BMB201. NESNE YÖNELİMLİ PROGRAMLAMA

Ders 4: Nesne Yönelimli Programlama Giriş

Erdinç Uzun
NKÜ Çorlu Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü

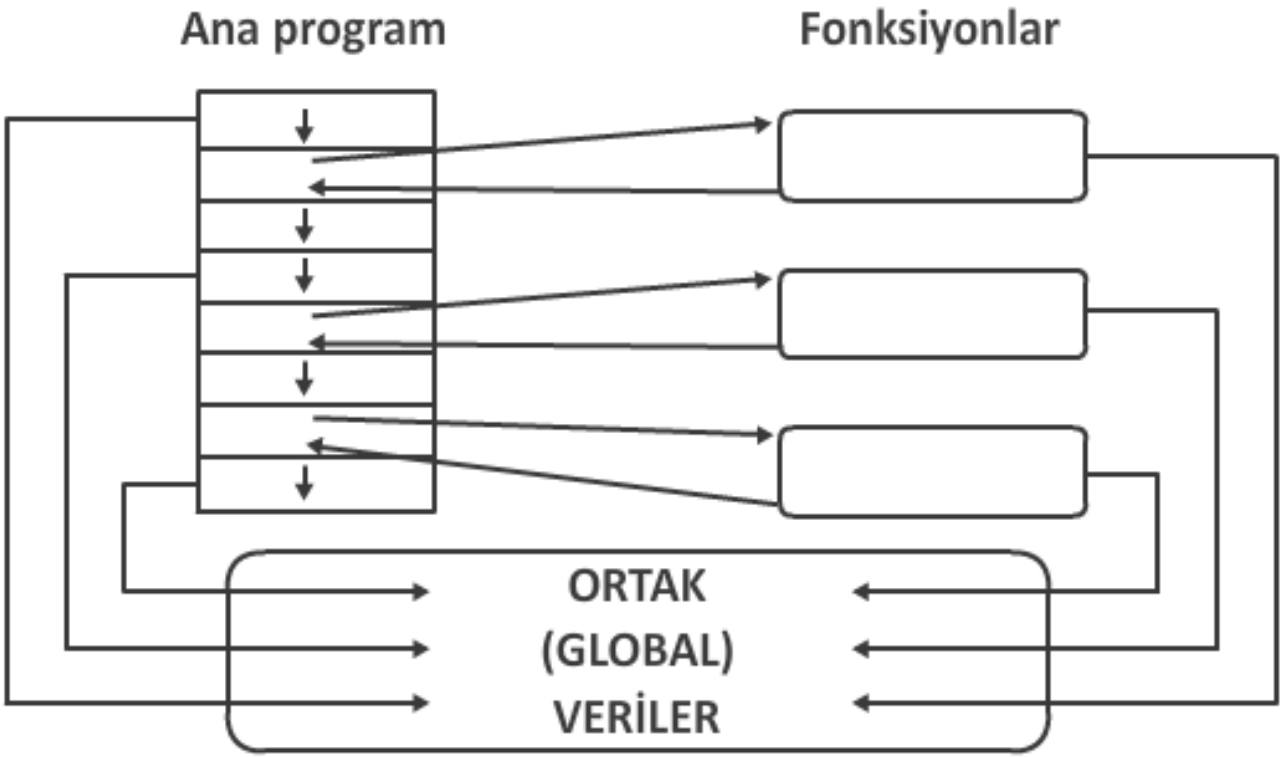
Nesne Yönelimli Programlama

- PBP - Procedure Based Programming (1. Hafta)
- NYP - Object Oriented Programming (NYP)
- Amaç:
  - Yazılımların programlaması ve bakımı kolaylaştırmak
  - Bununla birlikte zaman ve çaba maliyetini düşürmektir.

Prosedür Tabanlı Programlama

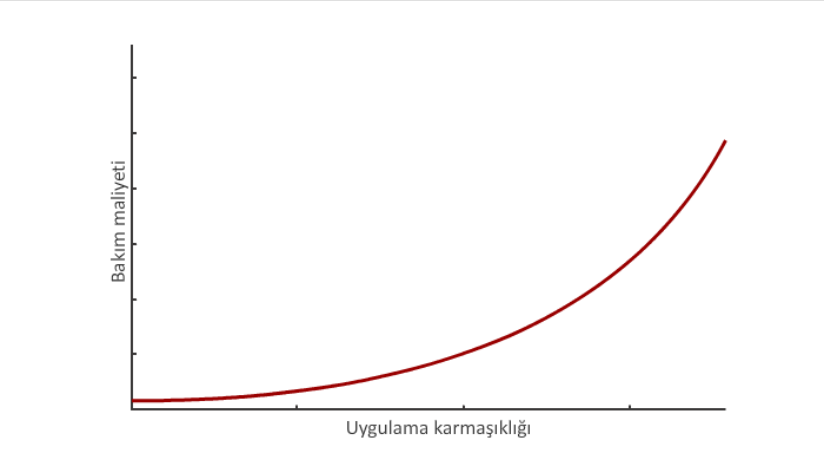
- C#, Java gibi bir Nesne Yönelimli Programlama (NYP) dilidir. NYP 1960'lardan bugüne yazılım dünyasını etkisine almış bir metodolojidir. 1970 yılından bugüne kadar geliştirilen birçok dil NYP desteğine sahiptir.
- NYP temel olarak, ortaya çıktığı güne kadar süregelen programlama mantığını kökten değiştirmiştir. NYP'den önce kullanılan yazılım metodolojisi, Prosedür Tabanlı Programlama adı ile anılır. Bu metodoloji, belirli bir yönde ilerleyen kodlar ve iş yükünü hafifletmek için ortak işlerin yüklendiği fonksiyonların çağırılması esasına dayalıydı.

Prosedür Tabanlı Programlama



Prosedür Programlama

- Geliştirilen uygulama parçalanamayan bir bütün halindeydi. Bu yüzden, uygulama üzerinde çalışan her geliştirici; uygulamanın hemen her yapısına hakim olmalıydı.
- Bu durum nedeniyle, projelere yeni yazılımcıların katılması önemli bir adaptasyon süreci gerektiriyordu.
- Uygulama tek bir bütün halinde olduğu için, ufak değişiklikler uygulamanın farklı noktalarında büyük sorunlara yol açabiliyordu.
- Yıllarla birlikte müşteri ihtiyaçları ve donanım kabiliyetleri arttı. Bunun getirisi olarak da, geliştirilen uygulamaların kapsamları ve boyutları büyüdü. Bu aşamada, yukarıda bahsedilen problemler gittikçe artmaya başladı. Başlanan projelerin çoğu istenen sürelerde yetiştirilememeye ve geliştirme zorluklarından ötürü iptal olmaya başladı.
- Uygulama maliyetleri giderek artmaya başladı.



Nesne Yönelimli Programlama

- Yazılım dünyasında bu çıkmazın aşılması NYP ile sağlanmıştır. NYP ile tüm yazılım anlayışı kökten değişmiştir. 1960'larda NYP fikrini ilk ortaya atan Alan Kay, önerdiği metodolojiyi şu şekilde ifade etmiştir:
  - Uygulama, nesneler ve onların ilişkileri çerçevesinde belirli bir iş yapmak için geliştirilebilmelidir.
  - Her nesnenin bir sınıfı olmalıdır ve sınıflar nesnelerin ortak davranışlarını ifade etmelidir.
  - Nesneler birbirleri ile iletişime geçebilmelidir.

Nesne Yönelimli Programlama

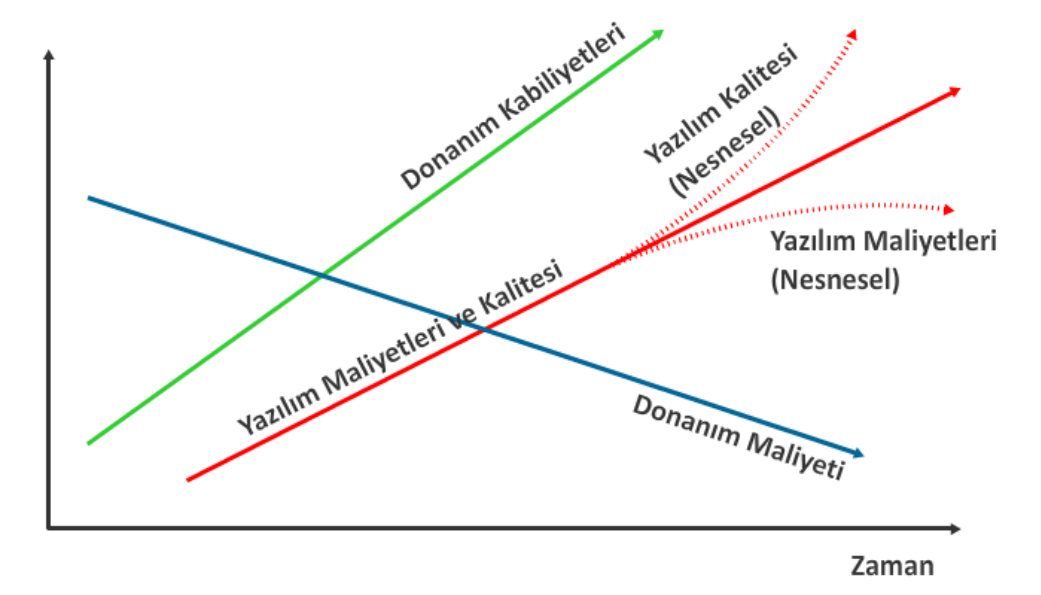
- Bunu basitçe, prosedür tabanlı programlamada bulunan soyut program geliştirme mantığını rafa kaldırıp, gerçek dünya modellemesi ile program geliştirme çabası olarak da düşünebiliriz.
- Gerçek dünya modellemesiyle anlatılmak istenen şudur:
  - Bir fabrika örneğini ele alalım. Bu fabrikada işçiler, makineler gibi birçok nesne bulunur ve bu nesnelerin ilişkisi çerçevesinde fabrika çeşitli işler yapıp çıktılar üretebilir. NYP ile programlama mantığında da, bu örnektekine benzer şekilde program kurgulanır. Çeşitli nesneler geliştirilip birbirleriyle ilişkilendirilerek, belirli amaçlara hizmet eden uygulamalar geliştirilir.

Nesne Yönelimli Programlama

- Bu yapının önemli getirileri şunlardır: Yazacağınız sınıflar birbirinden bağımsız olarak geliştirilebilir. Bu sayede program böl, parçala, fethet mantığı çerçevesinde çok kolay bir şekilde parçalanır ve her parça ayrı ayrı ele alınabilir. NYP mantığında gerçek dünya algılayışı temel alındığı için bu, anlaşılması çok daha kolay bir yapıdır.
- NYP, yapısı gereği kod tekrarlarının önüne geçer (doğru bir şekilde kullanılırsa) ve bu durum, özellikle ilk dönemlerde yazılımcıların hızlı bir şekilde NYP yapılarına geçmesinin temel nedenlerinden biri olmuştur.

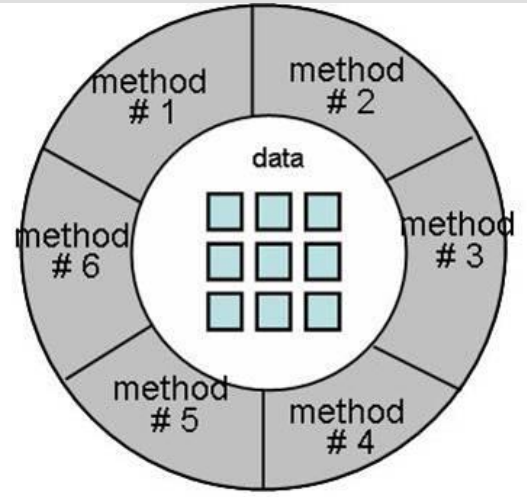
Nesne Yönelimli Programlama

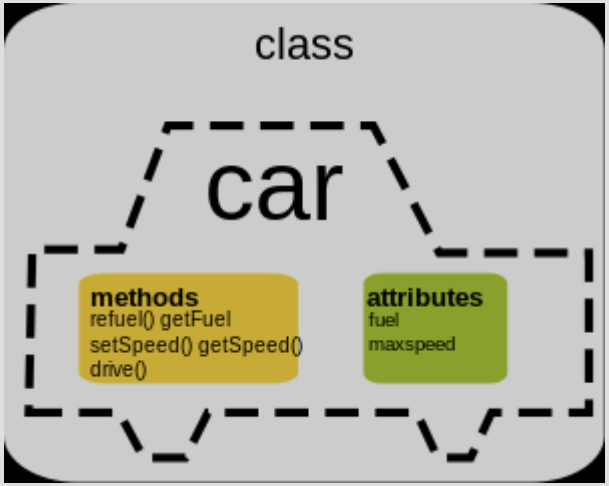
- Projelerin yönetilebilirliğini büyük miktarda artırdığı için daha büyük projeler çok daha az çaba ile yönetilebilir hale gelmiştir. Yine aynı getiriler sayesinde, projeler rahat bir şekilde büyütülebilmiştir. Alttaki grafikte noktalı çizgilerle gösterilen bölümler NYP'nin etkileridir. Yani, NYP ile yazılım kalitesi artmış, bunun yanısıra yazılım maliyetleri düşmüştür.



Sınıf

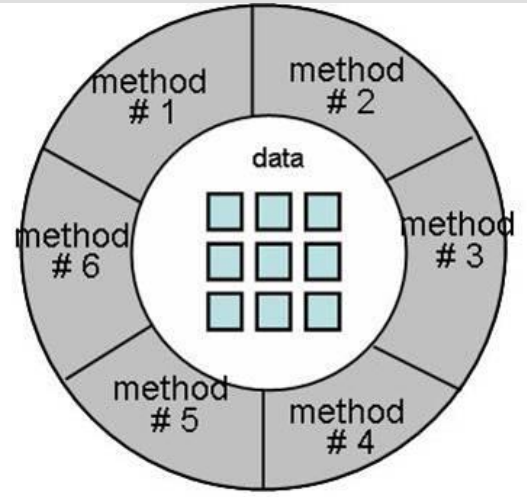
- Sınıf, nesne yönelimli programlama dillerinde nesnelerin özelliklerini, davranışlarını ve başlangıç durumlarını tanımlamak için kullanılan şablonlara verilen addır.
- Bir sınıftan türetilmiş bir nesne ise o sınıfın örneği olarak tanımlanır.
- Sınıflar genelde şahıs, yer ya da bir nesnenin ismini temsil ederler.

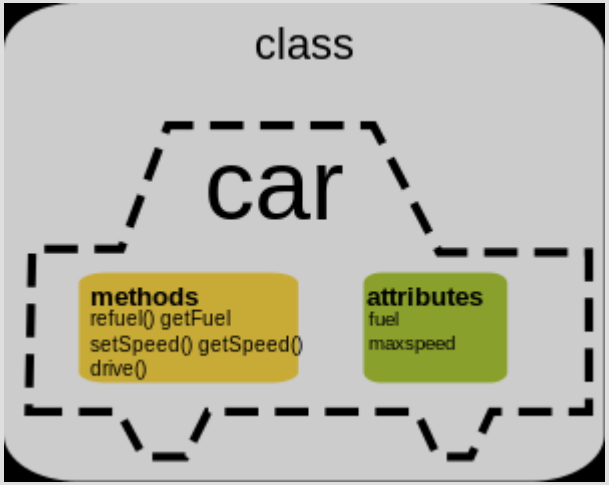




Sınıf

- Sınıflar metotları ile nesnelerin davranışlarını, değişkenleri ile ise nesnelerin durumlarını kapsül ederler.
- Sınıflar hem veri yapısına hem de bir ara yüze sahiptirler.
- Sınıflar ile nasıl etkileşime girileceği bu ara yüzler sayesinde sağlanır.
- Örneğin bir sınıf şablonu ile araba: yakıt ve maksimum hız özelliklerine ve ayrıca depoyu doldur, o anki yakıtı gör, hızı ayarla, hızı göster ve arabayı kullan gibi metotlara sahip olabilir.





Sınıf kullanılmasının başlıca nedenleri

- Gerçek hayat problemleri sınıf şablonları kullanılarak bilgisayar ortamına daha kolay ve anlaşılabilir bir biçimde aktarılabilir.
- Sınıflar kodlarımızın daha düzenli ve erişilebilir olmasını sağlar.
- Nesne yönelimli programlamada herhangi bir projede kullanılmak üzere yaratılan bir sınıf başka projelerde tekrar kullanılabilir. (Reusability)
- Yeni veri tiplerinin oluşturulabilir olması
- Hata ihtimalinin azalması ve ayıklamanın kolaylaşması
- Gerektiğinde değişikliğin hızlı ve kolayca yapılması
- Takım çalışmasının desteklenmesi

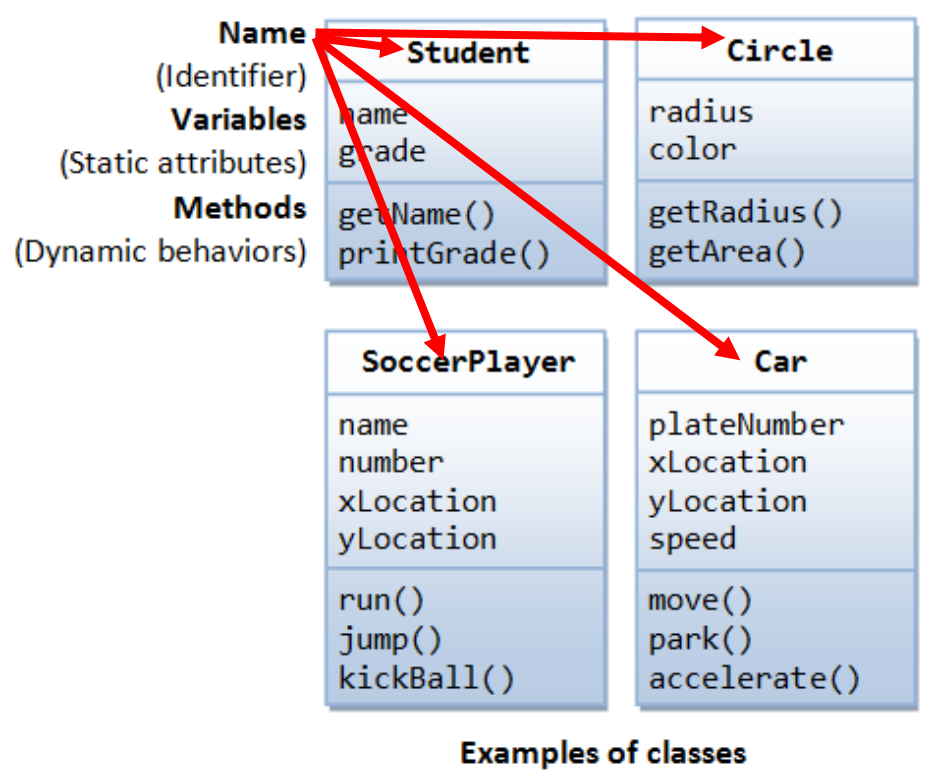
Sınıf - Metotlar

- Metotlar dört ayrı erişim kuralına göre tanımlanabilir. Bunlar public, protected, private ve internal olarak adlandırılmıştır. (Bu gün public kullanacağız.)
- Metotlar bir değer döndürebilir. (Return)
- Bir metodun geri dönüş değerinin boş olması istendiğinde bir prosedür ya da bunun mümkün olmadığı dillerde boş veri türü olan void kullanılmaktadır. (C# ve Java'da void var.)

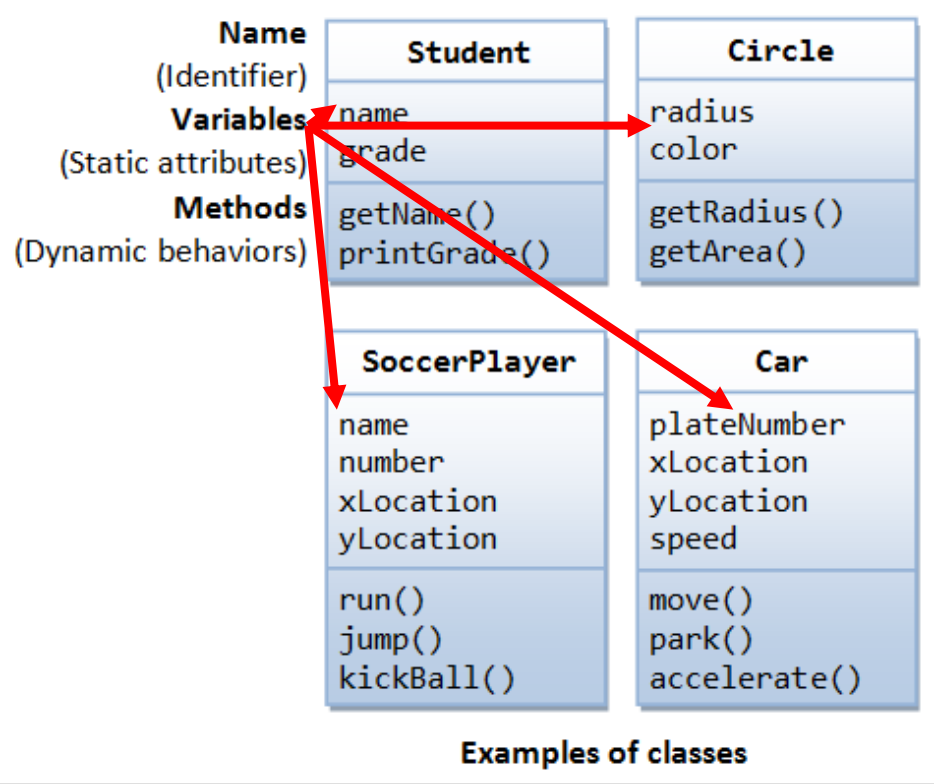
Sınıf - Veri - Özellik

- Verileri de metotlar da olduğu gibi dört erişim kuralı ile tanımlanabilir.
  - Bu erişim türleri gelecekte anlatılacaktır.

Sınıfa isim verme

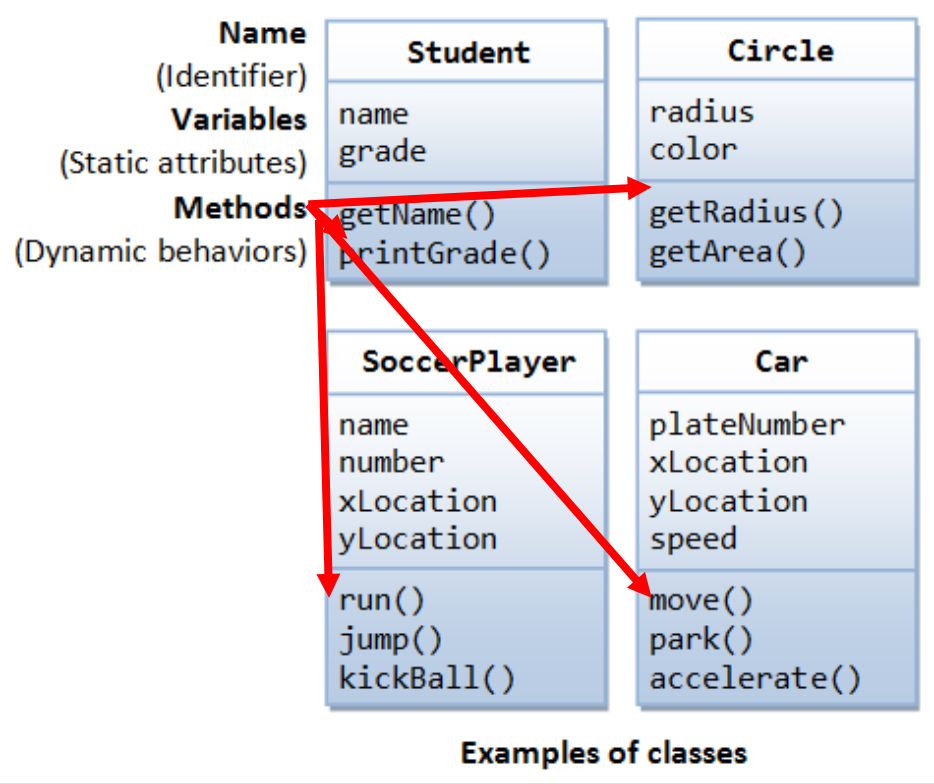


Sınıfın değişken/lerini belirleme



- Öğrencinin (Student) Ad (name) ve Grade (Not) değişkenleri.
- Dairenin (Circle) yarıçap (radius) ve renk (color) değişkenleri.
- Futbolcunun (SoccerPlayer) Adı (Name), FormaNo (Number), x ve y koordinatları (iki boyutlu bir oyun) değişkenleri.
- Arabanın (Car) Plakası (plateNumber), x ve y koordinatları, hızı (speed) değişkenleri.

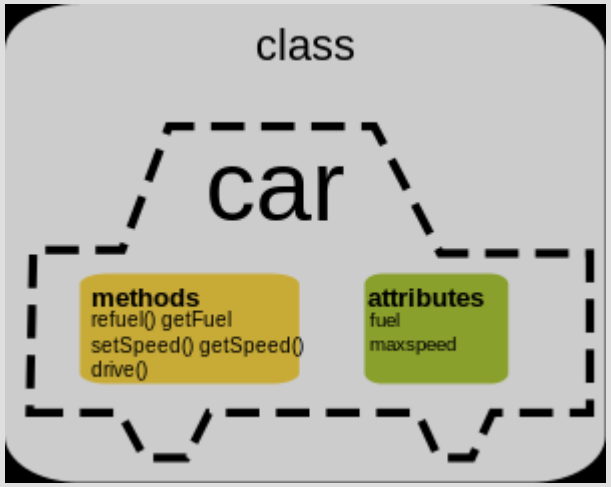
Sınıfın metot/larını belirleme

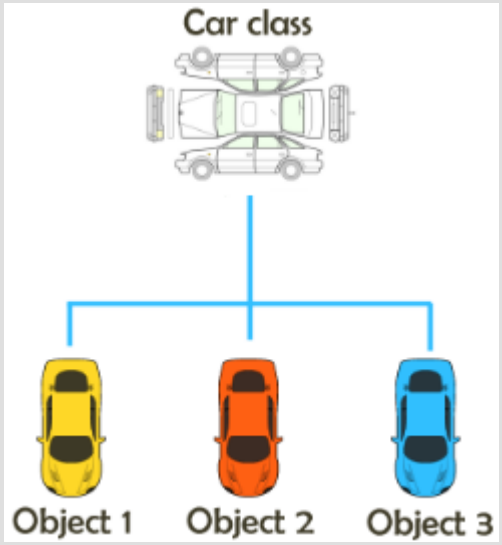


- Öğrencinin Adı Ne (getName) ve Notunu Görüntüle (printGrade).
- Dairenin yarıçapı ne (getRadius) ve alanı hesapla (getArea).
- Futbolcunun koş (run), zıpla (jump) ve topa vur (kickball).
- Arabanın hareket et (move), park et (park) ve accelerate (hızlan).

Nesne

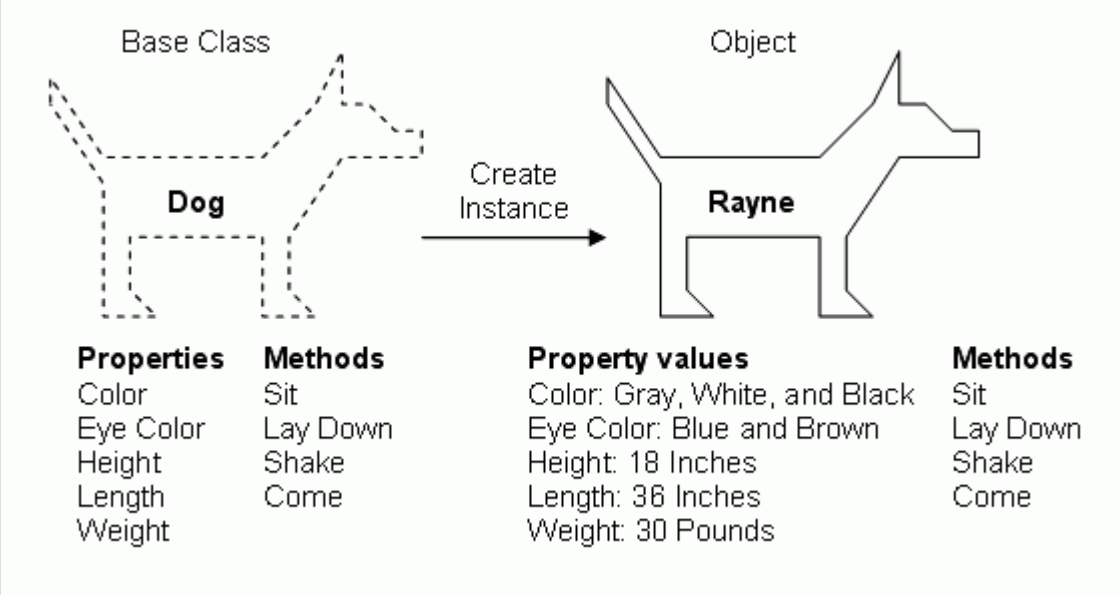
- OOP de objeler sınıflardan üretilir.
- Objeler, sınıfların aksine canlıdır ve kimlikleri vardır.
- Aynı sınıftan üretilmiş iki objenin sahip olduğu değişkenler değişik özelliklere sahiptir.
  - Örneğin araba sınıfı maxspeed değişkeni araba türüne göre farklılık gösterir.





Nesne

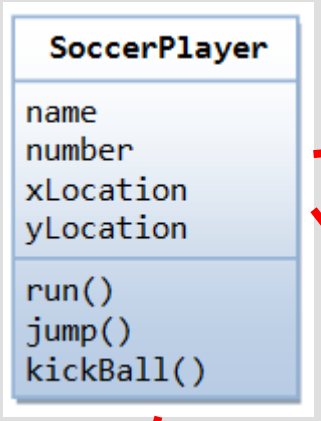
- Köpek (Dog) sınıfı ve bu sınıftan üretilen Rayne adında bir köpek (Object)



Not: Farklı kaynaklarda farklı ifadelerle karşılaşabilirsiniz.
Variable (Değişken), Veri, Attritubes ve Properties (Özellikler) aynı kapıya çıkar.
Method (Metot) bölümü yordam, fonksiyon olarak ta adlandırılabilir.

Nesne

- Futbolcu (Soccer Player - Class) ve Futbolcular (Objects)



Merhaba Nesne Dünyası

<Niteleyici> class <Sınıf Adı>

{

//Özellikler

//Metodlar

}

```
class Kutu
{
    public double length;   //Uzunluk
    public double breadth;  //Genişlik
    public double height;   //Yükseklik
}
```

- Niteleyici – Erişim Belirteçleri, şu an için public ve private kullanılacaktır. Eğer aynı namespace içinde isek bu bölümü boş geçebiliriz.
- İleri de bu konu daha geniş şekilde açıklanacaktır.

Merhaba Nesne Dünyası

- Nesne Yaratma
  - <Sınıf Adı> <Nesne Adı> = new <Sınıf Adı>;

```
class Kututester
{
    static void Main(string[] args)
    {
        Kutu Kutu1 = new Kutu();   //Birinci kutu - nesne 1
        Kutu Kutu2 = new Kutu();   //İkinci kutu - nesne 2
        double volume = 0.0;    //kutunun hacmini hesaplamak için

        //Nesne 1 özellikler
        Kutu1.height = 5.0;
        Kutu1.length = 6.0;
        Kutu1.breadth = 7.0;

        //Nesne 2 özellikleri
        Kutu2.height = 10.0;
        Kutu2.length = 12.0;
        Kutu2.breadth = 13.0;

        //Nesne 1 Hacmi
        volume = Kutu1.height * Kutu1.length * Kutu1.breadth;
        Console.WriteLine("Volume of Kutu1 : {0}", volume);

        //Nesne 2 Haci
        volume = Kutu2.height * Kutu2.length * Kutu2.breadth;
        Console.WriteLine("Volume of Kutu2 : {0}", volume);
        Console.ReadKey();
    }
}
```

```
using System;

namespace KutuUygulamasi
{
    class Kutu
    {
        public double length;   //Uzunluk
        public double breadth;  //Genişlik
        public double height;   //Yükseklik
    }
    class Kututester
    {
        static void Main(string[] args)
        {
            Kutu Kutu1 = new Kutu();   //Birinci kutu - nesne 1
            Kutu Kutu2 = new Kutu();   //İkinci kutu - nesne 2
            double volume = 0.0;    //kutunun hacmini hesaplamak için

            //Nesne 1 özellikler
            Kutu1.height = 5.0;
            Kutu1.length = 6.0;
            Kutu1.breadth = 7.0;

            //Nesne 2 özellikleri
            Kutu2.height = 10.0;
            Kutu2.length = 12.0;
            Kutu2.breadth = 13.0;

            //Nesne 1 Hacmi
            volume = Kutu1.height * Kutu1.length * Kutu1.breadth;
            Console.WriteLine("Volume of Kutu1 : {0}", volume);

            //Nesne 2 Haci
            volume = Kutu2.height * Kutu2.length * Kutu2.breadth;
            Console.WriteLine("Volume of Kutu2 : {0}", volume);
            Console.ReadKey();
        }
    }
}
```

Uygulama Uzayı

Sınıf Adı

Üç boyutlu kutu için üç değişken

Nesne 1

Nesne 2

Nesne 1'in özellikleri belirleniyor

Nesne 2'in özellikleri belirleniyor

Nesne 1'den özellikleri alınıp Hacim hesaplanıyor. Hacim ekranda görüntüleniyor.
Aynı işlemler Nesne 2 içinde yapılıyor.

Everthing is an Object

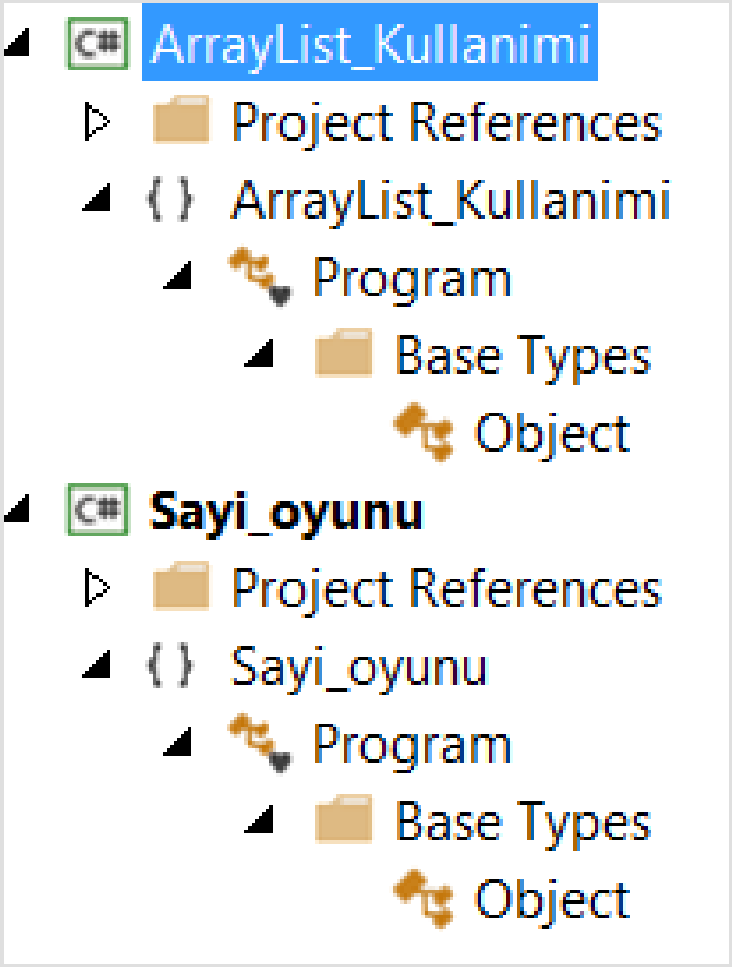
- Herşey bir nesnedir.
  - .NET framework sınıf hiyerarşisinde bütün sınıflar birer nesnedir.
  - Diğer yandan tüm nesneler, object sınıfından türetilmiştir.
  - Dünyadaki hatta evrendeki her şeyde bir nesnedir.
  - Nesneler doğar, yaşar ve kaçınılmaz son ölüm.
- Geçen haftalarda new kullandığımız birkaç durum:

```
ArrayList list = new ArrayList();

List<int> list = new List<int>();

FileInfo NewFile = new FileInfo(@"C:\Test\HedefDizini\NewFile.txt");

StreamReader reader01 = new StreamReader("file.txt");

StreamWriter writer =  new StreamWriter("important.txt")
```

Ayrıca bu güne kod yazdığımız Program bölümü de bir sınıftır.
Main bölümü bu sınıfın ilk çalışan metotudur.

Class View & Object Browser

- Class View: Sınıfımızı ve sınıfımızın özellik ve metotlarını ağaç gösterimi şeklinde bulabilirsiniz.
- Ve sonunda object'e ulaşırsınız.
- Object Browser, .Net kütüphanelerini ve kendi yazdığınız sınıfları hiyerarşik bir şekilde görebilirsiniz.



Sınıfın Üyeleri

- İç Sınıflar (Nested Class)
- Alanlar/Değişkenler (Variables)
- Yordamlar (Methods)
- Yapıcılar (Constructors)
- Yıkıcılar (Destructors)
- Özellikler (Properties)
- İndeksleyiciler (Indexers)
- Numaralandırmalar (Enumerations)
- Öznitelikler (Attritubes)
- Arabirimler (Interfaces)
- Yapılar (Structures)
- Temsilci ve Olaylar (Delegates & Events)

Derslerimizde bu üyelerin her birini ayrıntılı bir şekilde göreceğiz.

Sınıf İçine Yordam Tanımlama

- 3 yordam eklendi:
  - 1. yordam genişlik değiştirsin.
  - 2. yordam yükseklik değiştirsin.
  - 3. yordam hacmi hesaplasın.
- public bir erişim belirtecine nesneden ulaşılabilir.
- private bir değişkene nesneden ulaşılamaz.
- Ulaşmak için 1. ve 2. yordam tanımlanmıştır.
  - Yordam sayesinde if kontrolü yaparak veriyi alabilme şansı kazandığınızı unutmayın.

```
class Kutu
{
    private double length;
    private double breadth;
    private double height;
    public void setLength(double len)
    {
        length = len;
    }

    public void setBreadth(double bre)
    {
        breadth = bre;
    }

    public void setHeight(double hei)
    {
        height = hei;
    }
    public double getVolume()
    {
        return length * breadth * height;
    }
}
```

Abstraction (Soyutlama)

- «Soyutlama» önemli özelliklere odaklanabilmek için ayrıntıları göz ardı etme sürecidir. Bir yazılım birçok sınıftan oluşabilir. Her sınıfın kendine ait özellikleri vardır. Soyutlamanın amacı her sınıfın özelliklerin iyi bir şekilde belirlenmesidir. Örneğin kutu uygulamamız için uzunluk, genişlik ve yükseklik özelliklerinin belirlenmesi işlemi bir soyutlamadır. Daha genel anlamda Her sınıfın kendine ait özelliklerinin belirlenmesi işlemine soyutlama diyoruz.

```
class Kutu
{
    public double length;   //Uzunluk
    public double breadth;  //Genişlik
    public double height;   //Yükseklik
}
```

Encapsulation (Saklama, Paketleme)

- «Saklama», soyutlamayı desteklemek ya da güçlendirmek için bir sınıfın iç yapısının gizlenmesidir. Saklama sayesinde özelliklerinize ve yordamlarınıza ulaşılmayı engelleyebilirsiniz. Sınıf içindeki özellik ve yordamları yukarıda olduğu gibi public (genel) veya private (özel) olarak tanımlanabilir. Public, nesne oluşturulduktan sonra o özellik veya yordama ulaşabildiğimiz anlamına gelir. Private bir özellik veya yordama ise sadece sınıf içinden ulaşabiliriz.

```
class Kutu
{
    private double length;
    private double breadth;
    private double height;
    public void setLength(double len)
    {
        length = len;
    }

    public void setBreadth(double bre)
    {
        breadth = bre;
    }

    public void setHeight(double hei)
    {
        height = hei;
    }
    public double getVolume()
    {
        return length * breadth * height;
    }
}
```

Saklama işleminden sonra Nesne

- Private özelliklere nesne üzerinden ulaşamazsınız.
- Yordamlar üzerinden ulaşabilirsiniz. Ayrıca bu yordamlara if şartları yazıp veriyi kontrol altına alabilirsiniz. Örneğin yükseklik değerini 0'dan küçük ise 1 olsun diye kural girebilirsiniz.

```
class Kutu
{
    private double length;
    private double breadth;
    private double height;
    public void setLength(double len)
    {
        if (len <= 0)
            length = 1;
        else
            length = len;
    }

    public void setBreadth(double bre)
    {
        if (bre <= 0)
            breadth = 1;
        else
            breadth = bre;
    }

    public void setHeight(double hei)
    {
        if (hei <= 0)
            height = 1;
        else
            height = hei;
    }

    public double getVolume()
    {
        return length * breadth * height;
    }
}
```

Yeni Nesneyi Çağırma

```
class Kututester
{
    static void Main(string[] args)
    {
        Kutu Kutu1 = new Kutu();
        Kutu Kutu2 = new Kutu();

        //Nesne 1 yordam üzerinden
        Kutu1.setLength(6.0);
        Kutu1.setBreadth(7.0);
        Kutu1.setHeight(5.0);

        //Nesne 2 yordam üzerinden
        Kutu2.setLength(12.0);
        Kutu2.setBreadth(13.0);
        Kutu2.setHeight(10.0);

        //nesne 1 için hacim yordam üzerinden
        Console.WriteLine("Volume of Kutu1 : {0}", Kutu1.getVolume());

        //nesne 2 için hacim yordam üzerinden
        Console.WriteLine("Volume of Kutu2 : {0}", Kutu2.getVolume());

        Console.ReadKey();
    }
}
```

Nesne 1'in özelliklerine private olduğu için ulaşamıyoruz. Nasıl Ulaşırsınız?

Private Özelliklere Ulaşma

- Bir fonksiyon üzerinden private değerlere nesne üzerinden ulaşabilirsiniz.
- Set ile başlayan yordamlar void yordam iken get ile başlayan yordamlar değer döndürdükleri için return içerir. Ve uygun private özelliği return eder.

```
class Kutu
{
    private double length;
    private double breadth;
    private double height;

    Değer alma işlemleri (SET)

    #region Değer okuma işlemleri (GET)
    public double getLength()
    {
        return length;
    }

    public double getBreadth()
    {
        return breadth;
    }

    public double getHeight()
    {
        return height;
    }
    #endregion

    public double getVolume()
    {
        return length * breadth * height;
    }
}
```

Doğum - Yaşam - Ölüm

- Dünya farklı sınıflardan oluşur. Bu sınıflar doğar, yaşar ve ölüm gelir.
- Aynı şekilde NYP bir sınıf üzerinden Nesne doğar, yaşar ve ölümle karşılaşır.
- NYP, tüm tasarım size kalmıştır.
  - Sınıfın doğumu sırasında (Nesnenin oluşumu) özelliklerinin belirlenmesi
  - Sınıfın davranışlarına / yeteneklerine karar vermek (Yordamlar)
  - Sınıfın ölümü ve ölümü sırasındaki işlemlere karar vermek

Constructor (Yapıcı)

- Bir canlı dünyaya gelirken daha en başta bazı özellikleri belirlenir. NYP'da, sınıftan bir nesne oluşturduğumuzda özellikleri daha nesne oluşurken belirleme şansımız vardır.
- Yapıcı sınıf ile aynı isimde olur ve kesinlikle bir değer döndürmez.
- Bir yapıcı, parametresiz veya dışarıdan parametre alabilir.

```
class Kutu
{
    private double length;
    private double breadth;
    private double height;

    //Yapıcı Constructor
    public Kutu(double len, double bre, double hei)
    {
        setLength(len);
        setBreadth(bre);
        setHeight(height);
    }

    Değer alma işlemleri

    public double getVolume()
    {
        return length * breadth * height;
    }
}
```

Constructor

Main'den değişen constructor durumunu çağırma

```
class Kututester
{
    static void Main(string[] args)
    {
        Kutu Kutu1 = new Kutu(6.0, 7.0, 5.0);
        Kutu Kutu2 = new Kutu(12.0, 13.0, 10.0);

        //nesne 1 için hacim yordam üzerinden
        Console.WriteLine("Volume of Kutu1 : {0}", Kutu1.getVolume());

        //nesne 2 için hacim yordam üzerinden
        Console.WriteLine("Volume of Kutu2 : {0}", Kutu2.getVolume());

        Console.ReadKey();
    }
}
```

- Dikkat edilirse artık özellikler ilk contructor alınırken belirlenmektedir.

- İnsan doğarken göz özelliği bellidir.
  - Değişmez yapılmak istenirse private özellik olarak tanımlanır ve değiştirilemez.
  - Lens takınca göz rengini değiştirmek isterseniz public özellik yapabilirsiniz

  veya private özellik yapıp bir fonksiyonla set edebilirsiniz.
  Bu dünyanın tasarımı size kalmış!!!

Destructor (Yıkıcı)

- Nesne doğar, yaşar ve kaçınılmaz son ölüm.
- Nesnenin başlangıç özellikleri yapıcı ile belirledik. Nesnemiz, yaşaması sırasında yordam/lar ile bazı işlemleri yerine getirebilir. Ama ölüm günü gelecektir.
- Yıkıcı, ölümden önce sınıfın son işlemlerinin yapıldığı yordamdır. Yıkıcı da yapıcı da olduğu gibi sınıf ile aynı isme sahip olur ancak ~ karakteri ile başlar.

```
class Kutu
{
    private double length;
    private double breadth;
    private double height;

    //Yapıcı Constructor
    public Kutu(double len, double bre, double hei)
    {
        setLength(len);
        setBreadth(bre);
        setHeight(height);
    }

    ~Kutu()
    {
        Console.WriteLine("Güle güle C# Dünyası");
    }

    Değer alma işlemleri (SET)

    Değer okuma işlemleri (GET)

    İşlemsel Fonksiyonlar

}
```

Nesne programımıza veda ederken

Destructor (Yıkıcı)

- Destructor metodu illa yazmamıza gerek yok. Ama sınıf bellekten çıkarılırken bazı işlemler yapmamız gerekebilir. Sınıfın son durumunu kaydetme, log tutma, başka sınıfları tetikleme vb. birçok işlem bu esnada yapılabilir.
- C++ gibi programlama dillerinde bir nesnenin bellekten alınması manuel bir süreç olarak yönetiliyordu. Bu bellekten alma işlemi içi destructor metodu kullanılması gerekiyordu. Java ve C#'ta bu metodu sadece gerektiğinde kullanılmasına imkan tanıyor. Başka bir deyişle destructor işlemi yapmamıza gerek yok. Garbage Collection (GC) (çöp toplama) mekanizması sayesinde işlemi biten nesneler bellekten siliniyor. GC'ı, Azrail'e benzetebiliriz. Ölen nesnelerin ruhlarını almaya geliyor.

Destructor (Yıkıcı)

- GC işlemi yazılım kapatıldıktan sonra çalışır. Ancak nesne içindeki değerler değiştiyse GC.Collect() fonksiyonu çağırılarak otomatik çalıştırılabilir.
- Zorunlu olmadıkça otomatik çalıştırmanıza hiç gerek yoktur.
- Console uygulamanınız veya diğer uygulamalarınız kapatıldığı anda GC devreye otomatik olarak girecektir.

```
class Kututester
{
    static void Main(string[] args)
    {
        Kutu Kutu1 = new Kutu(6.0, 7.0, 5.0);
        Kutu Kutu2 = new Kutu(12.0, 13.0, 10.0);

        //nesne 1 için hacim yordam üzerinden
        Console.WriteLine("Volume of Kutu1 : {0}", Kutu1.getVolume());

        //nesne 2 için hacim yordam üzerinden
        Console.WriteLine("Volume of Kutu2 : {0}", Kutu2.getVolume());

        Kutu1 = null;
        Kutu2 = null;
        GC.Collect(); //GC hareket geçirme, ancak bu işlemi çağırmaya gerek yok.

        Console.ReadKey();
    }
}
```

Yordama farklı parametre aktarma teknikleri

- Referans yolu ile parametre aktarma
  - Ref
  - Out
- Params özelliği

Referans yolu ile parametre aktarma

- Bir yordam bir sonuç döndürür veya void bir yordam ise değer döndürmez.
  - Void bir yordamdan değer ve değerler almak için
  - Bir fonksiyon içinden birden fazla değer almak için
- 1. Global değişkenler tanımlanır.
- 2. parametre yolu ile veri alınabilir.

```
class Program
{
    //ref anahtar kelimesi değer tipinden önce kullanılmalı.
    static void kareAl(ref double d)
    {
        d = d * d;
    }
    static void Main(string[] args)
    {
        double i = 3.45;
        Console.WriteLine("Double sayi: {0}", i);
        kareAl(ref i);   //ref parametresini içeren metod çağırılıyor.
        Console.WriteLine("Karesi :  {0} ", i);
        Console.ReadKey();
    }
}
```

Bu değişken üzerinden main'e değer gönderilebilir.

ref

- Metot çağrımının da "ref" anahtar sözcüğü ile birlikte yapılması zorunludur.
- "ref" anahtar sözcüğünü içeren metodu çağırmadan önce "ref" olarak aktarılan argümana mutlaka bir değer atanması gerektiğidir.
  - Bunun nedeni, referans olarak argüman alan bir metodun kendisine gelen referansın başka bir veriyle ilişkili olduğunu varsaymasıdır. Yani "ref" kullanarak bir argümana ilk değer atamak için metot tanımlamak mümkün değildir. Eğer değer verilmemiş bir değişken kullanılırsa "Use of unassigned local variable" hata mesajı ile karşılaşılır.

out

- "out" anahtar sözcüğünün kullanımı da "ref" anahtar sözcüğünün kullanımından çok farklı değildir.
- Tek fark; "out" sözcüğü kullanılacak değişkene ilk değer atamak zorunda olunmasıdır. Çünkü; "out" ifadesi her zaman için değer atanmamış olarak kabul edilir ve metot sınırları içerisinde "out" sözcüğü için bir değer metot tarafından atanır.
- "out" anahtar sözcüğü bir metottan birden fazla geri dönüş bekleniyorsa kullanılır.

out

```
static void Main()
{
    bool b;
    int max = Max(9, 2, out b);
    Console.WriteLine(b);
}

static int Max(int x, int y, out bool b)
{
    if (x > y)
        b = true;
    else
        b = false;
    return Math.Max(x, y);
}
```

- Bu örnekte iki sayıdan büyüğünü geri dönen bir metot yazılmıştır.
- Aynı zamanda bu metot iki sayıdan hangisinin büyük olduğuna da karar verebilmektedir.
- Her metodun tek bir sonuca ulaştığı düşünülerek ilk değer verilmemiş, yeni bir parametre daha Max() isimli metoda gönderilmiştir.

Params

- Params, birden fazla parametrenin alınmasına olanak sağlar.
- Params'lar derleyici tarafından diziye dönüştürülürler.

```
static void Main()
{
    //Farklı toplama işlemleri, parametre sayısına dikkat.
    int sum1 = SumParameters(1);
    int sum2 = SumParameters(1, 2);
    int sum3 = SumParameters(3, 3, 3);
    int sum4 = SumParameters(2, 2, 2, 2);

    //Sonuçlar gösteriliyor.
    Console.WriteLine(sum1);
    Console.WriteLine(sum2);
    Console.WriteLine(sum3);
    Console.WriteLine(sum4);
}

static int SumParameters(params int[] values)
{
    //toplama işlemi
    int total = 0;
    foreach (int value in values)
    {
        total += value;
    }
    return total;
}
```

Params kullanmadan

```
static void Main()
{
    //Params kullanılmayınca uzun uzun diziyi yaratmak gerekli
    //Params kullanınca diziyi yaratmaya gerek yok, derleyici sizin için diziyi yaratır
    int num = SumParameters(new int[] { 1 });
    int num2 = SumParameters(new int[] { 1, 2 });
    int num3 = SumParameters(new int[] { 3, 3, 3 });
    int num4 = SumParameters(new int[] { 2, 2, 2, 2 });

    //Sonuçlar gösteriliyor.
    Console.WriteLine(num);
    Console.WriteLine(num2);
    Console.WriteLine(num3);
    Console.WriteLine(num4);
}

static int SumParameters(int[] values)
{
    //toplama işlemi
    int total = 0;
    foreach (int value in values)
    {
        total += value;
    }
    return total;
}
```

Console uygulamasına dışarıdan argüman (argument) alma

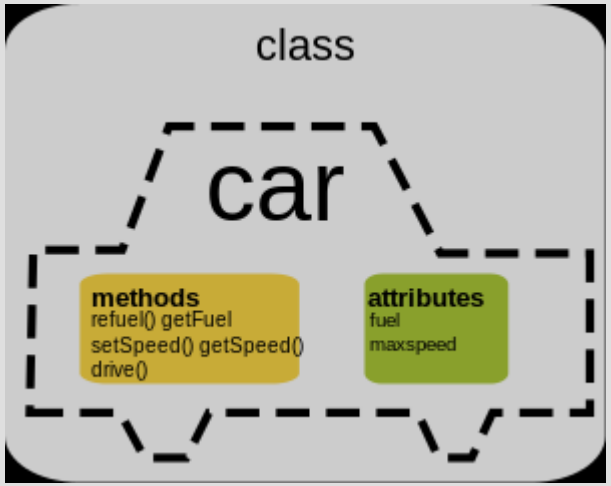
- Oluşturduğumuz exe'nin yanına string ifadeler yazıp uygulamamızı yönlendirebiliriz.
- Bunun için «command» ekranına gitmeliyiz. Geçmiş slaytlardan cd, dir ve exe çalışmayı hazırlayın.

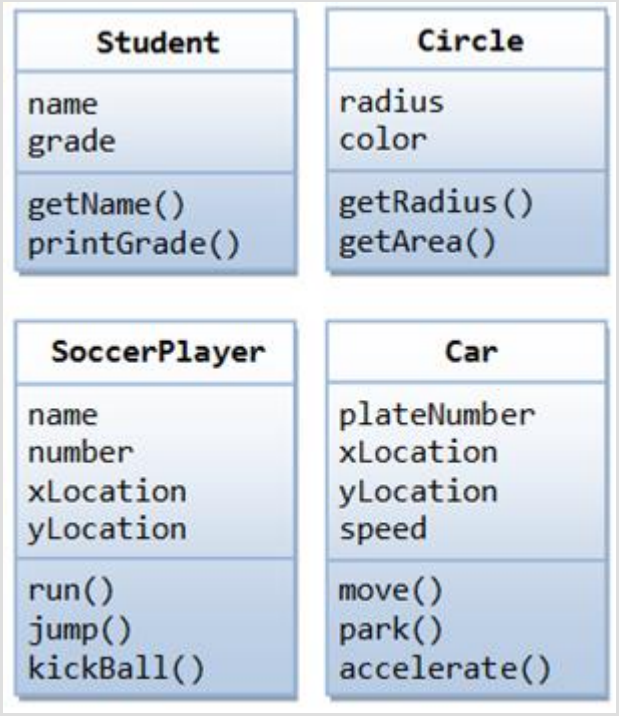
```
static void Main(string[] CmdArgs)
{
    foreach(string Arg in CmdArgs)
        Console.WriteLine("Argumanlar = {0}", Arg);

    Console.ReadKey();
}
```

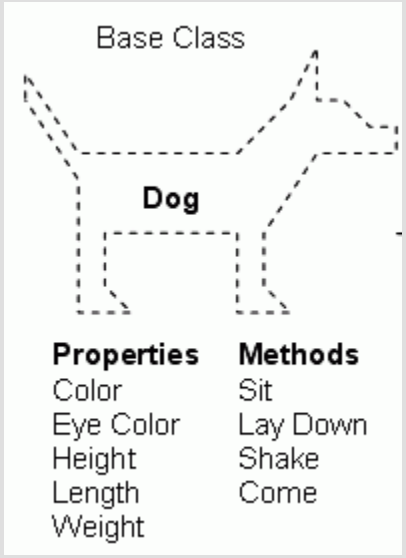
```
C:\>cd myspace

C:\MySpace>proje.exe arguman1 arguman2_
```

cd yanına klasör ismi girilmiştir. cd yanındaki klasör ismi bir argümandır. proje.exe adlı projemiz için bir veya birden fazla ifade tanımlayabiliriz.

Aşağıdaki konulardan hangisi üzerine kod yazalım?







Kaynaklar

- Wikipedia
- http://slideplayer.biz.tr/slide/2308848/
- https://gelecegiyazanlar.turkcell.com.tr/konu/windows-phone/egitim/windows-phone-101/nesne-yonelimli-programlama-object-oriented-programming
- https://bidb.itu.edu.tr/seyirdefteri/blog/2013/09/06/c-'-ta-de%C4%9Fer-ve-referans-tipleri