

Görsel Programlama

C# ile Görsel Programlama

Dr. Öğr. Üyesi Hayri Volkan Agun

Kaynaklar:

Doç. Dr. Erdinç Uzun - Ders Notları

Konular

- Nesneye Yönelik Programlama
 - Struct, Sınıf ve Metodlar
 - Kalıtım (Inheritance)
 - Erişim
 - Çok şekillilik
- Örnekler

Struct

- C#, bir yapı değer tipli (value type) bir veri tipidir.
- Özellikle ilişkili farklı veri tiplerini tek değişkende tutmakta yardımcı olur. (Sınıfa benziyor.)
- Struct anahtar kelimesi ile bir yapı kurulur.
- Yapılar kayıtların içeriklerini belirlemek için kullanılır. Örneğin bir kütüphanedeki kitaplar (books) için bir uygulama düşünelim. Bu uygulamada aşağıdaki içerikleri ön plana çıkar:

Title (Başlık)

Author (Yazar)

Subject (Konu)

Book ID (Kitap ID)

Matematiksel operatörler

```
struct Books
{
    public string title;
    public string author;
    public string subject;
    public int book_id;
};
```

```
public class testStructure
{
    public static void Main(string[] args)
    {
        Books Book1; /* Book1 tanımlandı */
        Books Book2; /* Book2 tanımlandı */

        /* Book1 özellikleri */
        Book1.title = "C Programming";
        Book1.author = "Nuha Ali";
        Book1.subject = "C Programming Tutorial";
        Book1.book_id = 6495407;

        /* Book2 özellikleri */
        Book2.title = "Telecom Billing";
        Book2.author = "Zara Ali";
        Book2.subject = "Telecom Billing Tutorial";
        Book2.book_id = 6495700;

        /* Book1 içeriği */
        Console.WriteLine("Book 1 title : {0}", Book1.title);
        Console.WriteLine("Book 1 author : {0}", Book1.author);
        Console.WriteLine("Book 1 subject : {0}", Book1.subject);
        Console.WriteLine("Book 1 book_id : {0}", Book1.book_id);

        /* Book2 içeriği */
        Console.WriteLine("Book 2 title : {0}", Book2.title);
        Console.WriteLine("Book 2 author : {0}", Book2.author);
        Console.WriteLine("Book 2 subject : {0}", Book2.subject);
        Console.WriteLine("Book 2 book_id : {0}", Book2.book_id);

        Console.ReadKey();
    }
}
```

Struct Kavramı

```
struct Books
{
    public string title;
    public string author;
    public string subject;
    public int book_id;
};

public class testStructure
{
    public static void Main(string[] args)
    {
        Books Book1; /* Book1 tanımlandı */
        Books Book2; /* Book2 tanımlandı */


        /* Book1 özellikleri */
        Book1.title = "C Programming";
        Book1.author = "Nuha Ali";
        Book1.subject = "C Programming Tutorial";
        Book1.book_id = 6495407;

        /* Book2 özellikleri */
        Book2.title = "Telecom Billing";
        Book2.author = "Zara Ali";
        Book2.subject = "Telecom Billing Tutorial";
        Book2.book_id = 6495700;

        /* Book1 içeriği */
        Console.WriteLine("Book 1 title : {0}", Book1.title);
        Console.WriteLine("Book 1 author : {0}", Book1.author);
        Console.WriteLine("Book 1 subject : {0}", Book1.subject);
        Console.WriteLine("Book 1 book_id : {0}", Book1.book_id);

        /* Book2 içeriği */
        Console.WriteLine("Book 2 title : {0}", Book2.title);
        Console.WriteLine("Book 2 author : {0}", Book2.author);
        Console.WriteLine("Book 2 subject : {0}", Book2.subject);
        Console.WriteLine("Book 2 book_id : {0}", Book2.book_id);


        Console.ReadKey();
    }
}
```



Struct Kavramı

```
struct Books
```

```
{  
    public string title;  
    public string author;  
    public string subject;  
    public int book_id;  
};
```



```
public class testStructure
```

```
{  
    public static void Main(string[] args)  
    {  
  
        Books Book1; /* Book1 tanımlandı */  
        Books Book2; /* Book2 tanımlandı */  
  
        /* Book1 özellikleri */  
        Book1.title = "C Programming";  
        Book1.author = "Nuha Ali";  
        Book1.subject = "C Programming Tutorial";  
        Book1.book_id = 6495407;  
  
        /* Book2 özellikleri */  
        Book2.title = "Telecom Billing";  
        Book2.author = "Zara Ali";  
        Book2.subject = "Telecom Billing Tutorial";  
        Book2.book_id = 6495700;  
  
        /* Book1 içeriği */  
        Console.WriteLine("Book 1 title : {0}", Book1.title);  
        Console.WriteLine("Book 1 author : {0}", Book1.author);  
        Console.WriteLine("Book 1 subject : {0}", Book1.subject);  
        Console.WriteLine("Book 1 book_id : {0}", Book1.book_id);  
  
        /* Book2 içeriği */  
        Console.WriteLine("Book 2 title : {0}", Book2.title);  
        Console.WriteLine("Book 2 author : {0}", Book2.author);  
        Console.WriteLine("Book 2 subject : {0}", Book2.subject);  
        Console.WriteLine("Book 2 book_id : {0}", Book2.book_id);  
  
        Console.ReadKey();  
    }  
}
```

C# Struct Özellikleri

- C#'ta Yapılar C ve C++ gibi dillere göre farklılıklar gösterir. C# Yapılarının özellikleri şöyledir:
- Yapılar metotlar (methods), alanlar (fields), özellikler (properties), operator metotları (operator methods) ve olaylar (events) içerebilir.
- Yapılara constructor tanımlanabilir, ama destructor (yıkıcı) tanımlanamaz. Ancak, yapılara geçerli bir constructor tanımlanmaz. İllaki parametrelili bir constructor olmalı.
- Bir struct yaratmada new operatörü kullanıldığı zaman, yaratılmış constructor çağrılmış olur. Sınıflardan farklı olarak, new operatörü kullanmadan da struct'ları kullanmak mümkündür.
- Eğer new operatörü kullanılmazsa, alanlar tanımlanmamış kalır ve nesne tüm alanlar tanımlanmadan kullanılamaz.

C# Struct Özellikleri

- Sınıflardan farklı olarak yapılar diğer sınıf veya yapılardan kalıtım uygulanamaz. Diğer bir deyişle, yapılar diğer yapılara veya sınıflara base olarak kullanılamaz.
- Bir yapı bir veya birden fazla interface'e uygulayabilir. (DateTime'ı hatırla)
- Yapı üyeleri üzerinde özel anahtar kelimeleri ile farklı özellikler kullanılamaz. Örneğin: abstract, virtual veya protected olarak belirlenemez.

C# Struct Özellikleri

```
public struct ogrenci_s  
{
```

```
    public string AdSoyad;  
    private int _Not;
```

```
    public ogrenci_s(string a, int n)
```

```
{  
    AdSoyad = a;  
    _Not = n;  
    //Not değişkeni kullanılamıyor ama class kullanılabiliyor...  
}
```

```
//structta parametresiniz constructor olmaz.  
//public ogrenci_s(){  
//    AdSoyad = "Test";  
//    _Not = 0;  
//}
```

```
    public int Not {  
        get { return _Not; }  
        set {  
            _Not = value;  
            if (_Not < 0)  
                _Not = 0;  
            if (_Not > 100)  
                _Not = 100;  
        }  
    }  
}
```

```
    public void Bilgi()  
    {  
        Console.WriteLine("Adı Soyadı: (struct)" + AdSoyad + " : " + Not);  
    }  
}
```

```
//struct sonu
```

Sadece public veya private erişim kullanılır.

Constructor – İlk değer belirleme fonksiyonu

Özellik

C# Sınıf vs Struct

```
public class ogrenci_c
{
    public string AdSoyad;
    private int _Not;
    //Sadece class için tanımlanabilir
    public ogrenci_c(){
        AdSoyad = "Guest";
        Not = 0;
    }

    //struct ile aynı
    public ogrenci_c(string a, int n){
        AdSoyad = a;
        Not = n; //struct'tan farklı olabiliyor
    }

    //struct ile aynı
    public int Not
    {
        get { return _Not; }

        set {
            _Not = value;
            if (_Not < 0)
                _Not = 0;
            if (_Not > 100)
                _Not = 100;
        }
    }

    //struct ile aynı
    public void Bilgi()
    {
        Console.WriteLine("Adı Soyadı: (Class) " + AdSoyad + " : " + Not);
    }
}
```

- Struct'tan farklı olarak Class'ta boş constructor tanımlanabilir.
- Erişim belirteçlerinin tümü kullanılabilir.
 - Private, public, protected sınıf dışından erişim için kullanılan belirteçlerdir.
 - Private ve protected dışarıdan erişime izin vermez.
 - Public dışarıdan metod veya değişkene erişime izin verir.

C# İç içe yapılar (Nested)

- Sınıf ve yapılar iç içe kullanılabilir.

```
//iç içe yapılar (nested structure)
public struct S1
{
    public string A;

    public struct S2
    {
        public string B;
    }
}
```

```
//iç içe sınıflar (nested class)
public class C1
{
    public string A;

    public class C2
    {
        public string B;
    }
}
```

İç yapıyı tanımlama

```
class Program
{
    static void Main(string[] args)
    {
        /*S1 _s1 = new S1();
        _s1.A = "Değer 1";

        S1.S2 _s12 = new S1.S2();
        _s12.B = "İç Değer";*/

        //new kullanmadan
        S1 _sn1;
        _sn1.A = "Değer 1";

        S1.S2 _sn12;
        _sn12.B = "İç Değer";

        //sınıf yaratılmak zorundadır.
        C1 _c1 = new C1();
        C1.C2 _c2 = new C1.C2();
    }
}
```

Struct ve Sınıf Farkları

Class'la referans tipindendir (reference types)

Struct'lar ise de değer tipindendir. (value types)

Yapılar kalıtım / miras desteklemez.

Yapıların geçerli constructor'a sahip değildir. Diğer deyişle constructor'ları parametre içermek zorundadır.

Struct ve Sınıf Farkları Örnek

```
public class sinif
{
    public int x;
}

public struct yapı
{
    public int x;
}
```

- Bu örnekte referans tipinden olan sınıf ile değer tipinden olan yapı yaratılacak ve aktarımlar üzerinden fark anlatılacaktır.

Struct ve Sınıf Farkları Örnek

c2=_c1 işleminde adres ataması yapılır. Bu aşamadan sonra _c1 üzerindeki her değişiklik _c2'ye yansır. Başka _c1 ve _c2 adresleri eşitlenmiştir.

```
static void Main(string[] args)
{
    sınıf _c1 = new sınıf();
    _c1.x = 5;

    sınıf _c2 = new sınıf();
    _c2 = _c1; // bellek üzerine işlem yapılır. Class referans türüdür.
```

```
    Console.WriteLine("Class ve Nesne üzerine olan işlemler");
    _c1.x = 8;
    Console.WriteLine(_c2.x.ToString());
```

```
    _c2.x = 7;
    _c1.x = 10;
    Console.WriteLine(_c1.x.ToString());
    Console.WriteLine(_c2.x.ToString());
```

Her iki çıktı sonucu: 10
_c1.x'teki değişim her iki sınıfı etkiler.

```
    Console.WriteLine("Struct üzerine olan işlemler");
    yapı _s1, _s2;
    _s1.x = 5;
    _s2 = _s1;
    Console.WriteLine(_s2.x.ToString());
```

```
    //Struct değer türüdür. bellek adresleri üzerine değil, direkt bellekteki içerik üzerine işlem yapılır.
    _s1.x = 13;
    _s2.x = 1;
    Console.WriteLine(_s1.x.ToString());
    Console.WriteLine(_s2.x.ToString());
```

Çıktılar sırasıyla: 13 ve 1'dir.
_s1.x'teki değişim sadece kendi yapısını etkiler.

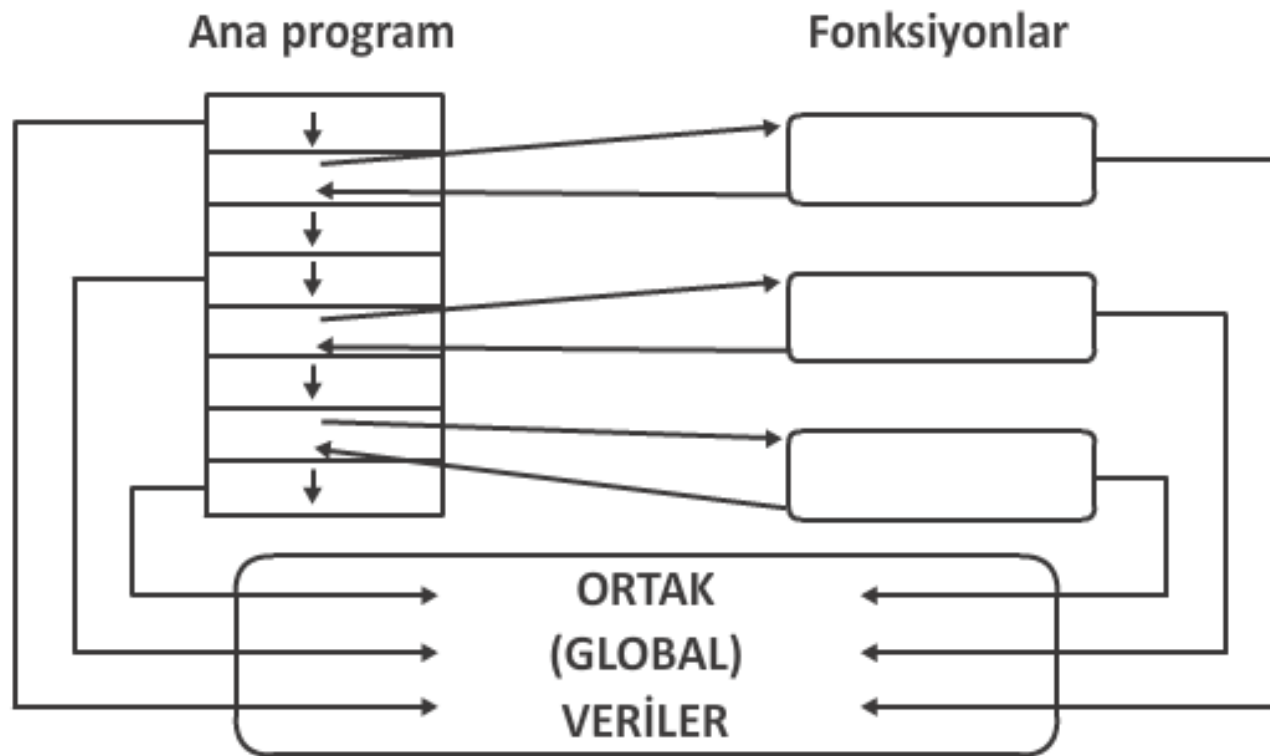
```
    Console.ReadLine();
```

```
}
```

Prosedür 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 Programlama



Nesneye Yönelik 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 kapsamı 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.

Nesneye Yönelik 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.

Nesneye Yönelik Programlama

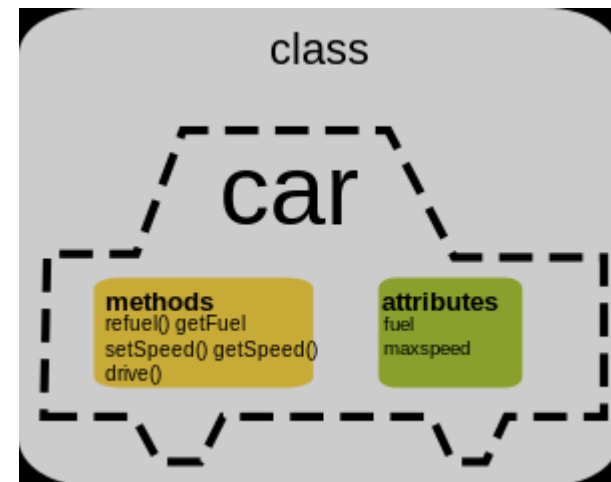
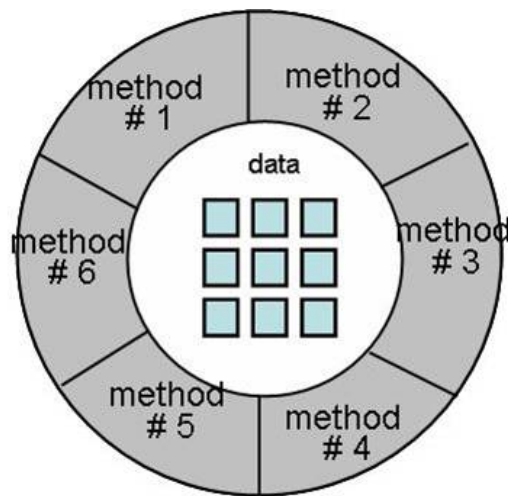
- Bunu basitçe, prosedür tabanlı programlamada bulunan soyut program geliştirme mantığını rafa kaldırıp, gerçek dünya modellemesi ile program geliştirme çabası olarak da düşünebiliriz.
- Gerçek dünya modellemesiyle anlatılmak istenen şudur:
 - Bir fabrika örneğini ele alalım. Bu fabrikada işçiler, makineler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

Nesneye Yönelik 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.

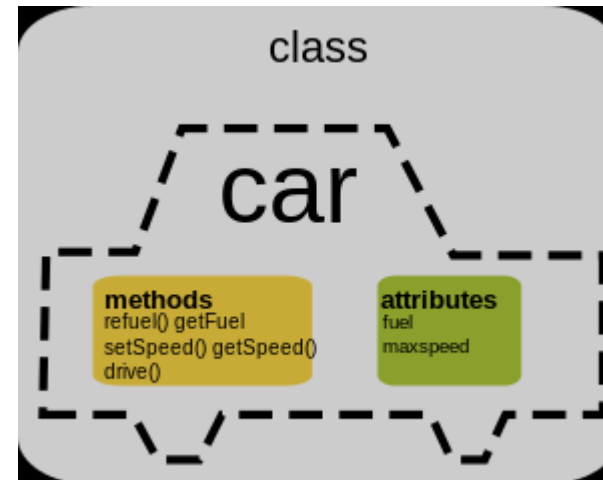
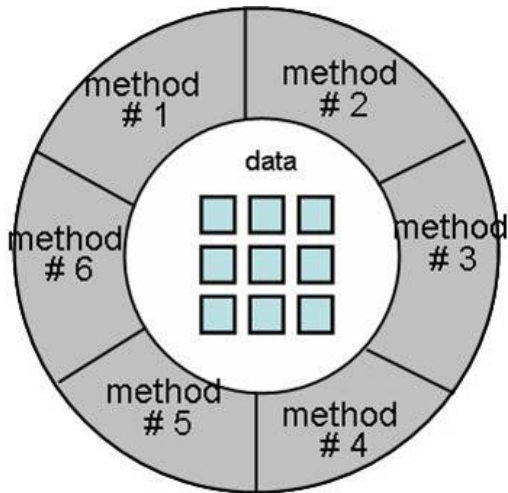
Sınıf

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



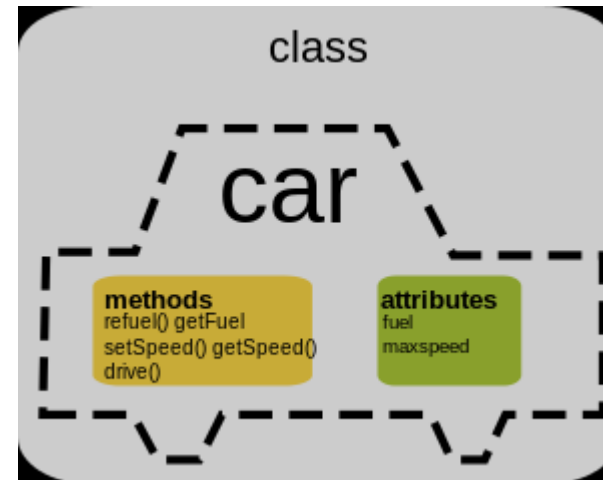
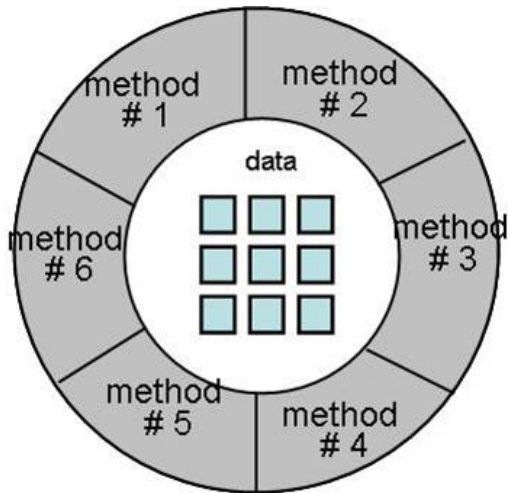
Sınıf

- 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.



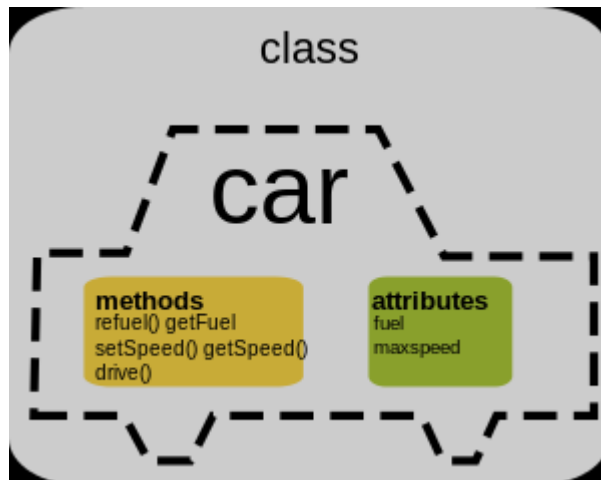
Sınıf - Metodlar

- 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 - Özellikler

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



Sınıf - İsimlendirme

Name (Identifier)	Student	Circle
Variables (Static attributes)	name grade	radius color
Methods (Dynamic behaviors)	getName() printGrade()	getRadius() getArea()

SoccerPlayer	Car
name number xLocation yLocation	plateNumber xLocation yLocation speed
run() jump() kickBall()	move() park() accelerate()

Examples of classes

C# Nesne Tanımlama

<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.

C# Nesne Tanımlama

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

C# Nesne Tanımlama

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

Tanımlama

Kutu 1 - Özellikleri
değiştirme

Kutu 2 - Özellikleri
değiştirme

C# Metodlar ve Erişim Belirteçleri

```
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;
    }
}
```

- public bir erişim belirtecine nesneden ulaşılabilir.
- private bir değişkene nesneden ulaşamaz.
 - Length, breadth, height özelliklerine nesne dışından ulaşamaz.
- Ulaşmak için 1. ve 2. yordam tanımlanmıştır.

Özet

- Struct
 - Tanımlama
 - Sınıf kavramından farkı
- Nesne
 - Nesne kavramı
 - C# üzerinde nesne yapıları
 - C# erişim belirteçleri