



Nesne Yönelimli Programlama

Genelleştirme (Generalization)

Öğr. Gör. İbrahim AYAZ



Genelleştirme Nedir ?

Genelleştirme, benzer nesne sınıflarını, daha genel bir sınıfta birleştirme mekanizmasıdır. Genelleme, bir dizi varlık arasındaki ortak noktaları tanımlar. Ortaklık; niteliklerde, davranışlarda veya her ikisinde birden olabilir. Başka bir deyişle üst sınıf, alt sınıflarla paylaşılacak en genel özelliklere, işlemlere ve ilişkilere sahiptir.

Genelleştirmeyi bir programlama dilinde uyguladığımızda buna genellikle Kalıtım (Inheritance) denir. Genelleme ve kalıtım aynıdır. Terminoloji, kullanıldığı bağlama bağlı olarak farklılık gösterir.



Gerçek dünya örneğinde müşteri bir kişidir. Aynı şekilde öğrenci de bir kişidir, çalışan da bir kişidir. Hepsinin bazı ortak noktaları var; örneğin hepsinin bir adı, ikinci adı ve soyadı var. Bunu nesne yönelimli programlamaya dönüştürmek için ad, soyad ve yaş gibi özelliklerine sahip Kisi sınıfı oluşturabilir ve Musteri, Ogrenci ve Calisan sınıflarını Kisi sınıfından miras/kalıtım alabiliriz. Bu şekilde tüm sınıflarda aynı özellikleri oluşturmamız gerekmeyecek ve kendini tekrar etme ilkesinin ihlalinin kaçınılmaz olacağız.

Kalıtım (Inheritance) Nedir ?

- ◎ Kalıtım, Nesneye Yönelik Programlamanın temel özelliklerinden biridir. **Kalıtım, bir sınıfın başka bir sınıfın özelliğini miras almasına izin veren mekanizmadır.** Bir sınıf başka bir sınıfı genişlettiğinde alanlar ve metotlar da dahil olmak üzere private olmayan tüm üyeleri miras alır. C#'da kalıtım, **base sınıf (parent)** ve **sub sınıf (child)** ilişkisiyle de anlaşılabilir.
- ◎ Kalıtım, bir **base (temel, üs) sınıf** ile onun **sub (alt) sınıfı** arasındaki ilişkiyi tanımlar. C# kalıtımı tanımlamak için **:** sembolü kullanılır.
- ◎ Kalıtım **is-a** ilişkisidir.



- ⦿ Aşağıdaki şekile göre A sınıfı, B sınıfından kalıtım almıştır. B sınıfı base (temel) sınıf, A sınıfı sub (alt) sınıf olarak tanımlanır. Bu durumda B sınıfındaki nitelikler ve yöntemler A sınıfına kalıtım olarak aktarılır.

```
1 namespace FirstProject.Inheritance;
  1 reference
2 public class B{
3     //...
4 }
5
6 0 references
7 public class A:B{
8     //...
9 }
```

Base class

Sub class



- 
- 
- ◎ C#'da private erişim belirtecine sahip bir alan (field), özellik (property) veya metot (method) kalıtım olarak alt sınıfa **aktarılamaz**.
 - ◎ C#'da kalıtım uygularken erişim belirteçleri ve soyutlamanın önemli bir konusu olan geçersiz kılma (overriding) kavramlarını göz önünde bulundurmanız gerekir.

Kalıtım Türleri

- ◎ Tekli kalıtım
- ◎ Çok seviyeli kalıtım
- ◎ Hiyerarşik kalıtım
- ◎ Çoklu kalıtım (C# ve Java dillerinde desteklenmiyor.)
- ◎ Hibrit kalıtım

Tekli Kalıtım

```
1 namespace FirstProject.Inheritance;
  1 reference
2 public class B{
3     //...
4 }
5
  0 references
6 public class A:B{
7     //...
8 }
```

Tekli kalıtımda, tek bir türetilmiş sınıf, tek bir temel sınıftan miras alır.

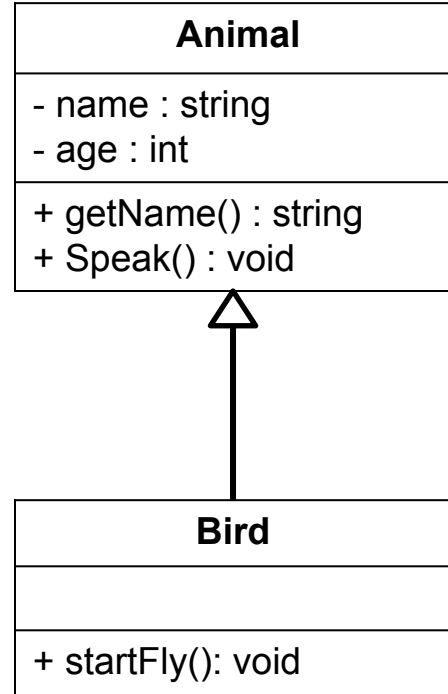


Tekli Kalıtım Sınıf Diyagramı ve C# kodu

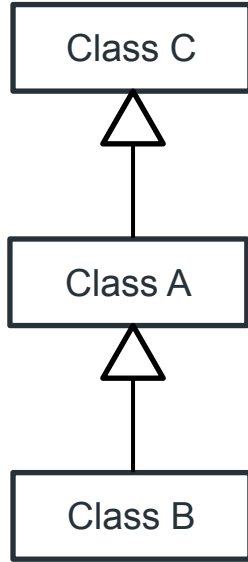
```
public class Animal
{
    private string name;
    private int age;

    0 başvuru
    public string getName()
    {
        return name;
    }
    0 başvuru
    public void Speak()
    {
        Console.Write("Miyav miyav..");
    }
}
```

```
0 başvuru
public class Bird:Animal
{
    0 başvuru
    public void startFly()
    {
        Console.WriteLine("Uçuyorum");
    }
}
```



Çok Seviyeli Kalıtım

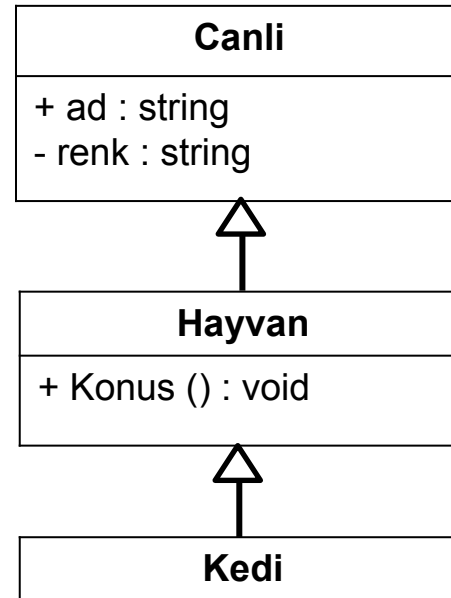


Çok seviyeli kalıtımda, türetilmiş bir sınıf bir temel sınıftan miras alır ve daha sonra aynı türetilmiş sınıf, başka bir sınıf için temel sınıf görevi görür.

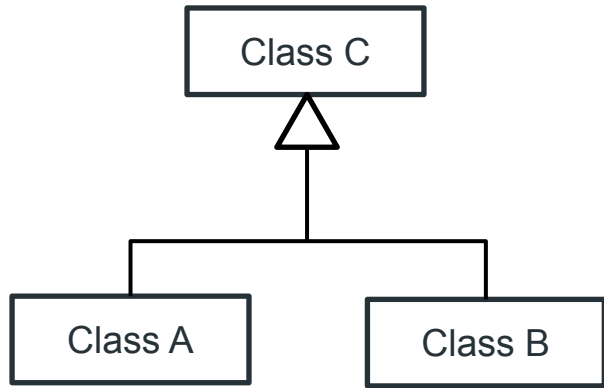
```
1 namespace FirstProject.Inheritance2;
2
3 1 reference
4 public class C{
5     //...
6 }
7 1 reference
8 public class A:C{
9     //...
10 }
11 0 references
12 public class B:A{
13     //...
14 }
```

Çok Seviyeli Kalıtım Sınıf Diyagramı ve C# Kodu

```
1 namespace FirstProject.Inheritance2;  
  1 reference  
2 public class Canli{  
    0 references  
3     public string ad;  
    0 references  
4     private string renk;  
5  
6 }  
  1 reference  
7 public class Hayvan:Canli{  
    0 references  
8     public void Konus(){  
9         Console.WriteLine("Hayvanlar Konusur");  
10    }  
11 }  
12  
13 public class Kedi:Hayvan{  
14 }
```



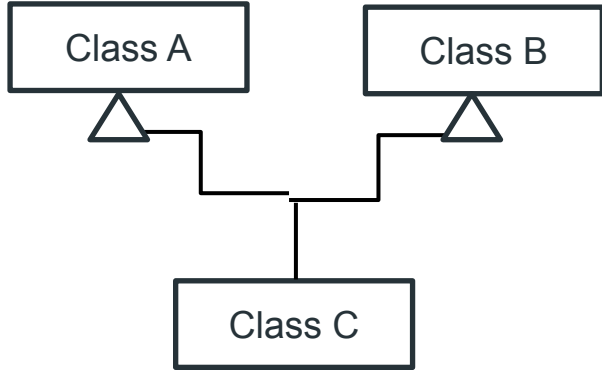
Hiyerarşik Kalıtım



Hiyerarşik kalıtımda, birden çok türetilmiş sınıf, tek bir temel sınıftan miras alır.

```
1 namespace FirstProject.Inheritance4;
2
3 2 references
4 public class C
5 {
6     //..
7 }
8 0 references
9 public class B : C
10 {
11     //..
12 }
13 0 references
14 public class A : C
15 {
16     //..
17 }
```

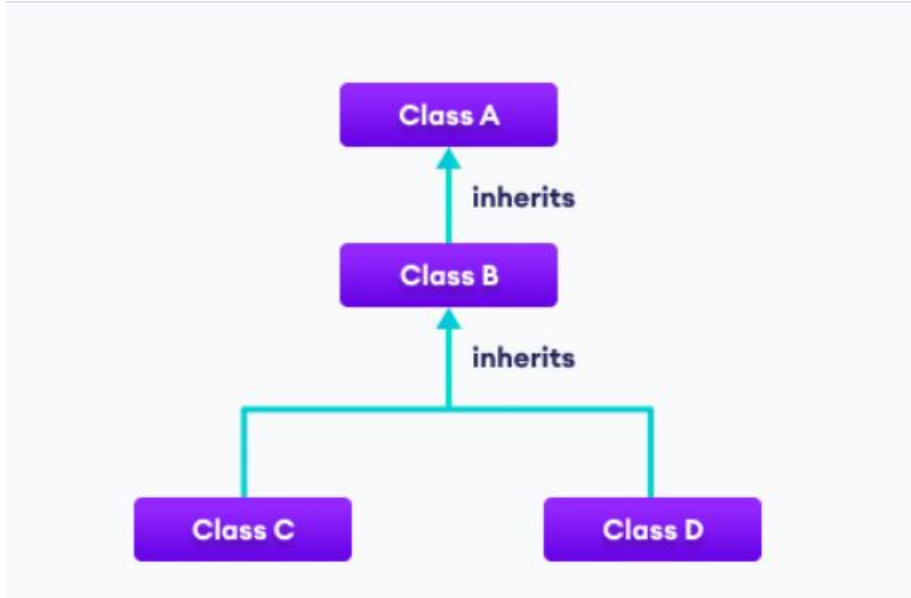
Çoklu Kalıtım



Çoklu kalıtımda, tek bir türetilmiş sınıf, birden fazla temel sınıftan miras alır. **C# çoklu kalıtımı desteklemez.** Ancak **arayüzler (interfaces)** aracılığıyla çoklu kalıtım elde edebiliriz.

```
1 namespace FirstProject.Inheritance4;
  1 reference
2 public class A
3 {
4     //..
5 }
  1 reference
6 public class B
7 {
8     //..
9 }
  0 references
10 public class C : A, B
11 {
12     //..
13 }
```

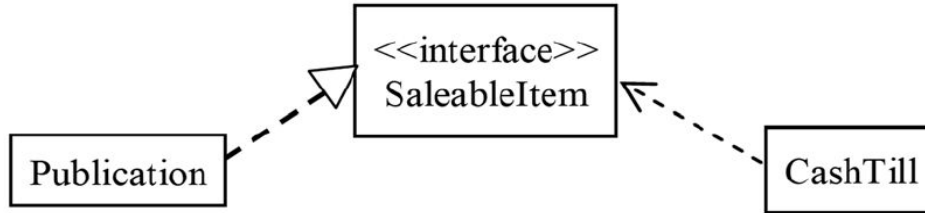
Hibrit Kalıtım



Hibrit kalıtım, iki veya daha fazla kalıtım türünün birleşimidir. Çok seviyeli ve hiyerarşik kalıtımın birleşimi Hibrit kalıtımın bir örneğidir.

Arayüz İlişkisi (Interface) (Realization(Gerçekleme))

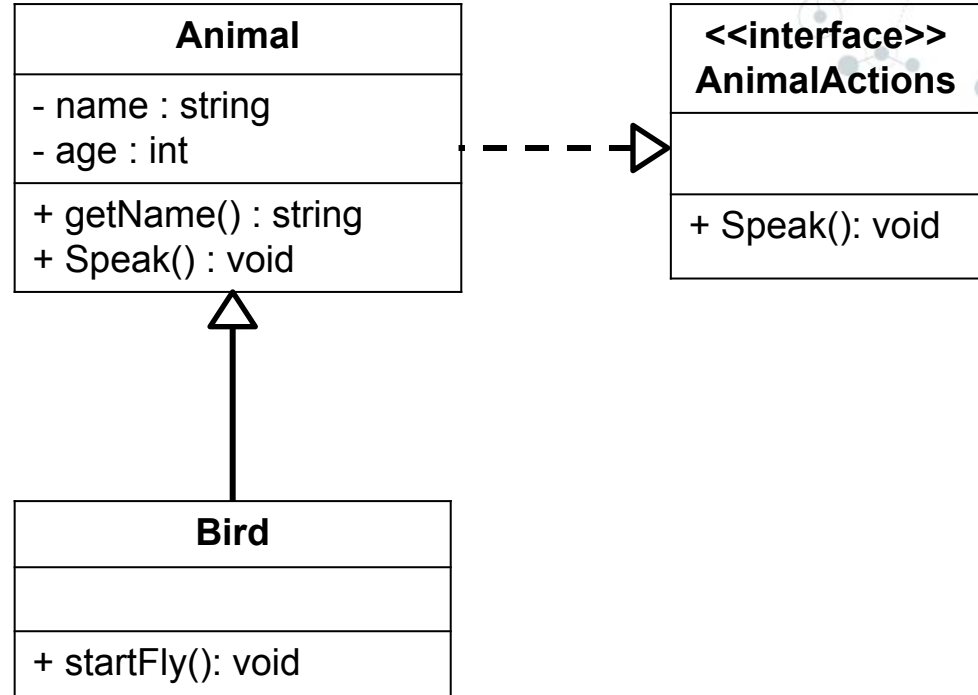
- Arayüzler kalıtıma benzer ancak arayüzlerde sadece arayüz kalıtılır. Arayüz tarafından tanımlanan yöntemler, arayüzü uygulayan her sınıfta uygulanmalıdır.
- Arayüzler **<<interface>>** anahtar sözcüğü kullanılarak gösterilebilir:



- Her iki durumda da bu örnekler, SaleableItem arayüzünün CashTill(Yazarkasa) tarafından gerekli olduğunu ve Publication tarafından uygulandığını gösterir.
Not: Publication, SaleableItem arayüzünü 'uyguladığını' veya 'gerçekleştirdiğini' gösteren **miras çizgisinin/okun** noktalı çizgi

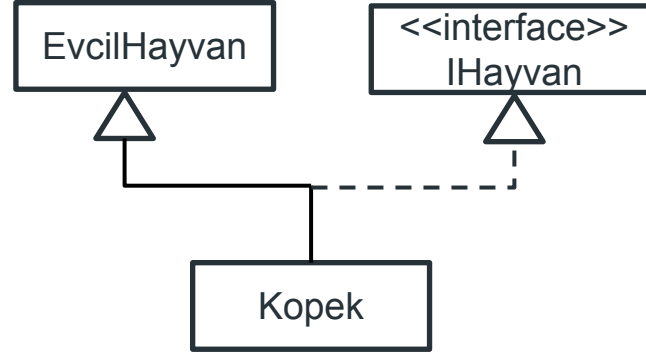
Interface ilişkisinin C# kodu üzerindeki gösterimi

```
C# ImplementsInterface.cs 1 •
C# ImplementsInterface.cs > {} FirstProject.Realization > Bird
1 namespace FirstProject.Realization;
2 public interface AnimalActions{
3     void Speak();
4 }
5 public class Animal : AnimalActions
6 {
7     private string name;
8     private int age;
9     public string getName()
10    {
11        return name;
12    }
13    public void Speak()
14    {
15        Console.WriteLine("Hayvan konuşuyor.");
16    }
17 }
18 public class Bird : Animal
19 {
20     public void startFly()
21     {
22         Console.WriteLine("Kuş uçuyor.");
23     }
24 }
```



Interface Kullanarak Çoklu Kalıtım Örneği

```
1 namespace FirstProject.Inheritance5;
  1 reference
2  public interface IHayvan{
  1 reference
3      void Konus();
  1 reference
4      void Dur();
5  }
  1 reference
6  public class EvcilHayvan{
  0 references
7      public string ad;
8  }
  0 references
9  public class Kopek : EvcilHayvan, IHayvan
10 {
11     1 reference
12     public void Dur()
13     {
14         Console.WriteLine("Köpek durdu");
15     }
16     1 reference
17     public void Konus()
18     {
19         Console.WriteLine("Köpek konuştu");
20     }
}
```



Çoklu kalıtımı C# desteklemez ancak bir sınıf birden fazla interfacesi kalıtım alabilir yani uygulayabilir.

UML de kullanılan bazı anahtar kelimeler

⊙ <<interface>>

⊙ <<abstract>>

⊙ <<struct>>

⊙ <<record>>

⊙ <<enum>>

⊙ <<get>>

⊙ <<set>>

Yukarıdaki anahtar kelimeler C# programlama diline özgüdür.

```
1 namespace FirstProject.Inheritance6;
  1 reference
2 public class Kisi
3 {
4     1 reference
    protected string Ad;
5 }
  0 references
6 public class Personel:Kisi{
7     1 reference
    private string Ad;
8     0 references
    public void Detail(){
9         base.Ad="İbrahim";
10        this.Ad="Ahmet";
11    }
12 }
```

C#'da alt sınıftan temel/üst (base) sınıfa erişmek için **base** anahtar kelimesi kullanılır.

Bir sınıfın kendi alan veya özelliklerini kullanması için **this** anahtar kelimesi kullanılır.



ÖDEV

Ürün, Satın Alma ve Müşteri ilişkilerini modelleyecek bir UML diyagramı çiziniz. Çizdiğiniz UML diyagramının C# kodunu yazınız.