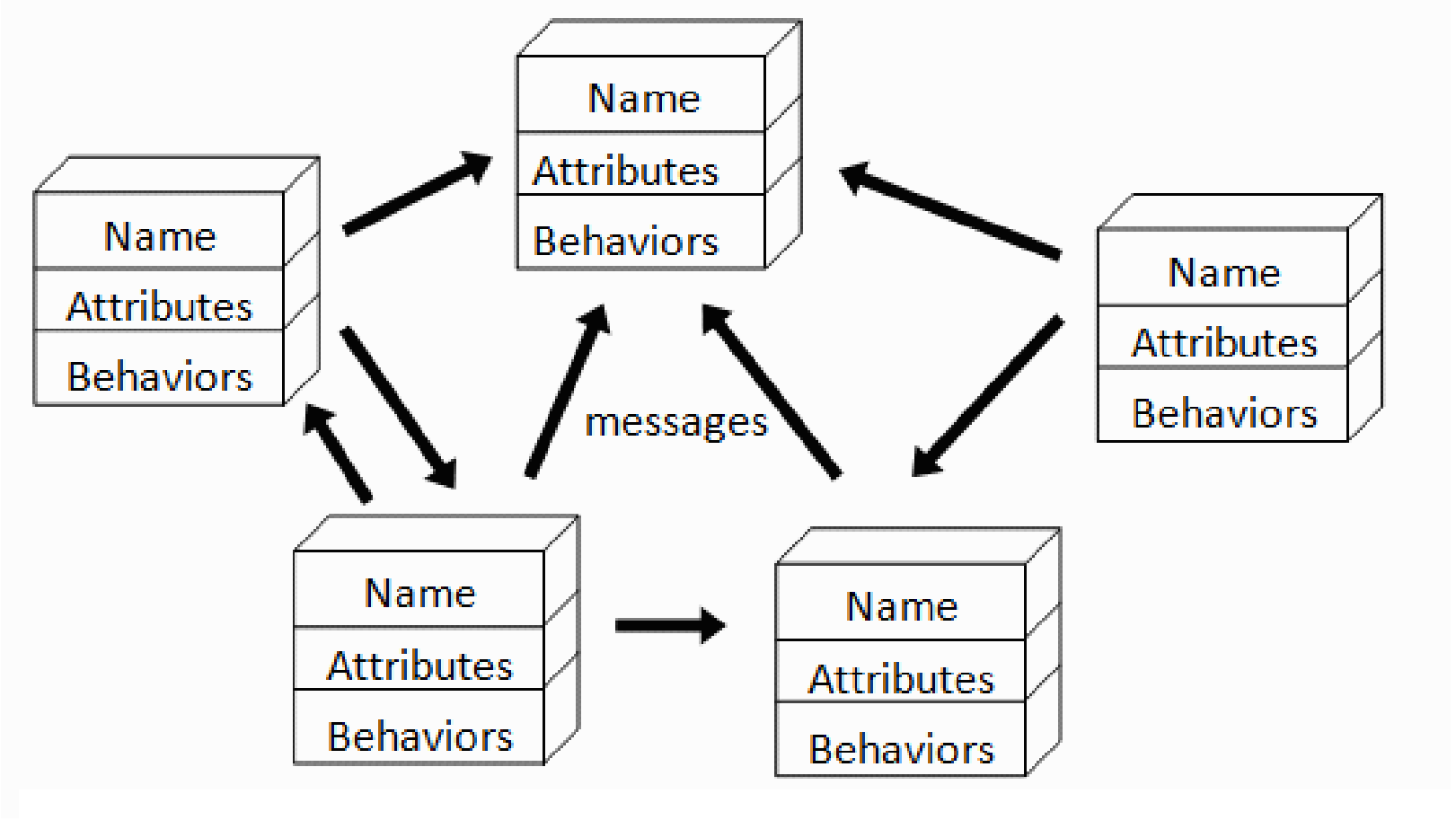
# YZM 3017
# Yazılım Tasarımı ve Mimarisi

**Prof. Dr. Hamdi Tolga KAHRAMAN**
**Arş. Gör. M. Hakan BOZKURT**
**Arş. Gör. Sefa ARAS**

# OOP Nedir?

Nesne yönelimli programlama olarak da ifade edebileceğimiz OOP, yapılması istenen işi küçük parçalara bölüp her bir parça arasındaki ilişkiyi kurarak büyük çapta uygulamalar yazmaya yarayan bir metodolojidir.

# OOP Nedir?


Name
Attributes
Behaviors
Name
Attributes
Behaviors
messages
Name
Attributes
Behaviors
Name
Attributes
Behaviors
Name
Attributes
Behaviors

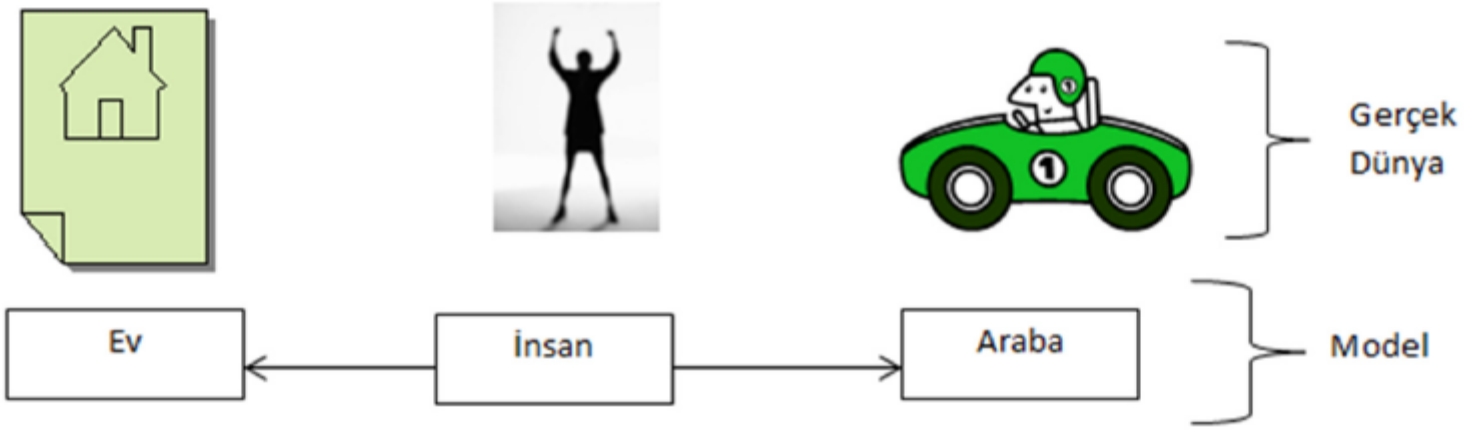
# Nesne Tabanlı Programlama Kavramları

- Nesne Tabanlı Programlama (OOP) tekniği, gerçek dünyadaki somut nesnelerin bir yazılımın sunduğu soyut modelde birer karşılığı olmasına dayanır.
- Bu karşılıklara da **Nesne (object)** denir.

- Nesne tabanlı programlama ile gerçek dünyadaki varlıklar ile bu varlıkların yer aldığı ilişkilerin bire bir modellenmesi ve bu modele göre kod yazılması mümkündür.

# Nesne Tabanlı Programlama Kavramları

- Örneğin, Bir insanın sahip olabileceği iki varlık ve bu üçlünün yazılıma eşlenmesi



# Neden OOP?

OOP kullanılmadan da uygulamalar yazılabilir. Fakat aşağıda OOP' un sağladığı bazı kolaylıklar sıralanmıştır.

- Çok daha az kod yazmak
- Hatalarda merkezi kontrol
- Daha anlamlı kod yazmak
- Belirli bir hiyerarşiye göre kod yazmak
- Gerçek hayattaki her şeyi programınızda simule edebilmek

# Nesne Tabanlı Programlama Kavramları

- Nesne tabanlı geliştirmede de diğer sistem geliştirme süreçlerinde olduğu gibi modelleme aynı zamanda bir sadeleştirmedir. Ne kadar sade modellerimiz olursa o kadar sade kodlarımız olur.
- Örneğin bir bordro programında insanların boy ve kilolarının önemi olmaz.
- Bazı durumlarda da aynı kişi tamamen aynı verilerden beslenen başka bir role sahip olabilir. Örneğin, sağlık sisteminde aynı veriler bir hastayı simgeleyecektir.

## Nesne Tabanlı Programlama Kavramları

- Nesneye Yönelik programlamada üzerinde işlem yapılacak bu nesnelerin yaratılması ve tanınması gerekir.
- Nesnelerin programda nasıl yaratılacağı dilden dile farklılık gösterir (C++, Java).
- Ancak nesnenin yaratılması için nesnenin bir plana ihtiyacı vardır. Bu plana **sınıf** (class) denir

# Class(Sınıf)

OOP’ un temel yapı taşı sınıflardır. Sınıflar, verileri modellemeye yarar ve nesneler oluşturabilmemizi sağlar.

Array, Random, Convert, Int32, String, MessageBox gibi sıkça kullandığımız komutlar da aslında birer sınıftır. Sınıflar sayesinde kendi tiplerimizi tanımlayacağız.

# SINIF

Bir sınıf, ortak özellikleri ve davranışları olan nesnelerin sözkonusu ortak özelliklerini ve ortak davranışlarını barındıran soyut bir kavramdır.

- Yani bir sınıf temel olarak bir tanımdır.
- Bir özellik ve davranış yığını sunar ve bu yığına topluca erişmemiz için bize birer isim verir.
- Bu ismi taşıyan nesneler bu yığındaki özellik ve davranışlara sahip olurlar.
- Bu ilişkiyi nesne sınıfa aittir (belongs to) diyerek açıklarız.
- ***Bir nesne bir sınıfta tanımlanan özellik ve davranışlara sahipse o sınıfa aittir.

# Class Tanımlaması

```
class <Class İsmi>
{
// Bir class özellikler, metotlar ve eventlar içerebilir.
}
```

Örnek:Aşağıda bir şirkette çalışan personelleri programatik olarak anlamlandırmak ve tanımlamak için Personel sınıfı oluşturulacaktır.

```
class Personel
{
/* Personellere ait özellikler ve işlemler yer alacaktır.
   ad, soy ad, yas, sigorta numarası gibi özellikleri ve maaş hesapla gibi
    işlemleri bulunacaktır. */
}
```

# NESNE

- Sınıf nesne ilişkisinde kullanılan önemli diğer bir kavramda bir nesnenin ait olduğu sınıfın bir **örneği** (instance) olmasıdır.

Bir nesne, uygulama çalışırken bellekte ya da başka bir kayıt ortamında (dosya, veritabanı, vs) bulunan ve ait olduğu sınıfın tanımladığı davranış biçimlerine uyan bir örnektir.

# Nesne

Class'lar birer şablondur, nesne ise bu şablondan oluşturulmuş bir örnektir(instance).

## Class (Şablon)

Personel Formu
TC Kimlik : ……
Ad : ……
Soy ad : ……
Sigorta No : ……
…
…
…
…

## Nesne (Instance)

Personel Formu
TC Kimlik : 17231550122
Ad : …Abdülkadir……
Soy ad : …BARIK……
Sigorta No : …123123131
…
…
…
…

# Nesne

Nesne yönelimli programlama tekniğinin en temel bileşeni nesnelerdir. Nesneler içeriklerinde veriler barındırırlar. Veriler arası ilişkiler sağlayan fonksiyonlara da sahiptirler. Nesnelerin veri ve fonksiyon gibi bileşenleri içermesine sarmalama (encapsulation) denilir.

# Nesne

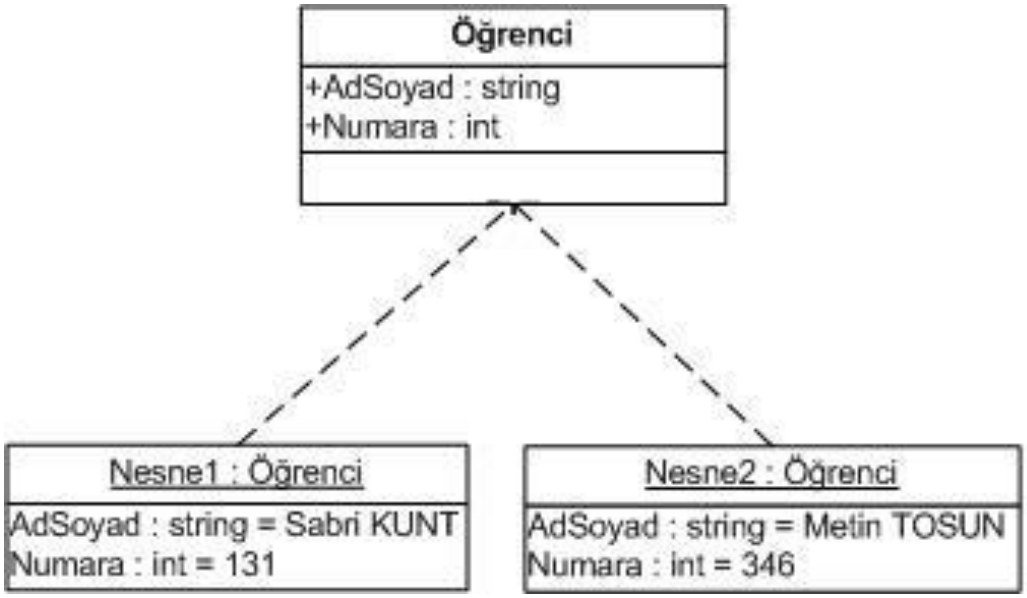
Nesne içindeki veriler ve fonksiyonlar nesnenin dışarıya nasıl hizmet verdiğini belirler. Fakat bu hizmeti nasıl verdiği belli değildir. Nesnenin hizmetlerinden faydalanmak için nesnenin dış dünyadan erişilen arayüzünün bilinmesi yeterlidir. Buna bilgi saklama (information hiding) adı verilir.

# Nesne

Nesnelerin birbirlerinden bağımsız olmasına rağmen aralarında haberleşebilirler. Hangi nesnenin hangi nesneye mesaj göndereceği, hangi nesnelerin fonksiyonlarının kullanılacağı derleme aşamasında belli olmayabilir. Bu durumda geç bağlama (late binding) mekanizmasından faydalanılır.

# Nesne

Nesneler referans tipi değişkenler grubuna girerler yani hafızada heap bölümünde oluşturulurlar.



# Class ve Nesne

## Class Tanımlaması

```
class Personel
{
public string _ad, _soyad;
public int _yas;
public decimal _tckimlikno;
private decimal _primorani;
public decimal MaasHesapla()
   {
     //Hesaplama Komutları
   }
}
```

## Nesne Oluşturulması

```
Personel pers1 = new Personel();
pers1._ad = "Abdülkadir";
pers1._soyad = "BARLIK";
pers1._tckimlikno = 17232423454;
pers1._yas = 25;
Pers1._primorani = 1.12; // hatalı
```

Not: Private olan değişkenlere başka yerden erişilemez sadece o class içerisinde erişilir.

## Nesne Tabanlı Programlama Kavramları

- Bir uygulama çalışırken bir sınıfa ait değişik sayılarda örnek (yani nesne) bulunabilir.
- Herhangi bir anda o sınıfa ait hiç nesne olmayabileceği gibi bir yada daha fazla sayıda nesne de olabilir.
- Bir nesne genelde yalnızca bir sınıfın örneğidir.

# Access Modefiers (Erişim Belirteçleri)

Erişim belirteçleri class içerisindeki özellik ve metotlar için belirlenir. Bu erişim belirteçleri sayesinde bir özellik veya metodun diğer classlardan erişilip erişilemeyeceğini belirtir. Aşağıda erişim belirteçleri sıralanmıştır;

- public
- private
- internal
- protected
- protectedinternal

**Not: Eğer bir özellik veya metodun erişim belirteci belirtilmemişse default olarak "private" değerini alır.**

# Access Modefiers (Erişim Belirteçleri)

| Access Modifier | Açıklama |
| --- | --- |
| **Private** | Sadece bu üyenin bulunduğu sınıf içerisinden erişilebilir. |
| **Internal** | Sadece bu üyenin bulunduğu proje içerisinden erişilebilir. |
| **Protected** | Sadece bu üyenin bulunduğu sınıf ve bu sınıftan türemiş alt sınıflardan erişilebilir. |
| **Protected Internal** | Bu üyeye aynı assembly içerisinden erişilmeye çalışıldığında “internal” gibi, başka assembly içerisinden erişilmeye çalışıldığında da “protected” gibi davranır. |
| **Public** | Public üyelerin erişiminde herhangi bir kısıtlama yapılmaz. İstenilen her yerden üyeye erişim açıktır. |

# Nesne Tabanlı Programlama Kavramları

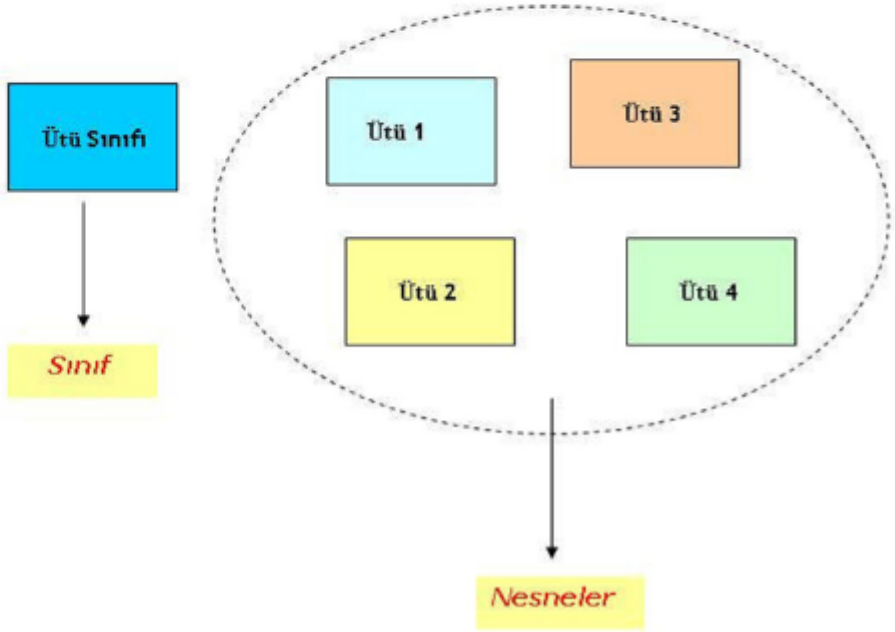
- Sınıflar ve nesnelerden bahsederken sürekli olarak nesnelerin **özellikleri** ve **davranışlarından** bahsederiz.
- Bir nesnenin **özellikleri** (attributes), o nesnenin içindeki **veriler** ve bu verilerin yapısını kapsar.
- Özellikler değişik seviyelerdeki soyutlamaları içerebilir.
- Yani bir nesnenin bir özelliği başka bir nesne olarak ifade edilebilir.
- Bu diğer nesne aynı sınıftan ya da farklı bir sınıftan olabilir.
  - **Örnek:** Bir Ev sınıfı düşünün; komşu özelliği de bir başka Ev nesnesi olabilir.

# Nesne Tabanlı Programlama Kavramları

- Bir nesne üzerindeki **işlemler** (operations) ile nesnenin **davranışını** (behavior) niteleriz.
- Davranış nesnenin sistemde kullanıldığı zaman yapacakları ve yapacaklarının sistemin durumu üzerinde yol açacağı değişikliklerden oluşur.
- Davranış sonunda değişime uğrayan sadece nesne olmak zorunda değildir. Çevresi de değişime uğrayabilir.
- Bazen davranış sonucu nesnenin kendisi hiç değişmezken çevresindeki diğer nesneler değişebilir.

# Nesne Tabanlı Programlama Kavramları

- Örneğin, elimizde bir ütümüz olsun. Ütünün markası, modeli, rengi, çalıştığı elektrik voltajı, ne tür kumaşları ütüleyebildiği bu ütüye ait özelliklerdir (veri).
- Aynı zamanda ütümüzü ısıtabiliriz, ütüleme işinde kullanabiliriz ve soğumaya bırakabiliriz. Bunlar ise ütünün fonksiyonlarıdır (metot).



# Nesne Tabanlı Programlama Kavramları

- Eğer ütü ile ilgili bir program yapmış olsak ve nesne tabanlı programlama tekniğini kullansak hemen bir ütü sınıfı (class) oluştururduk.
- Bu sınıfta ütüye ait bilgiler (veriler) ve ütü ile yapabileceğimiz işler (metot) bulunurdu.
- O zaman nesne tabanlı programlamada bir sınıfta, sınıfa ait veriler ve bu verileri işleyip bir takım faydalı sonuçlar üreten fonksiyonlar metotlar bulunur.
- Dahası, biz bir tane ütü sınıfı tasarlarsak bu sınıftan istediğimiz sayıda değişik ütüler (Object veya instance) yapabiliriz.

## Nesne Tabanlı Programlama Kavramları

- Bir nesne üzerindeki **kısıtlamalar** (constraints), nesnenin uyması gereken kuralları simgeler.
- Bu kurallar genellikle nesnenin özellikleri ile ilgilidir.
- Nesnenin veri bütünlüğünün korunması için gereken şeylerdir.
- Bu kısıtlamalar sayesinde nesnenin özellikleri anlamlı olmaya devam eder.
  - (Kısıtlamalar kötü bir şey olarak algılanmamalıdır.)

## Nesne Tabanlı Programlama Kavramları

- Örneğin, bir araba için sürücünün iki ayağı olacağından, debriyaj, fren ve gaz pedallarından aynı anda en çok ikisine basılabilir.
- Bunu bir kural olarak tanımlarsak, Araba sınıfından nesneler için bir **kısıtlama** olacaktır.

# Nesne Tabanlı Programlama Kavramları

- Kısıtlamalara benzer diğer bir kavram da bir nesnenin uyması beklenen **sözleşmesi** (**contract**) dir.
- Sözleşme, nesnenin üzerindeki işlemler gerçekleştirilirken nasıl davranması gerektiği üzerine kurallardır.
  - Kısıtlamalar ise nesnenin özelliklerinin her zaman uyması gereken kurallardır.
- Sözleşmelerin daha çok davranışa, kısıtlamaların ise veriye odaklı kurallar olduğunu söyleyebiliriz.

# Field(Alan)

- Field'lar sınıflarımız içerisinde veri barınmamızı sağlayan yapılardır. Yani sınıfa ait değişkenlerdir.
- Field'lar değer veya referans tipinden olabilir.
- Eğer field'lara değer atanmazsa default değerleri verilir.

| Veri Tipi | Default Değeri |
|---|---|
| int | 0 |
| long | 0 |
| float | 0.0 |
| double | 0.0 |
| bool | false |
| char | \0' (null karakteri |
| string | "" (boş metin) |
| Object | null |

# Field(Alan)

```
6  namespace OOP
7  {
8      public class Telefon
9      {
10         public string _model;
11         public string _marka;
12         public int _fiyat = 0;
13     }
14 }
15
```

```
private void Form1_Load(object sender, EventArgs e)
{
    Telefon tel = new Telefon();
    tel.
}
```

# Property(Özellik)

- Field'lara yani verilerin tutulduğu alana doğrudan erişim iznini kısıtlamak ve geçerli veriler sağlamak adına "Property" kullanılır.
- Property sayesinde Field'lara koşullu erişim sağlanır.
- Property'ler asla veri tutmaz. Sadece field'ı "kapsüller".

# Property(Özellik)

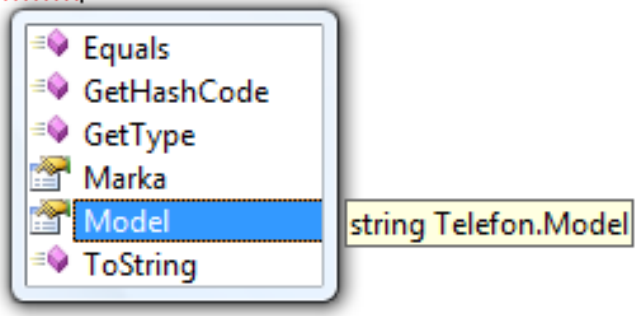
```
//Field
private string _model;

//Property
public string Model
{
    get { return _model; }
    set { _model = value; }
}

//Field
private string _marka;

//Property
public string Marka
{
    get { return _marka; }
    set { _marka = value; }
}
```

- Property, “get” ve “set” olmak üzere iki bloktan oluşur.
- Get bloğu değişkeninin değeri okunmak istendiğinde çalışır.
- Set bloğu değişkene değer atanmak istendiğinde çalışır.

# Property(Özellik)

```
//Field
private string _model;

//Property
public string Model
{
    get { return _model; }
    set { _model = value; }
}

//Field
private string _marka;

//Property
public string Marka
{
    get { return _marka; }
    set { _marka = value; }
}
```

```
Telefon tel = new Telefon();
tel.
```



# Property(Özellik)

- Eğer değer atanırken veya okunurken bir kısıtlama getirmek istiyorsak koşulları get ve set blokları içersinde yazmalıyız.

- "Value" değişkene atanmak istenen değeri belirtir.

```
public class Personel
{
    //Field
    private int _yil;

    //Property
    public int Yil
    {
        get { return _yil; }
        set
        {
            if (value >= 1900 && value <= 2100)
            {
                _yil = value;
            }
            else
            {
                throw new Exception("Tarih aralığını aştınız");
            }
        }
    }
}
```

```
private void Form1_Load(object sender, EventArgs e)
{
    Personel pers = new Personel();
    pers.Yil = 1200; //hata oluşacaktır.
}
```

# Constructor (Yapıcı Metot)

Nesneler için classların örneğidir demiştik. Constructor her nesne tanımlandığında devreye giren o nesneye ait özelliklerinin ilk değerlerini atamak için kullanılan metottur.

Constructor Tanımlama Kuralları :

- Metodun ismi class ismi ile aynı olmak zorundadır.
- Geri dönüş tipi olmaz.
- Eğer tanımlanmazsa bile mutlaka default constructor vardır.
- Overload edilebilir.

# Constructor (Yapıcı Metot)

```
class Personel
{
    public string _ad,_soyad;
    public int _yas;

    public Personel() //Constructor
    {
      _ad = “ ”;
      _soyad = “ ”;
      _yas = 0;
    }
}
```

```
Personel pers1 = new Personel();
```

## Constructor (Yapıcı Metot)

Yapıcı metotlar aşırı yüklenmişse türemiş sınıfın yapıcı metotları çağrılırken belli değerlerle temel sınıfında yapıcı metodunun çağrılması mümkündür ve bu işlem base anahtar sözcüğü ile yapılır:

- Public T(string s, int x, int z):base(int x, int z)

# Constructor (Yapıcı Metot)

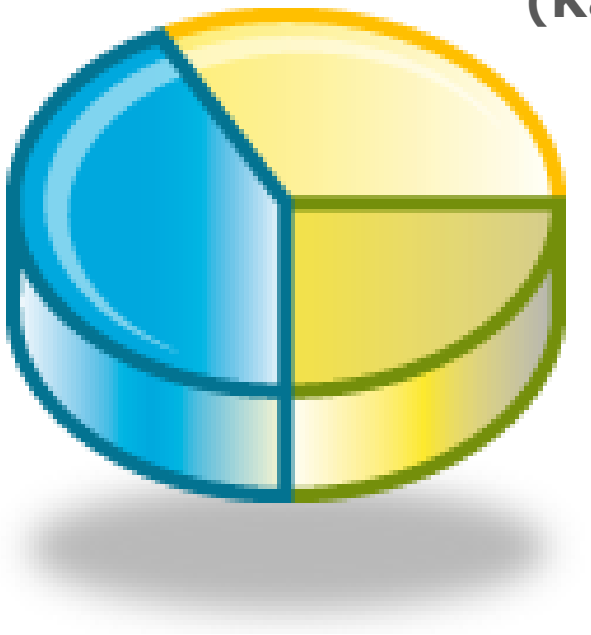
```
using System;
class A
{
    public int a;
    public A(int a)
    {
this.a = a;
Console.WriteLine("A yapıcısı çalıştı\n");
    }

}
class B : A
{
    public int b;
    public B(int a,int b):base(a)
    {
this.b=b;
Console.WriteLine("B yapıcısı çalıştı\n");
    }

}
class C:B
{   public int c;
    public C(int a,int b, int c):base(a, b)
    {
this.c = c;
Console.WriteLine("C sınıfının yapıcısı çağrıldı\n");
    }
}
```

```
class Program
{
    static void Main()
    {

        Console.WriteLine("C Nesnesi");
        Console.WriteLine("---------");
        C c = new C(2, 3, 4);
        Console.WriteLine("a="+c.a);
        Console.WriteLine("b=" + c.b);
        Console.WriteLine("c=" + c.c+"\n");

        Console.WriteLine("B Nesnesi");
        Console.WriteLine("---------");
        B b = new B(5, 6);
        Console.WriteLine("a=" + b.a);
        Console.WriteLine("b=" + b.b+"\n");

        Console.WriteLine("A Nesnesi");
        Console.WriteLine("---------");
        A a = new A(7);
        Console.WriteLine("a=" + a.a+"\n");
        Console.ReadLine();
    }
}
```

Yukarıdaki programda, base anahtar sözcüğü sınıf hiyerarşisinin en tepesindeki sınıfı temsil etmektedir. C sınıfında base anahtar sözcüğü B sınıfı anlamına gelirken, B sınıfında base anahtar sözcüğü A sınıfı anlamına gelmektedir.

# OOP’nin Temel Kavramları

Encapsulation
(Kapsülleme)

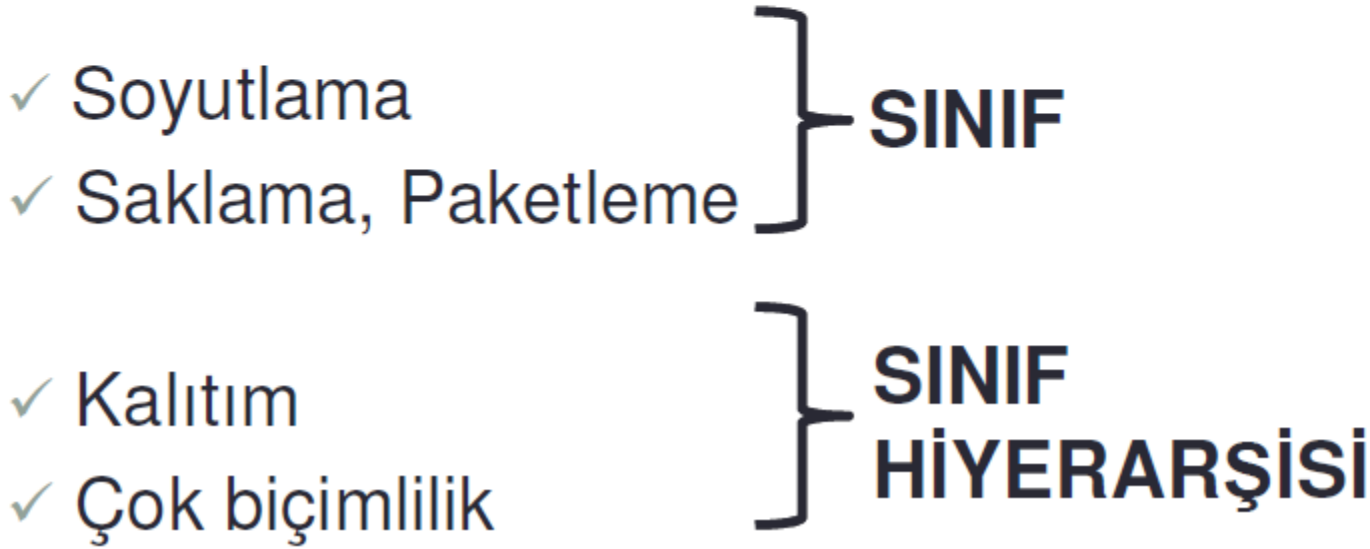


Inheritance
(Kalıtım/Miras)

Polymorphism
(Çok Biçimlilik)

# Nesne Tabanlı Programlama Kavramları- Devam

- Nesneye-yönelik programlamanın temel ilkeleri şunlardır:



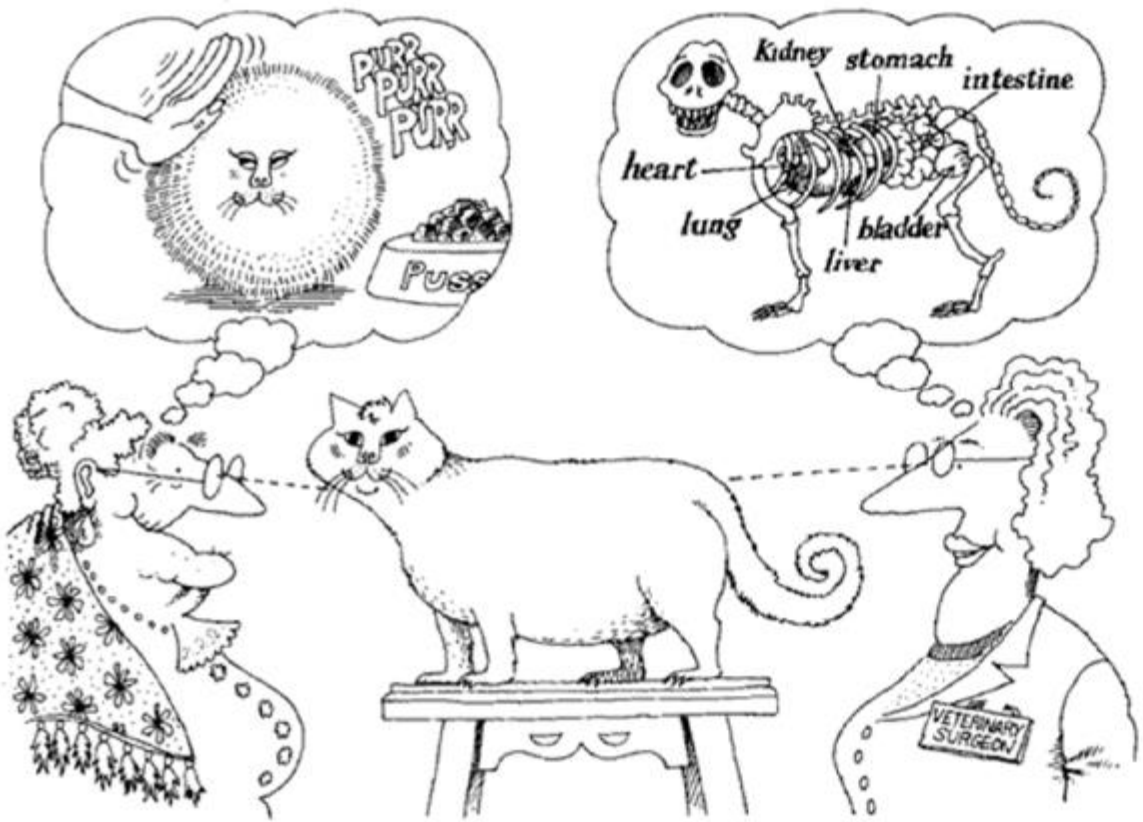
# Nesne Tabanlı Programlama Kavramları- Devam

- Nesneye-yönelik programlamanın temel kavramları,
  1. büyük programlar yazmayı kolaylaştıran **soyutlama** ve programları değiştirmeyi ve korumayı kolaylaştıran **saklama**
  2. **kalıtım** ve **çok biçimliğe** izin veren ve programları kolayca genişletilebilir kılan **sınıf hiyerarşisi**dir.
- Herhangi bir programlama dilinde bu kavramları uygulayabilirsiniz; fakat, nesneye-yönelik programlama dilleri salt bu amaçla tasarlanmışlardır.

# Soyutlama

- "Soyutlama" önemli özelliklere odaklanabilmek için ayrıntıları göz ardı etme sürecidir.
- Geleneksel olarak, bir programlama dili soyutlama yapmaya izin verdiği ölçüde yüksek-düzeyli (high-level) kabul edilir.

# Soyutlama




"Soyutlama" , belirli bir bakış açısından, önemli özelliklere odaklanabilmek için ayrıntıları göz ardı etme sürecidir.

# PROSEDÜREL SOYUTLAMA

- İşlemlere ilişkin ayrıntıları göz ardı etmemize izin veren "prosedürel soyutlama" en yaygın soyutlama tarzıdır.
- Programlarda kişileştirilmiş kod kullanmak yerine, belirli görevleri gerçekleştirmek için standart fonksiyonların oluşturulması bir prosedürel soyutlamadır.
- Kendi fonksiyonlarınızı yazarak, programın yaptığı bir dizi işleme bir isim vermiş olursunuz.

# PROSEDÜREL SOYUTLAMA

- Belirli bir dilde bir program yazarken programcı kendisini bu dilin sunmuş olduğu soyutlama düzeyiyle sınırlamak zorunda değildir. Birçok dil **kullanıcı-tanımlı fonksiyonlar** (rutinler, prosedürler) yardımıyla prosedürel soyutlama düzeyini daha yukarılara taşımaya izin verir.
- Prosedürel soyutlama ile kod tekrarlarından kurtulmak mümkündür.

# Veri Soyutlaması

- Bir veri tipinin nasıl yapılandığının ayrıntılarını göz ardı etmemize izin veren soyutlama tarzına "veri soyutlaması" denir.
- Örneğin, bilgisayardaki her tür veri ikili sayılar olarak düşünülebilir. Fakat, birçok programcı ondalık sayılarla düşünmeyi tercih ettiği için, dillerin çoğu tam ve "floating" sayıları destekler.
  - Basic dili karakter katarı (string) tipini bir veri soyutlaması olarak destekler. Diğer yandan, C dili string soyutlamasını doğrudan desteklemez. Bu dilde stringler ardışık bellek hücrelerini işgal eden bir dizi karakter olarak tanımlanmıştır.

# Veri Soyutlaması

- Prosedürel soyutlama kapasitelerinin aksine, birçok dil yeni veri soyutlaması düzeyleri yaratmak konusunda sınırlı destek sağlarlar.
- C kullanıcı tanımlı veri tiplerini "structure"lar ve "typedef"ler aracılığıyla destekler.
- C dili, içsel olarak birbirlerine bağlı olmalarına rağmen, prosedürel soyutlamayı ve veri soyutlamasını iki ayrı teknik olarak sunar.
- Bu tekniklerin birleştiği noktada nesne-tabanlı ya da nesneye-yönelik programlama yaklaşımı doğar.

## Saklama, Paketleme (Encapsulation)

- Saklama (Paketleme), soyutlamayı desteklemek ya da güçlendirmek için bir sınıfın iç yapısının gizlenmesidir.
- Bu şekilde veriler fonksiyonlarla gizlenebilir.
- Nesnenin içindeki kod, veri veya her ikisi bu nesneye **private (özel)** veya **public (genel)** olabilir.

# Saklama, Paketleme (Encapsulation)

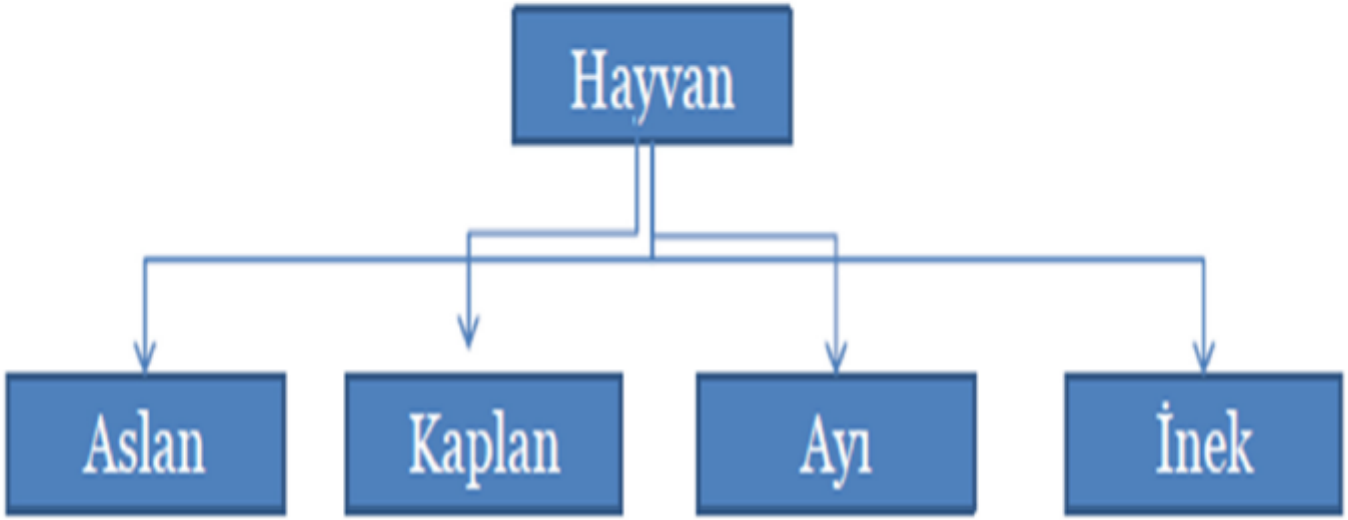
- **Private** kod ve veriler, sadece o nesneye ait bileşenler tarafından bilinebilir ve erişilebilir. Programda nesneye ait olmayan bileşenler, bu private kod ve verilere erişemez.
- Kod ve verilerimiz **public** ise, belirli bir nesnenin içinde tanımlansalar da programın diğer parçaları tarafından erişilebilirler.
- Nesnelere ait public elemanları, private elemanlara denetlenebilir şekilde erişebilmek amacıyla kullanırız.

# Sınıf Hiyerarşisi

- Nesneye-yönelik programlamanın, prosedürel programlamada bulunmayan, bir özelliği, tip hiyerarşisi tanımlayabilme yeteneğidir.
- Nesneye-yönelik programlama, bir sınıfın başka bir sınıfın alt-tipi olarak tanımlanmasına; sınıflar arası benzerlikleri bir ortak üst-sınıf altında toplamaya izin verir.
- Birkaç sınıf için ortak bir üst-sınıf tanımlama da bir tür soyutlamadır.
- Sınıfların ortaklaşa taşıdıkları bazı yönler üzerinde odaklaşıp diğerlerini göz ardı etmeye izin verir.

# Sınıf Hiyerarşisi

•"Aslan", "kaplan", "ayı" ve "inek" türünün üst-sınıfı ne olabilir?



# KALITIM (MİRAS ALMA) - Inheritance

**Kalıtım**, bir sınıfa ait özellikler ve işlemlerin yeni bir sınıfa birebir aktarılmasıdır.

Yeni sınıf eski sınıfın özellikleri ve işlemlerine sahip olmasına ek olarak kendisi de özellikler ve işlemler tanımlayabilir.

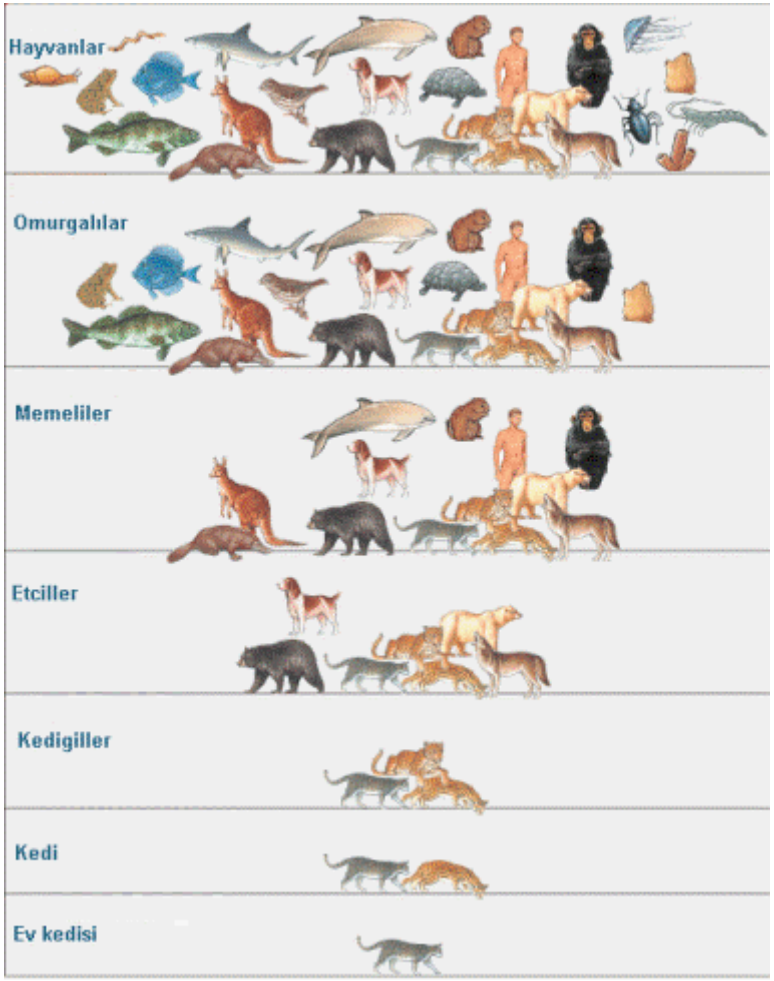
# Inheritance(Kalıtım)

- OOP'un en temel kavramlarından biri Inheritance'dır.
- Bir class'ın üyelerini(field, property, metot gibi) başka bir class'a ancak kalıtım yoluyla aktarılabiliriz.
- Reusability(Tekrar Kullanılabilirlik) sağlar.
- Inheritance yoluyla classlar birbirlerinden türetilirler.

# KALITIM (MİRAS ALMA) - Inheritance

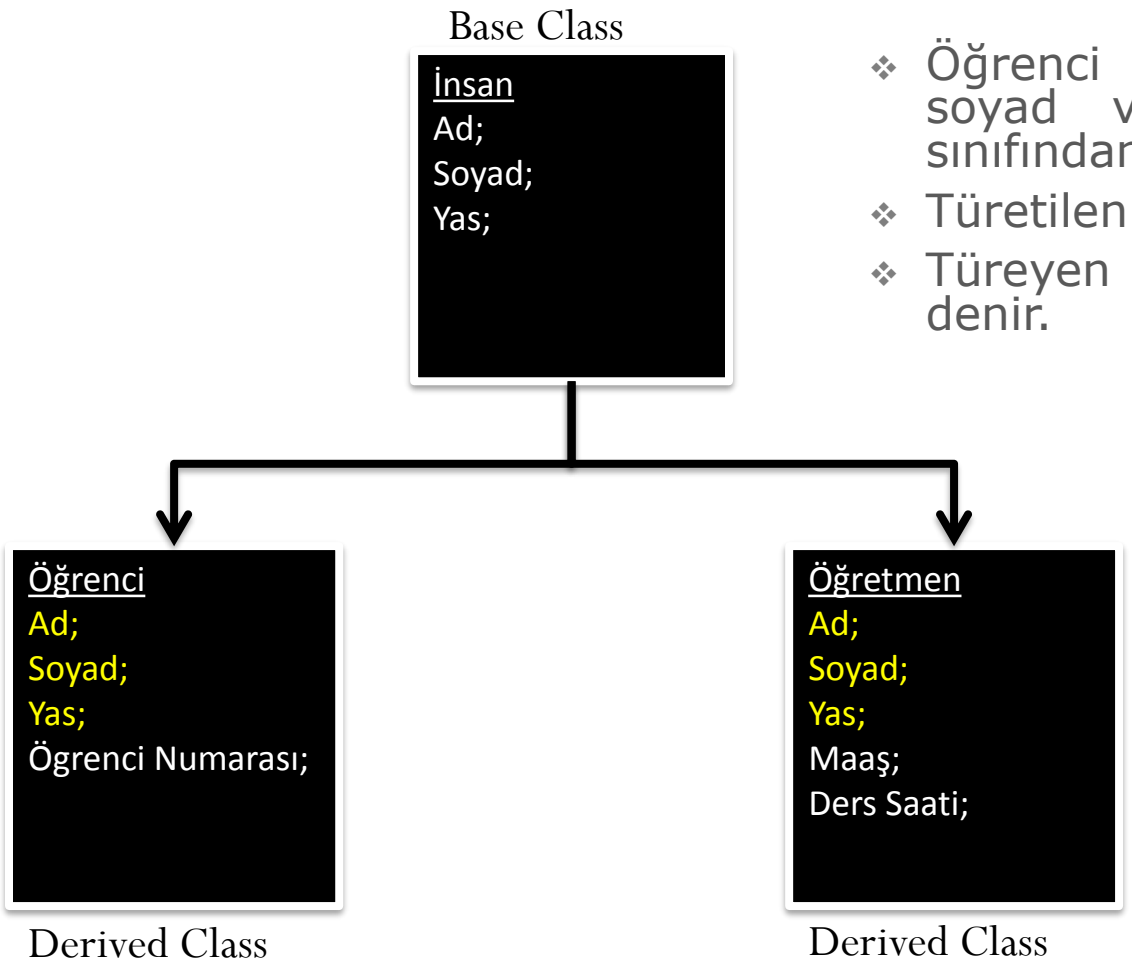
- Kalıtım yolu ile eldeki sınıflardan yeni sınıflar türetilir.
- Türeyen sınıflar türedikleri sınıfın özelliklerini kalıtım yoluyla devralırlar ve kendisi de yeni özellikler tanımlayabilir.
- Türetme ile sınıflar arasında hiyerarşik bir yapı kurulabilir.

# Inheritance(Kalıtım)




- Hayvanlardan ev kedisine doğru gidildikçe sahip olunan özellikler spesifikleşir.
- Hayvanlarda tüm hayvanlara ait genel özellikler bulunmalıdır.

# Inheritance(Kalıtım)

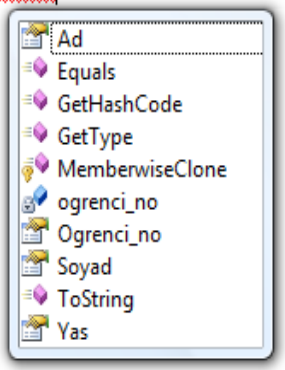



- Öğrenci ve Öğretmen sınıfları "ad, soyad ve yas" özelliklerini İnsan sınıfından miras alacaktır.
- Türetilen sınıfa "Base Class" denir.
- Türeyen sınıfa da "Derived Class" denir.

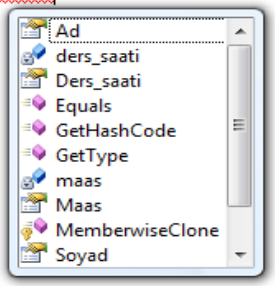
# Inheritance(Kalıtım)

```
class Insan
{
    private string ad;

    public string Ad
    {
        get { return ad; }
        set { ad = value; }
    }
    private string soyad;

    public string Soyad
    {
        get { return soyad; }
        set { soyad = value; }
    }
    private int yas;

    public int Yas
    {
        get { return yas; }
        set { yas = value; }
    }
}
```

```
class Ogrenci:Insan
{
    int ogrenci_no;

    public int Ogrenci_no
    {
        get { return ogrenci_no; }
        set { ogrenci_no = value; }
    }
    public Ogrenci()
    {
        this.
    }
}
```




```
class Ogretmen:Insan
{
    private int maas;

    public int Maas
    {
        get { return maas; }
        set { maas = value; }
    }
    private int ders_saati;

    public int Ders_saati
    {
        get { return ders_saati; }
        set { ders_saati = value; }
    }
    public Ogretmen()
    {
        this.
    }
}
```



# Inheritance(Kalıtım)

Türetme yapmak için sınıf tanımlaması şu şekilde yapılmalıdır:

```
class TüretilenSınıf : TemelSınıf
```

Türetme işleminden sonra türetilen sınıf temel sınıfın bütün özelliklerine sahip olur.

private özelliklere türetilen sınıflardan erişilemez. protected özellikler ise türeyen sınıfa private olarak geçer.

# Kalıtım

```
class A
{
    public int x;
    private int y;
    protected int z;

    public A()
    {
        x = 1;
        y = 2;
        z = 3;
        Console.WriteLine("A yapıcısı çalıştı...");
    }

    public void Listele()
    {
        Console.WriteLine("x={0}; y={1}; z={2};",x,y,z);
    }
}

class T : A
{
    public string s;

    public T()
    {
        s = "Türemiş Sınıf";
        Console.WriteLine("T yapıcısı çalıştı...");
    }

    public void Yaz()
    {
        Console.WriteLine("s={0}; x={1}; z={2};", s, x, z);
    }
}

class Program
{
    static void Main()
    {
        T t = new T();
        t.Listele();
        t.Yaz();
    }
}
```

# İsim Saklama (Name Hiding)

Türemiş sınıfta bazen temel sınıftaki üye elemanla aynı isimli bir eleman tanımlanmış olabilir. Bu durumda temel sınıftaki elemana normal yollarda erişmek mümkün değildir çünkü türeyen sınıftaki eleman temel sınıftaki elemanı gizlemiştir.

Temel sınıftaki elemana erişmek için yine base anahtar sözcüğünden faydalanılır.

Base ile hem özelliklere hemde metotlara erişilebilir.

# Kalıtım

```
using System;

class A
{
    public int a;

    public A()
    {
        a = 1;
    }
}

class T : A
{
    public int a;

    public T()
    {
        a = 2;
    }
}

class Program
{
    static void Main()
    {
        T t = new T();
        Console.WriteLine(t.a);
    }
}
```

```
using System;

class A
{
    public int a;

    public A()
    {
        a = 1;
    }
}
class T : A
{
    public new int a;
    public int b ;
    public T()
    {
        a = 2;
        b = base.a;

    }
}
class Program
{
    static void Main()
    {
        T t = new T();
        Console.WriteLine(t.a);
        Console.WriteLine(t.b);
        Console.ReadLine();
    }
```

# Kalıtım

base anahtar sözcüğünün örneklerdeki gibi kullanımı this referansına benzemektedir. this referansı kendisini çağıran sınıfı temsil ederken base anahtar sözcüğü türetmenin yapıldığı temel sınıfı temsil eder.

# Temel ve Türeyen Sınıf Nesneleri

- Tip güvenliği olan dillerde farklı türdeki nesnelerin birbirine atanması istisna durumlar dışında yasaktır.

- Bu istisna durumlardan biri de türemiş sınıfın referansının temel sınıfa ilişkin bir referansa atanabilmesidir.

- Bu durumda temel sınıf türeyen sınıfın tüm özelliklerine erişemeyecek olmasına rağmen atama işlemi yapılabilmektedir.

# Kalıtım

```
class Program
    {
        public static void Goster(oto Oto)
        {
            Console.WriteLine(Oto.Tur);
            Console.WriteLine(Oto.MotorGucu);
            Console.WriteLine(Oto.Tork);
            Console.WriteLine(Oto.Renk);
        }

        static void Main(string[] args)
        {
            oto oto1=new oto
    (75,100,"Kırmızı");
            Goster(oto1);
            Console.WriteLine("----------");
            model1 oto2=new
    model1("Fiat",100,110,"Beyaz");
            Goster(oto2);
            Console.WriteLine("----------");
            model2 oto3=new model2
    ("Renault",100,120,"Siyah");
            Goster(oto3);
            Console.ReadLine();

        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication9
{
    class oto
    {
        protected double motorgucu=5;
        protected double tork=800;
        protected string renk="Sarı";
        public oto(double guc, double tork, string
    renk)
        {
            this.motorgucu = guc;
            this.tork = tork;
            this.renk = renk;
        }
        public void ozellikgoster()
        {
            Console.WriteLine("Motor Gücü=" +
    motorgucu);
            Console.WriteLine("Tork=" + tork);
            Console.WriteLine("Renk=" + renk);
        }
        public double MotorGucu
        {
            get { return motorgucu; }
            set { motorgucu = value; }
        }
        public double Tork
        {
            get { return tork; }
            set { tork = value; }
        }
        public string Renk
        {
            get { return renk; }
            set { renk = value; }
        }
    }
```

```
class model1:oto
    {
        public string Tur;
        public model1(string tur,double guc,
    double tork,string renk):base
    (guc,tork,renk)
        {
            this.Tur=tur;
        }
        public void TurGoster()
        {
            Console.WriteLine("Türü"+Tur);
        }
    }

    class model2 : oto
    {
        public string Tur;
public model2(string tur,double guc, double
    tork,string renk):base (guc,tork,renk)
        {
            this.Tur=tur;
        }
        public void TurGoster()
        {
            Console.WriteLine("Türü"+Tur);
        }
    }
```

# KALITIM

- Bir sınıf hiyerarşisi tanımlamanın 2 pratik faydası vardır:
  - Türetilmiş sınıf üst-sınıfın kodunu paylaşabilir
    **(Kod Kalıtımı)**
  - Türetilmiş sınıf üst-sınıfın arayüzünü paylaşabilir
    **(Arayüz Kalıtımı)**

# KOD KALITIMI

- Eğer yeni bir sınıf tanımlıyorsanız ve mevcut bir sınıfın işlevselliğinden yararlanmak istiyorsanız, yeni sınıfınızı mevcut sınıftan türetirsiniz.
- Bu durumda **kalıtım** mekanizmaları size mevcut kodu yeniden kullanma imkanı sağlar.

- **Örnek:** Veri Girişi Formlarının tasarlanması.

# ARAYÜZ KALITIMI

- Bir diğer kalıtım stratejisi, türetilmiş sınıfın üst-sınıfının eleman fonksiyonlarının yalnızca isimlerini kalıtım yoluyla almasıdır.
- Türetilmiş sınıf bu fonksiyonlar için kendi kodunu kullanır.
- Arayüz kalıtımının temel faydası çok-biçimliliğe izin vermesidir.

- **Örnek:** Farklı veri tiplerindeki form girişleri ayrı ayrı alınır.

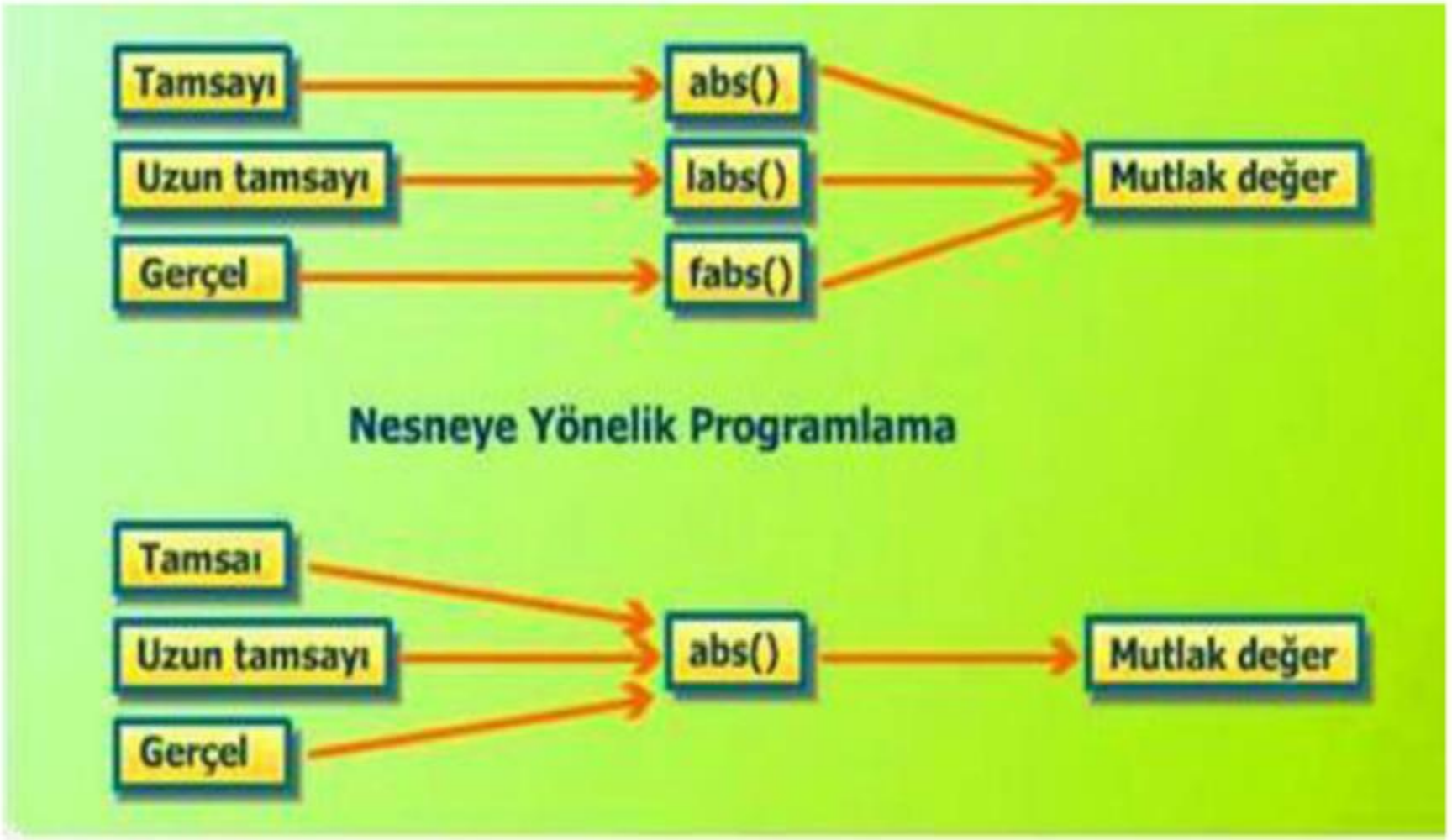
# ÇOK BİÇİMLİLİK (Polymorphism)

- Genel anlamı ile bir adın birbiriyle alakalı fakat teknik açıdan farklı iki veya daha fazla amaç için kullanılabilmesi yeteneğidir.
- NYP'de ise oluşturulan nesnelerin gerektiğinde başka bir nesne gibi davranabilmesine denir.
- Çok biçimlilikle programdaki her nesne kendi davranışını değiştirmeden, kalıtım hiyerarşisine göre farklı biçimlerde görülebilir.

# Bir Çok Biçimlilik Örneği

- Örneğin C dilinde, mutlak değer bulma işlemi için üç farklı fonksiyon tanımlıdır: abs(), labs() ve fabs().
- Fakat çok biçimliliği destekleyen C++'da bu fonksiyonlar, abs() gibi tek bir isimle adlandırılırlar.
- Fonksiyonu çağırmak için kullanılan veri tipi, gerçekte hangi fonksiyonun çalışacağını belirler.
- Böylece bir fonksiyon adının birkaç farklı amaç için kullanılması mümkündür.
- Buna fonksiyonların **aşırı yüklenmesi** denir.

# Bir Çok Biçimlilik Örneği



# Sanal Metotlar

Temel sınıf türünden bir nesneye türemiş sınıf referansı aktarılabilmekteydi. Bu aktarım sonrasında bazı metotların nesnelere göre seçilmesi istenebilir. Bu durumda sanal metotlar tanımlanır.

Sanal metotlar temel metotlar üzerinde bildirilmiş ve türeyen metotlar üzerinde tekrar bildirilen metotlardır.

İsim saklamaya benzemesine rağmen kullanımda farklıdır.

Sanal metotlar sayesinde temel sınıf türünden bir referansa türeyen sınıf referansı aktarıldığında, temel sınıf referansı üzerinden kendisine aktarılan türeyen sınıfın sanal metodunu çağırabilir.

Eğer türeyen sınıf sanal metodu devre dışı bırakmamışsa temel sınıfın sanal metodu çağrılır.

# Nesne Yönelimli Programlamanın Temel İlkeleri

```
using System;

class A
{
    public A()
    {
    }

    public virtual void Metot()
    {
        Console.WriteLine("A sanal metot...");
    }
}

class T : A
{
    public T()
    {
    }
}

class S : A
{
    public S()
    {
    }

    public override void Metot()
    {
         Console.WriteLine("S sanal metot...");
    }
}
class Program
{
    static void Main()
    {
        T t = new T();
        S s = new S();
        A a = new A();

        a = t;
        a.Metot();

        a = s;
        a.Metot();
```

# Çok Biçimlilik (Polymorphism)

Bu şekilde aynı nesne referansı üzerinden bir çok sınıfa ait farklı versiyonlardaki metotların çağrılabilmesi çok biçimlilik (polymorphism) olarak adlandırılır.

# Nesne Yönelimli Programlamanın Temel İlkeleri

Eğer metot sanal olarak bildirilmemişse, derleyici nesnelerin tür bilgisinden faydalanarak derleme zamanında hangi metodun çağrılacağını bilir.

Eğer metot sanal olarak bildirilmişse, derleyici derleme aşamasında ürettiği kod ile çalışma zamanında referansın türüne göre ilgili sınıfın devre dışı bırakılmış metodunu çağırır.

Hangi metodun çağrılacağının çalışma zamanında belirlenmesine geç bağlama (late binding) olarak isimlendirilir.

# Nesne Yönelimli Programlamanın Temel İlkeleri

Sanal metot bildirmek için virtual anahtar sözcüğü kullanılır.

Türeyen sınıfta, temel sınıftaki sanal metodu devre dışı bırakmak için override anahtar sözcüğü kullanılır.

Türeyen sınıfta devre dışı bırakılan metotların temel sınıftaki sanal metotların ismi ile aynı olmalıdır.

Türeyen sınıfta devre dışı bırakılan metotların parametrik yapısı temel sınıftaki metodun parametrik yapısı ile aynı olmalıdır.

Statik metotlar sanal olarak bildirilemez.

Türeyen sınıflar, temel sınıftaki sanal metotları devre dışı bırakmak zorunda değildir. Bu durumda temel sınıf referansları üzerinden temel sınıfa ait metot çağrılır.

# Özet(Abstract) Sınıflar

Nesne yönelimli programlamada sınıf hiyerarşisi oluşturulurken bazen hiyerarşinin en tepesinde bulunan sınıf türünden nesnelerin programcı için bir anlamı olmayabilir. Hiyerarşinin en tepesinde bulunan sınıfın kendisinden türetilecek diğer sınıflar için ortak özellikleri bir arada toplayan bir arayüz gibi davranması istenebilir. Bu tür sınıflara özet sınıflar adı verilir. Metotlar ve Özellikler de özet olarak tanımlanabilir.

# Nesne Yönelimli Programlamanın Temel İlkeleri

Özet sınıflar abstract anahtar sözcüğü ile bildirilirler. Özet sınıf türünden nesneler tanımlanamaz.

```
abstract class Sinif
{
}
```

Özet sınıflar tek başlarına anlamlı bir nesne ifade etmezler. Kullanılabilmeleri için mutlaka o sınıftan başka bir sınıf türetilmesi gerekmektedir.

Özet metotlar da yine abstract anahtar sözcüğüyle tanımlanır. Bu tür metotların gövdesi yoktur.

Türeyen sınıfta mutlaka devre dışı bırakılmalıdır. Özet sınıflarda yapı itibariyle sanal oldukları için ayrıca virtual kullanılmaz.

```
abstract public void Metot();
```

Özet sınıflar içinde özet olmayan metotlar bildirilebilir. Fakat tersi özet olmayan sınıflarda özet metotların tanımlanması söz konusu değildir.

Özet metotlar türeyen sınıfta devre dışı bırakılabilmeleri için private olarak tanımlanamazlar. public ya da protected olabilirler.

# Nesne Yönelimli Programlamanın Temel İlkeleri

Özellikler de özet olarak bildirilebilir.

- abstract public int X
  {
  get;
  set;
  }

Özet özellik bildiriminde kullanılan set veya get ifadelerinden hangileri kullanılmışsa türeyen sınıf bunları override ile mutlaka uygulamalıdır.

# Nesne Yönelimli Programlamanın Temel İlkeleri

```
using System;

abstract class A
{
    public int x;
    abstract public int y
    {
        set;
        get;
    }

    public A(int x)
    {
        this.x = x;
    }

    abstract public void Metot();
}

class S : A
{
    int z;

    public S(int x):base(x)
    {
    }
```

```
 public override int y
    {
        get
        {
            return z;
        }
        set
        {
            z = value;
        }
    }

    public override void Metot()
    {
        Console.WriteLine(x);
        Console.WriteLine(y);
    }
}

class Program
{
    static void Main()
    {
        S s = new S(5);
        s.y = 2 * s.x;
        s.Metot();
    }
}
```

# sealed Anahtar Sözcüğü

Bazı durumlarda sınıflardan türetme yapılması istenmeyebilir. Buna sağlamak için sınıf tanımlamasının başına sealed anahtar sözcüğü eklenir.

```
sealed class Sinif
{
}
```

Sınıflardan türetme yapılmaması türeyen sınıfın anlamsız olması ya da bazı üyelerin güvenliğini sağlamak olabilir. Tüm üyeleri statik olan sınıflar için de kullanılabilir.

# Arayüzler (Interfaces)

Özet sınıfların benzeri olan bir yapı da arayüzlerdir (interface), diğer sınıflar için ara yüz görevini üstlenir.

Bütün metotları ve özellikleri özet olarak bildirilmiş sınıflardan çok fazla bir farkı yoktur. Dolayısıyla arayüzlerdeki metot ve özelliklerin gövdesi yazılamaz.

Arayüzler kişisel uygulamalarda çok fazl kullanılmayabilir. Ancak bir grup tarafında geliştirilen projelerde ortak yapılar tanımlama için arayüzlerden faydalanılır.

# Nesne Yönelimli Programlamanın Temel İlkeleri

Arayüzler, interface anahtar sözcüğü ile bildirilirler. Bir arayüzde özellik, metot, indeksleyici (indexer), temsilci (delegate) ve olay (event) bildirimi yapılabilir. Arayüz tanımlamalarının zorunlu olmasa da başına "I" harfinin eklenmesi tanımlamanın arayüz olduğunun kolayca anlaşılmasını sağlar.

Arayüz tanımlamalarında dikkat edilecek bazı kısıtlamalar vardır:

Arayüz elemanları statik olamaz.

Arayüz elemanları public yapıdadır. Ayrıca erişim belirteci ile bildirilemez.

Üye değişken içeremezler.

Yapıcı ve yıkıcı metotlar tanımlanamaz ya da bildirilemez.

# Nesne Yönelimli Programlamanın Temel İlkeleri

```
using System;

interface IArayuz
{
    int Metot1();
    void Metot2();

    int Sayi
    {
        get;
        set;
    }

    int this[int indeks]
    {
        get;
    }
}
class Program
{
    static void Main()
    {
        Sinif s = new Sinif();
    }
}
```

```
class Sinif : IArayuz
{
    private int sayi;

    public int Metot1()
    {
        return 0;
    }

    public void Metot2()
    {
    }

    public int Sayi
    {
        get { return sayi; }
        set { sayi = value; }
    }

    public int this[int indeks]
    {
        get { return indeks; }
        set { }
    }
}
```

Sınıflar arasında çoklu türetme olmamasına rağmen arayüzler çoklu olarak uygulanabilir. Uygulanacak arayüzler virgül ile ayrılır:

```
class Sinif : IDisposable, IEnumerable
{
}
```

Arayüzü uygulayan sınıf arayüz dışında da elemanlara sahip olabilir. Yani istenildiği kadar üye eleman eklenebilir.

Sınıflarda olduğu gibi arayüzlerde birbirinden türetilebilir. Temel arayüzdeki tüm elemanlar türeyen arayüze aktarılır.

Arayüzler türetilirken new anahtar sözcüğü ile temel arayüzdeki elemanlar gizlenebilir. Bu şekilde aynı isimli yeni elemanlar tanımlanabilir.

Arayüzler ile referanslar oluşturulabilir. Arayüz referansları tek başına bir anlam ifade etmez fakat kendisini uygulayan bir sınıf nesnesinin referansı atanabilir. Bu durumda arayüz referansı ile arayüzde bulunan metot ya da özellikler hangi sınıf referansı tutuluyorsa oradan çağrılabilir.

# Nesne Yönelimli Programlamanın Temel İlkeleri

```
using System;

interface IArayuz
{
    int Metot1();
    void Metot2();
}

class Sinif : IArayuz
{
    int IArayuz.Metot1()
    {
        Console.WriteLine("Metot1");
        return 0;
    }

    public void Metot2()
    {
        Console.WriteLine("Metot2");
    }
}

class Program
{
    static void Main()
    {
        Sinif s = new Sinif();
        IArayuz a;

        a = s;
        a.Metot1();
    }
}
```