göstermek için** form üzerinde **KopekBilgisiGoster()** isimli bir metot oluşturalım.

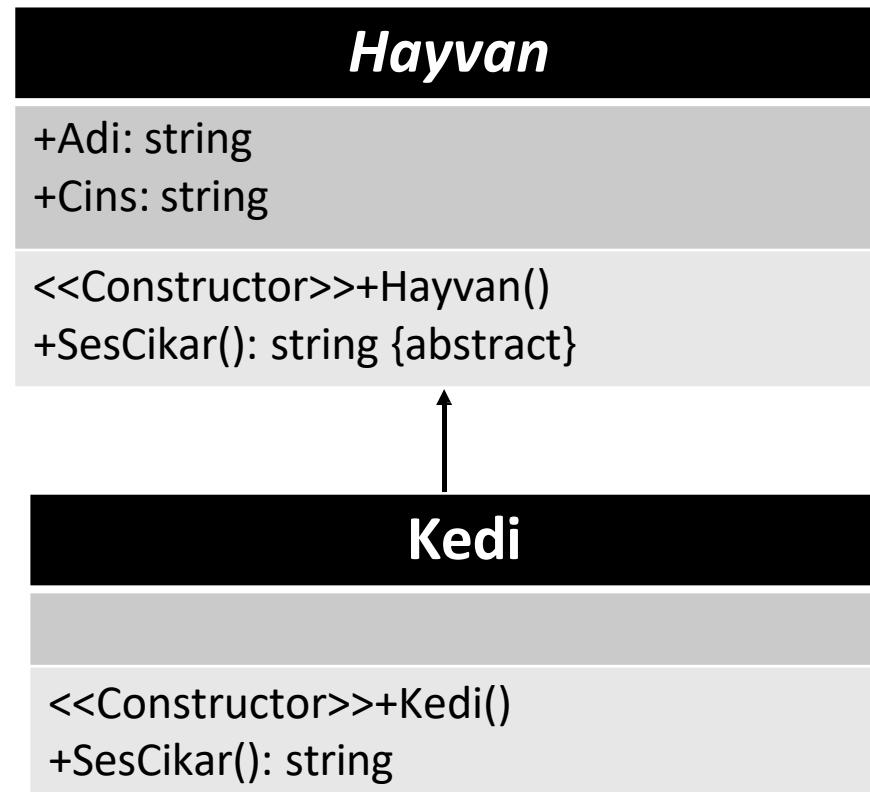
```
1 reference
private void KopekBilgisiGoster(Kopek kopek)
{
    MessageBox.Show("Köpek adı: " + kopek.Adi + Environment.NewLine +
                    "Ses çıkar: " + kopek.SesCikar());
}

1 reference
private void Form1_Load(object sender, EventArgs e)
{
    Kopek kopek = ...;
    kopek.Adi = "Zeytin";
    KopekBilgisiGoster(kopek);
}
```

Soyut Sınıflar (devam...)

- Soyut bir metot **tanımlamadan da soyut bir sınıf oluşturulabilir** fakat **soyut sınıf oluşturulmadan soyut bir metot oluşturulamaz.**
- Child sınıfında override edilmesi gereken metot kendi tanımına sahipse, bu metot **virtual** olarak tanımlanır. Eğer kendine ait bir tanım/gövde yoksa **abstract** olarak tanımlanır.
- Örnekteki; **Kopek** sınıfında **SesCikar()** metodu **ezilmek zorundadır.** **Bu metodun içerisinde herhangi bir işlem yapılmayabilir fakat bu metot mutlaka olmalıdır.** Bu metodu ezerken **override** anahtar kelimesi kullanılır.

Örnek1: Hayvan Soyut Sınıfının Yaratılması



Örnek1: Hayvan Soyut Sınıfının Yaratılması

```
public class Kedi : Hayvan
{
    |
    | 4 references
    |
    public override string SesCikar()
    {
        |
        | return "Miyavvvv";
        |
    }
}
```

Örnek1: Hayvan Soyut Sınıfının Yaratılması

- Form üzerinde bir **Kedi nesnesi** yaratalım.
- Kopek bilgilerini **ekranda göstermek için** form üzerinde **KediBilgisiGoster()** isimli bir metot oluşturalım.

```
1 reference
private void KediBilgisiGoster(Kedi kedi)
{
    MessageBox.Show("Kedi adı: " + kedi.Adi + Environment.NewLine +
                    "Ses çıkar: " + kedi.SesCikar());
}
```

```
Kedi kedi = ...
kedi.Adi = "Boncuk";
KediBilgisiGoster(kedi);
```

Örnek1: Hayvan Soyut Sınıfının Yaratılması

Dikkat: İki metot arasında ne fark var?? Daha iyi bir çözümü olan var mı?

```
1 reference
private void KopekBilgisiGoster(Kopek kopek)
{
    MessageBox.Show("Köpek adı: " + kopek.Adi + Environment.NewLine +
                    "Ses çıkar: " + kopek.SesCikar());
}

1 reference
private void KediBilgisiGoster(Kedi kedi)
{
    MessageBox.Show("Kedi adı: " + kedi.Adi + Environment.NewLine +
                    "Ses çıkar: " + kedi.SesCikar());
}
```

Örnek1: Hayvan Soyut Sınıfının Yaratılması

- Kedi ve Kopek sınıfından yaratılmış nesneler Hayvan parametresi alan **HayvanBilgisiGoster()** metoduna aktarılabilirler.
- Bu yönteme teknik olarak **upcast** de denir.
- Ortaya konulan yazılım tasarım prensibine LSP (Liskov Substitution Principle - Liskov Yerine Geçme/Değiştirme) prensibi denilir (Barbara Liskov).

Örnek1: Hayvan Soyut Sınıfının Yaratılması

```
2 references
private void HayvanBilgisiGoster(Hayvan hayvan)
{
    MessageBox.Show("Hayvan adı: " + hayvan.Adi + Environment.NewLine +
                    "Ses çıkar: " + hayvan.SesCikar());
}

1 reference
private void Form1_Load(object sender, EventArgs e)
{
    Kopek kopek = ...;
    kopek.Adi = "Zeytin";
    HayvanBilgisiGoster(kopek);

    Kedi kedi = ...;
    kedi.Adi = "Boncuk";
    HayvanBilgisiGoster(kedi);
}
```

Senaryo: Çalışan Maaş Hesapla

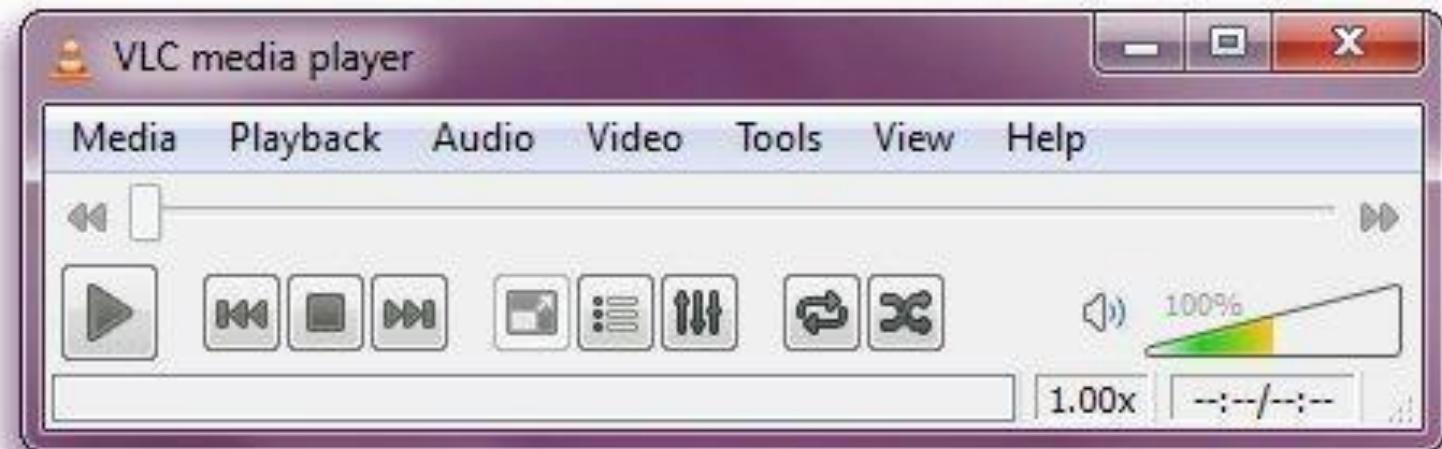
DeepBlue şirketinde 3 tip çalışan bulunmaktadır: Standart, Muhasebeci ve Satışçı. Her çalışan İsim ve Maaş bilgisine sahiptir. Tüm çalışanlar baz olarak, aylık çalışıkları 21 iş günü karşılığı asgari ücretin 2 katı maaş almaktadır. Muhasebeciler o ay yaptıkları her fazla mesai için gün başına (Maaş/21) tutarında mesai ücreti almaktadırlar. Satışçılar ise fazla mesai ücreti almayıp, o ay gerçekleştirdikleri satışın %10'u kadar satış primi almaktadırlar.

- Veli Gümüş standart bir çalışandır.
- Ahmet Demir Muhasebeci olup, bir ay boyunca 3 gün fazla mesai yapmıştır.
- Mehmet Çelik Satışçı olup, bir ay boyunca 1000 TL'lik satış yapmıştır.
- Asgari ücret 2000 TL olduğuna göre her üç personelin bir aylık maaşlarını hesaplayınız.
- DeepBlue firmasına Ahmet Demir, Mehmet Çelik ve Veli Gümüş isimli kullanıcıları ekleyelim. Firmanın o ayki çalışanlarına ödemesi gereken toplam maaşı bulalım.
- **Not:** Çok biçimlilik ve soyut sınıf kullanınız.

Soyut Sınıflar Özeti

- Soyut sınıflar, birden fazla child (yavru) sınıf için ortak özelliklerin ve/veya metodların ihtiyaç olduğu durumlarda **ortak kısımları toplama amacı** ile kullanılabilirler.
- Soyut sınıf **kullanımları önemlidir**, ancak zorunlu değildir.
- Soyut sınıfın **nesne türetilemez**.
- Soyut sınıflar **private olamazlar**.
- Soyut sınıflar **sealed (mühürlü)** olamazlar.

Arayüzler



Arayüzler (devam...)

- Bazı Nesne Yönelimli Programlama dilleri, bir yavru (child) sınıfının **birden fazla ana sınıfın özelliklerini ve davranışlarını barındırmamasına** izin vermektedir.
- Birden fazla sınıfın özelliklerini ve davranışlarını miras alma işlemeye **çoklu kalıtım (multiple inheritance)** denmektedir.
- Çoklu kalıtım **zor bir kavramdır** ve kullanıldığında ***çok fazla hata*** ile karşılaşılmaktadır.
- En açık örneği ise **parametre alan kurucu metoda** sahip **iki ayrı sınıfın özelliklerini barındırıiyorsa** hangisinin kurucu olarak kullanılacağıdır.
 - **Hangi ana sınıf temel alınmalıdır?**

Arayüzler (devam...)

- Bütün bu sıkıntılarından dolayı çoklu kalıtım C# ta engellenmiştir. Fakat C# çoklu kalıtımıma **arayüz (interface)** adında bir alternatif sunmaktadır.
- Soyut sınıflar gibi **arayüzler**, herhangi bir sınıfın üyelerinin ya da metodlarının koleksiyonudur.
- Bu metodlar herhangi bir sınıf tarafından, arayüzün soyut metodlarının tanımı sağlandığı sürece (override edildiği sürece) kullanılabilirler.
- **Soyut sınıfların** içerisinde **bazı metodlar soyut olmayabilir**.
- Fakat bir **arayüzün** içerisindeki **tüm metodlar soyut olmalıdır**.

Arayüzler (devam...)

- Bir arayüz tanımlanırken **interface** anahtar kelimesi kullanılır.
- **i** Arayüz adlandırmaları, başına 'I' harfi eklenerek sonuna da "-able / -ebilen" eki getirilerek yapılır.

```
interface ICalisabilen
{
    string Calis();
}
```

- Herhangi bir sınıf **ICalisabilen** arayüzüünü implemente ettiğinde, ayrıca **geriye string ifadesi döndüren Calis() metodunu** da içermiştir.

Örnek 1: ICalisabilen Arayüzü

- **ICalisabilen** adında, string Calis() metoduna sahip arayüzü yaratınız.
- Yarattığınız bu arayüzü **Ogretmen**, **Mudur** ve **Satisci** sınıflarına implemente ediniz.
- Calis() metodu sınıflara göre geriye geleceği değerler şu şekilde düzenlenmelidir:

Sınıf Adı	Calis() Metodunun Geriye Dönceği İfade
Ogretmen	«Öğretirim...»
Mudur	«İdarecilik yaparım...»
Satisci	«Satış yaparım...»

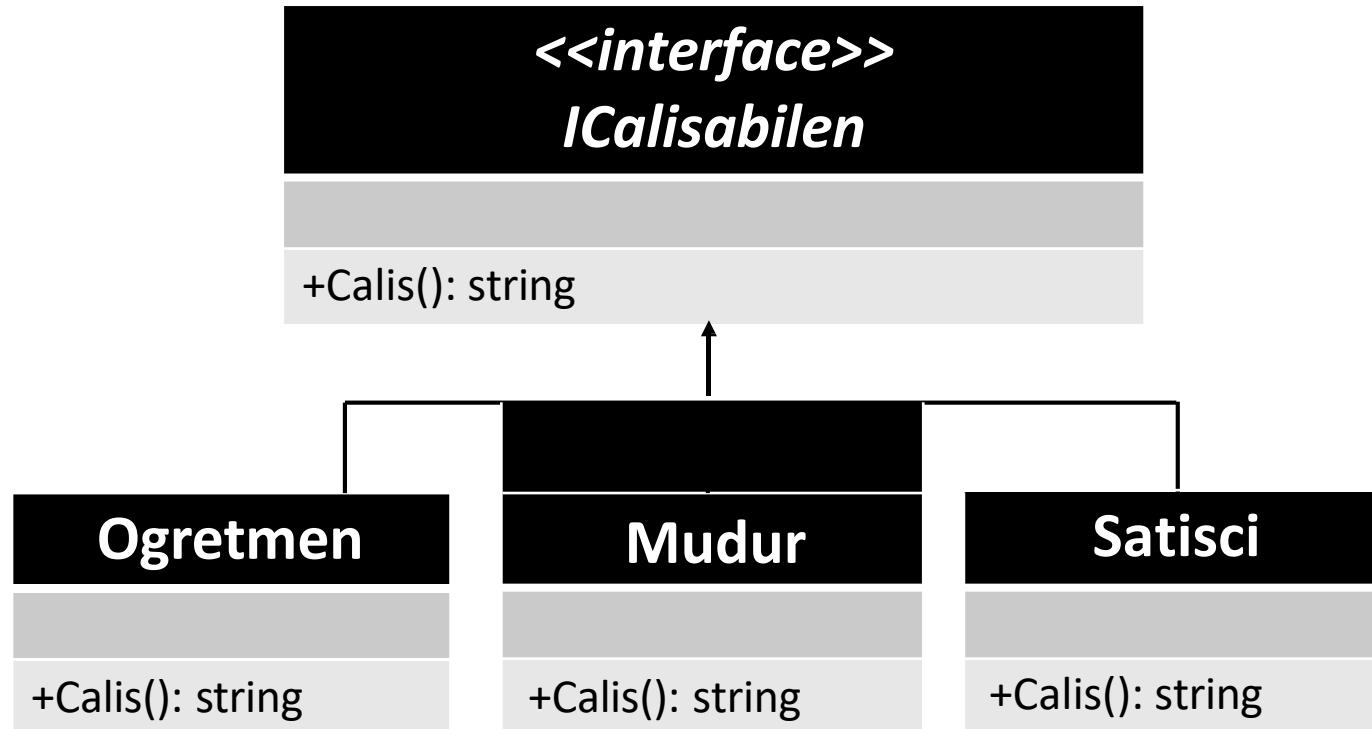
Örnek 1: ICalisabilen Arayüzü

- **ICalisabilen** arayüzüünü implemente edelim (gerçekleştirimini yapalım / kod parçalarını oluşturalım / program satırlarını yazalım):

```
public interface ICalisabilen
{
    string Calis();
}
```

Örnek 1: ICalisabilen Arayüzü

- **Senaryo1:** UML diyagramına bakarak implementasyon yapalım, test kodunu yazalım.



Örnek 1: ICalisabilen Arayüzü

```
public class Satisci : ICalisabilen
{
    3 references
    public string Calis()
    {
        return "Satış yaparım...";
    }
}
```

Senaryo1

```
public class Mudur : ICalisabilen
{
    3 references
    public string Calis()
    {
        return "İdarecilik yaparım...";
    }
}
```

```
public class Ogretmen : ICalisabilen
{
    3 references
    public string Calis()
    {
        return "Öğretirim...";
    }
}
```

```
private void Form1_Load(object sender, EventArgs e)
{
    Ogretmen ogretmen = new Ogretmen();
    Mudur mudur = new Mudur();
    Satisci satisci = new Satisci();

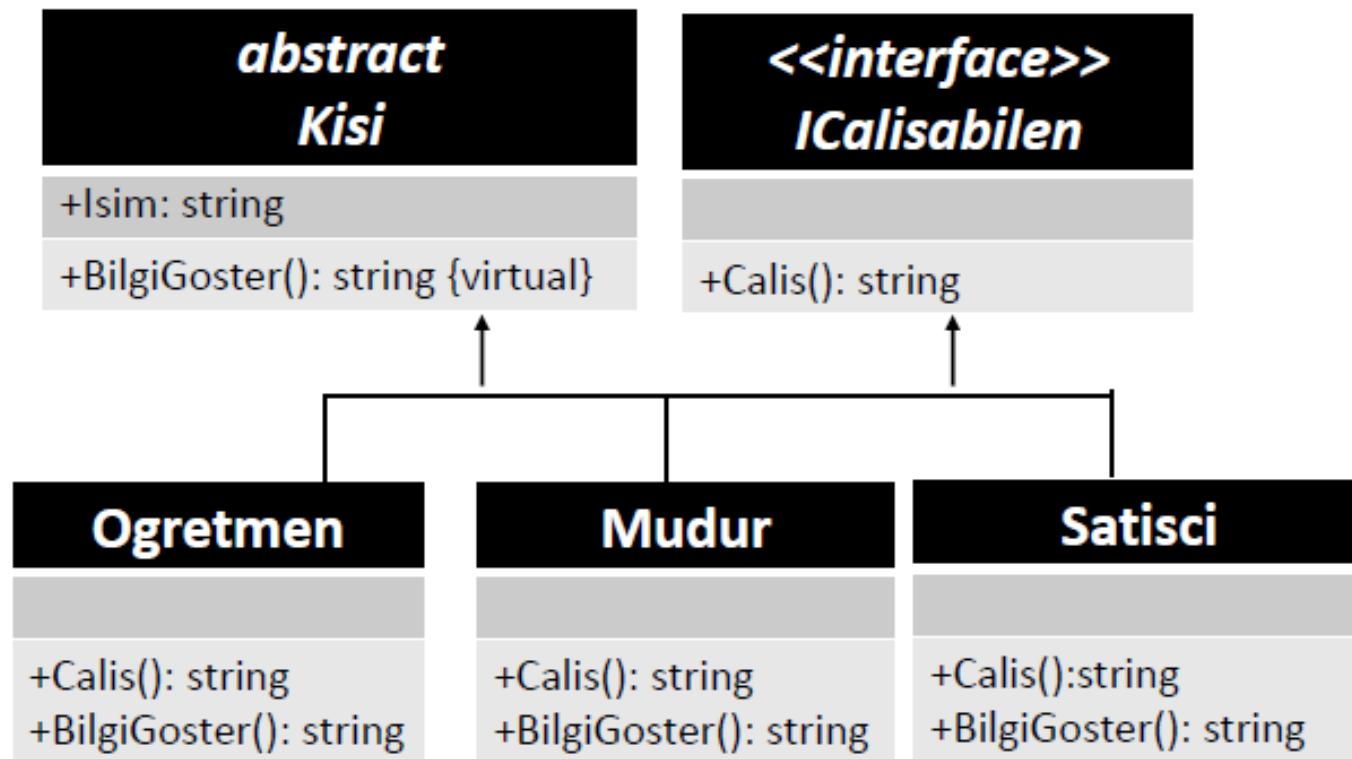
    MessageBox.Show(ogretmen.Calis());
    MessageBox.Show(mudur.Calis());
    MessageBox.Show(satisci.Calis());
}
```

Arayüzler (devam...)

- ICalisabilen arayüzü her implemente edildiğinde, Calis() metoduna bir **gövde** tanımlanmalıdır.
- Calis() metodu tanımlandığı sınıfı göre **işlem yapar**.
- Bu işlem **polimorfizm** yanı **çok biçimlilik**tir.
- Arayüzlerden nesne yaratılamaz.
- Soyut sınıfların arayüzlerden farkı soyut olmayan metodlar barındırıbmeleridir.
- Arayüzlerin barındırdığı tüm metodlar **soyut olmalıdır**.

Örnek 1: ICalisabilen Arayüzü

- **Senaryo2:** UML diyagramına bakarak gerçekleştirim yapalım, test kodunu yazalım.



Örnek 1: ICalisabilen Arayüzü

```
public class Ogretmen : Kisi, ICalisabilen
{
    6 references
    public string Calis()
    {
        return "Öğretirim...";
    }
    6 references
    public override string BilgiGoster()
    {
        return base.BilgiGoster();
    }
}
```

```
public class Satisci : Kisi, ICalisabilen
{
    6 references
    public string Calis()
    {
        return "Satış yaparım...";
    }
    6 references
    public override string BilgiGoster()
    {
        return base.BilgiGoster();
    }
}
```

```
public class Mudur : Kisi, ICalisabilen
{
    6 references
    public string Calis()
    {
        return "İdarecilik yaparım...";
    }
    6 references
    public override string BilgiGoster()
    {
        return base.BilgiGoster();
    }
}
```

```
public abstract class Kisi
{
    1 reference
    public string Isim { get; set; }
    6 references
    public virtual string BilgiGoster()
    {
        return "İsim: " + Isim;
    }
}
```

Senaryo2

Örnek 1: ICalisabilen Arayüzü

Senaryo2

```
private void Form1_Load(object sender, EventArgs e)
{
    Ogretmen ogretmen = new Ogretmen();
    ogretmen.Isim = "Ahmet Demir";
    Mudur mudur = new Mudur();
    mudur.Isim = "Mehmet Çelik";
    Satisci satisci = new Satisci();
    satisci.Isim = "İsmet Gümüş";

    MessageBox.Show(ogretmen.BilgiGoster() + " - " + ogretmen.Calis());
    MessageBox.Show(mudur.BilgiGoster() + " - " + mudur.Calis());
    MessageBox.Show(satisci.BilgiGoster() + " - " + satisci.Calis());
}
```

Arayüzler (devam...)

*Bir sınıf sadece **tek sınıfın türetilebilir** fakat
birden fazla arayüzü gerçekleştirebilir.*

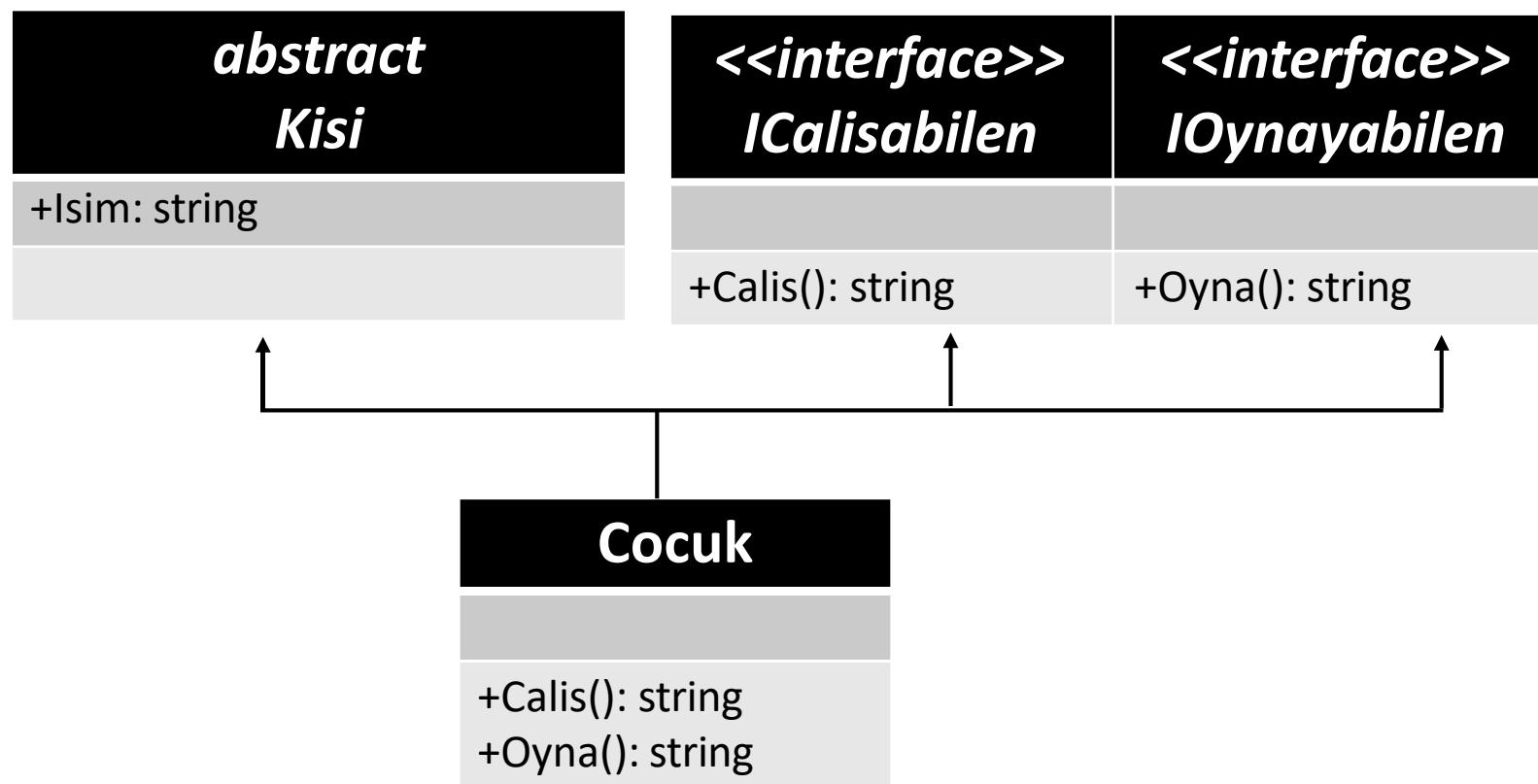
- Örneğin; **Cocuk** sınıfını ele aldığımızda, **Kisi** sınıfından türetilebilir ve aynı zamanda iki arayüzü de gerçekleştirebilir: **ICalisabilen** ve **IOynayabilen**.

class Cocuk: Kisi, ICalisabilen, IOynayabilen

- Arayüzler bir **sözleşme** olarak düşünülebilir.
- Bir sınıf bu arayüzü implemente ediyorsa sözleşmeye uymak zorundadır.

Örnek 2: Çoklu Arayüz Implementasyonu

- UML diyagramına bakarak implementasyon yapalım, test kodunu yazalım.



Örnek 2: Çoklu Arayüz Implementasyonu

```
public abstract class Kisi
{
    0 references
    public string Isim { get; set; }
}
```

```
interface IOynayabilen
{
    1 reference
    string Oyna();
}
```

```
interface ICalisabilen
{
    1 reference
    string Calis();
}
```

```
public class Cocuk : Kisi, ICalisabilen, IOynayabilen
{
    1 reference
    public string Calis()
    {
        return "Ders çalışırıım...";
    }
    1 reference
    public string Oyna()
    {
        return "Ders oynarııım...";
    }
}
```

Arayüzler (devam...)

Başlangıçta **ne zaman arayüz (Interface) ne zaman soyut sınıf (Abstract Class) kullanılacağına karar vermek zordur.** Abstract bir sınıfın bütün metodlarını **abstract yaparak** onu bir **interface** gibi **kullanabiliriz.**

- Tipik olarak, child sınıflara *bazı özelliklerin ve metodların aktarılması isteniyor* ve bunlardan **bazlarının ezilmesi gerekiyorsa soyut sınıflar** kullanılır.
- Her metod ezilmesi zorunluysa, **arayüz** oluşturulur.

Arayüzler (devam...)

Nesnelerin daha spesifik bir örneğini oluşturmak istiyorsak soyut sınıflar, davranışlarının benzemesini istiyorsak arayüzler yaratılır.

- **Abstract** sınıfların genellikle **IS-A (dır, dir)** ilişkilerinde, kalıtım(inheritance) özelliğini kullanarak kod tekrarını azaltmak için kullanılır.
- **Interface** sınıfların ise daha çok **CAN-DO (yapabilir..)** tarzı ilişkilerde değişen kavramları uygulamadan soyutlamak için kullanılır.

Arayüzler (devam...)

```
1 reference
interface IDoJob
{
    1 reference
    void DoJob1();
    1 reference
    void DoJob2();
    1 reference
    void DoJob3();
}
```

```
public class Worker : IDoJob
{
    1 reference
    private void Job1InterProcess()
    {
        //Todo
    }
    1 reference
    public void DoJob1()
    {
        Job1InterProcess();
    }
    1 reference
    public void DoJob2()
    {
        throw new NotImplementedException();
    }
    1 reference
    public void DoJob3()
    {
        throw new NotImplementedException();
    }
}
```

Problem:

- DoJob2 ve DoJob3 fonksiyonlarının gerçekleştirimi yapılmamış.
- Worker sınıfının sadece DoJob1'e ihtiyacı var.

Çözüm:

- SOLID tasarım prensipleri
- Interface Segregation (ISP)
- Nesneler asla ihtiyacı olmayan metotları içeren interface'leri implemente etmeye zorlanmamalıdır.