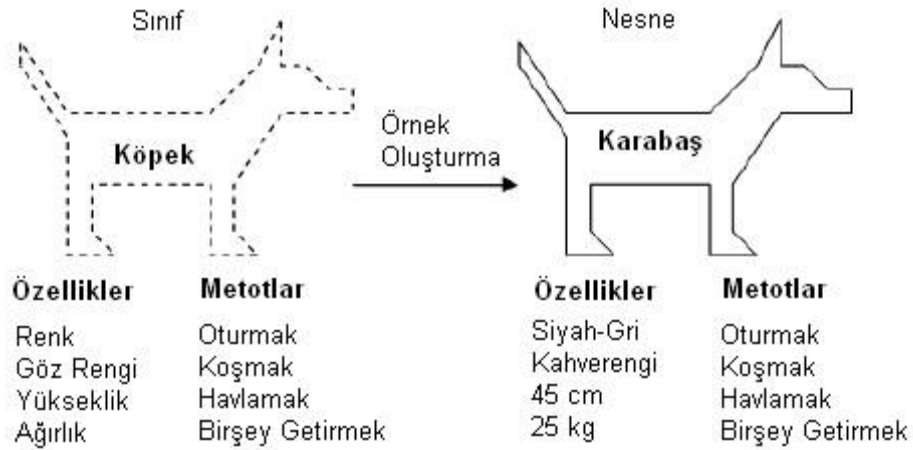


5.1. Sınıflar

Nesneler çevremizde bulunan her şeydir. Araba, uçak, kitap, personel, müdür, fatura, öğrenci gibi. Programcılar her zaman gerçek yaşamdaki durumlara yakın senaryolar oluşturmaya çalışırlar. Bu yöndeki ilk adım bilgisayarın yaşadığımız dünyadan nesnelerle ilişki kurmasını sağlamaktır. Hepimizin bildiği gibi, bilgisayar sadece bir elektronik makinedir. Bilgisayarın bizim bildiğimiz nesneleri tanımasını sağlayacak bilgiyi vermek, bizim sorumluluğumuzdadır. İşte bu noktada nesneye dayalı modelleme tekniği devreye girer. Bu modelde gerçek problemlerde karşılaştığımız nesneleri, bilgisayarda benzer nesneler olarak modelleyebiliriz.

Nesneye dayalı programlamanın esasını sınıf (class) oluşturur. Sınıf aynı cins nesnelerin genel tanımıdır. Örneğin kullandığımız araba bir nesnedir (Ford gibi). Aynı şekilde başkalarının da kullandığı Opel, BMW, Mercedes, Renault arabalarının her biri ayrı birer nesnedir. Bu nesnelerin hepsi araba sınıfı ile tanımlanabilir. Aynı sınıfa ait nesneler ortak özelliklere sahiptir.



5.2. Nesneye Dayalı Programlama

1. Nesnelerin ortak özelliklerinin sınıf (class) yapıları kullanılarak tanımlanması,
2. Bu sınıfları kullanarak nesnelerin oluşturulması,
3. Bu nesnelerle uygulamaların gerçekleştirilmesi,

Aşamalarından oluşur.

Sınıf yeni bir tip veri tanımlar. Bu yeni tip, bu tipte yeni nesneler oluşturmada kullanılır. Bu yüzden sınıf nesne için bir şablondur. Ve bir nesne sınıfın bir örneğidir.

Sınıfın Genel Biçimi: Bir sınıf, class anahtar sözcüğü ile tanımlanır. Sınıf içinde değişkenler ve metotlar yer alır.

Sınıfı oluşturan kod ve veri, söz konusu **sınıfın üyeleri (members)** olarak adlandırılır.

Sınıf üyeleri:

1. **Alan (field):** Sınıf tarafından tanımlanan veriler.
2. **Özellik (Properties):** Sınıf tarafından tanımlanan sınırlandırılmış veya özelleştirilmiş veriler.
3. **Metot (Method):** Veri üzerindeki operasyonlar.
4. **Olaylar (Events):** Diğer nesneler için operasyonel yapılar.
5. **Temsilciler (Delegates):** Olayları oluşturabilmek için gereken kapsüllenmiş yapılar.

```
class sinif_ismi
{
    //alan(fields, değişken) tanımlamaları

    tip degisken1;
    tip degisken2;
    .....

    ....

    //özellik(properties) tanımlamaları

    tip ozellik1
    {
        get {
```

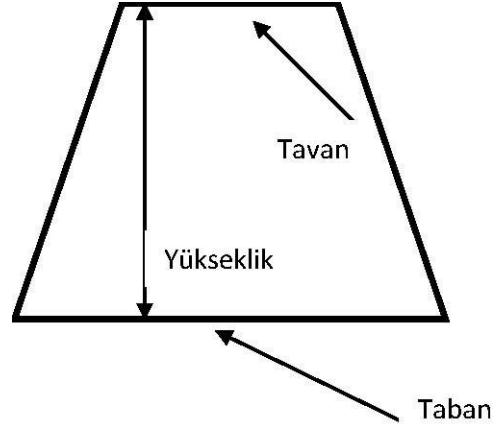
```

        ....
    }
    set{
        ....
    }
}
tip ozellik2
{
get {
    ....
}
    set{
        ....
    }
}
// metot(method) tanımlamaları
tip metot_ismi1
(parametreler)
{
    metodun gövdesi
    ....
}
tip metot_ismi2 (parametreler)
{
    metodun gövdesi
    ....
}
....
}

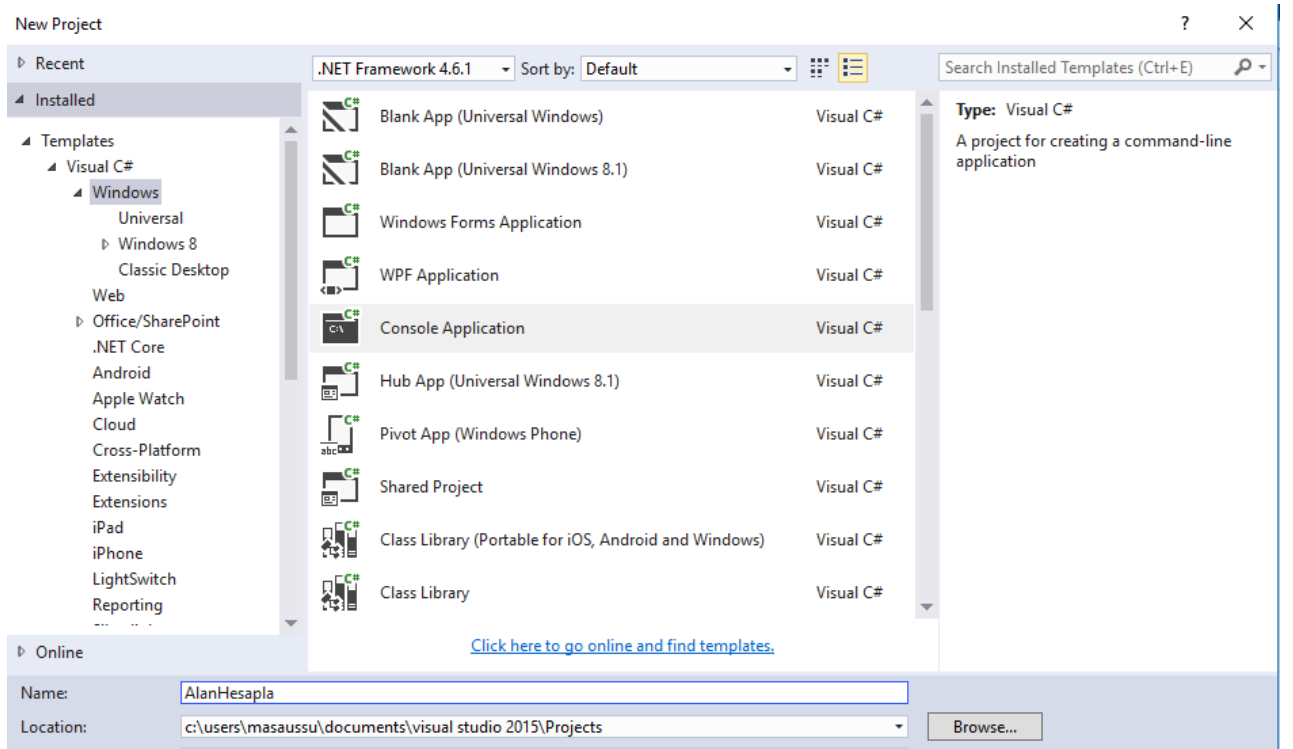
```

Örnek 1: Yamuk geometrik elemanının alanını hesaplayacak bir program geliştireceğiz.

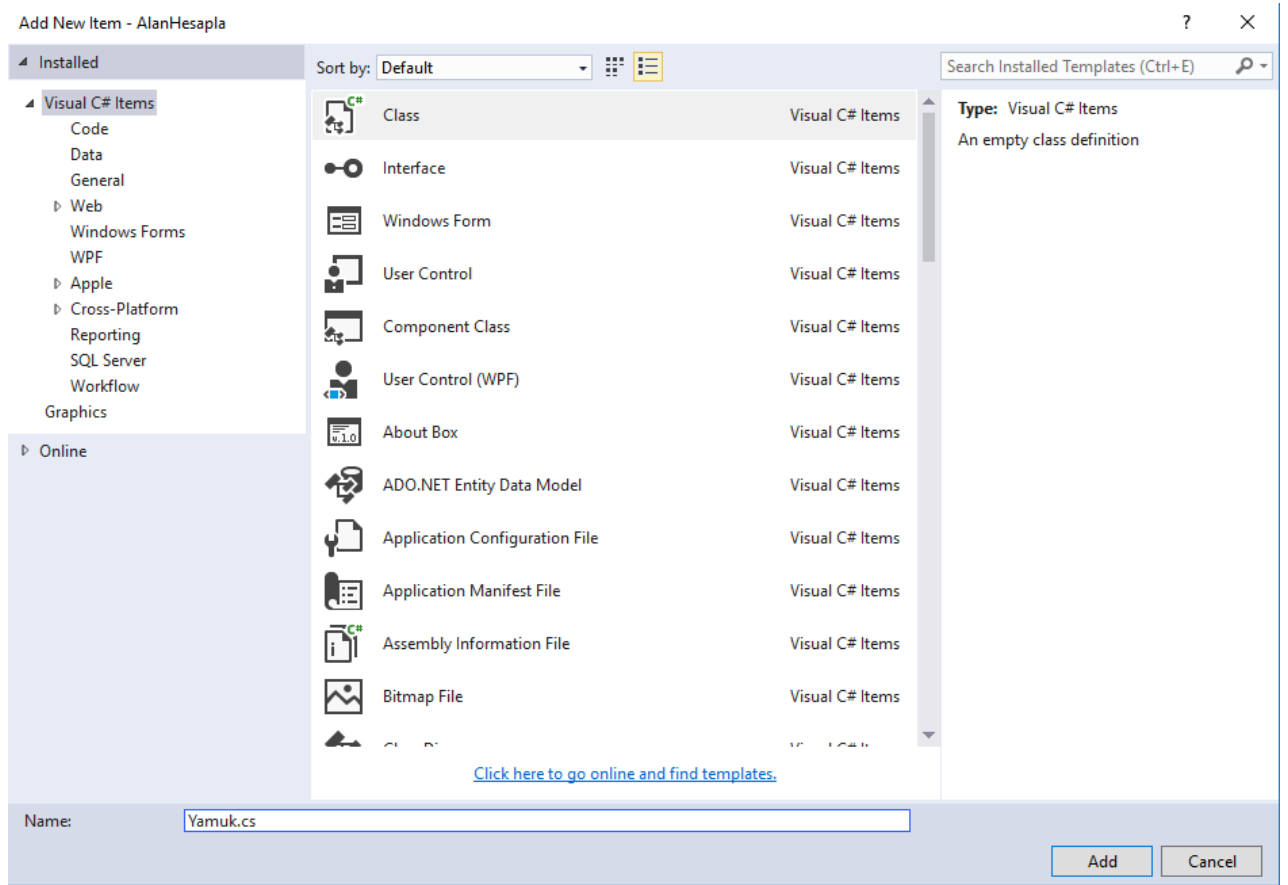
$$\text{Alan} = (\text{taban} + \text{tavan}) / 2 * \text{yükseklik}$$



VS 2015 ortamında yeni bir proje açalım.



Projemize Yamuk isimli yeni bir class ekleyelim.



Bir yamuk nesnesi tarif edebilmek için taban, tavan ve yüksekliğe ihtiyaç vardır. Bu elemanları sınıfımıza dahil edelim.

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

namespace AlanHesapla
{
    class Yamuk
    {
        double taban;
        double tavan;
        double yukseklik;
    }
}
```

`double` türünden tanımladığımız taban, tavan, yüksekli bilgilerine eğer sınıf içerisinde değer verirse yamuk nesnesinden ziyade her seferinde ölçüleri belli bir yamuk tanımı yapılmış olur. Halbuki isteğimiz yamuk nesnesi her oluşumunda kendisini tarif edilebilen değerler alabilmesi olmalıdır. Dolayısıyla Yamuk sınıfının bir örneği (instance) oluştuktan sonra değer verilmelidir.

Bir sınıfın bir örneğini oluşturma “Nesne” oluşturma denir ve `new` anahtar sözcüğü ile olur ve aşağıdaki şekilde tanımlanır.

```
Sinif_ismi degisken_ismi = new Sinif_ismi();
```

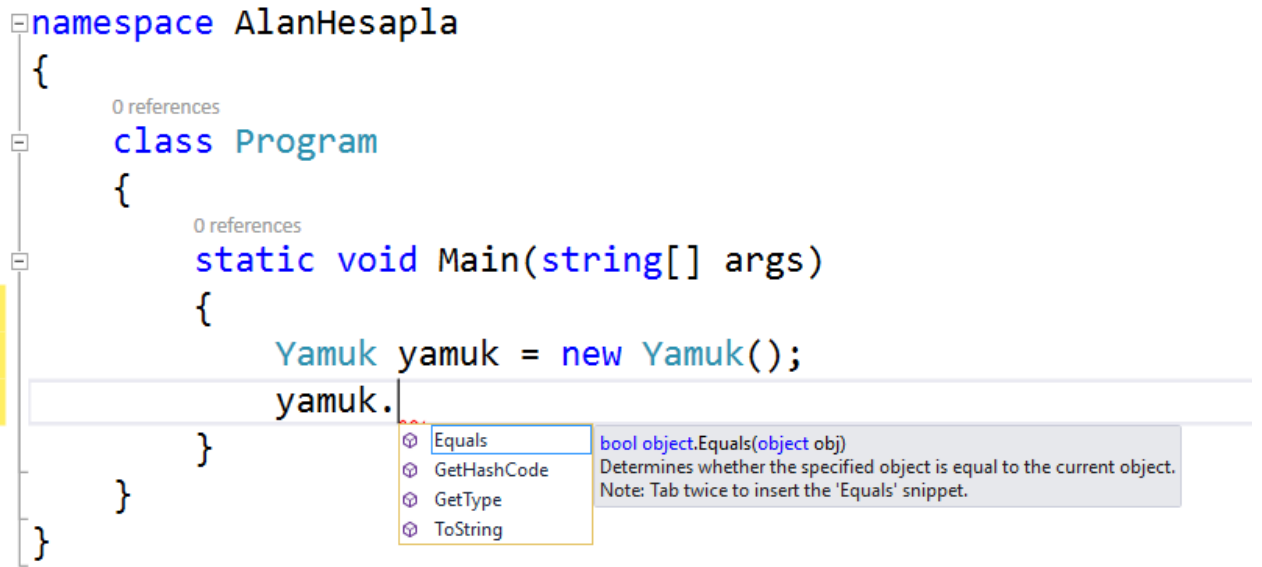
Örneğimizde ise;

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

namespace AlanHesapla
{
    class Program
    {
        static void Main(string[] args)
        {
            Yamuk yamuk = new Yamuk();
        }
    }
}
```

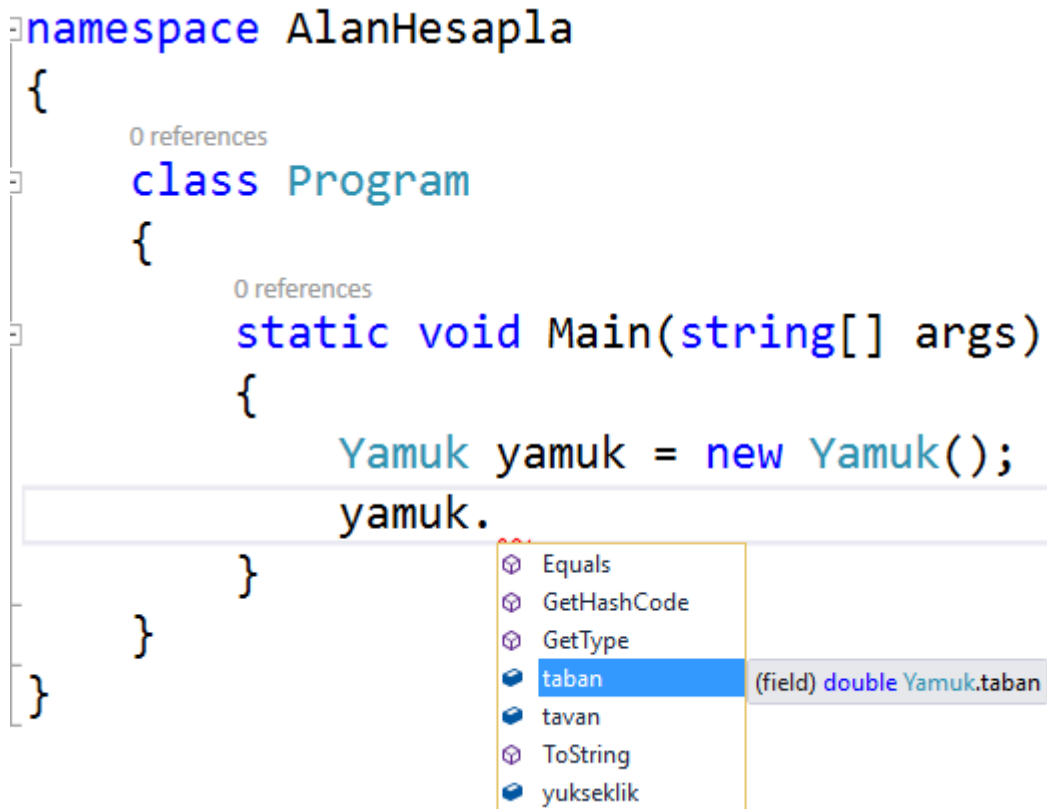
Artık şablonumuz içerisinde oluşturduğumuz alanlara değer verebiliriz. Bunun içinde “yamuk” isimli nesne isminden sonra nokta (.) operatörü kullanılır.

```
namespace AlanHesapla
{
    class Program
    {
        static void Main(string[] args)
        {
            Yamuk yamuk = new Yamuk();
            yamuk.
        }
    }
}
```



Resimden de görüleceği üzere Yamuk sınıfı içerisinde oluşturduğumuz alanları (taban, tavan, yükseklik) kullanmıyoruz. Halbuki şablonumuzda oluşturabildiğimiz **double** türünden taban, tavan ve yükseklik değerlerine ulaşabilmeliydik.

```
namespace AlanHesapla
{
    class Program
    {
        static void Main(string[] args)
        {
            Yamuk yamuk = new Yamuk();
            yamuk.
        }
    }
}
```



Bunun nedeni NDP özellikleri olan soyutlama (abstraction) ve sarmalamadır(encapsulation, bilgi saklama).

3.3. Soyutlama

Nesne tanımlanırken verinin kullanılması veya veriye erişilmesi için gerekli detayların azaltılması işlemidir. Soyutlama kabaca, iki şekilde yapılır. Bunlar veri (data) ve kontrol (control) soyutlamasıdır. Veri Soyutlaması, veri yapıları üzerinde yapılan soyutlamadır. Kontrol Soyutlaması ise yapısal programlama ile gelen altprogram (subprogram), işlev (function) gibi kavramlar üzerinde yapılan soyutlamadır.

Örnek olarak bir "kişi" nesnesinde, kişinin yaşını tutan bir tamsayı değişkeni olan "yas" değişkenini ele alalım. Bu "yas" değişkenine, programın çalıştırma anında, "-10" gibi bir değer atanmasını kimse engelleyemez. İşte burada Nesneye Dayalı Programlamanın getirdiği görünürlük (visibility) ve özellik (property) tanımlama gibi yetenekler kullanarak "yas" değerine gerçek bir değer girilmesi garanti edilir. Bu işleme Veri Soyutlaması adı verilir.

Benzer şekilde soyutlama işleminin verileri taşıyan değişkenlerin değil de yöntem (method) veya işlevlerin (function) üzerinde yapılması işlemine Kontrol Soyutlaması adı verilir. Yani yöntem ile yapılan işlemlerin bir başka nesne tarafından müdahale edilemeyecek hale sokulması işlemidir¹.

3.4. NDP Temel Kavram I: Sarmalama (Encapsulation)

Bir nesne üzerinde hem veri soyutlama, hem de kontrol soyutlaması yapılıyor ise buna sarmalama (encapsulation) adı verilir².

Kodu, veriyi veya nesneyi dış müdahalelerden koruyabilmek, oluşturulma amacı dâhilinde kullanabilmek ve kullanılmayacağı yerlerde gösterimini engellemek için sarmalamaya ihtiyaç duyulmuştur.

Bir sınıf içinde bulunan üyelerin, o sınıfı kullananlar tarafından erişilip, erişilemeyeceğini belirlemeye yarar. Örnek vermek gerekirse arabamızı çalıştırdığımızı düşünelim. Biz sadece anahtarı yerine sokup çeviriyoruz değil mi? O anda arka planda olan hiç bir işleme karışmıyor ve müdahale edemiyoruz. Yani anahtar çevrilince o motora giden enerjiyi açıyor, yakıtı pompalıyor ve

¹

[http://tr.wikipedia.org/wiki/Soyutlama_\(bilgisayar_bilimi\)](http://tr.wikipedia.org/wiki/Soyutlama_(bilgisayar_bilimi))

²

[http://tr.wikipedia.org/wiki/Sarma_\(Bilgisayar_Bilimleri\)](http://tr.wikipedia.org/wiki/Sarma_(Bilgisayar_Bilimleri))

buna benzer bir dizi işlemi sırayla yapıyor ama biz bununla ilgilenmiyoruz ve de erişemiyoruz. İşte bu mekanizma bizim erişemememiz için kapsüllenmiştir. Aksi taktirde kaputu açıp enerjiyi biz vermeye kalksaydık istenmeyen durumlarla karşılaşabilirdik. Bu çok uç bir örnek oldu farkındayım o yüzden daha basit bir örnek vermek gerekirse hepimiz cep telefonu kullanıyoruz, mesaj yazarken birini ararken dokunmatik ekrandaki klavyeye basıyoruz ama telefonun içindeki devrelere erişmemiz engellenmiştir. Bu işlemi yapabilmemiz için yazılım ile klavye takımı geliştirilmiştir. Yani devreler bir nevi kapsüllenmiştir.

Sarmalama erişim denetleyicileri ile sağlanmaktadır.

3.5. Erişim Denetleyicileri (Access Modifiers)

Diğer sınıfların veya sınıf gruplarının, sınıf ya da sınıf üyelerine erişimi düzenleyen/kısıtlayan anahtar kelimelerdir.

Pratikte yaygın olarak kullanılan anlayış, bir programcı bir sınıfı (class) oluşturur ve daha sonra diğer programcılar bu sınıfı kendi kodlarında kullanırlar. Nesneye dayalı programlamanın önemli özelliklerinden bir tanesi olan **encapsulation**, sınıf değişkenlerine erişimi kısıtlayarak sınıfı oluşturan kişinin sınıfını korumasını sağlamaktadır. Aşağıdaki anahtar sözcükler, metot veya değişkenlere erişimi kısıtlamak veya izin vermek için kullanılırlar.

public	Bu şekilde tanımlanmış bir metot veya değişkene herhangi bir yerden ulaşmak mümkündür.
private	Bu şekilde tanımlanmış bir metot veya değişkene sadece sınıf içinden ulaşmak mümkündür.
protected	Bu şekilde tanımlanmış bir metot veya değişkene tanımlandıkları sınıftan veya bu sınıftan türetilmiş sınıflardan ulaşılabilir.
internal	Bu şekilde tanımlanmış bir metot veya değişkene tanımlandıkları sınıftan ve tanımlandıkları sınıfla aynı assembly'de olan sınıflardan ulaşmak mümkündür. (C#'a ait anahtar kelimedir.)
protected internal	Bu şekilde tanımlanmış bir metot veya değişken hem protected hem de Internal özelliğindedir. (C#'a ait anahtar kelimedir.)

Bu noktada unutulmaması gereken 5 husus vardır:

1. İşletim sistemleri bir tipin public, protected ya da private olmasıyla ilgilenmez. Dolayısıyla hafıza takibi ile üyelerin değerlerine ulaşılabilir. Dolayısıyla hafıza takibi ile üyelerin değerlerine ulaşılabilir.
2. Her programcı kendi yazdığı sınıfın güvenliğinden ve güvenilirliğinden sorumludur. Eğer herhangi bir sınıf üyesinin private ile tanımlanması yeterli olduğu halde public ile dış dünyaya açılırsa programcılar bu üyeyi istedikleri gibi kullanılabılırler. Bu durum NDP'nin soyutlama prensibine aykırıdır.
3. Her dilin kabul ettiği default değerleri farklıdır. C# private, Java public olarak kabul etmektedir. Ancak programın okunabilirliği açısından erişim denetleyicilerin default değeri her ne olursa olsun belirtilmesi tavsiye edilmektedir.
4. Sınıf üyelerinden alanların(field) public ile sınıf dışına açılması önerilmemektedir. (Sınıf üyelerinden özellikler konusunda açıklanacaktır.)
5. Bir sınıfın kendisinin private olarak tanımlanmasının hiç bir anlamı yoktur. Dolayısıyla derleyiciler hata vermektedir.

Erişim denetleyicileri ile Yamuk sınıfını tekrar düzenlersek;

```
public class Yamuk
{
    public double taban;
    public double tavan;
    public double yukseklik;
}
```

haline gelecektir.

Main kodlarımız ise;

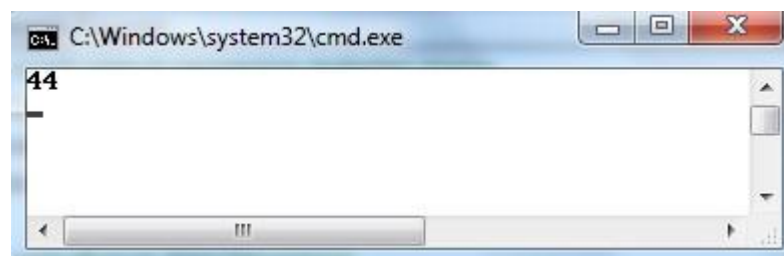
```
class Program
{
    static void Main(string[] args)
    {
        // yamuk isimli yamuk nesnesini oluşturma
        Yamuk yamuk = new Yamuk();

        // y isimli yamuk nesnesine değerler atama
        yamuk.taban = 5;
        yamuk.tavan = 6;
        yamuk.yukseklik = 8;

        // y yamuğunun alanı hesaplanır
        double alan = (yamuk.taban + yamuk.tavan) / 2 *
yamuk.yukseklik;

        // y yamuğunu ekrana yazdırır
        Console.WriteLine("Y: " + alan);
    }
}
```

F5 ile programı çalıştırdığımızda ekrana 44 sonucunu verecektir.



İkinci bir yamuk nesnemizi oluşturarak sonuçları ekrana yazdıralım.

```
class Program
{
    static void Main(string[] args)
    {
        // yamuk isimli yamuk nesnesini oluşturma
        Yamuk yamuk = new Yamuk();

        // yamuk isimli yamuk nesnesine değerler atama
        yamuk.taban = 5;
        yamuk.tavan = 6;
        yamuk.yukseklik = 8;

        // yamuk yamuğunun alanı hesaplanır
        double alan = (yamuk.taban + yamuk.tavan) / 2 *
yamuk.yukseklik;

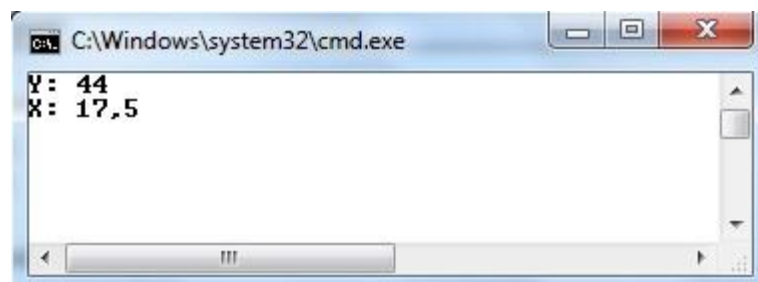
        // yamuk yamuğunu ekrana yazdırır
        Console.WriteLine("Y: " + alan);

        // x isimli yamuk nesnesini oluşturma
        Yamuk x = new Yamuk();

        // x isimli yamuk nesnesine değerler atama
        x.taban = 3;
        x.tavan = 4;
        x.yukseklik = 5;

        // x yamuğunun alanı hesaplanır
        double alan2 = (x.taban + x.tavan) / 2 * x.yukseklik;

        // x yamuğunu ekrana yazdırır
        Console.WriteLine("X: " + alan2);
    }
}
```



3.6. partial Anahtar kelimesi

Bir tipe ait kodun birden fazla dosyaya yazılabilmesi özelliğidir. Sadece C# ait bir anahtar kelimedir.

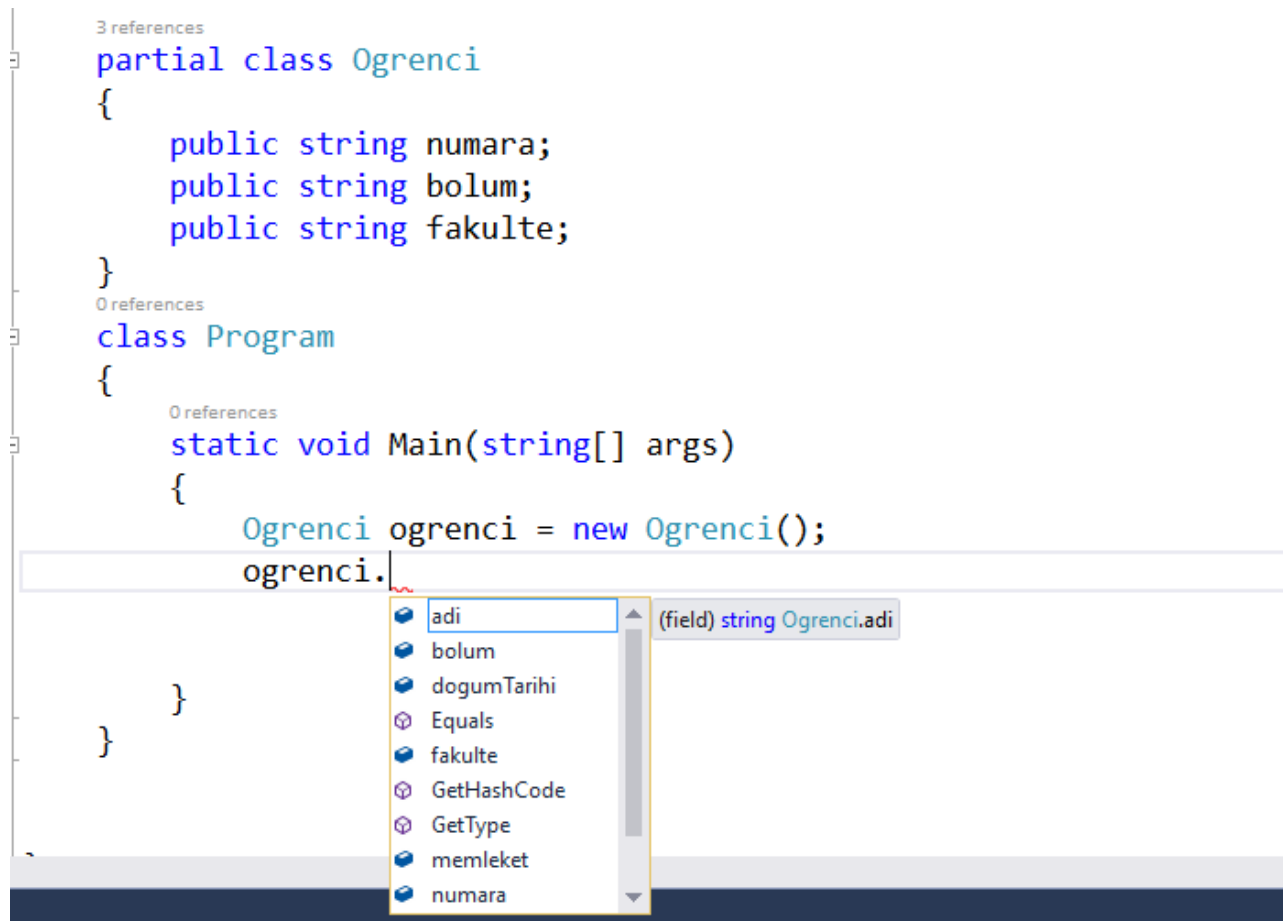
Örnek 2: Öğrencilerin ad, soyad, doğum tarihi, memleketi, öğrenci numarası, bölümü, fakültesi bilgilerini tutan sınıfı oluşturalım.

VS 2010' da yeni bir proje açıp adını **PartialClassOrnegi** olarak değiştirelim. Öğrenci isimli sınıfımızı ekleyelim.

Oğrenci.cs kodları;

```
partial class Öğrenci
{
    public string adi;
    public string soyadi;
    public DateTime dogumTarihi;
    public string memleket;
}
```

Program.cs sınıfında partial class ile Öğrenci sınıfının devamını yazabiliriz.



Program.cs kodları;

```
class Program
{
    static void Main(string[] args)
    {
        // ogrenci1 isimli ilk öğrenci nesnesi oluşturuluyor
        Ogrenci ogrenci1 = new Ogrenci();

        //ogrenci1 nesnesine değerleri atanıyor
        ogrenci1.adi = "Barış";
        ogrenci1.soyadi = "Çalışkan";
        ogrenci1.dogumTarihi = new DateTime(1981, 3, 1, 7, 0, 0);
        ogrenci1.memleket = "Sakarya";
        ogrenci1.numara = "0001.10001";
        ogrenci1.bolum = "Bilgisayar Mühendisliği";
        ogrenci1.fakulte = "Mühendislik";

        //ogrenci1 nesnesi ekrana yazdırılıyor
        Console.WriteLine("Adı          : " + ogrenci1.adi);
    }
}
```

```

        Console.WriteLine("Soyadı      : " + ogrenci1.soyadi);
        Console.WriteLine("Doğum Tarihi : " +
ogrenci1.dogumTarihi.ToShortDateString());
        Console.WriteLine("Memleket    : " + ogrenci1.memleket);
        Console.WriteLine("Numarası    : " + ogrenci1.numara);
        Console.WriteLine("Fakültesi   : " + ogrenci1.fakulte);
        Console.WriteLine("Bölümü     : " + ogrenci1.bolum);

        Console.WriteLine();

Console.WriteLine("*****");

        Console.WriteLine();

        // o2 isimli ikinci öğrenci nesnesi oluşturuluyor
        Öğrenci o2 = new Öğrenci();

        //o2 nesnesine değerleri atanıyor
        o2.adi = "Ahmet";
        o2.soyadi = "Şanslı";
        o2.dogumTarihi = new DateTime(1983, 7, 19, 7, 0, 0);
        o2.memleket = "Sakarya";
        o2.numara = "0101.10001";
        o2.bolum = "Bilgisayar Mühendisliği";
        o2.fakulte = "Mühendislik";

        //o2 nesnesi ekrana yazdırılıyor
        Console.WriteLine("Adı      : " + o2.adi);
        Console.WriteLine("Soyadı    : " + o2.soyadi);
        Console.WriteLine("Doğum Tarihi : " +
o2.dogumTarihi.ToShortDateString());
        Console.WriteLine("Memleket    : " + o2.memleket);
        Console.WriteLine("Numarası    : " + o2.numara);
        Console.WriteLine("Fakültesi   : " + o2.fakulte);
        Console.WriteLine("Bölümü     : " + o2.bolum);
    }
}

```

Ekran çıktımız ise;

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

Adı      : Barış
Soyadı   : Çalışkan
Doğum Tarihi : 01.03.1981
Memleket : Sakarya
Numarası : 0001.10001
Fakültesi : Mühendislik
Bölümü   : Bilgisayar Mühendisliği

*****

Adı      : Ahmet
Soyadı   : Şanslı
Doğum Tarihi : 19.07.1983
Memleket : Sakarya
Numarası : 0101.10001
Fakültesi : Mühendislik
Bölümü   : Bilgisayar Mühendisliği
Devam etmek için bir tuşa basın . . .
```