

Görsel Programlama

Ders 2

Dr. Öğr. Üyesi Mehmet Dinçer Erbaş
Bolu Abant İzzet Baysal Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü

Nesne odaklı programlama

- Sınıflar, oluşturacağımız yazılımda kullanacağımız nesneleri sınıflandırmak için kullandığımız araçlardır.
 - 📖 Bu sayede yazılımın kullanacağı verileri düzenlemiş oluruz.
 - 📖 Bu işlem sadece programcıların değil günlük hayatta herkesin yaptığı bir işlemdir.
 - 📖 Örnek: Araba sınıfı
 - Ortak davranışlar:
 - 📖 Yönlendirilebilir, durdurulabilir, hızlandırılabilir vb.
 - Ortak özellikler:
 - 📖 Direksiyon, motor, koltuk, kapı vb.
 - Bu ortak davranış ve özelliklere sahip bütün nesneleri anlatmak için araba sözcüğünü kullanırız.
 - Araba dediğinizde herkes ne tür bir araçtan bahsettiğinizi anlar.

Sınıflar ve nesneler

- İletişimde olduğu gibi bir sorunun çözümünün oluşturulmasında farklı kavramları sınıflandırmak ve bu sınıfları bir programlama dili ile modellemek oldukça yararlıdır.
 - 📖 Bu işlem Visual C# gibi nesne yönelimli programlama dillerinin kullandığı yöntemdir.
- Kapsülleme önemli bir sınıf tanımlama aracıdır.
 - 📖 Yöntem olarak sınıfın elemanı olan nesneler ile ilgili özellikler ve nesnelere uygulanacak işlemleri bir araya koymadır.
 - 📖 Bu sayede sınıfı kullanan programlar o sınıfın ne şekilde çalıştığını bilmek zorunda değildir.
 - Sadece sınıfın bir örneğini yaratır ve o sınıfın kapsüllenmiş yöntemlerini çalıştırır.
 - 📖 Yöntemler istenilen işlemleri gerçekleştirdikleri sürece geri kalan detaylar program tarafından bilinmez.

Sınıflar ve nesneler

- Kapsülleme

📖 Örneğin Console.WriteLine yöntemini kullandığımızda ne tür işlemler yapıldığını bilmek zorunda değiliz.

- Tek istediğimiz şey ekranda belli bir metnin görüntülenmesidir.
- Kendimize uygun Console.WriteLine versiyonunu seçer ve uygun girdiler ile ekrana istediğimiz metni yazdırırız.

📖 Sonuç olarak kapsüllemenin iki önemli fonksiyonu vardır:

- Yöntem ve verileri bir sınıf içerisinde birleştirir ve bu sayede sınıflandırmayı destekler.
- Yöntem ve verilerin kullanımını denetler.

Sınıflar ve nesneler

- C# dilinde yeni bir sınıf tanımlamak için **class** anahtar sözcüğünü kullanılır.
- Sınıfın veri ve yöntemleri sınıf tanımının gövde kısmında tanımlanır.

 Örnek: Circle sınıfı

```
class Circle
{
    double Area()
    {
        return Math.PI * radius * radius;
    }

    int radius;
}
```

Sınıflar ve nesneler

- Örnekte görüldüğü üzere Math sınıfı matematik sabitleri içeren alanlar ve matematiksel hesaplamalar içerir.

📖 Örneğin `Math.PI = 3.14159265358979323846`

- Sınıf gövdesinde `Area()` gibi fonksiyonlar ve `radius` gibi alanlar içerir.
- Bu sınıfa ait bir nesne yaratmak için `new` anahtar sözcüğü kullanılır

📖 `Circle c;` // Bir Circle değişkeni yaratır

📖 `c = new Circle();` // Başlangıç değeri atar

- Sınıfa ait bir nesneye direk olarak ilk değer atayamazsınız. Örneğin:

📖 `int i;`

📖 `i = 42;`

- Yapılabilir ancak

📖 `Circle c;`

📖 `c = 42`

- Yapılamaz.


Sınıflar ve nesneler

- Sınıfın bir örneğini aynı türden başka bir örneğe atayabilirsiniz.


 Circle c;

 c = new Circle();


 Circle d;

 d = c;

- Sınıflar ile ilgili bir diğer önemli konu erişebilirliktir.


 Önceki slaytta yapılan şekilde sınıf tanımlandığında sınıfın veri veya fonksiyonlarına dışardan erişim şansı yoktur.

- Yani bir Circle nesnesi yaratsanız bile radius alanına veya area() fonksiyonuna erişemezsiniz.


 Sınıfların alan ve fonksiyonlarına erişimi kontrol edebilmek için public ve private anahtar kelimelerini kullanabiliriz.

Sınıflar ve nesneler

- Erişebilirlik

 Bir yöntem veya alan sadece sınıfın içerisinde erişebilir olmalı ise o yöntem veya alan private ismi ile tanımlanır.

- Varsayılan ayar olarak herhangi bir anahtar kelime içermeyen alan veya fonksiyonlar private olur ancak bu durumu açık bir biçimde belirtmekte yarar vardır.

 Bir yöntem veya alan hem sınıf içerisinde hem de dışarıdan erişilebiliyorsa, o yöntem veya alan public olarak tanımlanır.

```
class Circle
{
    public double Area()
    {
        return Math.PI * radius * radius;
    }

    private int radius;
}
```


Sınıflar ve nesneler

- Erişebilirlik

📖 Alan ve yöntem tanımlarken aşağıda belirtilen isimlendirme kurallarına uymak gerekir:

- Sınıf içerisindeki public tanımlayıcılar büyük harf ile başlamalıdır.
- Sınıf içerisindeki private tanımlayıcılar küçük harf ile başlar.
- 📖 Bu kurala tek istisna kurucuların adlarıdır. Sınıf içerisindeki private kurucular büyük harf ile başlar.

📖 Kurucular → Constructor

- Kurucu bir sınıfın örneğini yarattığınızda otomatik olarak çalışan özel bir yöntemdir.
- Sınıf ile aynı ada sahip fonksiyonlardır.
- Parametre alabilirler.
- Herhangi bir değer döndürmezler.

→ `public classname () {
}`

Sınıflar ve nesneler

- Erişebilirlik

- 📖 Kurucular

- Oluşturduğunuz sınıfın kurucu fonksiyonunu tanımlamazsanız derleyici tarafından otomatik olarak bir kurucu oluşturulur ve kullanılır.
 - Kendi kurucunuzu oluşturmak için
 - 📖 Sınıf ile aynı ada sahip
 - 📖 Tanımı public olan bir fonksiyon oluşturulmalıdır.

```
class Circle
{
    public Circle() // varsayılan kurucu
    {
        radius = 0;
    }

    public double Area()
    {
        return Math.PI * radius * radius;
    }

    private int radius;
}
```

Sınıflar ve nesneler

- Kurucular

📖 Önceki slaytta gösterilen sınıfta Circle nesnesi oluşturulabilir ve Area yöntemi çalıştırılabilir.

📖 Ancak bu sınıfta bir sorun mevcuttur.

- Oluşturulan bütün nesnelerin radius değeri 0 olur ve bunu değiştirmenin yolu yoktur.

📖 Bu sorunu çözmek için kurucuya aşırı yükleme yaparak ilk değer atanan yeni bir kurucu yöntem oluşturmalıyız.

- Console.WriteLine yönteminde olduğu gibi farklı girdiler alan birden fazla Circle() yöntemi yazacağız.



Override

Sınıflar ve nesneler

```
class Circle
{

    public Circle() // varsayılan kurucu
    {
        radius = 0;
    }

    public Circle(int initialRadius) // aşırı yüklenmiş kurucu
    {
        radius = initialRadius;
    }

    public double Area()
    {
        return Math.PI * radius * radius;
    }

    private int radius;
}
```


Sınıflar ve nesneler


- Yeni bir nesne yaratırken gerekli argümanlar verilmelidir.


 Circle c;


 c = new Circle(45);


- Kısmi sınıflar

 Bir sınıf çok sayıda yöntem, alan, kurucu ve diğer yöntemleri içerir.

 Bu sebeple sınıf oldukça genişleyebilir.

 Geniş bir sınıfı kolay yönetilbilmek için ufak parçalar oluşturmak mümkündür.

 Bir sınıfı birçok dosyaya ayırdığımızda partial anahtar sözcüğünü kullanabilirsiniz.

 Örneğin Circle sınıfını iki parçaya bir sonraki slaytta görülecek şekilde iki parçaya ayırabiliriz.

Sınıflar ve nesneler

```
partial class Circle
```

```
{  
    public Circle() // varsayılan kurucu  
    {  
        radius = 0;  
    }  
  
    public Circle(int initialRadius) // aşırı yüklenmiş kurucu  
    {  
        radius = initialRadius;  
    }  
}
```

circ1.cs

```
partial class Circle
```

```
{  
    public double Area()  
    {  
        return Math.PI * radius * radius;  
    }  
  
    private int radius;  
}
```

circ2.cs

Sınıflar ve nesneler

- Static yöntemler

📖 Daha önceki örneklerimizde Math sınıfına ait veri ve yöntemleri kullandık

- Math.PI;
- Math.Sqrt(25);

📖 Burada Math sınıfına ait bir nesne yaratmak yerine direk olarak metodu çağırıyoruz.

📖 Math sınıfına ait bir nesne yaratsaydık, aşağıdaki gibi yapabilirdik:

- Math m = new Math();
- double d = m.Sqrt(25);

📖 Ancak bu verimli bir kullanım şekli değildir.


- Yapmak istediğimiz standart bir yöntemi kullanmak. Bu sebeple nesne oluşturmamıza gerek yoktur.

Sınıflar ve nesneler

- C# dilinde tüm yöntemler bir sınıf içinde bildirilmelidir.
- Ancak bir yöntem veya alan static olarak tanımlanırsa sınıfın adını kullanarak yöntemi çağırabilir yada alana erişebiliriz.

 Bu durumda o sınıfa ait bir nesne yaratmamıza gerek kalmaz.

```
class Math
{
    public static double Sqrt(double d) { ... }
    ...
}
```

 Statik bir yöntem tanımladığınızda bu yöntem sadece statik olarak tanımlanmış alanlara erişebilir.

Sınıflar ve nesneler

- Static alanlar

- 📖 Paylaşılan bir alan

- Alan tanımlarken `static` anahtar sözcüğünü kullanılabilir.
 - Bu sayede sınıfa ait tüm nesneler tarafından paylaşılan bir alan yaratabiliriz.

- 📖 Statik olmayan alanlar her bir nesne oluşumu için yereldir.

- Önceden oluşturduğumuz Circle sınıfına oluşturulan circle sayısını sayan bir paylaşılmış statik alan yaratalım.

Sınıflar ve nesneler


```
class Circle
{
    public Circle() // varsayılan kurucu
    {
        radius = 0;
        NumCircles++;
    }


    public Circle(int initialRadius) // aşırı yüklenmiş kurucu
    {
        radius = initialRadius;
        NumCircles++;
    }


    ...
    private int radius;
    public static int NumCircles = 0;
}
```

Sınıflar ve nesneler

- const anahtar sözcüğü

 Statik bir alanın değerinin hiç değişmeyeceğini biliyorsak const anahtar sözcüğü ile bu alanı oluşturabiliriz.

 const anahtar sözcüğü ile oluşturulan alanlarda static kelimesi bulunmaz ancak bu alanlar static alanlardır.

 Sadece int veya double türünde alanlar const olarak tanımlanabilir.

```
class Math
{
    ...
    public const double PI = 3.14159265358979323846;
}
```

Sınıflar ve nesneler

- Static sınıflar

- 📖 C# dilinde bir sınıf static olarak tanımlanabilir.
- 📖 Static bir sınıf sadece static yöntem ve alanlar içerir.
- 📖 Bu tür sınıfların oluşturulmasında amaç standart olan bazı yöntem ve alanların toplanmasıdır.
- 📖 Static sınıfın oluşum verisi olmaz veya new anahtar kelimesi ile bu sınıfa ait bir nesne oluşturulamaz.
 - Bir başlangıç değeri atamanız gerekiyorsa static sınıf varsayılan bir kurucuya sahip olabilir.
 - 📖 Ancak kurucunun da static olarak bildirilmesi gerekir.
- 📖 Math sınıfının sadece static üyeleri içeren bir sürümünü bir sonraki slaytta görüldüğü gibi oluşturabiliriz.

Sınıflar ve nesneler

```
public static class Math
{
    public static double Sin(double x) {...}
    public static double Cos(double x) {...}
    public static double Sqrt(double x) {...}
    ...
}
```

Sınıflar ve nesneler

- Anonim sınıflar

- 📖 Anonim sınıflar adı olmayan sınıflardır.

- 📖 New anahtar sözcüğü kullanarak anonim sınıfının içinde olmasını istediğimiz alanlar ve değerler tanımlanabilir.

- benimAnonimNesnem = new { Isim = "Ali", Yas = 21};

- 📖 Bu durumda Isim ve Yas isminde iki public alan bulunur.

- 📖 Bir anonim sınıf tanımlandığında derleyici sınıf için kendi adını üretir.

- 📖 Aynı anonim sınıfa ait bir başka nesne yaratabiliriz.

- digerAnonimNesnem = new { Isim = "Ayse", Yas = 22};

- 📖 Alan özelliklerine bakarak derleyici iki nesnenin aynı türden olduğunu tespit eder.

- digerAnonimNesnem = benimAnonimNesnem;