

C# ile Nesneye Dayalı Programlama

İçerik

C# ile Nesneye Dayalı Programlama

Yapılandırıcılar (Constructors)

Örnek 1

Birden Fazla Yapılandırıcı Oluşturmak

Örnek 2

Dizi Parametrelili Yapılandırıcı Oluşturmak

Örnek 3

Statik Yapılandırıcı (Static Constructors)

Örnek 4

Statik Metotlar (Static Methods)

Örnek 5

Statik Sınıflar (Static Classes)

Örnek 6

Statik Sınıf İçerisinde Başka Sınıflar Oluşturmak

Örnek 7

İç İç Statik Sınıflar (**Static Nested Classes**)

Örnek 8

Yıkıcılar (**Destructors**)

Örnek 9

Örnek 9-2

New Komutu

Örnek 11

This Komutu

Örnek 12

Sadece Okunabilir Özellik Tanımlamak (ReadOnly Property)

Örnek 13

Sadece Yazılabilir Özellik Tanımlamak (WriteOnly Property)

Örnek 14

Yazılabilen ve Okunabilen Özellik Tanımlamak

Örnek 15

Indexer Oluşturmak

Örnek 16

String Indexli Indexer Oluşturmak

Örnek 16

Typeof Komutu

Örnek 18

Sınıf Değerlerini Dosyaya Yazdırmak

Örnek 19

Sınıf İçerisinde Fonksiyonlar Oluşturmak

Örnek 20

Parametre İçermeyen Fonksiyon Tanımlamak

Örnek 20

Parametrelili Fonksiyon Tanımlamak

Örnek 20
Dizi Parametrelili Fonksiyon Tanımlamak
Örnek 20
Birden Fazla Parametrelili Fonksiyon Tanımlamak
Örnek 20
Fonksiyonlarda Dizi Değişken Değeri Döndürmek
Örnek 20
Metotların Aşırı Yüklenmesi
Örnek 21
Sınıf İçerisinde Prosedür Oluşturmak
Örnek 22
Parametre İçermeyen Prosedür
Örnek 22
Parametrelili Prosedür
Örnek 22
Ref Bildirili Prosedürler
Örnek 22
Out Bildirili Prosedür
Örnek 22
Kalıtım
Örnek 27
Örnek 27-2
Private Bildiri Yapmak
Örnek 27
Public Bildiri Yapmak
Örnek 27
Internal Bildiri Yapmak
Örnek 27
Protected Bildiri Yapmak
Örnek 27
Partial Class Bildirisi
Örnek 27
Base Komutu
Örnek 27
Virtual Ve Override Metod Tanımlamak
Örnek 34
Override Metod Tanımlamak
Örnek 34
Değişken Tanımlarken Farklı Class'lar Kullanmak
Örnek 35
Abstract Class'lar
Örnek 36
Örnek 36-2
Interface
Örnek 37
Örnek 37-2
Delege Tanımlamaları
Örnek 39
Delege İle Bizden Fazla Metodu Aynı Anda Çağırarak

Örnek 39

Delege İle Kontrollere Yordam Belirlemek

Örnek 40

Event Oluşturmak

Örnek 41

Ad Alanları (Namespaces)

Örnek 48

İç İçe Ad Alanları (Nested Namespaces)

Örnek 48

Yapı (Struct)

Örnek 50

Yapı İçerisinde Metot Oluşturmak (Methods in Struct)

Örnek 50

Yapı İçerisinde Başka Bir Yapı Oluşturmak (Nested Struct)

Örnek 50

Yapılarda Yapılandırıcı (Constructors in Struct)

Örnek 50

Yapılarda Kalıtım

Örnek 55

Generic İfadeler

Örnek 51

Generic Sınıflar

Örnek 52

Generic Sınıflar İçerisindeki Static Metodları İşletmek

Örnek 52

Generic Metodlar (Generic Methods)

Örnek 54

Kaynakça

Kitap

Web Sayfaları

Blog Yazıları

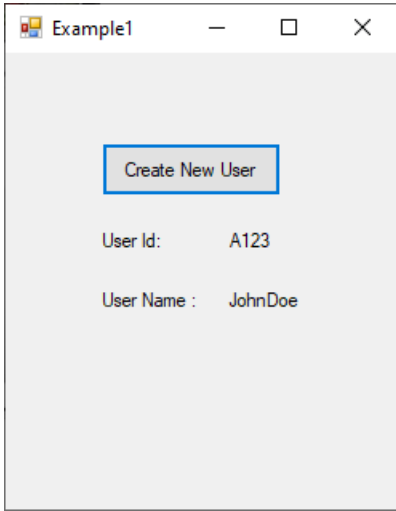
Yapılandırıcılar (Constructors)

Yapılandırıcı (constructor), bir nesne oluşturulduğu anda bu nesneyi ilk kullanıma hazırlar. Yapılandırıcı kendi sınıfı ile aynı isme sahiptir ve açık bir dönüş tipi yoktur.

Genellikle, bir sınıf tarafından tanımlanan değişkenlere ilk değer atarken kullanılır.

Örnek 1

Bu uygulamada butona basıldığında User sınıfından bir nesne oluşturulmakta ve yapılandırıcı kullanılarak nesnenin değişkenlerine veri atandıktan sonra bu veriler ekrana yazılmaktadır.



Sınıf içerisinde tanımlanması:

```
public class User
{
    public string userId;
    public string userName;

    public User(string userId, string userName)
    {
        this.userId = userId;
        this.userName = userName;
    }
}
```

Nesne oluşturulurken çağırılması:

```
User newUser = new User("A123", "JohnDoe");
```

Uygulamanın Kaynak Kodları: [Example1](#)

Birden Fazla Yapılandırıcı Oluşturmak

Bir nesnenin, parametrelerine göre farklı sürümlerinin oluşturulmasıdır. Hangi yapılandırıcının çalıştırılacağı parametrenin sayısına ve tipine bağlıdır.

Örnek 2

Bu uygulamada basılan butona göre kullanıcı verisi ekrana yazdırılmaktadır.

Sınıfta iki adet yapılandırıcı bulunmaktadır. Parametre almayan yapılandırıcı ile oluşturulan nesnede John Doe, parametre ile oluşturulan nesnede kullanıcıdan alınan veriler ekrana yazdırılacaktır. Nesnenin hangi yapılandırıcı ile oluşturulacağı butonlara bağlıdır.

```
public class User
{
    public string userName;
    public string userSurname;

    public User()
    {
        this.userName = "John";
        this.userSurname = "Doe";
    }

    public User(string userName, string userSurname) {
        this.userName = userName;
        this.userSurname = userSurname;
    }
}
```

Parametrelili Yapılandırıcı ile oluşturulan nesne.

Parametresiz Yapılandırıcı ile oluşturulan nesne.

Uygulamanın Kaynak Kodları: [Example2](#)

Dizi Parametrelili Yapılandırıcı Oluşturmak

Yapılandırıcılar parametre olarak dizileri alabilirler. Dinamik dizi için ArrayList sınıfı kullanılabilir.

Örnek 3

Bu örnekte kullanıcıdan alınan isimler bir listeye eklenmekte ve eklenen ilk ve son isimler ekrana yazdırılmaktadır.

Main sınıfının yapılandırıcısı parametre olarak aldığı diziyi, dizinin ilk elemanını ve dizinin son elemanını nesnenin içindeki değişkenlere atar.

```
class Main
{
    private ArrayList UserList;
    private string firstName;
    private string lastName;

    public Main(ArrayList Users)
    {
        this.UserList = Users;
        this.firstName = Users[0].ToString();
        this.lastName = Users[this.UserList.Count - 1].ToString();
    }

    public string getFirstUser()
    {
        return firstName;
    }

    public string getLastUser()
    {
        return lastName;
    }
}
```

Main sınıfından dizi parametrelili nesne oluşturmak.

```
ArrayList Users = new ArrayList(); //dizi oluşturuldu
Users.Add("Nuri Melih Sensoy"); //diziye eleman eklendi

var MainClass = new Main(this.Users); //nesne yaratıldı
```

Uygulamanın Kaynak Kodları: [Example3](#)

Statik Yapılandırıcı (Static Constructors)

Bir yapılandırıcı başına 'static' eklenerek statik olarak tanımlanabilir. Statik yapılandırıcı, sınıftan yaratılmış nesneye değil de sınıfın niteliklerine başlangıç değeri atamak için kullanılır. İlk örnek nesne oluşturulmadan otomatik olarak çağrılır.

- Statik bir Oluşturucu erişim değiştiricileri almaz veya parametrelere sahip değildir.
- Bir sınıf yalnızca bir statik oluşturucuya sahip olabilir.
- Statik oluşturucular devralınamaz veya aşırı yüklenemez.

- Statik bir Oluşturucu doğrudan çağrılmaz ve yalnızca ortak dil çalışma zamanı (CLR) tarafından çağrılabilir. Otomatik olarak çağrılır.
- Statik bir Oluşturucu çalışma zamanı boyunca ikinci bir kez çağrılmaz.

Örnek 4

```
Static Constructor
Non-Static Constructor
Non-Static Constructor
Value: 143
Value2: 100

Value: 143
Value2: 200
Press any key to close
```

Program çıktısında görüldüğü gibi normal yapılandırıcıdan önce ve yalnızca bir kere çalıştırılmaktadır.

```
class SomeClass
{
    static int value;
    int value2;

    static SomeClass()
    {
        Console.WriteLine("Static Constructor");
        value = 143;
    }

    public SomeClass(int val)
    {
        Console.WriteLine("Non-Static Constructor");
        value2 = val;
    }

    public void printData()
    {
        Console.WriteLine("Value: " + value);
        Console.WriteLine("Value2: " + value2);
    }
}

class Program
{
    static void Main(string[] args)
    {
        SomeClass MyClass1 = new SomeClass(100);
        SomeClass MyClass2 = new SomeClass(200);
    }
}
```

```
MyClass1.printData();
Console.WriteLine("");
MyClass2.printData();

//Console Wait
Console.WriteLine("Press any key to close");
Console.ReadKey();
}
}
```

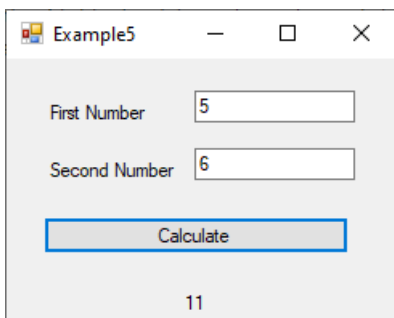
Uygulamanın Kaynak Kodları: [Example4](#).

Statik Metotlar (Static Methods)

Metotlar başına 'static' eklenerek statik olarak tanımlanabilirler. Statik olarak tanımlanan bir metot nesne oluşturulmadan sadece bulunduğu sınıfın adı kullanılarak çağırılabilir.

Örnek 5

Bu uygulama basit bir toplama işlemi yapılmaktadır.



Statik metot tanımlama :

```
class MyClass
{
    private static int result = 0;

    public static int sum(int a, int b)
    {
        return a + b;
    }

    public static void storeResult(int val) {
        result = val;
    }

    public static int getResult()
    {
```

```
        return result;
    }

}
```

Statik metot kullanma :

```
MyClass object1 = new MyClass();
int result = MyClass.sum(int.Parse(textBox1.Text), int.Parse(textBox2.Text));
MyClass.storeResult(result);
```

Uygulamanın Kaynak Kodları: [Example5](#)

Statik Sınıflar (Static Classes)

Sınıflar başına 'static' eklenerek statik olarak tanımlanabilirler. Temelde statik olmayan sınıflar ile aynıdır, ancak bir fark vardır: statik bir sınıfın nesnesi oluşturulamaz. Diğer bir deyişle, new komutu kullanılamaz. Üyelerine sadece sınıf adının kendisi kullanılarak erişilebilir. Yalnızca statik üyeleri içerir.

Örnek 6

Statik sınıf tanımlama:

```
static class Operation
{
    public static int sum(int a, int b) {
        return a + b;
    }
}
```

Statik sınıf üyelerine erişim:

```
int result = Operation.sum(5, 6);
result = Operation.sum(int.Parse(textBox1.Text), int.Parse(textBox2.Text));
```

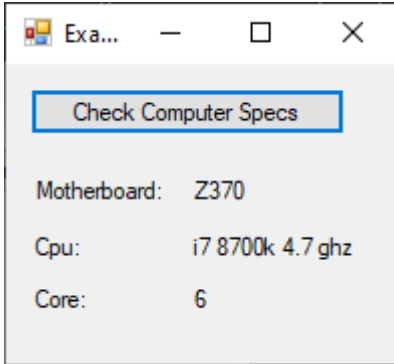
Uygulamanın Kaynak Kodları: [Example6](#)

Statik Sınıf İçerisinde Başka Sınıflar Oluşturmak

Statik bir sınıf içerisinde statik olmayarak tanımlanabilen tek yapı sınıflardır. Benzer iş yapan sınıfları iç içe tanımlamak sınıfların mantıksal olarak gruplanmasına olanak sağlar.

Örnek 7

Bu uygulama bilgisayar özelliklerini gösteren bir uygulamayı taklit etmektedir.



Motherboard Sınıfı :

```
static class Motherboard
{
    public static string Model = "Z370";
    public class CPU{
        public string Model = "i7 8700k";
        public string Speed = "4.7 ghz";
        public int Core = 6;
    }
}
```

Dış Sınıfın Elemanlarına Erişim :

```
Motherboard.Model = "X570";
```

İç Sınıfın Elemanlarına Erişim :

```
Motherboard.CPU Processor = new Motherboard.CPU();
Processor.Model = "Ryzen 9 3900X";
Processor.Speed = "4.6 ghz";
Processor.Core = 12;
```

Görüldüğü gibi dış sınıf statik olmasına rağmen iç sınıftan nesne oluşturulabiliyor.

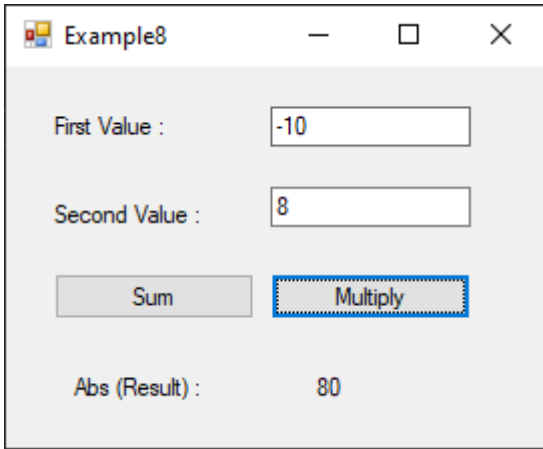
Uygulamanın Kaynak Kodları: [Example7](#)

İç İçe Statik Sınıflar (Static Nested Classes)

Statik sınıflar iç içe tanımlanabilirler. Sınıfların iç içe tanımlanması hiyerarşi sağlar.

Örnek 8

Bu uygulama girilen iki sayı ile toplama veya çarpma işlemi yaparak sonuçları mutlak değer olarak ekrana yazar.



İç İçe Statik Sınıf Tanımlama :

```
static class Operation
{
    public static class Utility
    {
        public static int absVal(int a)
        {
            if (a < 0) return a * -1;
            else return a;
        }
    }
    public static int sum(int a, int b)
    {
        return a + b;
    }
    public static int multiply(int a, int b)
    {
        return a * b;
    }
}
```

İç içe Statik Sınıf Kullanma :

```
int a = Operation.Utility.absVal(int.Parse(textBox1.Text));  
int b = Operation.Utility.absVal(int.Parse(textBox2.Text));  
int result = Operation.sum(a, b);
```

Görüldüğü gibi iki sınıfın da metotlarına sınıf ismiyle ulaşıyor.

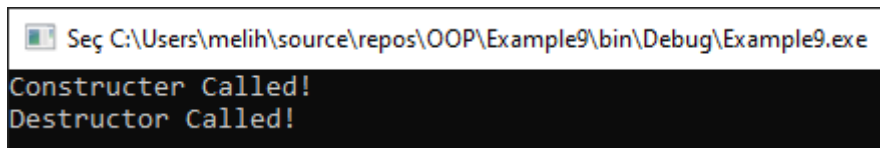
Uygulamanın Kaynak Kodları: [Example8](#)

Yıkıcılar (Destructors)

Bir nesnenin çöp toplayıcı tarafından tamamen yok edilmesinden hemen önce çağrılmak üzere bir metot tanımlamak mümkündür. Bu metoda yok edici (destructor) denir ve bir nesnenin kesinlikle sonlandırıldığını garanti etmek için kullanılabilir. Sınıf isminin başına tilda karakteri eklenerek tanımlanır.

Örnek 9

Uygulamanın Çıktısı :



```
Seç C:\Users\melih\source\repos\OOP\Example9\bin\Debug\Example9.exe  
Constructor Called!  
Destructor Called!
```

Örnek Sınıf :

```
public class MyClass  
{  
    public MyClass()  
    {  
        Console.WriteLine("Constructor Called!");  
    }  
  
    ~MyClass() {  
        Console.WriteLine("Destructor Called!");  
    }  
}
```

Sınıftan Nesne Oluşturma ve Mesajları Görme :

```
static void Main(string[] args)
{
    ClassInstance();
    GC.Collect(); //for seeing the destructor message

    Console.ReadLine();//preventing console from closing
}

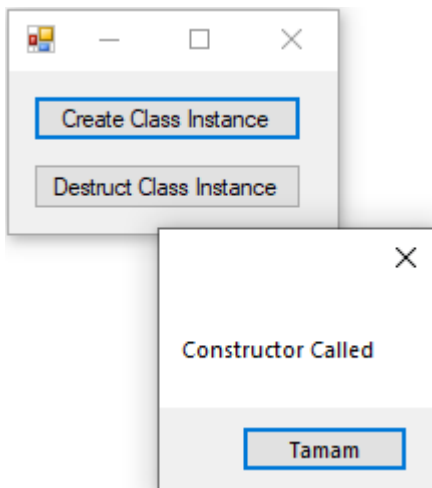
public static void ClassInstance() {
    MyClass MyClass1 = new MyClass();
}
```

Uygulamanın Kaynak Kodları: [Example9](#).

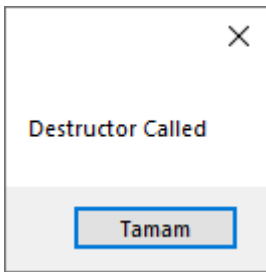
Örnek 9-2

Bu uygulama sınıf yapılandırıcısı ve yıkıcısının çağırıldığını kullanıcıya bir mesaj kutusu kullanarak göstermektedir.

"Create Class Instance" butonuna basıldığında sınıftan nesne oluşturulur ve kullanıcıya mesaj kutusu gösterilir.



"Destruct Class Instance" butonuna basıldığında veya pencere kapatıldığında nesne yok edilmeden önce kullanıcıya bir mesaj kutusu gösterir.



```
public class MyClass
{
    public MyClass() {
        MessageBox.Show("Constructor Called");
    }

    ~MyClass() {
        MessageBox.Show("Destructor Called");
    }
}
```

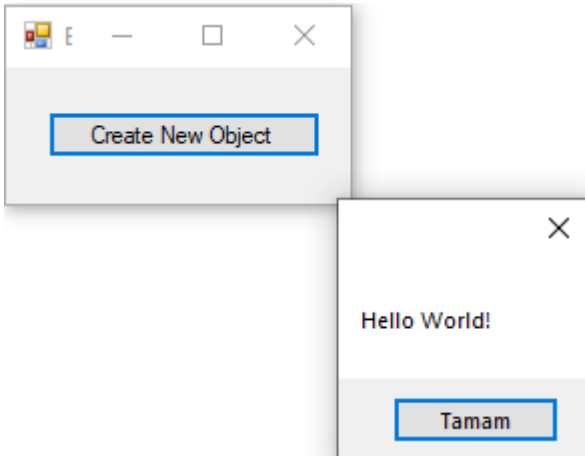
Uygulamanın Kaynak Kodları: [Example9-2](#)

New Komutu

Bir türden veya sınıftan yeni nesne üretmek için kullanılır. Bu yolla oluşturulan nesnenin Yapılandırıcısı çalışır.

Örnek 11

Bu uygulamada butona basıldığında yeni bir nesne oluşturulur ve sınıfın bir metodu çalıştırılır.



Sınıf :

```
public class MyClass
{
    public void helloWorld() {
        MessageBox.Show("Hello World!");
    }
}
```

Nesne Oluşturma ve Metot Çalıştırma:

```
MyClass MyClass1 = new MyClass();
MyClass1.helloWorld();
```

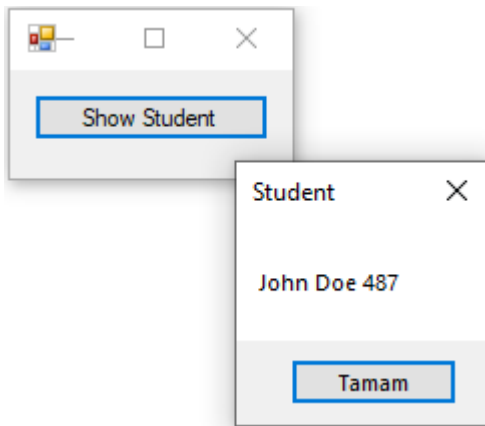
Uygulamanın Kaynak Kodları: [Example11](#)

This Komutu

"this" komutu çağırıldığı nesnenin referansını gösterir.

Örnek 12

Bu uygulama butona basıldığında ekrana bir öğrencinin bilgilerini getirir.



Burada görüldüğü üzere sınıfın Yapılandırıcı parametrelerinin isimleri sınıfın özellikleriyle aynı. Bu durum bir belirsizliğe yol açmaktadır. This komutu kullanılarak yerel değişken yerine nesnenin değişkenine erişim sağlanabilir.

```
public class Student
{
    private string fullName;
    private int studentId;
```

```
public Student(string fullName, int studentId)
{
    this.fullName = fullName;
    this.studentId = studentId;
}

public void showStudent()
{
    MessageBox.Show(this.fullName + " " + this.studentId, "Student");
}
}
```

Butona basıldığında çalışacak kod :

```
Student Student1 = new Student("John Doe", 487);
Student1.showStudent();
```

Uygulamanın Kaynak Kodları: [Example12](#)

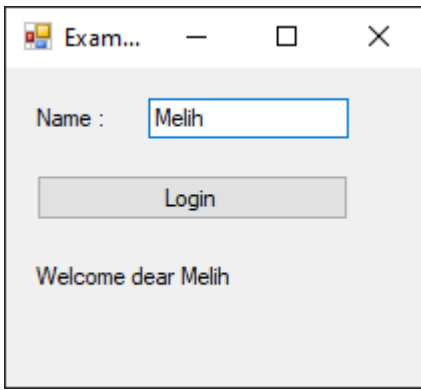
Sadece Okunabilir Özellik Tanımlamak (ReadOnly Property)

Bir sınıf içerisinde bulunan bazı özel alanlara dışarıdan kontrollü olarak ulaşılması istenebilir. Properties bu özel alanlar üzerinde okuma, yazma, hesaplama gibi işlemler yapılmasını sağlayan bir özelliktir.

Sadece okunabilir özellikler değiştirilemezler. Değişkenlere aktarılabilirler. Sadece GET bloğu barındırırlar.

Örnek 13

Bu örnekte kullanıcıdan alınan isim bilgisine göre ekrana bir karşılama cümlesi yazdırılmaktadır.



Burada sınıf içerisindeki 'fullName' alanındaki veriyi 'formalName' özelliği ile kontrol edilmekte. Özelliğe sadece GET bloğu eklediğimizden dolayı veriyi sadece okuyabiliriz. GET Bloğunun içinde yaptığımız işlemler sayesinde veri nesne içinde saklandığı ham hali yerine başına "Welcome dear " eklenmiş hali ile okunabilir. Bu sayede veriye kontrollü olarak ulaşılmış olur.

```
class Login
{
    private string fullName;

    public string formalName
    {
        get
        {
            return string.Format("Welcome dear {0}", this.fullName);
        }
    }

    public void setName(string name)
    {
        this.fullName = name;
    }
}
```

Veriyi Okuma :

```
Login LoginClass = new Login();
label3.Text = LoginClass.formalName;
```

Uygulamanın Kaynak Kodları: [**Example13**](#)

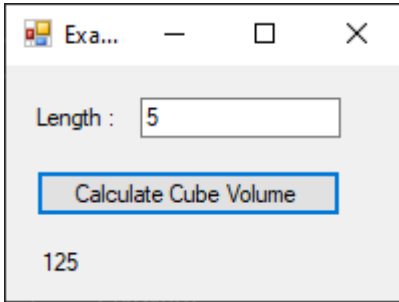
Sadece Yazılabilir Özellik Tanımlamak (WriteOnly Property)

Bir sınıf içerisinde bulunan bazı özel alanlara dışarıdan kontrollü olarak ulaşılması istenebilir. Properties bu özel alanlar üzerinde okuma, yazma, hesaplama gibi işlemler yapılmasını sağlayan bir özelliktir.

Sadece yazılabilir özellikler okunamazlar. Değişkenlere aktarılabilirler fakat içerikleri öğrenilemez. Sadece SET bloğu barındırırlar.

Örnek 14

Bu örnekte kullanıcıdan alınan kenar uzunluğu değerine göre bir küpün hacmi hesaplanmaktadır.



Burada sınıfın içerisindeki "height", "width" ve "depth" özellikleri "allDimensions" özelliği ile kontrol edilmektedir. Özellik sadece SET bloğu içerdiği için sadece yazılabilir halde tanımlanmış. Bu örnekteki gibi aynı değerın sınıftaki birden fazla alana yazılması gereken durumlarda kullanılması kolaylık sağlar.

```
class Cube
{
    public double height;
    public double width;
    public double depth;

    public double allDimensions
    {
        set
        {
            height = width = depth = value;
        }
    }

    public double getVolume()
    {
        return height * width * depth;
    }
}
```

```
}
```

Butona basıldığında işletilen kod bloğu :

```
Cube MyCube = new Cube();  
MyCube.allDimensions = double.Parse(textBox1.Text);  
label2.Text = MyCube.getVolume().ToString();
```

Görüldüğü gibi nesneye veri atanması işlemi "allDimensions" özelliği ile yapılıyor.

Uygulamanın Kaynak Kodları: [Example14](#).

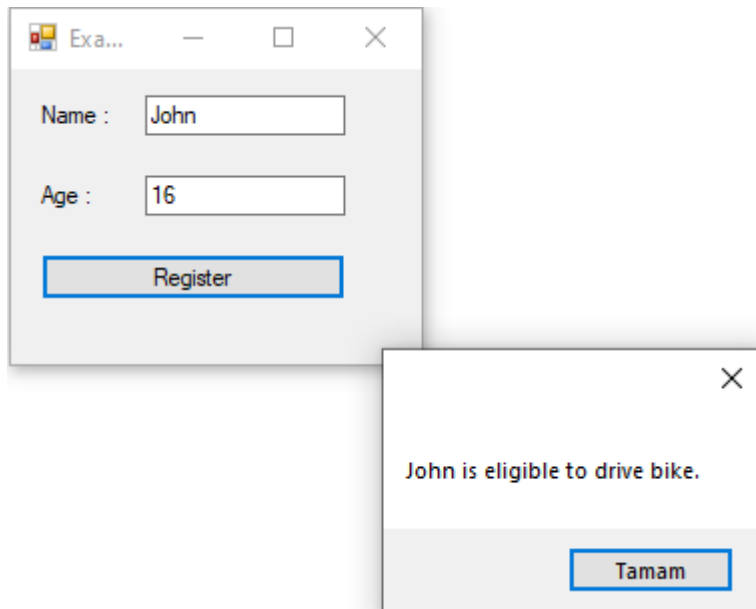
Yazılabilen ve Okunabilen Özellik Tanımlamak

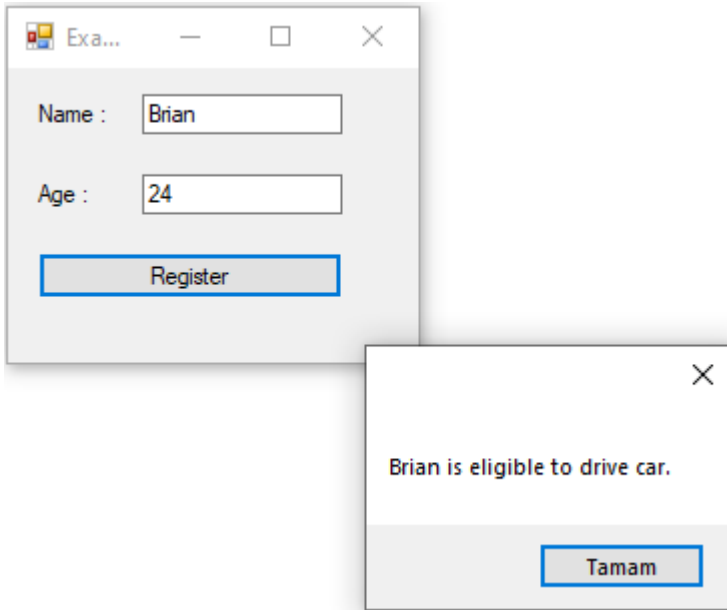
Bir sınıf içerisinde bulunan bazı özel alanlara dışarıdan kontrollü olarak ulaşılması istenebilir. Properties bu özel alanlar üzerinde okuma, yazma, hesaplama gibi işlemler yapılmasını sağlayan bir özelliktir.

Bu şekilde tanımlanan özellikler hem yazılıp hem okunabilirler. SET ve GET kod bloğu barındırırlar.

Örnek 15

Bu örnekte kullanıcıdan isim ve yaş bilgisi alınmakta, alınan yaş bilgisine göre de araba mı bisiklet mi sürebileceğini ekrana yazdırır.





Bu örnekte "`_age`" alanına erişim "`Age`" özelliği üzerinden sağlanmaktadır. Veri okunmak istendiğinde olduğu halinde döndürülmekte fakat yazılmak istendiğinde yaş değeri 18'den küçükse sınıfın "`eligible`" alanı "`false`" yapılmaktadır. Bu sayede veri yazıldığı anda hesaplama yapılmaktadır.

```
class Licence
{
    public string name { get; set; }
    public bool eligible = true;
    private int _age;

    public int Age
    {
        get { return _age; }
        set
        {
            if (value < 18)
            {
                this.eligible = false;
            }
            this._age = value;
        }
    }
}
```

Uygulamanın Kaynak Kodları: [**Example15**](#)

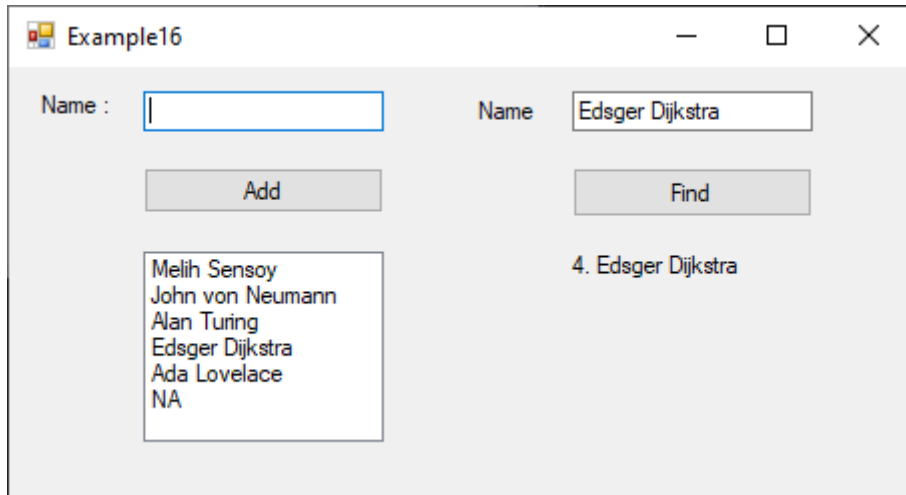
Indexer Oluşturmak

İndeksleyici, bir nesnenin tıpkı bir dizi gibi indekslenmesine imkan verir. Tanımlanışları özelliklere(properties) benzer.

```
<return type> this[<parameter type> index]
{
    get{
    }
    set{
    }
}
```

Örnek 16

Bu uygulamada kullanıcıdan alınan isimler bir listeye eklenmekte ve aralarında arama yapılmaktadır.



Burada "Users" sınıfı için bir indeksleyici tanımlanmıştır. İndeksleyicinin GET bloğunda sınır kontrolü yapılarak nesnenin içindeki "names" dizisinden "index" isimli değişkenin tuttuğu indisteki elemanı döndürülmekte, SET bloğunda ise yine sınır kontrolü yapılarak nesnenin içindeki "names" dizisinin istenilen indisine veri yazılmakta.

```
class Users
{
    private string[] names;
    private int size;

    public Users(int size)
    {
        this.size = size;
    }
}
```

```

        names = new string[size];
        for (int i = 0; i < size; i++)
            names[i] = "NA";
    }
    public string this[int index]
    {
        get
        {
            string tmp;

            if (index >= 0 && index <= size - 1)
            {
                tmp = names[index];
            }
            else
            {
                tmp = "";
            }
            return (tmp);
        }
        set
        {
            if (index >= 0 && index <= size - 1)
            {
                names[index] = value;
            }
        }
    }
}

```

Görüldüğü gibi "Users" sınıfının nesnesine indis kullanılarak veri yazılabilmekte.

```

Users UsersClass = new Users(6);
UsersClass[0] = "Melih Sensoy";
UsersClass[1] = "John von Neumann";
UsersClass[2] = "Alan Turing";
UsersClass[3] = "Edsger Dijkstra";
UsersClass[4] = "Ada Lovelace";

```

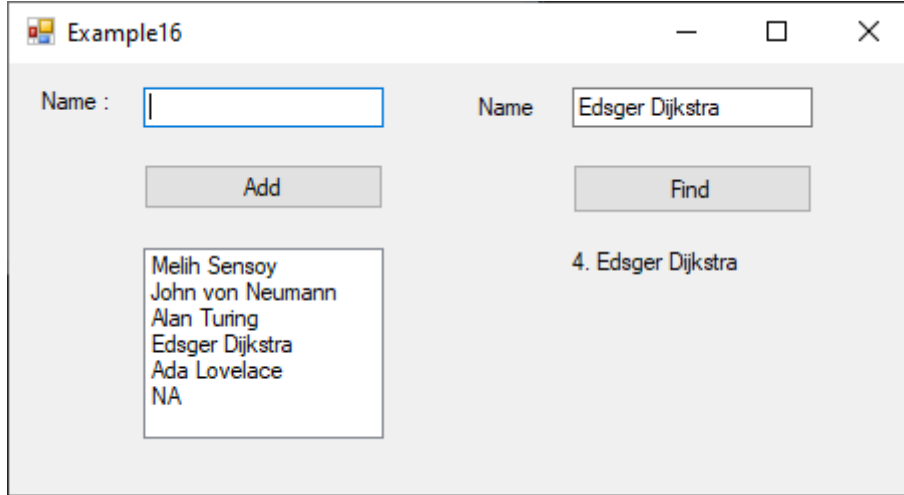
Uygulamanın Kaynak Kodları: **Example16**

String Indexli Indexer Oluşturmak

İndis numarası yerine string tipi indis kullanılır. İndisleyici aşırı yüklenerek tanımlanır.

Örnek 16

Bu uygulamada kullanıcıdan alınan isimler bir listeye eklenmekte ve aralarında arama yapılmaktadır.



Burada indeksleyici GET bloğunda string tipinde aldığı parametredeki elemanı "names" dizisinde arar, bulursa indisini bulamazsa -1 döndürür. Önceki örnekte yazılan indeksleyici aşırı yüklenerek tanımlanmıştır.

```
public int this[string name]
{
    get
    {
        int index = 0;
        while (index < this.size)
        {
            if (names[index] == name)
            {
                return index;
            }
            index++;
        }
        return -1;
    }
}
```

Bu şekilde kullanılır :

```
int foundIndex = UsersClass["Alan Turing"];
```

Burada eklenen liste içerisinde "Alan Turing" aranacak ve bulunursa bulunduğu indis, bulunamazsa -1 değeri "foundIndex" değişkenine atanacaktır.

Uygulamanın Kaynak Kodları: [Example16](#)

Typeof Komutu

Bir değişkenin veya nesnenin derleme zamanında bilinen tip bilgilerinin Type nesnesi olarak elde edilmesini sağlar.

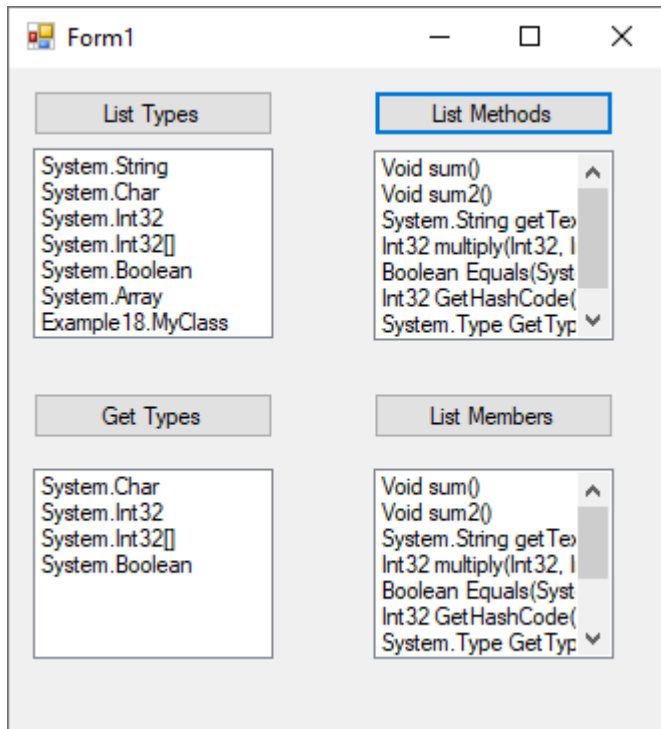
GetType() : Programın çalışma sırasında bir nesnenin tipini elde eder.

GetMethods() : Elde edilen Type nesnesini kullanarak tip tarafından desteklenen metotları getirir.

GetMembers(); Elde edilen Type nesnesini kullanarak tipin sahip olduğu üyeleri getirir.

Örnek 18

Bu uygulamada çeşitli sınıf ve değişkenlerin tipleri ekrana yazdırılır.



Örnek Sınıf:

```
public class MyClass
{
    private string text = "Hello";
    public int number = 15;
}
```

```

        public static int number2 = 10;

        public static void sum()
        {
            number2 += number2;
        }

        public void sum2()
        {
            number += number;
        }

        public string getText()
        {
            return this.text;
        }

        public int multiply(int a, int b)
        {
            return a * b;
        }
    }

```

"List Types" butonunun çalıştırdığı kod bloğu :

```

Type a = typeof(string);

listBox1.Items.Add(a);
listBox1.Items.Add(typeof(char));
listBox1.Items.Add(typeof(int));
listBox1.Items.Add(typeof(int[]));
listBox1.Items.Add(typeof(bool));
listBox1.Items.Add(typeof(Array));
listBox1.Items.Add(typeof(MyClass));

```

Burada "typeof" komutu kullanılarak çeşitli veri tipleri ekrana yazdırılıyor.

"Get Types" butonunun çalıştırdığı kod bloğu :

```

char a = 'a';
int b = 32;
int[] c = { 0, 1, 2 };
bool d = false;

listBox2.Items.Add(a.GetType());
listBox2.Items.Add(b.GetType());
listBox2.Items.Add(c.GetType());
listBox2.Items.Add(d.GetType());

```

Burada "GetType" metodu kullanılarak değişkenlerin çalışma zamanı tipleri ekrana yazdırılıyor.

"List Methods" butonunun çalıştırdığı kod bloğu :

```
Type ClassType = typeof(MyClass);
MethodInfo[] methodList = ClassType.GetMethods();
foreach (MethodInfo Method in methodList)
{
    listBox3.Items.Add(Method);
}
```

"MyClass" sınıfının Type nesnesi "ClassType" değişkenine atanıyor. Atandıktan sonra "GetMethods" metodu kullanılarak ulaşılan metod listesi ekrana yazdırılıyor.

"List Members" butonunun çalıştırdığı kod bloğu :

```
Type ClassType = typeof(MyClass);
MemberInfo[] memberList = ClassType.GetMembers();
foreach (MemberInfo Member in memberList)
{
    listBox4.Items.Add(Member);
}
```

"MyClass" sınıfının Type nesnesi "ClassType" değişkenine atanıyor. Atandıktan sonra "GetMembers" metodu kullanılarak ulaşılan üye listesi ekrana yazdırılıyor.

Uygulamanın Kaynak Kodları: [Example18](#)

Sınıf Değerlerini Dosyaya Yazdırmak

Nesnenin durumunu gerektiğinde yeniden oluşturmak için kaydetmek önemlidir. Nesneleri kaydetmek, depolamak, bir veritabanı ya da bir dosyaya aktarmak için nesneyi bayt akışına dönüştürmek gerekir. Bu dönüştürme işlemine Serileştirme(Serialization) denir. Bu işlemin tersine yani bayt akışından tekrardan nesneyi oluşturma işlemine ise Seriyi Kaldırma(Deserialization) denir. Bu işlem için çeşitli yöntem ve standartlarda yapılabilmektedirler. Bunlar : İkili Serileştirme (Binary Serializaton), SOAP Serileştirme, JSON Serileştirme.

Örnek 19

Bu uygulama kullanıcıdan alınan verileri "Users.dat" isimli dosyaya kaydeder ve kaydedilen dosyadaki verileri kullanıcıya gösterir. İkili Serileştirme yöntemi kullanır.

Serileştirilecek sınıf:

```
[Serializable]
public class Users
{
    public string userName { get; set; }
    public string email { get; set; }
    public string country { get; set; }

    public Users(string userName, string email, string country) {
        this.userName = userName;
        this.email = email;
        this.country = country;
    }
}
```

Serileştirilmesi istenen sınıfa "[Serializable]" eklenmesi yeterlidir. Eğer sınıf içerisinde serileştirmenin dışında tutmak istenilen bir alan varsa "[NonSerialized]" kullanılmalıdır.

Serileştirme ve seriyi kaldırma işlemini yapan sınıf :

```
public static class SerializeWrapper
{
    public static string fileName = "DataFile.dat";

    public static void binarySerialize(Users data)
    {
        FileStream fs = new FileStream(fileName, FileMode.Create);

        BinaryFormatter formatter = new BinaryFormatter();
        try
        {
            formatter.Serialize(fs, data);
        }
    }
}
```

```

    }
    catch (SerializationException e)
    {
        Console.WriteLine("Failed to serialize. Reason: " +
e.Message);
        throw;
    }
    finally
    {
        fs.Close();
    }
}

public static Users binaryDeserialize()
{
    Users UsersClass = null;
    FileStream fs = new FileStream(fileName, FileMode.Open);
    try
    {
        BinaryFormatter formatter = new BinaryFormatter();

        UsersClass = (Users)formatter.Deserialize(fs);
        return UsersClass;
    }
    catch (SerializationException e)
    {
        Console.WriteLine("Failed to deserialize. Reason: " +
e.Message);
        throw;
    }
    finally
    {
        fs.Close();
    }
}
}

```

Serileştirme sınıfının kullanımı :

```

//Serialize
Users user = new Users("melih", "melih@sensoy.com", "Turkey");
SerializeWrapper.fileName = "Users.dat";
SerializeWrapper.binarySerialize(user);

//Deserialize
Users user = SerializeWrapper.binaryDeserialize();
label7.Text = user.userName;
label8.Text = user.email;
label9.Text = user.country;

```

Uygulamanın Kaynak Kodları: [Example19](#)

Sınıf İçerisinde Fonksiyonlar Oluşturmak

Sınıflar içerisinde istenilen işlemleri yapacak fonksiyonlar oluşturulabilir. Bu fonksiyonlar sınıfın içinde de dışında olduğu gibi tanımlanabilirler. Metotlarla eş anlamlıdır.

Örnek 20

Bu uygulama sınıfın içerisindeki bir değişkenin içeriğini değiştiren bir fonksiyona sahiptir.

"MyClass" sınıfı :

```
class MyClass
{
    public string text;

    public void helloWorld()
    {
        text = "Hello World!";
    }
}
```

Görüldüğü üzere sınıfın içinde "helloWorld" isimli bir değişken tanımlanmış. Bu değişken çağırıldığında sınıfın "text" isimli alanının içeriğini "Hello World!" yapmakta ve herhangi bir değer döndürmemektedir.

Uygulamanın Kaynak Kodları: [Example20](#)

Parametre İçermeyen Fonksiyon Tanımlamak

Fonksiyonlar herhangi bir işlemde kullanmak üzere veriler alabilirler. Bu verilere parametre denir. Bu tarz bir fonksiyon yaratmak isteniyorsa fonksiyonda parantez içi boş bırakılmalıdır.

Örnek 20

Bu uygulama sınıfın içerisindeki bir değişkenin içeriğini değiştiren bir fonksiyona sahiptir.

"MyClass" isimli sınıf ve "resetText" isimli parametre içermeyen fonksiyon :

```
class MyClass
{
    public string text;
    public void resetText()
    {
        text = " ";
    }
}
```

"resetText" isimli fonksiyon sınıfın içindeki "text" isimli alanın değerini boş karakter yapmaktadır.

Uygulamanın Kaynak Kodları: [Example20](#)

Parametrelili Fonksiyon Tanımlamak

Fonksiyonlar herhangi bir işlemde kullanmak üzere veriler alabilirler. Bu verilere parametre denir. Bu tarz bir fonksiyon yaratmak isteniyorsa parametreler fonksiyon tanımlanırken parantez içine yazılmalıdır. Parametre değişken tanımlanır gibi tanımlanmalıdır.

Örnek 20

Bu uygulama sınıfın içerisindeki bir değişkenin içeriğini değiştiren bir fonksiyona sahiptir.

"MyClass" isimli sınıf ve "setText" isimli parametrelili fonksiyon :

```
class MyClass
{
    public string text;
    public void setText(string text)
    {
        this.text = text;
    }
}
```

"setText" isimli fonksiyon parametre olarak aldığı değeri sınıfın içindeki "text" isimli alana yazar.

Uygulamanın Kaynak Kodları: [Example20](#)

Dizi Parametrelili Fonksiyon Tanımlamak

Fonksiyonlar herhangi bir işlemde kullanmak üzere veriler alabilirler. Bu verilere parametre denir. Bu parametreler dizi olabilirler. Fonksiyonlara dizi parametre göndermek bazı durumlarda çok kullanışlı olabilir.

Örnek 20

Bu uygulama parametre olarak aldığı kelimeleri cümle haline getiren bir fonksiyona sahiptir.

"MyClass" isimli sınıf ve "makeSentence" isimli dizi parametrelili fonksiyon :

```
class MyClass
{
    public string makeSentence(string[] words)
    {
        string sentence = "";
        foreach (string word in words)
        {
            sentence += " " + word;
        }

        return sentence;
    }
}
```

"makeSentence" isimli fonksiyon parametre olarak aldığı string tipindeki dizinin elemanlarını arasına bir boşluk koyarak cümle haline getirir ve string tipinde döndürür.

Uygulamanın Kaynak Kodları: [Example20](#)

Birden Fazla Parametrelili Fonksiyon Tanımlamak

Fonksiyonlar herhangi bir işlemde kullanmak üzere veriler alabilirler. Bu verilere parametre denir. Bazen birden fazla parametre alması çok faydalı olabilir. Bu gibi durumlarda parantez içerisinde her bir parametre arasına virgül koyarak oluşturulmalıdır.

Örnek 20

Bu uygulama iki string tipinde değeri karşılaştıran bir fonksiyona sahiptir.

"MyClass" isimli sınıf ve "compareStrings" isimli iki parametrelili fonksiyon :

```
class MyClass
{
    public bool compareStrings(string a, string b)
    {
        if (a == b)
        {
            return true;
        }
        return false;
    }
}
```

"compareStrings" isimli fonksiyon parametre olarak aldığı iki string tipindeki değeri karşılaştırır. Aynı değerde iseler "true" değeri, farklı değerde iseler "false" değeri döndürür.

Uygulamanın Kaynak Kodları: [Example20](#)

Fonksiyonlarda Dizi Değişken Değeri Döndürmek

Fonksiyonlar "void" ifadesiyle tanımlanmadıysa değer döndürürler. Fonksiyonların dizi değişken döndürmesi bazı durumlarda faydalı olabilir. Böyle bir fonksiyon tanımlamak için fonksiyon döndürülen değer tipinde tanımlanmalı ve içerisinde "return" ifadesi ile döndürülen değer belirtilmelidir.

Örnek 20

Bu örnek parametre olarak aldığı cümlemin kelimelerini dizi olarak döndüren bir fonksiyona sahiptir.

"MyClass" isimli sınıf ve "seperateWords" isimli bir parametre alıp, dizi döndüren fonksiyon :

```
class MyClass
{
    public string[] seperateWords(string sentence)
    {
        string[] words = sentence.Split(' ');
        return words;
    }
}
```

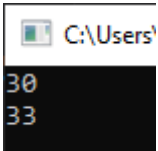
"seperateWords" isimli fonksiyon string tipinde aldığı cümleyi boşluklarına göre ayırarak cümledeki kelimeleri bir diziye ekler ve bu diziye döndürür.

Metotların Aşırı Yüklenmesi

Metotları aşırı yükleme, bir metodun farklı versiyonlarını oluşturmayı sağlar. Aynı sınıfın içindeki iki yada daha fazla metod parametre tipleri veya sayıları farklı olduğu sürece aynı ismi paylaşabilirler. Bu metotlar aynı veya farklı işi yapabilirler.

Örnek 21

Bu uygulama ekrana 10, 20 ve 10.5, 22.5 sayılarının toplamını ekrana yazdırmaktadır.



Sınıf ve ilgili metotlar :

```
class Calculation
{
    static public int sum(int a, int b)
    {
        return a + b;
    }

    static public double sum(double a, double b)
    {
        return a + b;
    }
}
```

Burada "sum" isimli metod aşırı yüklenmiştir. Bir versiyonu tam sayılarla işlem yaparken diğeri ondalıklı sayılarla işlem yapmaktadır.

Sınıfın kullanımı :

```
int a = 10;
int b = 20;

Console.WriteLine(Calculation.sum(a, b));

double c = 10.5;
double d = 22.5;

Console.WriteLine(Calculation.sum(c, d));
```

Görüldüğü üzere aynı metot ile iki farklı tipte toplama işlemi gerçekleştiriliyor.

Uygulamanın Kaynak Kodları: [Example21](#)

Sınıf İçerisinde Prosedür Oluşturmak

Prosedürler fonksiyonun değer döndürmeyen halidir. "void" tipinde tanımlanır ve "return" ifadesi içermez.

Örnek 22

Bu uygulama kullanıcıdan iki sayı almakta ve aldığı iki sayıyı toplayıp, tıklanılan butona göre sonuç üzerinde işlemler yapmaktadır.

The screenshot shows a Windows application window with the title 'Exam...'. Inside the window, there is a 'Hello World' button at the top. Below it is a section titled 'Sum Operation'. This section contains two input fields: 'Number 1' with the value '6' and 'Number 2' with the value '88'. Below the input fields are two buttons: 'Calculate and Show' and 'Calculate and Save'. At the bottom of the window, there is a display showing the calculation '6 + 88 = 94' and a 'Swap' button.

"helloWorld" isimli çalıştığında ekrana mesaj gösteren prosedür :

```
class Operation
{
    static public void helloWorld()
    {
        MessageBox.Show("Hello World!");
    }
}
```

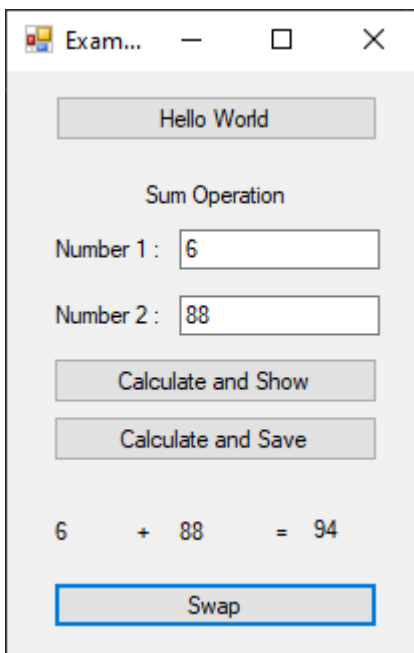
Uygulamanın Kaynak Kodları: [Example22](#)

Parametre İçermeyen Prosedür

Prosedürler, parantez içleri boş bırakılırsa parametresiz olarak isimlendirilirler.

Örnek 22

Bu uygulama kullanıcıdan iki sayı almakta ve aldığı iki sayıyı toplayıp, tıklanılan butona göre sonuç üzerinde işlemler yapmaktadır.



"Operation" isimli sınıfın içindeki "resetValues" isimli prosedür :

```
class Operation
{
    public int a;
    public int b;
```

```
private int _result;

public void resetValues()
{
    a = 0;
    b = 0;
    result = 0;
}

}
```

Bu prosedür nesnenin içindeki "a", "b" ve "_result" isimli değişkenlerin değerini 0 yapmaktadır. Parantez iç boş bırakıldığı için parametresiz olarak sınıflandırılır.

Uygulamanın Kaynak Kodları: [Example22](#)

Parametrelili Prosedür

Prosedürler işlemlerinde kullanmak üzere dışarıdan veri alabilirler. Bu tarz veriler parametre olarak isimlendirilir. Parantezin içinde, değişkenler ile aynı şekilde tanımlanır. Birden fazla eklenmek istendiğinde aralarına virgül koyularak yazılır.

Örnek 22

Bu uygulama kullanıcıdan iki sayı almakta ve aldığı iki sayıyı toplayıp, tıklanılan butona göre sonuç üzerinde işlemler yapmaktadır.

The screenshot shows a Windows application window with the title bar 'Exam...'. The window contains a 'Hello World' button at the top. Below it is a section titled 'Sum Operation'. This section has two input fields: 'Number 1' with the value '6' and 'Number 2' with the value '88'. There are two buttons: 'Calculate and Show' and 'Calculate and Save'. Below these buttons, the calculation '6 + 88 = 94' is displayed. At the bottom of the window is a 'Swap' button.

"Operation" isimli sınıfın içindeki "resetValues" isimli prosedür :

```
class Operation
{
    public int a;
    public int b;
    private int _result;

    public void setValues(int a, int b)
    {
        this.a = a;
        this.b = b;
    }
}
```

Bu prosedür nesnenin içindeki "a" ve "b" isimli değişkenlere parametre olarak aldığı değerleri atar. Parantez içindeki iki adet değişken tanımlandığı için parametre sayısı ikidir.

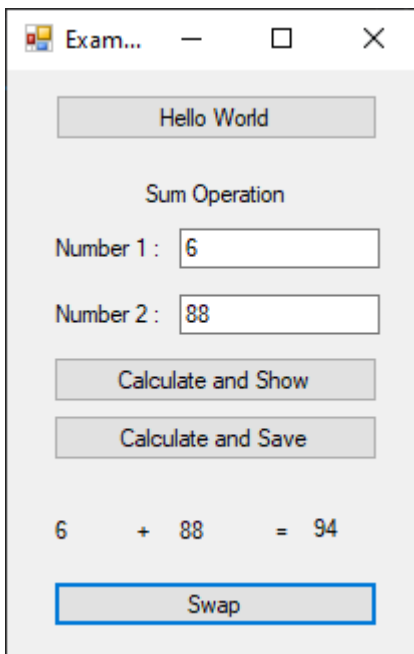
Uygulamanın Kaynak Kodları: [Example22](#)

Ref Bildirili Prosedürler

Prosedürlere gönderilen parametreler normal şartlarda değer olarak gönderilir. Değer olarak gönderilen parametreler prosedüre birebir kopyası oluşturularak aktarılırlar ve içeride yapılan değişiklikler orijinal değer üzerinde bir etkili oluşturmaz. Bazı durumlarda gönderilen parametrenin orijinal değeri üzerinde işlem yapılması gerekebilir. Bu durumlarda orijinal değere erişmek için parametre referans olarak gönderilmelidir. Bir parametreyi referans olarak göndermek demek ramdeki adresini göndermek demektir. Referans olarak göndermek için parametrenin tip değerinden önce "ref" deyimini kullanılmalıdır. Başlangıç değeri almak zorundadır.

Örnek 22

Bu uygulama kullanıcıdan iki sayı almakta ve aldığı iki sayıyı toplayıp, tıklanılan butona göre sonuç üzerinde işlemler yapmaktadır.



"Operation" isimli sınıfın içindeki "swap" isimli iki adet ref parametre alan prosedür :

```
class Operation
{
    public int a;
    public int b;
    private int _result;

    public static void swap(ref int a, ref int b)
    {
        int t;
        t = a;
        a = b;
        b = t;
    }
}
```

Bu prosedür nesnenin içindeki "a" ve "b" isimli parametrelerin değerlerini kendi aralarında değiştirir.

"Operation" isimli sınıfın içindeki "swap" isimli prosedürün kullanılması :

```
Operation.swap(ref op2.a, ref op2.b);
```

Ref parametre alan prosedürler kullanılırken de parametrenin önüne "ref" eklenmelidir.

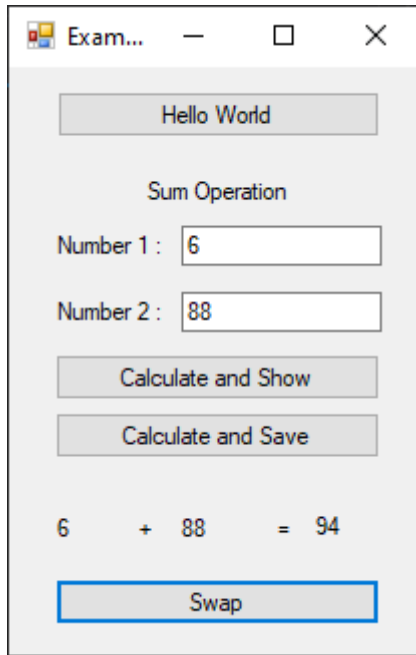
Uygulamanın Kaynak Kodları: [Example22](#)

Out Bildirili Prosedür

Prosedürlerde dışarıya değer aktarmak için kullanılır. Ref ile aynı amaca hizmet eder. Parametrenin tip değerinden önce "out" deyimi kullanılmalıdır. Başlangıç değeri almak zorunda değildir.

Örnek 22

Bu uygulama kullanıcıdan iki sayı almakta ve aldığı iki sayıyı toplayıp, tıklanılan butona göre sonuç üzerinde işlemler yapmaktadır.



"Operation" isimli sınıfın içindeki "setValues" isimli iki adet out parametre alan prosedür :

```
class Operation
{
    public int a;
    public int b;
    private int _result;

    public void setValues(int a, int b, out bool error, out string message)
    {
        if ((a < 0) || (b < 0))
        {
            error = true;
            message = "Values cannot be negative";
        }
        else
        {
            this.a = a;
            this.b = b;
        }
    }
}
```

```
        error = false;
        message = " ";
    }
}
```

Bu prosedürde parametre olarak aldığı "a" ve "b" değerlerinin sadece pozitif olduğu durumdaki değerinin nesne içine alınması, negatif olduğu durumda ise out bildirisi ile hata mesajı yollaması amaçlanmıştır. Pozitif olduğu durumda nesne içindeki ilgili değişkene direkt atama yapar, "error" parametresini "false" atar. "Message" parametresini boş bırakır. Negatif olduğu durumda "error" parametresini "true" atar. "Message" parametresine hata mesajı atar.

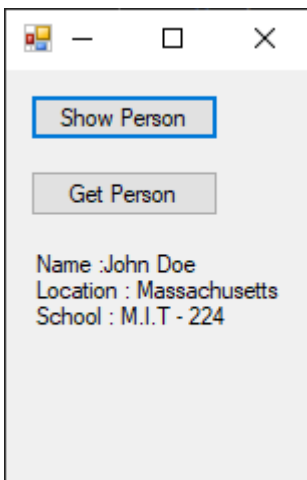
Uygulamanın Kaynak Kodları: [Example22](#)

Kalıtım

Kalıtım, nesnelerin birbirinden türetilmesidir. Nesneler birbirinden türetildiğinde türetilen nesne genel özelliklerini atalarından alır, eşsiz özelliklerini tanımlaması sınıfı oluşturmak için yeterli olur.

Örnek 27

Bu uygulama bir kişinin isim, konum ve okul bilgisini ekrana yazdırmaktadır.



"Location", "School" ve "Person" isimli sınıflar :

```
class Location
{
```

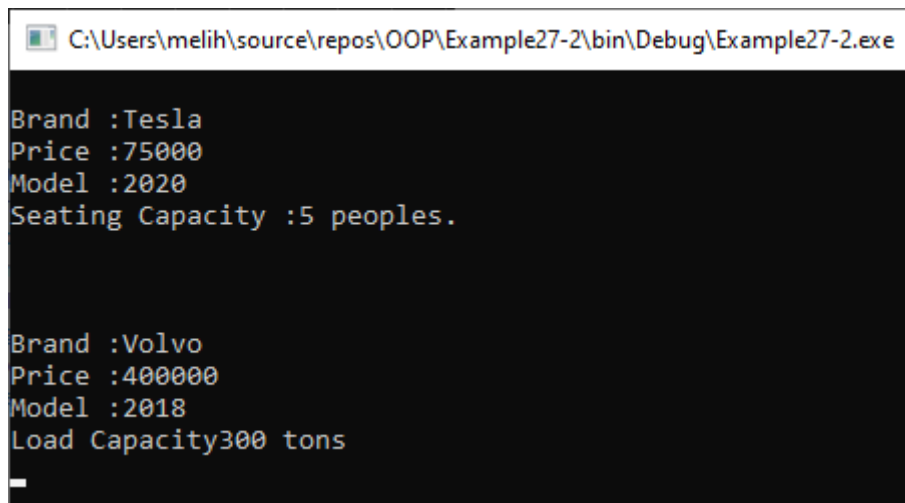
```
}  
  
class School : Location  
{  
  
}  
  
class Person : School  
{  
  
  
}
```

İki nokta operatöründen anladığımız üzere "School" sınıfı "Location" sınıfından, "Person" sınıfı ise "School" sınıfından türetilmektedir.

Uygulamanın Kaynak Kodları: [Example27](#).

Örnek 27-2

Bu uygulamada iki farklı tür aracın özellikleri yazdırılmaktadır.



```
C:\Users\melih\source\repos\OOP\Example27-2\bin\Debug\Example27-2.exe  
  
Brand :Tesla  
Price :75000  
Model :2020  
Seating Capacity :5 peoples.  
  
Brand :Volvo  
Price :400000  
Model :2018  
Load Capacity300 tons  
_
```

"Vehicle" isimli sınıf :

```
public class Vehicle
{
    public string brand;
    public int price;
    public int model;

    public Vehicle(string brand, int price, int model)
    {
        this.brand = brand;
        this.price = price;
        this.model = model;
    }
}
```

Sınıfın üç adet alanı vardır.

"Vehicle" sınıfından türeyen "Car" sınıfı :

```
public class Car : Vehicle
{
    public int seatCapacity;

    public Car(string brand, int price, int model, int capacity) : base(brand,
price, model)
    {
        seatCapacity = capacity;
    }

    public void DisplayInfo()
    {
        Console.WriteLine("\nBrand :" + brand);
        Console.WriteLine("Price :" + price);
        Console.WriteLine("Model :" + model);
        Console.WriteLine("Seating Capacity :" + seatCapacity + " peoples.");
    }
}
```

"Vehicle" sınıfından türeyen "Truck" sınıfı :

```
public class Truck : Vehicle
{
    public int loadCapacity;
    public Truck(string brand, int price, int model, int load) : base(brand,
price, model)
    {
        this.loadCapacity = load;
    }

    public void DisplayInfo()
```

```
{  
    Console.WriteLine("\nBrand :" + brand);  
    Console.WriteLine("Price :" + price);  
    Console.WriteLine("Model :" + model);  
    Console.WriteLine("Load Capacity" + loadCapacity + " tons");  
}  
}
```

Uygulamanın "Main" Bloğu :

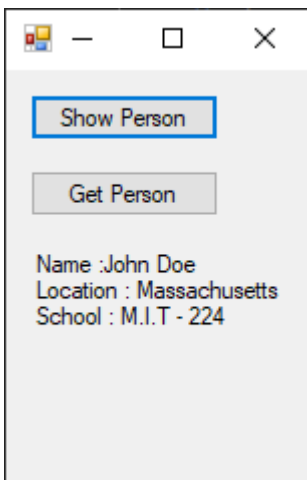
```
Car car1 = new Car("Tesla", 75000, 2020, 5);  
Truck truck1 = new Truck("Volvo", 400000, 2018, 300);  
  
car1.DisplayInfo();  
Console.WriteLine("\n");  
truck1.DisplayInfo();  
Console.ReadLine();
```

Private Bildiri Yapmak

Erişim belirteçleri, bir sınıfa ait öğelere erişebilme kısıtlarını ve yetilerini belirleyen anahtar sözcüklerdir. Erişim belirteçleri ilgili öğenin başına yazılarak kullanılır. "private" sözcüğü bir erişim belirteçidir. Öğenin sadece bulunduğu sınıf içerisinde erişilmesini sağlar, dışa kapalıdır.

Örnek 27

Bu uygulama bir kişinin isim, konum ve okul bilgisini ekrana yazdırmaktadır.



"School" isimli sınıf :

```
class School : Location
```

```
{  
    protected string schoolName;  
    private int schoolId;  
  
    public string schoolData  
    {  
        get  
        {  
            return schoolName + " - " + schoolId;  
        }  
    }  
  
    private void setSchoolId()  
    {  
        var rand = new Random();  
        schoolId = rand.Next(100, 999);  
    }  
  
    public School(string sName)  
    {  
        schoolName = sName;  
        setSchoolId();  
    }  
}
```

Sınıfta bulunan schoolId" değişkeni ve "setSchool" metodu "private" bildirisiyle tanımlanmıştır. Yapılandırıcı ile ilk değer ataması yapılmıştır. "School" sınıfının içinden başka bir yerden ulaşılamazlar.

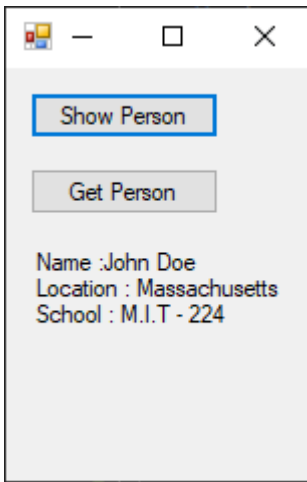
Uygulamanın Kaynak Kodları: [Example27](#)

Public Bildiri Yapmak

Erişim belirteçleri, bir sınıfa ait öğelere erişebilme kısıtlarını ve yetilerini belirleyen anahtar sözcüklerdir. Erişim belirteçleri ilgili öğenin başına yazılarak kullanılır. "public" sözcüğü bir erişim belirteçidir. Öğenin başka sınıflar içerisinden erişilmesini sağlar. Erişim kısıtı yoktur, her yerden erişilir.

Örnek 27

Bu uygulama bir kişinin isim, konum ve okul bilgisini ekrana yazdırmaktadır.



"Location" isimli sınıf :

```
class Location
{
    public string locationName;
}
```

"locationName" değişkeni "public" bildiriyile tanımlanmıştır.

"Person" isimli sınıf :

```
class Person : School
{
    internal string fullName;

    public Person(string fName, string schoolName) : base(schoolName)
    {
        fullName = fName;
        locationName = "Empty";
    }

    public string getAllData()
    {
        return "Name :" + fullName + "\nLocation : " + locationName +
            "\nSchool : " + schoolData;
    }
}
```

"getAllData" metodu "public" bildiri ile tanımlanmıştır.

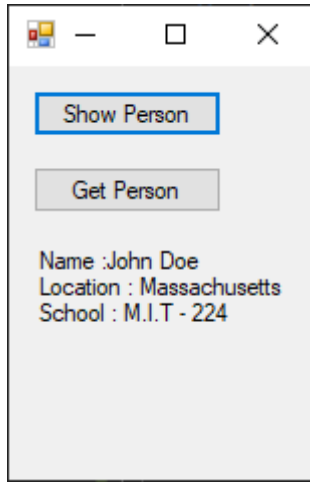
Uygulamanın Kaynak Kodları: [Example27](#)

Internal Bildiri Yapmak

Erişim belirteçleri, bir sınıfa ait öğelere erişebilme kısıtlarını ve yetilerini belirleyen anahtar sözcüklerdir. Erişim belirteçleri ilgili öğenin başına yazılarak kullanılır. "internal" sözcüğü bir erişim belirteçidir. Bir üyenin bir assembly içindeki tüm dosyalar tarafından bilindiğini ancak assembly(Derlenmiş .exe veya .dll dosyaları) dışında tanınmadığını bildirir. Yazılım bileşenleri oluştururken faydalıdır.

Örnek 27

Bu uygulama bir kişinin isim, konum ve okul bilgisini ekrana yazdırmaktadır.



"Person" isimli sınıf :

```
class Person : School
{
    internal string fullName;

    public Person(string fName, string schoolName) : base(schoolName)
    {
        fullName = fName;
        locationName = "Empty";
    }

    public string getAllData()
    {
        return "Name : " + fullName + "\nLocation : " + locationName +
"\nSchool : " + schoolData;
    }
}
```

"fullName" değişkeni "internal" bildiri ile tanımlanmıştır.

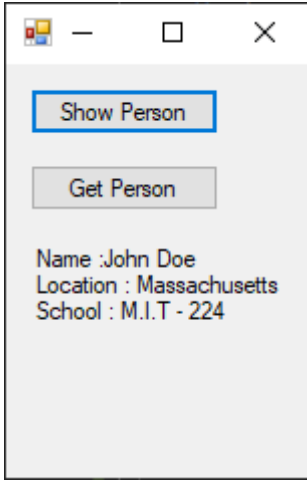
Uygulamanın Kaynak Kodları: [Example27](#)

Protected Bildiri Yapmak

Erişim belirteçleri, bir sınıfa ait öğelere erişebilme kısıtlarını ve yetilerini belirleyen anahtar sözcüklerdir. Erişim belirteçleri ilgili öğenin başına yazılarak kullanılır. "protected" sözcüğü bir erişim belirteçidir.Öğenin bulunduğu sınıftan ve o sınıftan türetilen sınıflar içerisinde erişilmesini sağlar.

Örnek 27

Bu uygulama bir kişinin isim, konum ve okul bilgisini ekrana yazdırmaktadır.



"School" isimli sınıf :

```
class School : Location
{
    protected string schoolName;
    private int schoolId;

    public string schoolData
    {
        get
        {
            return schoolName + " - " + schoolId;
        }
    }

    private void setSchoolId()
    {
        var rand = new Random();
        schoolId = rand.Next(100, 999);
    }
}
```

```
public School(string sName)
{
    schoolName = sName;
    setSchoolId();
}
}
```

"schoolName" değişkeni "protected" bildirisi ile tanımlanmıştır.

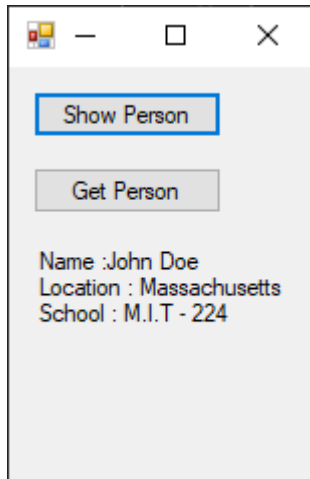
Uygulamanın Kaynak Kodları: [Example27](#)

Partial Class Bildirisi

Sınıflar, yapılar veya arayüz tanımlarını birden fazla dosyaya bölmek mümkündür. Her kısım ve tanımlama dosyalar arasına bölünebilir. Bölünen dosyalar derleme sırasında birleştirilir.

Örnek 27

Bu uygulama bir kişinin isim, konum ve okul bilgisini ekrana yazdırmaktadır.



"Person" isimli sınıf :

```
partial class Person : School
{
    internal string fullName;

    public Person(string fName, string schoolName) : base(schoolName)
    {
        fullName = fName;
        locationName = "Empty";
    }
}
```

```
        public string getAllData()
        {
            return "Name :" + fullName + "\nLocation : " + locationName +
"\nSchool : " + schoolData;
        }
    }

    partial class Person
    {
        public void setLocation(string location)
        {
            locationName = location;
        }
    }

    partial class Person
    {
        public void showAllData()
        {
            MessageBox.Show(getAllData());
        }
    }
}
```

"Person" sınıfı "partial" bildirisi ile tanımlanmıştır.

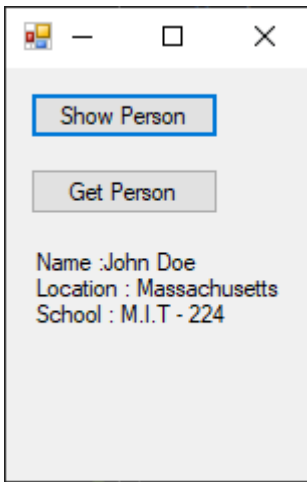
Uygulamanın Kaynak Kodları: [Example27](#)

Base Komutu

Base komutu, türetilmiş sınıf içerisinde temel sınıfın üyelerine erişmek için kullanılır. Bu üyelere yapılandırıcı da dahildir.

Örnek 27

Bu uygulama bir kişinin isim, konum ve okul bilgisini ekrana yazdırmaktadır.



"Person" isimli sınıf :

```
partial class Person : School
{
    internal string fullName;

    public Person(string fName, string schoolName) : base(schoolName)
    {
        fullName = fName;
        locationName = "Empty";
    }

    public string getAllData()
    {
        return "Name :" + fullName + "\nLocation : " + locationName +
"\nSchool : " + base.schoolData;
    }
}

partial class Person
{
    public void setLocation(string location)
    {
        locationName = location;
    }
}

partial class Person
{
    public void showAllData()
    {
        MessageBox.Show(getAllData());
    }
}
```

"Person" sınıfının yapılandırıcısında, kalıtım aldığı "School" sınıfının yapılandırıcısını çağırıp parametre göndermek için "base" komutu kullanılmıştır.

"getAllData" metodunun içinde "School" sınıfının özelliğine ulaşmak için "base" komutu kullanılmıştır.

Uygulamanın Kaynak Kodları: [Example27](#)

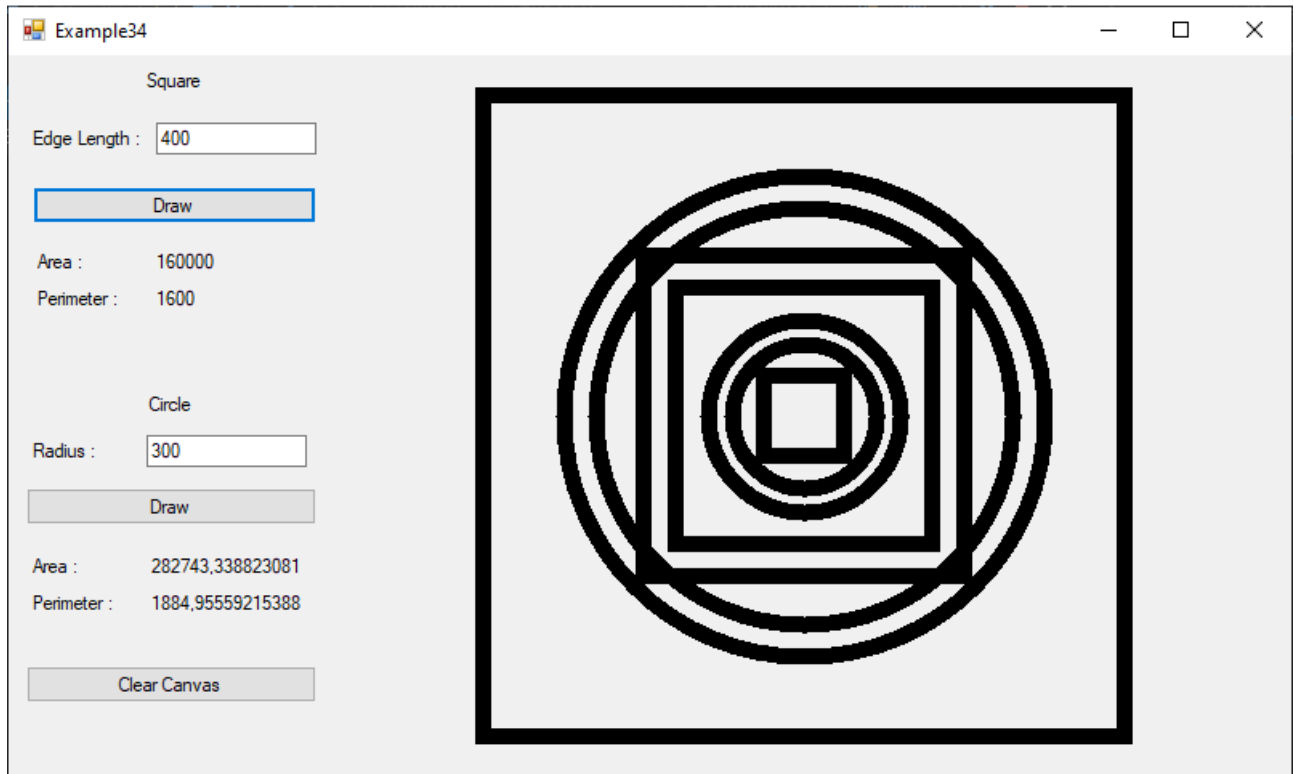
Virtual Ve Override Metod Tanımlamak

Sanal metod, temel sınıf içinde virtual olarak deklare edilen ve bir veya daha fazla türetilmiş sınıf içinde yeniden tanımlanan bir metottur. Böylece, her türetilmiş sınıf bir sanal

metodun kendine özgü bir versiyonuna sahip olabilir. Sanal bir metod oluşturmak için erişim belirleyicisinden sonra "virtual" anahtar kelimesi eklenir.

Örnek 34

Bu uygulama kullanıcıdan aldığı kenar uzunluğu ve yarıçap değerlerine göre şekiller çizer. Çizilen şekillerin alan ve çevre hesabını yaparak kullanıcıya gösterir.



"Shape" isimli temel sınıf :

```
public class Shape
{
    public const double PI = Math.PI;
```

```

protected double x, y;

public Shape()
{
}

public Shape(double x, double y)
{
    this.x = x;
    this.y = y;
}

public virtual double Area()
{
    return x * y;
}

public virtual double Perimeter()
{
    return x * 2 + y * 2;
}

public int getEdgeLength()
{
    return (int)x;
}
}

```

Bu sınıfta bir şeklin sahip olması gereken özellikler ve işlemler tanımlanmıştır. Hesaplama işlemleri bu şeklin bir dikdörtgen olduğu nu varsayarak yaptığı hesaplamanın değerini döndürmektedir. Şeklin dikdörtgenden başka olduğu durumlarda eski hesaplamalar devre dışı dışı bırakılıp, yeni hesaplamaların kalıtım alınan sınıfta yapılabilmesi için ilgili metotlar sanal olarak tanımlanmıştır. Bu metotlar başındaki "virtual" anahtar sözcüğünden tanınabilmektedir.

"Shape" sınıfından kalıtım alan "Square" sınıfı :

```

public class Square : Shape
{
    public Square(double edge) : base(edge, edge)
    {
        x = y = edge;
    }
}

```

Bu sınıf "Shape" sınıfından kalıtım almıştır. Hesaplama yapılması istenen şekil kare olarak kurgulanmıştır. Kare, eşit kenarlı bir dikdörtgen olduğundan dolayı tanımlanan Yapılandırıcıda tek bir kenar değeri almaktadır ve aldığı kenar değerini kalıtım aldığı sınıfta bulunan iki kenar değerine atanmaktadır. Hesaplamalarında bir farklılık bulunmadığından dolayı herhangi bir metot devre dışı bırakma işlemi gerçekleşmemiştir.

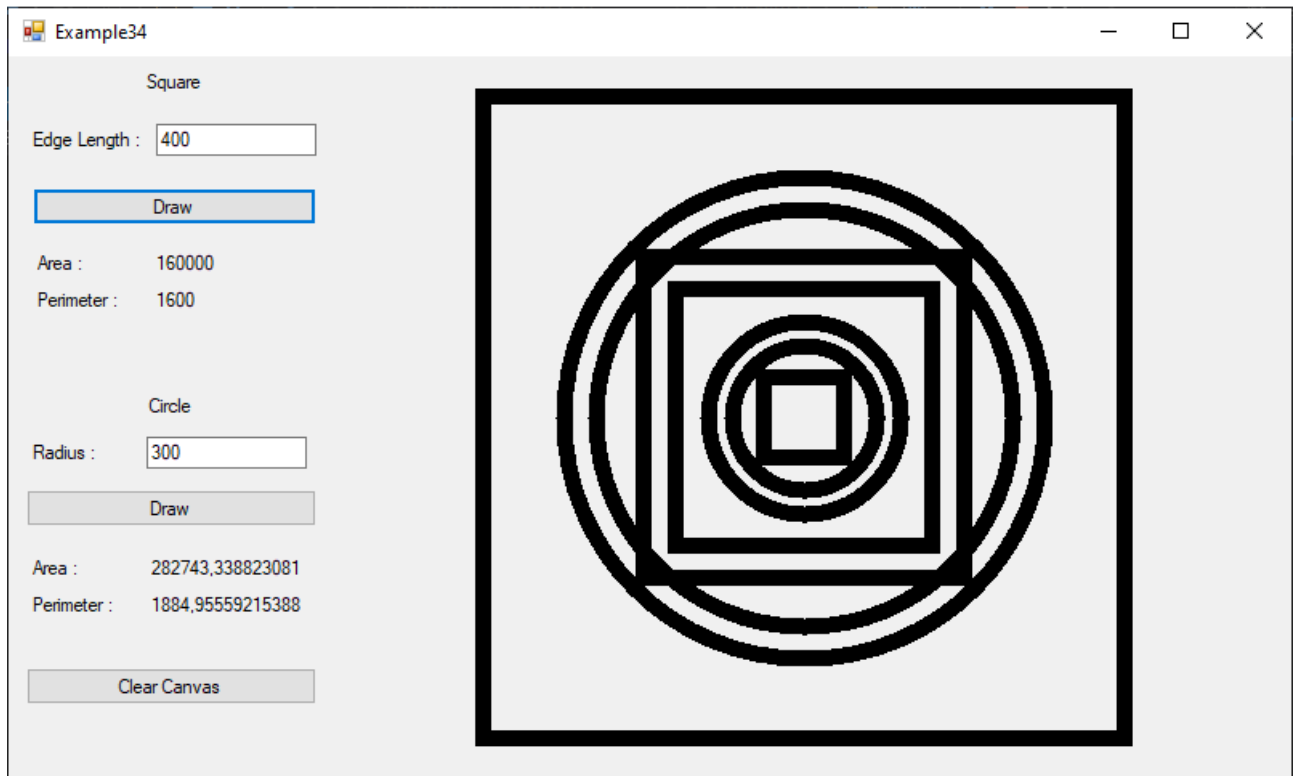
Uygulamanın Kaynak Kodları: [Example34](#).

Override Metod Tanımlamak

Bir sanal metod bir türetilmiş sınıf tarafından yeniden tanımlanırken override niteleyicisi kullanılır. Bu nedenle, bir sanal metodu bir türetilmiş sınıf içinde yeniden tanımlama işlemine metodu devre dışı bırakma (method overriding) denir. Bu işlem için metodun erişim belirleyicisinden sonra "override" anahtar kelimesi eklenir.

Örnek 34

Bu uygulama kullanıcıdan aldığı kenar uzunluğu ve yarıçap değerlerine göre şekiller çizer. Çizilen şekillerin alan ve çevre hesabını yaparak kullanıcıya gösterir.



"Shape" isimli temel sınıf :

```
public class Shape
{
    public const double PI = Math.PI;
    protected double x, y;

    public Shape()
    {
    }
}
```

```

public Shape(double x, double y)
{
    this.x = x;
    this.y = y;
}

public virtual double Area()
{
    return x * y;
}

public virtual double Perimeter()
{
    return x * 2 + y * 2;
}

public int getEdgeLength()
{
    return (int)x;
}
}

```

Bu sınıfta bir şeklin sahip olması gereken özellikler ve işlemler tanımlanmıştır. Hesaplama işlemleri bu şeklin bir dikdörtgen olduğu nu varsayarak yaptığı hesaplamanın değerini döndürmektedir. Şeklin dikdörtgenden başka olduğu durumlarda eski hesaplamalar devre dışı dışı bırakılıp, yeni hesaplamaların kalıtım alınan sınıfta yapılabilmesi için ilgili metotlar sanal olarak tanımlanmıştır. Bu metotlar başındaki "virtual" anahtar sözcüğünden tanınabilmektedir.

"Shape" sınıfıtan kalıtım alan "Circle" sınıfı :

```

public class Circle : Shape
{
    public Circle(double r) : base(r, 0)
    {
    }

    public override double Area()
    {
        return PI * x * x;
    }

    public override double Perimeter()
    {
        return 2 * PI * x;
    }
}

```


Bu sınıf "Shape" sınıfından kalıtım almıştır. Hesaplama yapılması istenen şekil daire olarak kurgulanmıştır. Yapılandırıda sadece yarıçap değeri almakta ve bunu kalıtım aldığı sınıfın tek bir kenar uzunluğu değerine atamakta, diğer değere ise sıfır atamaktadır. Dairenin hesaplamaları diktörtgenden farklı olduğu için hesaplama yapan metodları devre dışı bırakmış ve yeniden tanımlamıştır. Görüldüğü üzere bu metodlar tanımlanırken "override" anahtar kelimesi kullanılmaktadır.

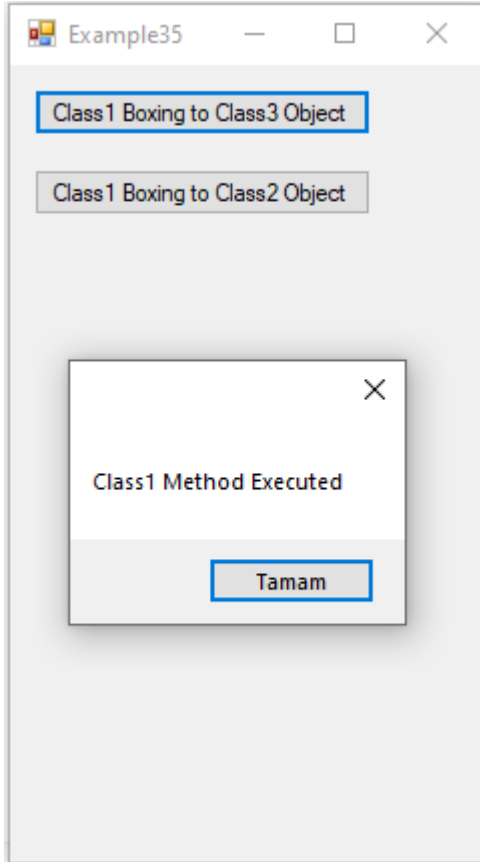
Uygulamanın Kaynak Kodları: [Example34](#).

Değişken Tanımlarken Farklı Class'lar Kullanmak

Kalıtım gerçekleşen sınıflarda nesne değişkeni "boxing" yapılarak kalıtımı aldığı üst sınıfın özelliklerine erişebilir. Yani "Child" nesnesiyle "Parent" özelliklerine erişilebilir.

Örnek 35

Bu uygulama basılan butona göre mesaj kutusu göstermektedir.



"Class1", "Class2" ve "Class3" isimli sınıflar :

```
class Class1
```

```
{  
    public string cl1 = "";  
    public void cl1Message()  
    {  
        MessageBox.Show(cl1);  
    }  
}  
  
class Class2 : Class1  
{  
    public string cl2 = "";  
    public void cl2Message()  
    {  
        MessageBox.Show(cl2);  
    }  
}  
  
class Class3 : Class2  
{  
    public string cl3 = "";  
    public void cl3Message()  
    {  
        MessageBox.Show(cl3);  
    }  
}
```

"Class2" sınıfı "Class1" sınıfından, "Class3" sınıfı "Class2" sınıfından kalıtım almaktadır.

"Class3" nesnesi ile "Class1" metodu çalıştırmak :

```
Class3 cl3 = new Class3();  
((Class1)cl3).cl1 = "Class1 Method Executed";  
((Class1)cl3).cl1Message();
```

Burada "Class1" sınıfı "cl3" nesnesine boxing yapmıştır.

"Class2" nesnesi ile "Class1" metodu çalıştırmak :

```
Class2 cl2 = new Class2();  
((Class1)cl2).cl1 = "Class1 Method Executed";  
((Class1)cl2).cl1Message();
```

Burada "Class1" sınıfı "cl2" nesnesine boxing yapmıştır.

Uygulamanın Kaynak Kodları: [Example35](#).

Abstract Class'lar

Ortak özellikleri olan nesneler için temel sınıf tanımlamak için kullanılır. Sınıfın ne olduğunu belirtmek için kullanılır. Nesnesi oluşturulamaz ancak bir sınıf tarafından kalıtım yoluyla kullanılabilir. Bir sınıf yalnızca bir abstract sınıftan türetilebilir. Abstract ve normal metotlar barındırabilir. Abstract tanımlanan metodun gövdesinin, kalıtım yapılan sınıf tarafından sağlanması zorunludur.

Örnek 36

Bu uygulama konsola x ve y'nin sahip olduğu değeri yazdırmaktadır.

```
C:\Users\melih\source\repos\OOP\Example36\bin\Debug\Example36.exe
x = 111, y = 161
```

"BaseClass" isimli soyut sınıf :

```
abstract class BaseClass
{
    protected int _x = 100;
    protected int _y = 150;
    public abstract void AbstractMethod();
    public abstract int X { get; }
    public abstract int Y { get; }
}
```

Bu sınıf başına "abstract" anahtar kelimesi kullanılarak soyut olarak tanımlanmıştır. İçerisinde bir adet soyut metot ve iki adet soyut özellik barındırmaktadır.

"DerivedClass" isimli "BaseClass" sınıfından türetilmiş sınıf :

```
class DerivedClass : BaseClass
{
    public override void AbstractMethod()
    {
        _x++;
        _y++;
    }

    public override int X
    {
        get
        {
            return _x + 10;
        }
    }
}
```

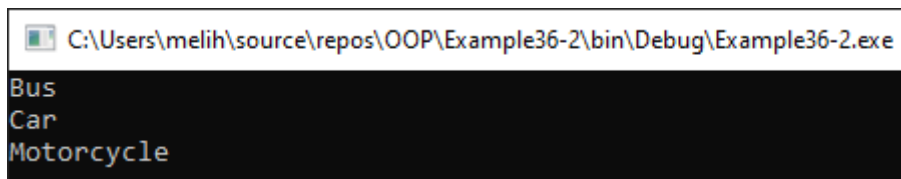
```
        public override int Y
        {
            get
            {
                return _y + 10;
            }
        }
    }
}
```

Görüldüğü üzere bu sınıfta "BaseClass" sınıfından kalıtım alınmakta ve soyut olarak tanımlanan bütün öğelerin gövdesi bu sınıfta oluşturulmakta. Bu işlem için tüm soyut öğelerin başında "override" anahtar kelimesi kullanılmıştır.

Uygulamanın Kaynak Kodları: [Example36](#)

Örnek 36-2

Bu uygulama taşıt türlerini ekrana yazmaktadır.



```
C:\Users\melih\source\repos\OOP\Example36-2\bin\Debug\Example36-2.exe
Bus
Car
Motorcycle
```

"Vehicle" isimli soyut sınıf :

```
public abstract class Vehicle
{
    public abstract void display();
}
```

"Bus" isimli sınıf :

```
public abstract class Vehicle
{
    public abstract void display();
}
```

"Car" isimli sınıf :

```
public class Car : Vehicle
{
    public override void display()
    {
        Console.WriteLine("Car");
    }
}
```

"Motorcycle" isimli sınıf :

```
public class Motorcycle : Vehicle
{
    public override void display()
    {
        Console.WriteLine("Motorcycle");
    }
}
```

Uygulamanın "Main" Bloğu :

```
Vehicle v;
v = new Bus();
v.display();
v = new Car();
v.display();
v = new Motorcycle();
v.display();
```

Uygulamanın Kaynak Kodları: **[Example36-2]**(

Interface

Sınıflar için bir şablon sağlar. ne yapması gerektiğini belirtir. Bir sınıf birden fazla interface'i kalıtım yoluyla kullanabilir. Tüm üyeleri varsayılan olarak "public" erişim erişim belirleyicisine sahiptir. Tanımlarken isminin başına I harfi getirmek sınıflar ile ayırt edilmesini sağlar.

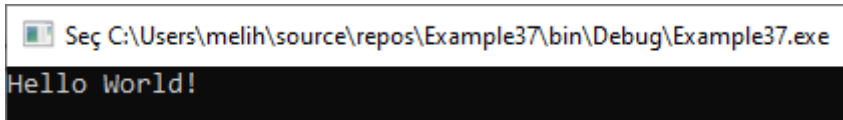
Sınıflardan farklı olarak arayüzlerin sahip olamayacakları öğeler şunlardır :

Yapıcılar (Constructors)

Yıkıcılar (Destructors)

Örnek 37

Bu uygulama ekrana "Hello World!" yazmaktadır.



"IHello" isimli arayüz :

```
interface IHello
{
    void helloWorld();
}
```

Burada "interface" anahtar kelimesiyle "IHello" isimli bir arayüz tanımlanmıştır.

"MainClass" sınıfı :

```
class MainClass : IHello
{
    public void helloWorld()
    {
        Console.WriteLine("Hello World!");
    }

    public static void Main(String[] args)
    {
        MainClass hello = new MainClass();
        hello.helloWorld();

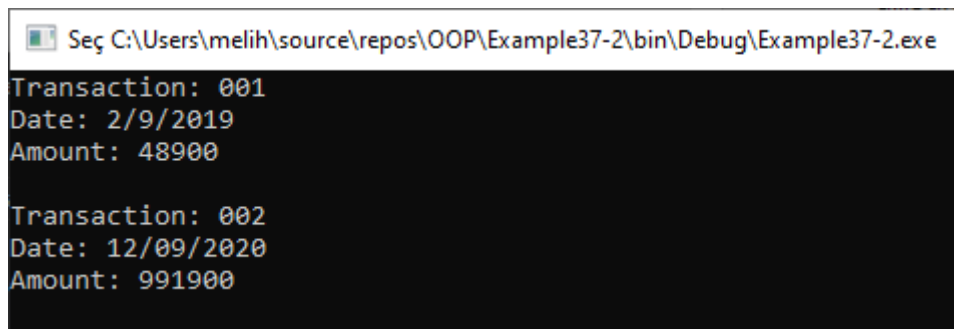
        Console.ReadLine();
    }
}
```

Burada "MainClass" sınıfı "IHello" arayüzünden türetilmekte ve "Main" metodunda sınıftaki "helloWorld" metodu işletilmekte.

Uygulamanın Kaynak Kodları: [Example37](#)

Örnek 37-2

Bu uygulama yapılan para transferlerini yazdırır.



```
Seç C:\Users\melih\source\repos\OOP\Example37-2\bin\Debug\Example37-2.exe
Transaction: 001
Date: 2/9/2019
Amount: 48900

Transaction: 002
Date: 12/09/2020
Amount: 991900
```

"ITransactions" isimli arayüz :

```
public interface ITransactions
{
    void showTransaction();
    double getAmount();
}
```

"Transaction" isimli "ITransactions" arayüzünden kalıtım alan sınıf :

```
public class Transaction : ITransactions
{
    private string tCode;
    private string date;
    private double amount;

    public Transaction()
    {
        tCode = " ";
        date = " ";
        amount = 0.0;
    }

    public Transaction(string c, string d, double a)
    {
        tCode = c;
        date = d;
        amount = a;
    }

    public double getAmount()
    {
        return amount;
    }

    public void showTransaction()
    {
        Console.WriteLine("Transaction: {0}", tCode);
        Console.WriteLine("Date: {0}", date);
    }
}
```

```
        Console.WriteLine("Amount: {0}", getAmount());  
    }  
}
```

Uygulamanın "Main" bloğu :

```
Transaction t1 = new Transaction("001", "2/9/2019", 48900.00);  
Transaction t2 = new Transaction("002", "12/09/2020", 991900.00);  
  
t1.showTransaction();  
Console.WriteLine();  
t2.showTransaction();  
Console.ReadLine();
```

Uygulamanın Kaynak Kodları: [Example37-2](

Delege Tanımlamaları

Delege, bir metoda referansta bulunabilen bir nesnedir. Delegeler kullanılarak referansı oluşturulan metod çağırılabilir. Programın çalışma zamanı sırasında referansta bulunduğu metodun değiştirilebilmesi en önemli özelliğidir.

Delegeler "delegate" anahtar kelimesiyle deklare edilir. Referansı oluşturulacak metot ile geri dönüş ve parametre tipleri uyumlu olmalıdır.

Örnek 39

Bu uygulama kullanıcıdan alınmayan bir cümle ve kullanıcıdan alınan bir metin üzerinde yine kullanıcının eklediği sıraya göre toplu işlemler yapmaktadır. Bu işlemler: Metni ters çevirme, boşlukları virgöl ile doldurma, boşlukları silme ve virgülleri boşlukla değiştirme.

Example39

— □ ×

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Reverse Sentence

Replace Spaces with Comma

Apply Both

Enter Text

Operation Chain

Add

Reverse

Replace Spaces with Comma

Remove Spaces

Replace Commas with Space

Remove

Reverse

Replace Spaces with Comma

Remove Spaces

Replace Commas with Space

Apply Chain

"StringOps" isimli sınıf ve "stringHandler" isimli delege :

```

delegate void stringHandler(ref string str);
class StringOps
{
    public void getStr(ref string str)
    {

    }

    public void replaceToComma(ref string str)
    {
        str = str.Replace(" ", ",");
    }

    public void replaceToSpace(ref string str)
    {
        str = str.Replace(",", " ");
    }

    public void removeSpaces(ref string str)
    {
        str = str.Replace(" ", "");
    }

    public void reverseString(ref string str)
    {
        string temp = "";
        for (int i = str.Length-1; i >= 0; i--)

```

```
        {  
            temp += str[i];  
        }  
  
        str = temp;  
    }  
  
}
```

Burada string tipindeki değerler üzerinde çeşitli işlemler yapan "StringOps" isimli bir sınıf ve o sınıfın metotlarını çağırması için tanımlanmış "stringHandler" isimli delege görülmektedir.

"Reverse Sentence" isimli butonun işlettiği metot :

```
class Example39  
{  
    StringOps str = new StringOps();  
    public string loremReverse(string text)  
    {  
        stringHandler strHandler = str.reverseString;  
        strHandler(ref text);  
  
        return text;  
    }  
}
```

Burada metotda "strHandler" ismiyle tanımlanan delege "str" nesnesinin "reverseString" metodunu göstermesi sağlanmış ve metodun tanımlanan delege ile çalışması sağlanmıştır.

"Replace Spaces With Commas" isimli butonun işlettiği metot :

```
class Example39  
{  
    StringOps str = new StringOps();  
    public string loremReplace(string text)  
    {  
        stringHandler strHandler = str.replaceToComma;  
        strHandler(ref text);  
  
        return text;  
    }  
}
```

Burada metotda "strHandler" ismiyle tanımlanan delege "str" nesnesinin "loremReplace" metodunu göstermesi sağlanmış ve metodun tanımlanan delege ile çalışması sağlanmıştır.

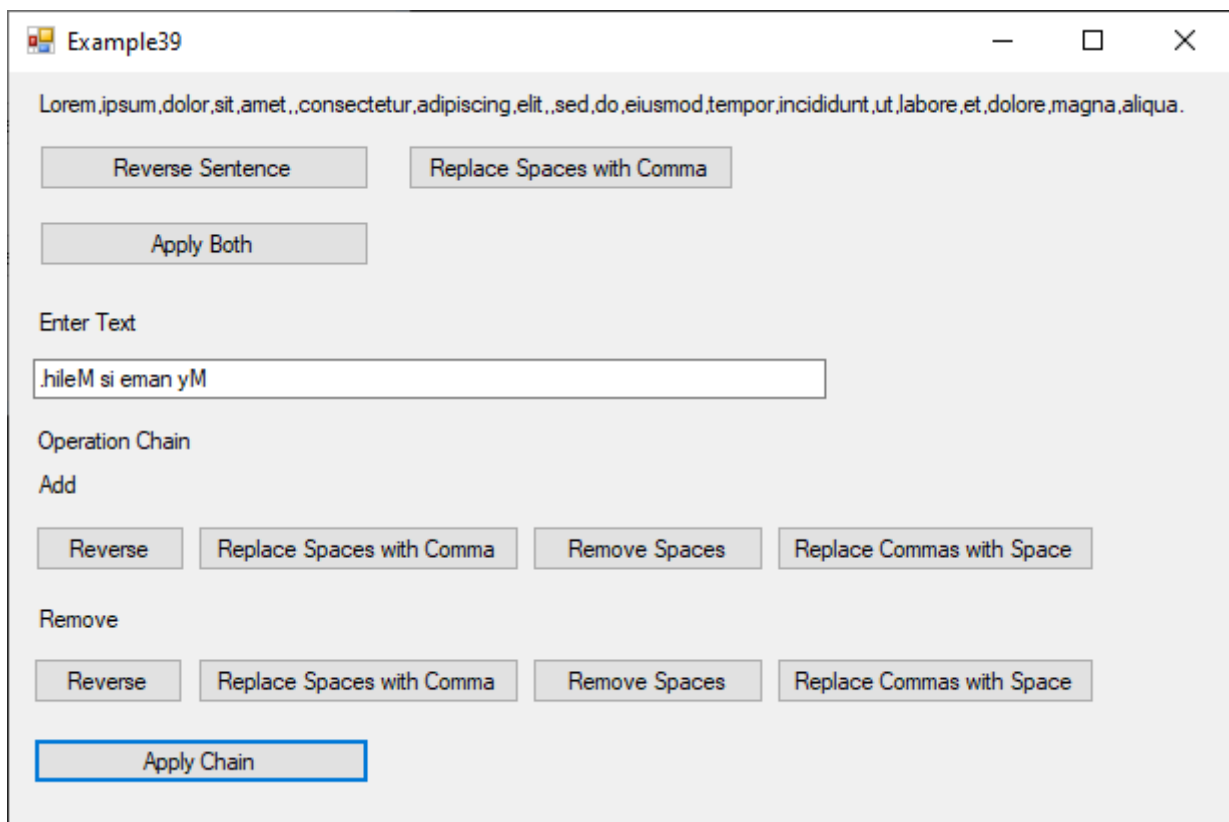
Uygulamanın Kaynak Kodları: [Example39](#)

Delege İle Bizden Fazla Metodu Aynı Anda Çağırarak

Delegeler kullanılarak metotlar için bir çağrı zinciri oluşturulabilir. Bu zincire metot eklemek için "+=", zincirden metot çıkarmak için "-=" operatörü kullanılır. Çoklu çağrı yapan bir delegenin dönüş tipi "void" olmalıdır.

Örnek 39

Bu uygulama kullanıcıdan alınmayan bir cümle ve kullanıcıdan alınan bir metin üzerinde yine kullanıcının eklediği sıraya göre toplu işlemler yapmaktadır. Bu işlemler: Metni ters çevirme, boşlukları virgül ile doldurma, boşlukları silme ve virgülleri boşlukla değiştirme.



"StringOps" isimli sınıf ve "stringHandler" isimli delege :

```
delegate void stringHandler(ref string str);  
class StringOps  
{  
  
    public void getStr(ref string str)  
    {  
  
    }  
  
    public void replaceToComma(ref string str)  
    {  
  
    }  
}
```

```

        str = str.Replace(" ", ",");
    }

    public void replaceToSpace(ref string str)
    {
        str = str.Replace(", ", " ");
    }

    public void removeSpaces(ref string str)
    {
        str = str.Replace(" ", "");
    }

    public void reverseString(ref string str)
    {
        string temp = "";
        for (int i= str.Length-1; i >= 0; i--)
        {
            temp += str[i];
        }

        str = temp;
    }
}

```

Burada string tipindeki değerler üzerinde çeşitli işlemler yapan "StringOps" isimli bir sınıf ve o sınıfın metotlarını çağırması için tanımlanmış "stringHandler" isimli delege görülmektedir.

"Apply Both" isimli butonun işlettiği metot :

```

public string chainedOp(string text)
{
    stringHandler op1 = new stringHandler(str.replaceToComma);
    stringHandler op2 = new stringHandler(str.reverseString);
    stringHandler op3 = op1;

    op3 += op2;
    op3(ref text);

    return text;
}

```

Burada "op*" isimleriyle delegeler tanımlanmıştır. Bu tanımlanan delegelerden "op3" isimli olanın "+" operatörü kullanılarak iki adet metoda referans vermesi sağlanmıştır. Son olarak "op3" isimli delege kullanılarak iki metot çalıştırılmıştır.

"Example39" isimli sınıfın bir kısmı :

```

class Example39
{
    StringOps str = new StringOps();
    public string lorem = "Lorem ipsum dolor sit amet, consectetur adipiscing
elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.";
    stringHandler strRev;
    stringHandler rToComma;
    stringHandler rToSpace;
    stringHandler removeSpaces;
    stringHandler chainOp;

    public Example39()
    {
        strRev = str.reverseString;
        rToComma = str.replaceToComma;
        rToSpace = str.replaceToSpace;
        removeSpaces = str.removeSpaces;
        chainOp = str.getStr;
    }

    /*
        0 - Reverse
        1 - Replace Spaces With Commas
        2 - Remove Spaces
        3 - Replace Commas With Spaces
    */
    public void addToChain(short opt)
    {
        if (opt == 0)
        {
            chainOp += strRev;
        }
        else if (opt == 1)
        {
            chainOp += rToComma;
        }
        else if (opt == 2)
        {
            chainOp += removeSpaces;
        }
        else if (opt == 3)
        {
            chainOp += rToSpace;
        }
    }

    /*
        0 - Reverse
        1 - Replace Spaces With Commas
        2 - Remove Spaces
        3 - Replace Commas With Spaces
    */
    public void removeToChain(short opt)
    {
        if (opt == 0)

```

```

        {
            chainOp -= strRev;
        }
        else if (opt == 1)
        {
            chainOp -= rToComma;
        }
        else if (opt == 2)
        {
            chainOp -= removeSpaces;
        }
        else if (opt == 3)
        {
            chainOp -= rToSpace;
        }
    }

    public string applyChain(string text)
    {
        chainOp(ref text);
        chainOp = str.getStr;
        return text;
    }
}

```

Bu sınıfta delegeler kullanarak bir işlem zinciri oluşturulmuştur. "addToChain" ve "removeToChain" metodları ile zincire işlem eklenmekte ve çıkartılmaktadır. "applyChain" metodu ile zincir çalıştırılmaktadır.

"Example39" isimli sınıfın kullanımı :

```

Example39 app = new Example39();

app.addToChain(0);
app.addToChain(1);
string out = app.applyChain(text);

```

Uygulamanın Kaynak Kodları: [Example39](#)

Delege ile Kontrollere Yordam Belirlemek

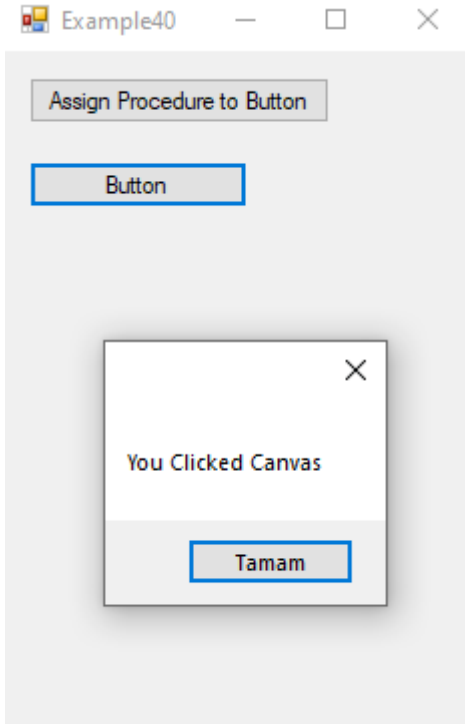
Windows form kontrollerinde olaylar "EventHandler" isimli delege ile bağlanır. Bu delegeye "+=" operatörü ile istenilen yordam bağlanarak ilgili olay tetiklendiğinde yordamın çalışması sağlanabilir. Bağlanan yordamın parametre değerleri "EventHandler" delegesiyle aynı olmalıdır.

"EventHandler" delegesi :

```
public delegate void EventHandler(object? sender, EventArgs e);
```

Örnek 40

Bu uygulamada form içerisinde eleman harici bir yere tıklandığında bir mesaj gösterilir. Bir butona çalışma zamanında yordam belirlenir. Ve bu butona tıklanıldığında mesaj gösterilir.



"EventMethods" isimli sınıf :

```
class EventMethods
{
    public void clickButton(object obj, EventArgs e)
    {
        MessageBox.Show("You Clicked Button");
    }
}
```

"Form1" isimli sınıf :

```
public partial class Form1 : Form
{
    EventMethods eM = new EventMethods();

    public Form1()
    {
        InitializeComponent();
        this.Click += formAlert;
    }
}
```

```
}

private void formAlert(object obj, EventArgs e)
{
    MessageBox.Show("You Clicked Canvas");
}

private void button1_Click(object sender, EventArgs e)
{
    EventMethods eM = new EventMethods();
    button2.Click += eM.clickButton;
}
}
```

Burada "Form1" sınıfının yapılandırıcısında formun arkaplanına tıklanma olayına "+=" operatörü ile "formAlert" yordamı bağlanmıştır. Ve butona tıklandığında bir diğer butona "EventMethods" sınıfının "clickButton" yordamının çalışma zamanında bağlanması gerçekleşmektedir.

Uygulamanın Kaynak Kodları: [Example40](#)

Event Oluşturmak

Olay, bir eylemin gerçekleştiğini bildirmek için nesne tarafından gönderilen mesajdır. Eylem, düğme tıklaması gibi kullanıcı etkileşimi veya bir özelliğin(property) değerinin değişmesi olabilir. Olaylar bir sınıfın üyeleridir ve "event" anahtar kelimesi ile deklare edilirler.

Örnek 41

Bu uygulama kullanıcıdan sayı alır ve kullanıcı 10'dan büyük sayı girerse ekrana uyarı yazar.


```
C:\Users\melih\source\repos\OOP\Example41\bin\Debug\Example41.exe
Enter Number :
1
Enter Number :
5
Enter Number :
7
Enter Number :
8
Enter Number :
15
Event Triggered! Entered Number is bigger than 10.
Enter Number :
2
Enter Number :
70
Event Triggered! Entered Number is bigger than 10.
Enter Number :
```

"NumberClass" isimli sınıf :

```
public delegate void myCustomHandler();

public class NumberClass
{
    private int _number = 0;
    public event myCustomHandler BigNumber;

    public int number
    {
        set
        {
            _number = value;
            if (value > 10) {
                BigNumber();
            }
        }
        get { return _number; }
    }
}
```

Görüldüğü üzere "myCustomHandler" adında bir delege ve bu delege kullanılarak ilgili sınıfın içinde "BigNumber" isimli olay tanımlanmıştır. Sınıfın içindeki "number" özelliği yazılırken değer 10'dan büyük ise "BigNumber" olayı gerçekleşecektir.

"Program" sınıfı ve içindeki "Main" bloğu :

```
class Program
{
    static void bigNumberAlert()
    {
```

```

        Console.WriteLine("Event Triggered! Entered Number is bigger than
10.");
    }

    static void Main(string[] args)
    {
        NumberClass numberObj = new NumberClass();
        numberObj.BigNumber += new myCustomHandler(bigNumberAlert);

        while (true)
        {
            Console.WriteLine("Enter Number :");
            int enteredNumber = Convert.ToInt32(Console.ReadLine());
            numberObj.number = enteredNumber;
        }
    }
}

```

"NumberClass" sınıfından bir "numberObj" nesnesi türetilmiş ve bu sınıfın "BigNumber" olayına "myCustomHandler" delegeesi kullanılarak "bigNumberAlert" isimli yordam atanmıştır. Bu yordam olayın tetiklendiğine dair ekrana bilgi yazar. Sonsuz döngünün içinde ise kullanıcıdan sayı alınıp sınıfa gönderilmektedir.

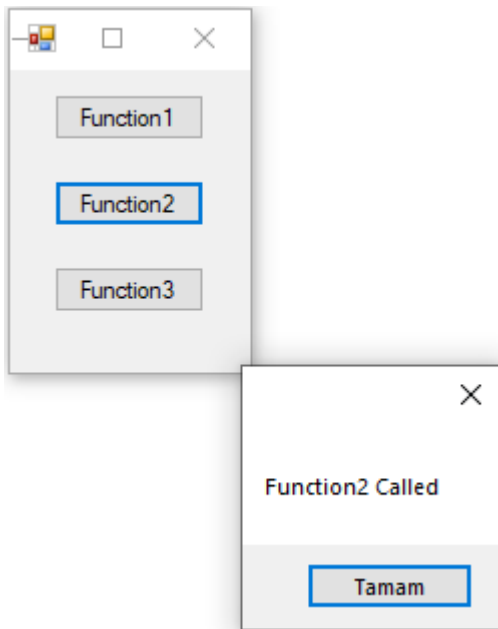
Uygulamanın Kaynak Kodları: [Example41](#)

Ad Alanları (Namespaces)

Aynı amaca hizmet eden bu sınıfları bir paket altında toplamak hiyerarşik yapı olarak aradığımızı daha kolay bulmamızı ve benzer üyeleri gruplamamızı sağlar. .NET bütün bu sınıfları ve türleri iyi bir şekilde organize edebilmek için isim uzayı(Namespaces) kavramını sıklıkla kullanır. Namespace'ler .NET sınıf kütüphanesindeki veri türlerini ve sınıfları kullanabilmemiz için C# dilinde **using** anahtar sözcüğü ile birlikte kullanılır ve derleyiciye bildirilir

Örnek 48

Bu uygulamada basılan butona göre hangi fonksiyonun çalıştırıldığı ekrana yazılmaktadır.



"FirstSpace" isminde bir ad alanı tanımlanması :

```
namespace FirstSpace
{
    class Class1
    {
        public void function1()
        {
            MessageBox.Show("Function1 Called");
        }
    }
}
```

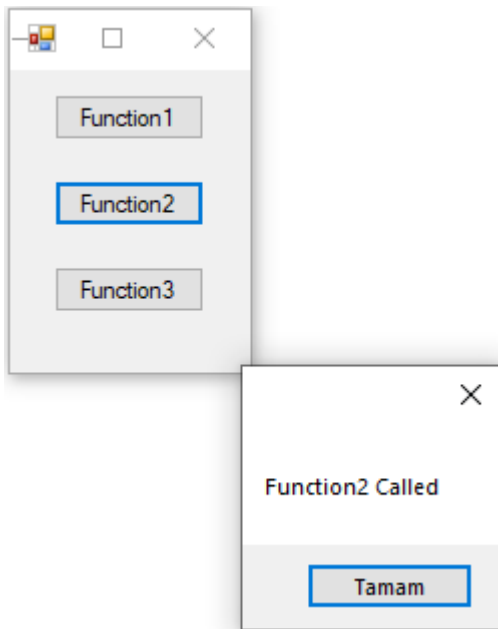
Uygulamanın Kaynak Kodları: [Example48](#)

İç İçe Ad Alanları (Nested Namespaces)

Ad Alanları iç içe tanımlanabilirler.

Örnek 48

Bu uygulamada basılan butona göre hangi fonksiyonun çalıştırıldığı ekrana yazılmaktadır.



"SecondSpace" ad alanı içinde "ThirdSpace" isimli ad alanı tanımlamanması :

```
namespace SecondSpace
{
    namespace ThirdSpace
    {
        class Class2
        {
            public void function2()
            {
                MessageBox.Show("Function2 Called");
            }
        }
    }
}
```

Uygulamanın Kaynak Kodları: [Example48](#)

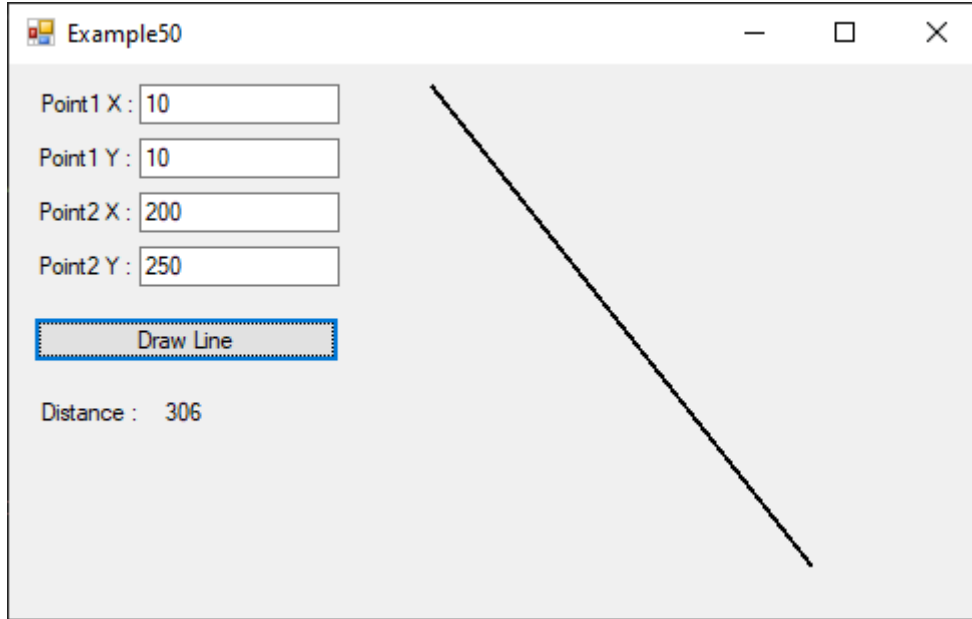
Yapı (Struct)

Bildiğiniz gibi sınıflar, referans tipidir. Bunun anlamı şudur: Sınıf nesneleri bir referans aracılığıyla erişilirler. Bu durum, doğrudan erişilen değer tiplerinden farklılık gösterir. Yine de, kimi zaman bir nesnenin, değer tiplerine erişir şekilde doğrudan erişilebilir olması yararlı olacaktır. Bunun nedenlerinden biri verimliliklidir. Sınıf nesnelerine bir referans aracılığıyla erişmek her erişime ilave yük bindirir. Ayrıca bellek alanı da harcar. Çok küçük nesneler için bu ekstra alan önemli olabilir. Bu sorunları dile getirmek için C#, yapıları önermektedir. Yapı (structure), sınıf ile aynıdır. Tek fark, yapının bir referans tipi yerine bir değer tipinde olmasıdır.

Yapılar struct anahtar kelimesi kullanılarak deklare edilir ve söz dizimsel olarak sınıflara benzer

Örnek 50

Bu uygulama kullanıcıdan aldığı iki noktanın koordinatları sınır olacak şekilde bir çizgi oluşturur. Çizilen çizginin uzunluğunu iki nokta arası uzaklık formülü ile hesaplayarak ekrana yazdırır.



Burada "Coordinate" isimli yapının tanımlanması görülmekte :

```
struct Coordinate
{
    public int x;
    public int y;

    public int distanceBetween(Coordinate point)
    {
        return (int)Math.Sqrt(Math.Pow((point.x - this.x), 2) +
Math.Pow((point.y -this.y), 2));
    }
}
```

Yapıların kullanılması :

```
Coordinate aPoint;
aPoint.x = 10;
aPoint.y = 10;

Coordinate bPoint = new Coordinate();
aPoint.x = 100;
aPoint.y = 100;
```

Burada bir yapının iki farklı kullanım yöntemi görülmektedir. Birinci yöntemde yapı, belleğin Stack kısmında tutulurken ikinci yöntemde "new" operatörü kullanıldığı için belleğin Heap kısmında muhafaza edilecektir.

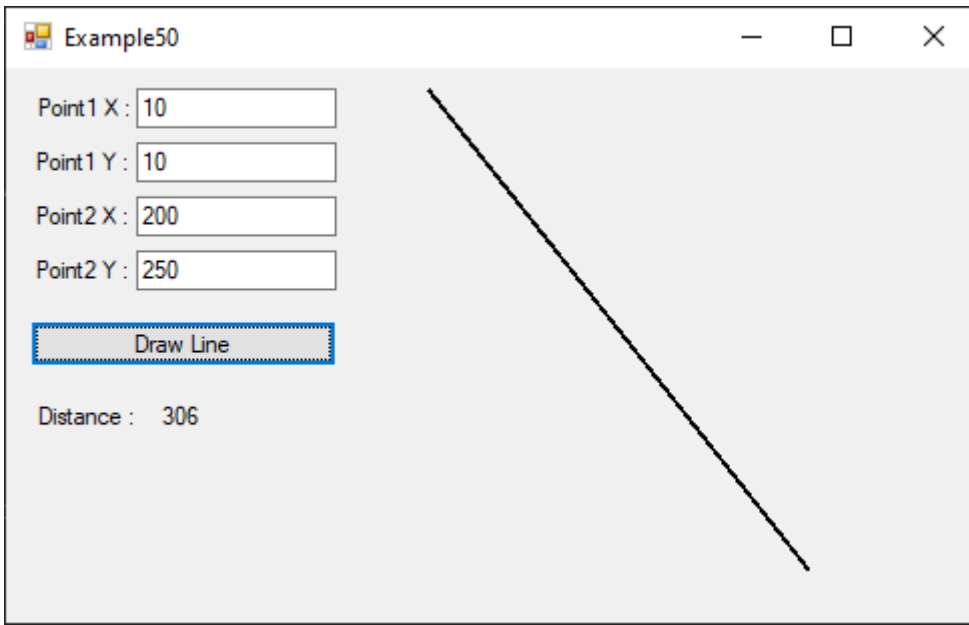
Uygulamanın Kaynak Kodları: [Example50](#)

Yapı İçerisinde Metot Oluşturmak (Methods in Struct)

Yapılar içinde sınıflarda oluşturulduğu gibi metot oluşturulabilir.

Örnek 50

Bu uygulama kullanıcıdan aldığı iki noktanın koordinatları sınır olacak şekilde bir çizgi oluşturur. Çizilen çizginin uzunluğunu iki nokta arası uzaklık formülü ile hesaplayarak ekrana yazdırır.



Burada "Coordinate" isimli yapının içinde "distanceBetween" isimli bir metodun tanımlanması görülmekte. Bu metod koordinatı verilen 2 nokta arasındaki mesafeyi hesaplayarak tam sayı tipinde döndürür.

```
struct Coordinate
{
    public int x;
    public int y;

    public int distanceBetween(Coordinate point)
    {
        return (int)Math.Sqrt(Math.Pow((point.x - this.x), 2) +
Math.Pow((point.y -this.y), 2));
    }
}
```

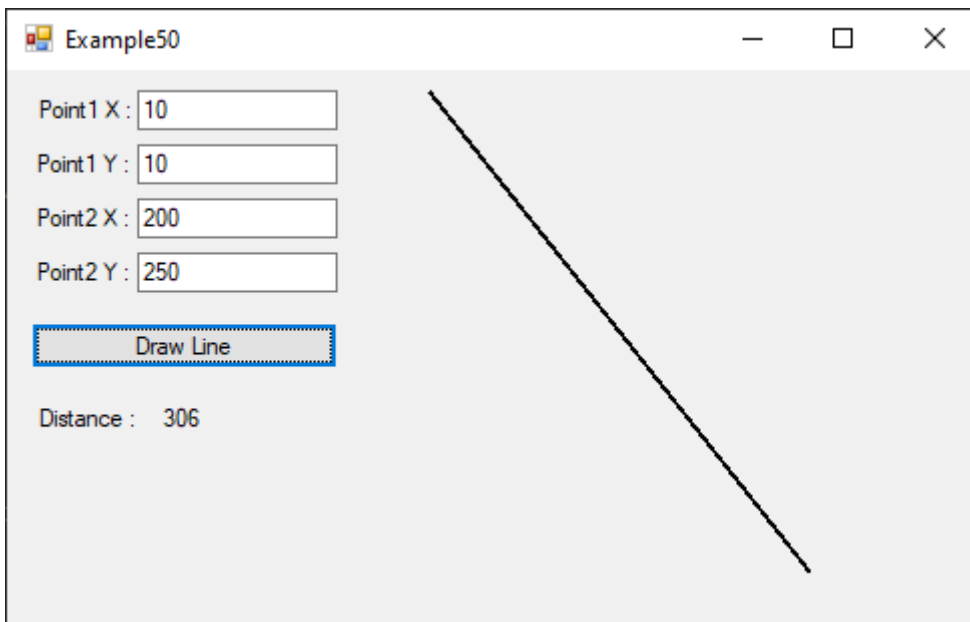
Uygulamanın Kaynak Kodları: [Example50](#)

Yapı İçerisinde Başka Bir Yapı Oluşturmak (Nested Struct)

Yapı içerisinde başka yapılar oluşturulabilir veya kullanılabilir. Yapı içinde yapı iki şekilde tanımlanır.

Örnek 50

Bu uygulama kullanıcıdan aldığı iki noktanın koordinatları sınır olacak şekilde bir çizgi oluşturur. Çizilen çizginin uzunluğunu iki nokta arası uzaklık formülü ile hesaplayarak ekrana yazdırır.



Yapılar içinde sadece o yapının kullanabileceği dışarıdan erişilemeyecek bir yapı tanımlanabilir. "Length" isimli yapı bu tip bir yapıya örnektir.

Yapılar içinde yapının dışında oluşturulmuş başka bir yapı kullanılabilir. 4. satırda görüldüğü gibi "Coordinate" isimli yapının kullanılması buna bir örnektir.

```
struct Edge
{
    Length _l;
    public Coordinate a, b;
    public int length {
        get
        {
            return _l.length;
        }
    }

    struct Length
    {
        public int length;
    }

    public Edge(int aX, int aY, int bX, int bY)
    {
        this.a.x = aX;
        this.a.y = aY;
        this.b.x = bX;
        this.b.y = bY;
        this._l.length = a.distanceBetween(b);
    }

    public void update()
    {
        this._l.length = a.distanceBetween(b);
    }
}
```



```
}  
  
}
```

Uygulamanın Kaynak Kodları: [Example50](#)

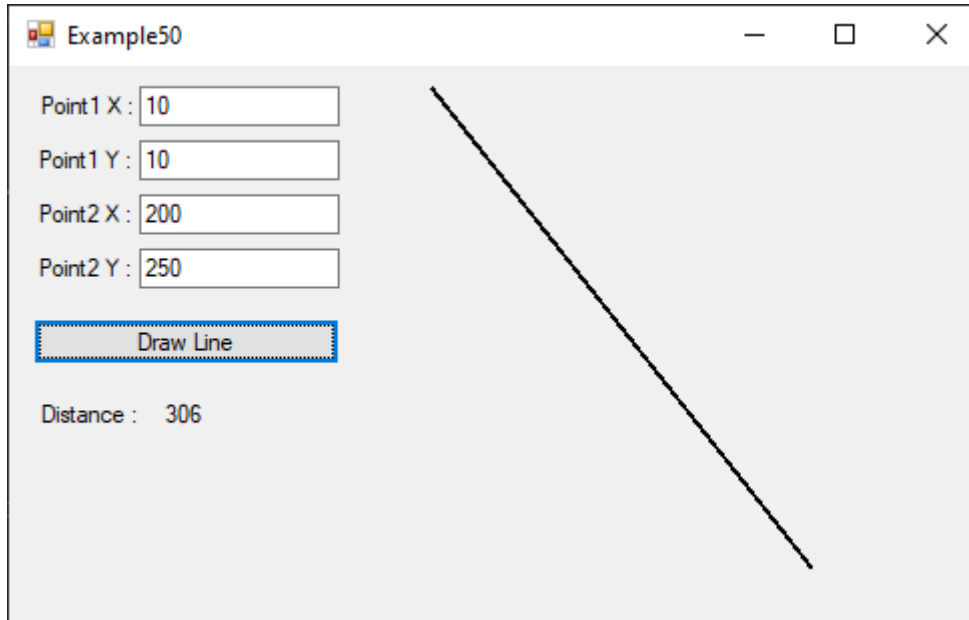
Yapılarda Yapılandırıcı (Constructors in Struct)

Yapılarda sadece parametrelili yapılandırıcı tanımlanabilir. Parametresiz yapılandırıcı tanımlanamaz. Bunun nedeni, varsayılan yapılandırıcının tüm yapılar için otomatik olarak tanımlı olmasıdır; bu varsayılan yapılandırıcı değiştirilemez.

Yapılarda yapılandırıcının çalıştırılması için "new" operatörü ile yapının nesnesi oluşturulmalıdır.

Örnek 50

Bu uygulama kullanıcıdan aldığı iki noktanın koordinatları sınır olacak şekilde bir çizgi oluşturur. Çizilen çizginin uzunluğunu iki nokta arası uzaklık formülü ile hesaplayarak ekrana yazdırır.



"Edge" isimli yapının ve onun yapılandırıcısının tanımlanması :

```
struct Edge  
{  
    Length _l;  
    public Coordinate a, b;  
    public int length {  
        get
```

```

        {
            return _l.length;
        }
    }

    struct Length
    {
        public int length;
    }

    public Edge(int aX, int aY, int bX, int bY)
    {
        this.a.x = aX;
        this.a.y = aY;
        this.b.x = bX;
        this.b.y = bY;
        this._l.length = a.distanceBetween(b);
    }

    public void update()
    {
        this._l.length = a.distanceBetween(b);
    }
}

```

Burada yapılandırıcı iki noktanın koordinatlarına ilk değer atamakta ve bu iki değer arasındaki uzaklığı hesaplamakta.

"Edge" isimli yapının kullanılması :

```
Edge aEdge = new Edge(0, 0, 0, 0);
```

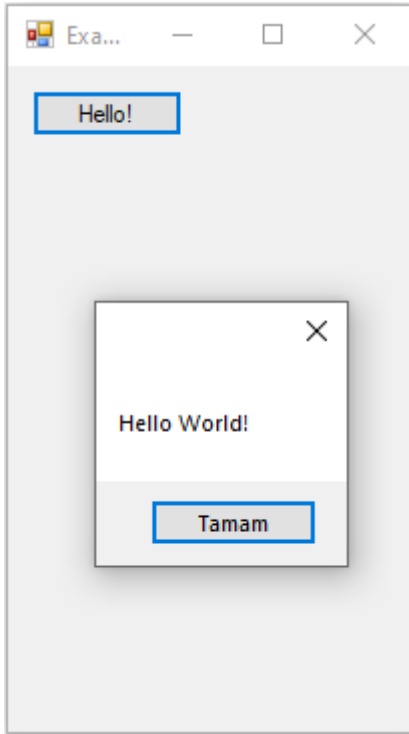
Uygulamanın Kaynak Kodları: [Example50](#)

Yapılarda Kalıtım

Yapılar, diğer yapıları veya sınıfları kalıtım yoluyla aktaramaz ya da diğer yapılar veya sınıflar için temel sınıf olarak kullanılamazlar. Bununla birlikte, bir yapı bir ya da daha fazla arayüzü uygulayabilir. Bu arayüzler, yapının isminden sonra virgül ile ayrılmış bir liste halinde belirtilir. Tıpkı sınıflar gibi, yapı üyeleri de metot, alan, indeksleyici, özellik, operatör metotlarını ve olayları kapsar.

Örnek 55

Bu uygulama butona basıldığında ekrana "Hello World!" mesajı getirmektedir.



"MyStruct" yapısı ve türetildiği arayüz :

```
interface IHello
{
    void helloWorld();
}

struct MyStruct : IHello
{
    public void helloWorld()
    {
        MessageBox.Show("Hello World!");
    }
}
```

Uygulamanın Kaynak Kodları: [Example55](#).

Generic İfadeler

Generic ifadeler bir sınıf veya metodun tipinin kullanılmadan önce belirlenmesini sağlar. Bu sayede her veri tipiyle çalışabilen sınıflar veya metotlar yazılabilir. Parametre türü olarak da tanımlanabilirler. Cast, boxing-unboxing işlemlerinin maliyetini ve riskini ortadan kaldırır. Kullanılırken "" ifadesinde x yerine istenilen tip yazılır.

C# System.Collections.Generic alan adında kullanıma hazır generic yapılar sunar. Kullanıma hazır yapılardan bazıları : List, LinkedList, Stack, Queue, HashSet .Bu yapılar haricinde kendi generic sınıflarımızı da oluşturabiliriz.

Örnek 51

Bu uygulamada kullanıcıdan tam sayı ve ondalıklı sayı listesine veri eklemesi istenmekte ve hatalı tipte veri girerse uyarılmakta.

Generic bir liste tanımlama :

```
List<int> intList = new List<int>();  
List<double> doubleList = new List<double>();
```

Burada "intList" adında tam sayı ve "doubleList" adında ondalıklı sayı tipinde iki adet generic liste tanımlanmıştır. Listenin hangi tipte tanımlandığı küçüktür ve büyüktür işareti arasındaki ifadeden anlaşılmaktadır.

Generic bir listeye veri ekleme :

```
intList.Add(51);  
doubleList.Add(5.5);
```

Burada tanımladığımız "intList" listesine 51 değeri, "doubleList" listesine 5.5 değeri eklenmektedir.

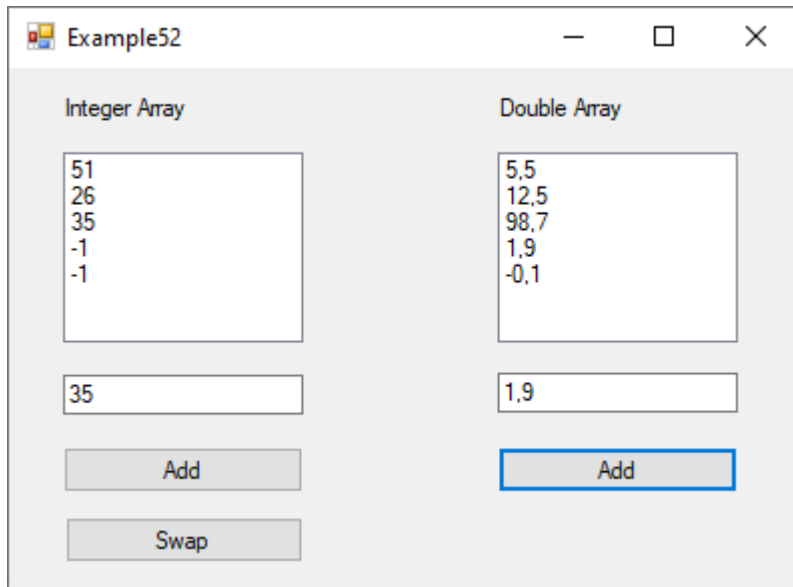
Uygulamanın Kaynak Kodları: [Example51](#)

Generic Sınıflar

Kendi generic sınıflarımızı oluşturabiliriz. Bunun için sınıf adının yanına "<T>" eklememiz ve tip yerine "T" kullanmamız yeterlidir.

Örnek 52

Bu uygulamada kullanıcıdan tam sayı ve ondalıklı sayı dizisine veri eklemesi istenmekte ve hatalı tipte veri girerse uyarılmakta.



"GenericArray" isimli generic olarak tanımlanmış dizi oluşturan sınıf :

```
class GenericArray<T>
{
    private T[] array;
    private int size;
    private T defaultVal;

    public GenericArray(int size, T defaultVal)
    {
        this.size = size;
        this.defaultVal = defaultVal;
        array = new T[size+1];
        for (int i=0; i<size; i++)
        {
            array[i] = defaultVal;
        }
    }
}
```

```

    }

    public void swapFirstSecond()
    {
        swap<T>(ref array[0], ref array[1]);
    }

    public static void swap<T>(ref T lhs, ref T rhs)
    {
        T temp;
        temp = lhs;
        lhs = rhs;
        rhs = temp;
    }

    public T this[int index]
    {
        get
        {
            if (index >= 0 && index<=size-1)
            {
                return array[index];
            }
            else
            {
                return defaultVal;
            }
        }
        set
        {
            if (index >= 0 && index <= size - 1)
            {
                if (value.GetType() == typeof(T))
                {
                    array[index] = value;
                }
                else
                {
                    throw new System.ArgumentException("Wrong Type");
                }
            }
        }
    }
}

```

Görüldüğü gibi sınıfın isminin yanında "" ifadesi var ve sınıfın içinde tür yerine T kullanılmakta. Yapılandırıcı kod bloğunda diziye varsayılan değerler atanıyor. Sınıfa indeksleyici tanımlanmış. Veri eklerken dizi için sınır ve tip kontrolü yapılmakta.

"GenericArray" isimli generic olarak tanımlanmış sınıf kullanılarak istenilen tipte dizi oluşturma:

```
GenericArray<int> intList = new GenericArray<int>(5, -1);
GenericArray<double> doubleList = new GenericArray<double>(5, -0.1);
GenericArray<string> stringList = new GenericArray<double>(5, "N.A");

intList[0] = 51;
intList[1] = 26;
intList[2] = 35;

doubleList[0] = 5.5;
doubleList[1] = 12.5;
doubleList[2] = 98.7;
doubleList[3] = 1.9;

stringList[0] = "Melih";
stringList[1] = "Hello World";
```

Uygulamanın Kaynak Kodları: [Example52](#)

Generic Sınıflar İçerisindeki Static Metodları İşletmek

Generic sınıflar içerisinde static metodlar tanımlanabilir. Normal sınıflarla herhangi bir farkı yoktur.

Örnek 52

Bu uygulamada kullanıcıdan tam sayı ve ondalıklı sayı dizisine veri eklemesi istenmekte ve hatalı tipte veri girerse uyarılmakta.

Example52

Integer Array

51
26
35
-1
-1

35

Add

Swap

Double Array

5.5
12.5
98.7
1.9
-0.1

1.9

Add

"GenericArray" isimli generic olarak tanımlanmış dizi oluşturan sınıf :

```
class GenericArray<T>
```

```
{
    private T[] array;
    private int size;
    private T defaultVal;

    public GenericArray(int size, T defaultVal)
    {
        this.size = size;
        this.defaultVal = defaultVal;
        array = new T[size+1];
        for (int i=0; i<size; i++)
        {
            array[i] = defaultVal;
        }
    }

    public void swapFirstSecond()
    {
        swap<T>(ref array[0], ref array[1]);
    }

    public static void swap<T>(ref T lhs, ref T rhs)
    {
        T temp;
        temp = lhs;
        lhs = rhs;
        rhs = temp;
    }

    public T this[int index]
    {
        get
        {
            if (index >= 0 && index<=size-1)
            {
                return array[index];
            }
            else
            {
                return defaultVal;
            }
        }
        set
        {
            if (index >= 0 && index <= size - 1)
            {
                if (value.GetType() == typeof(T))
                {
                    array[index] = value;
                }
                else
                {
                    throw new System.ArgumentException("Wrong Type");
                }
            }
        }
    }
}
```



```
    }  
    }  
}
```

"swap" metodu "static" tanımlanmıştır.

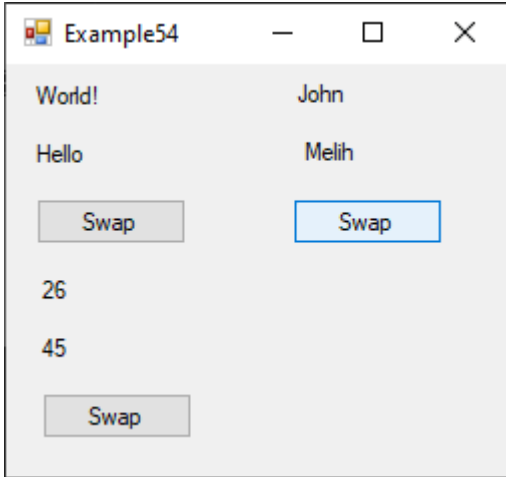
Uygulamanın Kaynak Kodları: [Example52](#)

Generic Metodlar (Generic Methods)

Metotlar generic olarak tanımlanabilirler.

Örnek 54

Bu uygulama "Swap" butonuna basıldığında üzerindeki yazıları birbirleriyle değiştirmektedir.



Generic olarak tanımlanmış bir "swap" isimli parametre olarak aldığı değerleri birbirleriyle değiştiren metot :

```
static void swap<T>(ref T lhs, ref T rhs)  
{  
    T temp;  
    temp = lhs;  
    lhs = rhs;  
    rhs = temp;  
}
```

"swap" metodunun yanındaki "<T>" ifadesinden ve tip yerine "T" kullanılmasından anlaşıldığı üzere generic bir metot. Parametre olarak referans almakta. Bu sayede tek bir fonksiyon ile tipten bağımsız olarak nesneler ve değişkenler kendi tipleri arasında birbirleriyle değiştirilebilir.

"Swap" fonksiyonunun çeşitli tiplerde uygulanması :

```
string text1 = "Hello";  
string text2 = "World!";  
swap<string>(ref text1, ref text2);  
  
int number1 = 45;  
int number2 = 26;  
swap(ref text1, ref text2);  
  
Class1 class1 = new Class1("Melih", 21);  
Class1 class2 = new Class1("John", 35);  
swap(ref class1, ref class2);
```

Uygulamanın Kaynak Kodları: [Example54](#).

Kaynakça

Kitap

- H. Schildt, Yağcı Duygu Arbatlı., and Tüzel Selçuk, *Herkes için C#*. İstanbul: Alfa, 2002.

Web Sayfaları

- *C# docs – get started, tutorials, reference.* | Microsoft Docs. [Online]. Available: <https://docs.microsoft.com/tr-tr/dotnet/csharp/>. [Accessed: 02-Sep-2020].
- Contributors to Wikimedia, “C# Programming – Wikibooks, open books for an open world,” *Wikibooks, open books for an open world*, 15-Apr-2020. [Online]. Available: https://en.wikibooks.org/wiki/C_Sharp_Programming. [Accessed: 02-Sep-2020].
- Tao, D. (n.d.). Re: Are set-only properties bad practice? [Web log comment]. Retrieved from <https://stackoverflow.com/a/4564971>
- “Java Nested and Inner Class,” *Java Nested and Inner Class (With Examples)*. [Online]. Available: <https://www.programiz.com/java-programming/nested-inner-class>. [Accessed: 02-Sep-2020].
- *Nested Class (İç içe sınıflar)*. [Online]. Available: [https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/nested-class-\(iç-içe-sınıflar\)](https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/nested-class-(iç-içe-sınıflar)). [Accessed: 02-Sep-2020].
- “C# – Generics,” *C# – Generics – Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/csharp/csharp_generics.htm. [Accessed: 02-Sep-2020].
- “C# – Indexers,” *C# – Indexers – Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/csharp/csharp_indexers.htm. [Accessed: 02-Sep-2020].
- “Abstract Classes in C#,” *Abstract Classes in C# – Tutorialspoint*. [Online]. Available: <https://www.tutorialspoint.com/Abstract-Classes-in-Csharp>. [Accessed: 02-Sep-2020].
- “C# – Interfaces,” *C# – Interfaces – Tutorialspoint**. [Online]. Available: https://www.tutorialspoint.com/csharp/csharp_interfaces.htm. [Accessed: 02-Sep-2020].

Blog Yazıları

- O. Salkaya, “C# Delegate Nedir?,” *Onur SALKAYA (Matematik Mühendisi): C# Delegate Nedir?*.
- B. S. Şenyurt , “C# Temelleri – Olayları(Events) Kavramak,” *Burak Selim Şenyurt | C# Temelleri – Olayları(Events) Kavramak.* .
- ankita_saini, “typeof Operator Keyword in C#,” *typeof Operator Keyword in C# – GeeksforGeeks.* .
- C. Çözvelioğlu, “Abstract Class ve Interface Arasındaki Farklar Nelerdir?,” *Abstract Class ve Interface Arasındaki Farklar Nelerdir? | by Ceyhun Çözvelioğlu | Software Development Turkey | Medium.* .