



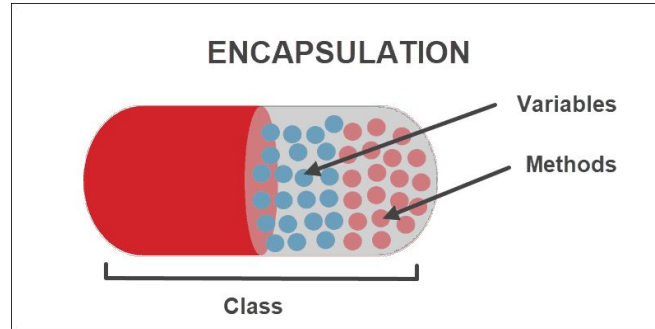
# Nesne Yönelimli Programlama

Kapsülleme (Encapsulation)

Öğr. Gör. İbrahim AYAZ

## Kapsülleme nedir ?



- © Kapsülleme, nesne yönelimli programlamanın en temel özelliklerinden biridir. Verilerin tek bir birim altında sarılması olarak tanımlanır. **Başka bir deyişle bir nesnenin belirli özellik ve metotlarının erişiminin kısıtlanması ve saklanmasıdır.** Kodu ve onun işlediği verileri birbirine bağlayan mekanizmadır. Farklı bir anlatımla kapsülleme, verilere bu kalkanın dışındaki kod tarafından erişilmesini engelleyen koruyucu bir kalkanıdır.





***NOT***

Kapsüllemedeki kısıtlamalar kötü amaçlı kullanım ve istenmeyen veri girişlerini engeller.

- 
- ◎ Teknik olarak kapsüllemeye, bir sınıfın değişkenleri veya verileri başka herhangi bir sınıftan gizlenir ve yalnızca içinde bildirildikleri kendi sınıfının herhangi bir üye işlevi aracılığıyla erişilebilir.
  - ◎ Kapsüllemeye olduğu gibi, bir sınıftaki veriler diğer sınıflardan gizlenir, dolayısıyla veri gizleme olarak da bilinir.
  - ◎ Kapsülleme şu şekilde gerçekleştirilebilir: Sınıftaki **tüm değişkenleri (alanları) özel (private)** olarak bildirmek ve değişkenlerin değerlerini ayarlamak ve almak için sınıf içerisinde **C# özelliğini (property)** kullanmak.
- 



## ***HATIRLATMA***

**Public** : Öğenin kod içerisindeki **her yerden** erişilebileceğini belirtir.

**Private** : Öğenin **sadece bulunduğu sınıf** içerisinde erişilebileceğini belirtir.

**Protected** : Öğenin **bulunduğu sınıftan veya o sınıftan türetilen sınıflardan** erişilebileceğini belirtir.

**Internal** : Öğenin **tanımlandığı namespace** içerisinde **her yerden** erişilebilir.

# Sınıf içerisinde alan(field) kapsülleme

```
C# Encapsulation.cs 2 X
C# Encapsulation.cs > Student > Age

2 public class Student
3 {
4     private string name;
5     private string surname;
6     private int age;
7     public int Age {
8         get{ return age; }
9         set{ age=value; }
10    }
11    public string Name{
12        get{
13            return name;
14        }
15        set{
16            name=value;
17        }
18    }
19    public string Surname{
20        get{
21            return surname;
22        }
23        set{
24            surname=value;
25        }
26    }
27 }
```

Alanlar(Fields) veya Sınıf değişkenleri

Özellikler(Property)

Yandaki sınıfta, değişkenler private olarak bildirildiğinden Student sınıfı kapsüllenmiştir. Bu private değişkenlere erişmek için, private alanların değerlerini almak ve ayarlamak amacıyla **get** ve **set** yöntemini içeren Name, Surname ve Age özelliklerini kullanıyoruz. Özellikler diğer sınıflardan erişilebilmeleri için **public** olarak tanımlanırlar.



## ***DİKKAT***

*Özellikler, diğer nesnelerin verilere erişmesi ve bunları değiştirmesi için uygun ve tutarlı bir yol sağlarken, bir sınıfın dahili verilerine erişimini kontrol etmenize olanak tanır.*

*C# özellik tanımlarında kullanılan **get** ilgili alana **okuma** (veriyi getirme), **set** ise **yazma** (veriyi değiştirme) yeteneği kazandırır.*

## Kapsüllemenin Avantajları

- ◎ **Veri Gizleme:** Kullanıcının sınıfın iç yapısı hakkında hiçbir fikri olmayacaktır. Sınıf değerlerinin değişkenlerde nasıl saklandığı kullanıcı tarafından görülmeyecektir.
- ◎ **Arttırılmış Esneklik:** Sınıfın değişkenlerini ihtiyacımıza göre salt okunur veya salt yazılır hale getirebiliriz. Değişkenleri salt okunur yapmak istiyorsak kodda yalnızca **Get** erişim aracını kullanmamız gerekir. Değişkenleri salt yazılır yapmak istiyorsak yalnızca **Set** erişim aracını kullanmamız gerekir.
- ◎ **Yeniden Kullanılabilirlik:** Kapsülleme aynı zamanda yeniden kullanılabilirliği de artırır ve yeni gereksinimlere göre değiştirilmesi kolaydır.
- ◎ **Test Edilebilirlik:** Kapsüllenmiş kodun birim testi için test edilmesi





Kapsüllemenin arkasındaki fikir, bir sınıfın uygulama ayrıntılarını dış dünyadan gizli tutmak ve yalnızca kullanıcıların sınıfla kontrollü ve güvenli bir şekilde etkileşime girmesine olanak tanıyan genel bir arayüzü açığa çıkarmaktır. Bu, yazılım sistemlerinin tasarımında modülerliğin, sürdürülebilirliğin ve esnekliğin desteklenmesine yardımcı olur.

## Kapsüllemenin Mantığı

Kitap
+ ad: string + yazarAdi: string + sayfaSayisi:int

Soru 1: Sayfa sayısı -15 olan kitap var mı ?

Soru 2: Kitap sınıfının değişkenlerine dışarıdan erişmek ne kadar doğrudur ?

```

2 public class Kitap{
3     | private string ad;
4     | private string yazarAdi;
5     | private int sayfaSayisi;
6     | Kitap(string ad, string yazarAdi, int sayfaSayisi){
7     |     this.ad=ad;
8     |     this.yazarAdi=yazarAdi;
9     |     setSayfaSayisi(sayfaSayisi);
10    | }
11    | public string getAd()=> this.ad;
12    | public string getYazarAd(){
13    |     return this.yazarAdi;
14    | }
15    | public int getSayfaSayisi()=> this.sayfaSayisi;
16    | public void setAd(string ad)=> this.ad=ad;
17    | public void setYazarAd(string yazarAd)=>this.yazarAdi=yazarAd;
18    | public void setSayfaSayisi(int sayfaSayisi){
19    |     if(sayfaSayisi<0)
20    |         this.sayfaSayisi=0;
21    |     else
22    |         this.sayfaSayisi=sayfaSayisi;

```

## Kitap

- ad: string
- yazarAdi: string
- sayfaSayisi:int

- + Kitap(ad:string, yazarAdi:string, sayfaSayisi:int)
- + getAd():string
- + getYazarAd():string
- + getSayfaSayisi():int
- + setAd(ad:string):void
- + setYazarAd(yazarAd:string):void
- + setSayfaSayisi(sayfaSayisi:int):void

Bir önceki slaytta public olan değişkenler bu örnekte private olarak değiştirilmiş ve sayfaSayisi alanı kapsüllenerek korunmuştur. Ayrıca private alanlara erişmek için get ve set işlemleri için metodlar kullanılmıştır. **C# get ve set ifadeleri için metotlar oluşturmak yerine direkt set ve get erişim araçlarıyla özellik oluşturulabilir.**

## Bir örnek

```
/// <summary>  
/// Bir Kitap sınıf oluşturunuz.  
/// Bu sınıf içerisinde sayfa sayısı negatif olma problemi vardır.  
/// Bu problemi kapsülleme tekniğini kullanarak problemi çözen C# kodunu yazınız.  
/// </summary>
```

0 başvuru

```
public class Kitap
```

```
{
```

```
    private int sayfaSayisi;
```

0 başvuru

```
    public int SayfaSayisi
```

```
{
```

```
        get { return sayfaSayisi; }
```

```
        set
```

```
{
```

```
            if (value < 0)
```

```
{
```

```
                throw new Exception("Sayfa Sayısı Negatif olamaz kardeş.");
```

```
            }
```

```
            else
```

```
{
```

```
                sayfaSayisi = value;
```

```
            }
```

```
        }
```

```
    }
```

```
/// Bir BankaHesap sınıfı oluşturunuz.  
/// Bu sınıf içerisinde bakiye alanı şifre olmadan değiştirilebiliyor.  
/// Bu problemi kapsülleme tekniğini kullanarak çözünüz.  
/// </summary>
```

3 başvuru

```
public class BankaHesap  
{  
    private string sifre;  
    2 başvuru  
    public string Sifre  
    {  
        get { return sifre; }  
        set { sifre = value; }  
    }  
    private double bakiye;  
    2 başvuru  
    public double Bakiye  
    {  
        get { return bakiye; }  
        set  
        {  
            if(Sifre == "123")  
            {  
                bakiye=value;  
            }  
            else  
            {  
                throw new Exception("Şifreniz hatalıdır kardeş !");  
            }  
        }  
    }  
    1 başvuru  
    public BankaHesap(string sifre)  
    {  
        Sifre = sifre;  
    }  
}
```

```
2  ✓ public class BankaHesap
3  {
4      | 5 references
      | private double bakiye;
      | 0 references
5  ✓ public BankaHesap(double baslangicBakiyesi)
6  {
7      |     bakiye = baslangicBakiyesi;
8      | }
      | 0 references
9  ✓ public void ParaYatirma(double miktar)
10 {
11     |     bakiye += miktar;
12     | }
      | 0 references
13 ✓ public void ParaCek(double miktar)
14 {
15     | if (bakiye >= miktar) {
16     |     bakiye -= miktar;
17     | }
18     | else {
19     |     Console.WriteLine("Yetersiz Bakiye");
20     | }
21     | }
      | 0 references
22 ✓ public double BakiyeGetir() {
23     |     return bakiye;
24     | }
25 }
```

Bu örnekte, yatırılabilen ve çekilebilen bir bakiyeye sahip basit bir banka hesabını temsil eden bir BankaHesap sınıfımız var. Bakiye alanı özel olarak işaretlenmiştir; bu, sınıf dışından doğrudan erişilemeyeceği anlamına gelir. Bunun yerine, bakiye alanına kontrollü bir ara yüz sağlayan genel Para Yatırma, Çekme ve Bakiye Getirme yöntemlerini sunulmuştur.



## DİKKAT

Özellikleri kullanarak, C#'ta kapsüllemeyi hem verimli hem de anlaşılması kolay bir şekilde uygulayabilir, kodunuzu daha bakımı kolay ve hataya daha az açık hale getirebilirsiniz.

## C#'da Özellik(Property) Tanımı

```
1 class Car {  
2     private int _speed;  
3     public int Speed {  
4         get { return _speed; }  
5         set { _speed = value; }  
6     }  
7 }
```

```
1 class Car {  
2     private readonly int _speed;  
3     public int Speed {  
4         get { return _speed; }  
5     }  
6 }
```

```
1 class Car {  
2     public int Speed { get; private set; }  
3 }
```

C#'ta özellikler get ve set anahtar cümleleriyle beraber kullanılır. Get; veriyi getirme, Set; veriyi düzenleme/değiştirme işlemlerinde kullanılır.

Ayrıca alanlar salt okunur (sadece okunur) olarak tanımlanabilir bunun için readonly anahtar cümlecisi kullanılmalıdır.

Özellikler, alanlar olmadanda oluşturulabilir. Bununla birlikte set işlemi private olarak tanımlanarak sadece sınıf içerisinde değişiklik izni verilebilir.



```
1 namespace FirstProject.Encapsulation;
2 0 references
3 class Makine
4 {
5     4 references
6     private int sicaklik;
7     0 references
8     public int Sicaklik
9     {
10         get { return sicaklik; }
11         set
12         {
13             if (value <= 100 && value >=0)
14             {
15                 sicaklik = value;
16             }
17             else if (value < 0)
18             {
19                 sicaklik = 0;
20                 throw new Exception("Fazla soğuk");
21             }
22             else
23             {
24                 Console.WriteLine("Aşırı ısıya engellendi");
25                 sicaklik = 100;
26                 throw new Exception("Aşırı derecede ısıya var");
27             }
28         }
29     }
30 }
```

Makine sınıfı içerisinde yer alan sicaklik alanı private tanımlanarak dışarıdan erişimi engellenmiştir. Ayrıca sicaklik ifadesini kontrol edebilmek için Sicaklik adında bir özellik oluşturarak sicaklik alanını kapsülledik.

```

2  ∨ public class Urun
3  {
    1 reference
4      private readonly string ad;
    4 references
5      private int miktar;
    0 references
6  ∨  public int Miktar
7      {
8  ∨      get
9          {
10             return miktar;
11         }
12  ∨      set
13          {
14             if (value < 0)
15                 miktar = 0;
16             else
17                 miktar = value;
18         }
19     }
    2 references
20     private double alisFiyat;
    2 references
21     private double karOrani;
    0 references
22     public double Fiyat => alisFiyat * karOrani;
    0 references
23  ∨  public Urun(string ad, int miktar, double alisFiyat, double karOrani)
24     {
25         this.ad = ad;

```

## Urun

- ad: string

- miktar: int

+ Miktar: int <<get>> <<set>>

- alisFiyat: double

- karOrani: double

+ Fiyat: double <<get>>

+ Urun(ad:string, miktar:int, alisFiyat:double, karOrani:double)



## ÖDEV

*Aggregation örneği içeren iki sınıf tanımlayınız.  
Tanımlanan sınıflar içerisinde kapsülleme  
tekniğini uygulayınız. Son olarak kodladığınız  
sınıfların Aggregation sınıf diyagramını çiziniz.*