Bölüm 17

Kalıtım (inheritance)

Kalıtım Kavramı

Mesaj İletme

Polimorfizm

Kalıtım kavramı nesne yönelimli programlamanın üç önemli kavramından birisidir. Öteki ikisi *polymorphism* ve *interface*'dir. Bu bölümde kalıtım kavramının ne olduğunu açıklayacağız. Bu kavramın çok önemli uygulamaları, bu kitabın kapsamı dışındadır.

Adından da anlaşılacağı üzere, en basit ifadesiyle kalıtım, atadan oğula bırakılandır. Bunu nesne yönelimli programlama için ifade edersek şöyle diyebiliriz: Kalıtım, bir sınıftan başka bir sınıfa miras bırakılandır. Bunu aşağıdaki basit örnek üzerinde açıklamak, algılanmasını kolaylaştıracaktır.

Kalıtım01.cs

```
using System;

public class AtaSınıf
{
    public AtaSınıf()
    {
        Console.WriteLine("Ata Kurucu.");
    }

    public void Yaz()
    {
        Console.WriteLine("Ben Ata sınıfındayım.");
    }
}

public class OğulSınıf : AtaSınıf
{
    public OğulSınıf()
```

```
    {
        Console.WriteLine("Oğul Kurucu.");
    }
}

class Uygulama
{
    public static void Main()
    {
        OğulSınıf oğul = new OğulSınıf();

        oğul.Yaz();
    }
}
```

Şimdi bu programı çözümleyelim. `AtaSınıf` içinde `public AtaSınıf()` kurucusu çağrılınca konsola "Ata Kurucu" metni yazılıyor. Ayrıca `AtaSınıf` içinde `Yaz()` metodu "Ben Ata sınıfındayım." metnini konsola yazıyor.

`OğulSınıf` bildirimi

```
public class OğulSınıf : AtaSınıf
```

sözdizimi ile veriliyor. Burada alışılmışın dışında

```
OğulSınıf : AtaSınıf
```

dizgesi var. Bu özel bir sözdizimidir (syntax) ve anlamı şudur: `OğulSınıf` sınıfının `AtaSınıf` sınıfından miras aldığını belirtir. `OğulSınıf : AtaSınıf` dizgesinde `OğulSınıf` miras alan, `AtaSınıf` ise miras bırakandır. İkisi arasına (:) konulması zorunludur. `OğulSınıf` içinde `public OğulSınıf()` kurucusu çağrılınca konsola "Oğul Kurucu" metni yazılıyor.

Şimdi bu iki sınıfı çağıran kodları çalıştıracak bir `Main()` metodu ve onu içerecek bir sınıf gerekiyor. `Uygulama` sınıfı bu işe yarıyor. `Uygulama` sınıfı içindeki `Main()` metodu `OğulSınıf` sınıfa ait `oğul` adlı bir nesne yaratıyor ve onun `Yaz()` metodunu çağırıyor. Oysa, `Yaz()` metodu `OğulSınıf` sınıfa değil, `AtaSınıf` 'a aittir. Ama `OğulSınıf` 'a miras kaldığı için, onu kendi öğesiymiş gibi kullanır. Böylece,

```
oğul.Yaz();
```

deyimi `AtaSınıf` içindeki `Yaz()` metodunu çağırır ve "Ben Ata sınıfındayım." metnini konsola yazdırır.

Miras bırakan sınıfa *taban sınıf* (base class), miras alan sınıfa *türemiş sınıf* (inherited class) denilir. Bunlara eşanlamlı olarak, taban sınıfa `ata`, türemiş sınıfa da `oğul` diyeceğiz.

Mesaj İletme

Uygulama sınıfından `oğul.Yaz()` metodunu çağırdık. Buna, nesne yönelimli programlamada bir sınıftan başka bir sınıfa *mesaj gönderme* denilir. Mesaj gönderme, aslında başka bir sınıftaki metodu çağırma; yani onu harekete geçirme eylemidir. Büyük bir programda, farklı eylemler farklı sınıflardaki metotlarla yapılır. Gerektiğinde, bir sınıftan başka bir sınıfın metodu çağrılıp, işlevini yapması istenebilir.

Aşağıdaki örnek, kullanıcıdan aldığı veriyi hem oğul (türemiş) sınıfındaki `apartmanYöneticisi`

alanına, hem de oğul sınıfı üzerinden ata sınıfındaki (taban sınıf) `evSahibi` alanına taşımaktadır.

Kalıtım02.cs

```
using System;

class Ev
{
    public string sahip;
    public Ev()
    {
        Console.WriteLine("Ev Kurucu.");
    }

    public void EvSahibiniYaz(string sahip)
    {
        Console.WriteLine("Ev 'in sahibi : {0}", sahip);

    }
}

class ApartmanKatı : Ev
{
    string apartmanYöneticisi;
    ApartmanKatı()
    {
        Console.WriteLine("ApartmanKatı Kurucu.");
    }
    void AptYöneticiniYaz(string s)
    {
        Console.WriteLine("Apartman Yöneticis : {0}", s);
    }

    class Uygulama
    {
        public static void Main()
        {
            ApartmanKatı apt = new ApartmanKatı();

            Console.WriteLine("Evin sahibi kim?  ");
            apt.sahip = Console.ReadLine();
            apt.EvSahibiniYaz(apt.sahip);

            Console.WriteLine("Apartman Yöneticisi kim?  ");
            apt.apartmanYöneticisi = Console.ReadLine();
            apt.AptYöneticiniYaz(apt.apartmanYöneticisi);
        }
    }
}
```

Burada `Ev` sınıfı, bütün evleri temsil eden genel bir sınıftır. `ApartmanKatı` ise `Ev` sınıfının bazı niteliklerini taşır, ama kendine özgü niteliklere de sahiptir. Örneğin, `Ev` sınıfında *salon, mutfak, oda sayısı, ısıtma sistemi, emlakVergisi, fiyatı* gibi değişken ve metotlar varsa, onlar `apartmanKatı` için de geçerli olur. Ama `apartmanKatında` her evde olması gerekmeyen bazı nitelikler var olabilir. Örneğin, *evin kaçıncı katta olduğu, apartman yöneticisi, kapıcı, kalorferci* gibi nitelikler.

Bu söylediklerimizin yukarıdaki `Ev` ve `apartmanKatı` sınıflarının gövdelerine yazabiliriz. Ancak, görünüş ne denli yalın olursa, konuyu kavramamız o kadar kolaylaşacaktır. O nedenle, yukarıdaki örnekte

ata sınıfa `evSahibi` adlı alan ile `EvSahibiniYaz()` metodunu koyduk. Benzer olarak oğula `apartmanYöneticisi` alanı ile `ApartmanYöneticisiniYaz()` metodunu koyduk. Alanlar ve metotlar, uygulamanın gerektirdiği kadar artırılabilir.

Şimddi yukarıdaki sözdizimini açıklayalım:

Kalıtım03.cs

```
class Ev
{
    public string evSahibi;
    public Ev()
    {
        Console.WriteLine("Ev Kurucu.");
    }

    public void EvSahibiniYaz (string evSahibi)
    {
        Console.WriteLine("Ev 'in sahibi : {0}", evSahibi);

    }
}
```

bildirimi `Ev` adlı ata sınıfın bildirimidir. Bunun gövdesindeki bütün öğeler `public` erişim belirteci ile nitelenmiştir. Oğuldan (alt sınıftan) bunlara erişilebilmesi için `public` nitelemesi gereklidir.

`public string evSahibi` bir veri alanıdır (field).

`public Ev() { }` deyimi Ev sınıfının kurucusudur.

`public void EvSahibiniYaz (string evSahibi) {}` deyimi bir metot bildirimidir.

Aynı düşünüşle,

Kalıtım04.cs

```
class ApartmanKatı : Ev
{
    string apartmanYöneticisi;
    public ApartmanKatı()
    {
        Console.WriteLine("ApartmanKatı Kurucu.");
    }
    public void AptYöneticiniYaz(string s)
    {
        Console.WriteLine("Apartman Yöneticis : {0}", s);
    }
}
```

bildirimi `apartmanKatı` adlı oğul sınıfının bildirimidir. Bu bildirimin başlığındaki

```
public class apartmanKatı : Ev
```

ifadesi, `apartmanKatı` sınıfının `Ev` sınıfından miras aldığını belirtir. `apartmanKatı : Ev` dizgesinde `apartmanKatı` miras alan (oğul), `Ev` ise miras bırakandır (ata). İkisi arasına (:) konulması zorunludur.

Oğulun gövdesindeki öğelere herhangi bir erişim belirteci nitelemesi konulmamıştır. Öyleyse hepsi `private` nitelemesine sahiptir. Böyle olduğu halde, erişimde bir sorun yaşanmaz. Çünkü, `Uygulama` sınıfındaki `Main()` metodu bu sınıfa ait `apt` adlı bir nesne yaratmaktadır. `Apt` nesnesi içinden `apt`

nesnesinin öğelerine kısıtsız erişmek mümkündür; bir erişim belirtecine gerek yoktur.

`string apartmanYöneticisi` oğul sınıfında bir veri alanıdır.

`public apartmanKatı() { }` deyimi oğul sınıfının kurucudur.

`public void` AptYöneticiniYaz(`string` s) oğul sınıfında bir metot bildirimidir.

Oğul sınıf, ata sınıfın `private` olmayan bütün öğelerine (alanlar ve metotlar) sahip olur. Ayrıca kendisine özgü öğeler olabilir. Oğul sınıftan ata sınıfa doğal dönüşüm (casting, implicit conversion) yapılabilir. Ters dönüşüm için apaçık (explicit) dönüşüm gerekir. Örneğin, yukarıdaki ata ve oğul için, şu dönüşümler geçerlidir.

```
Ev birEv = new Ev();                              ataya ait nesne yarat
ApartmanKatı birAptKatı = new ApartmanKatı();     oğula ait nesne yarat
birEv = birAptKatı;                               ataya atama
birAptKatı = (ApartmanKatı)birEv;                 oğula atama
```

Bu dönüşümlerde ata ve oğuldaki ortak öğelerde bir kayıp olmaz.

İpucu

Oğulsınıfa erişildiğinde ata sınıfa da erişilmiş olur; yani ata sınıfın private olmayan öğelerine oğul sınıf üzerinden erişilir.

Polimorfizm

Poly “çok” morph is “form”, “biçim” anlamlarını taşır. Bu ikisinin birleşimiyle oluşan “poymorphism” sözcüğü “çok biçimlilik” amlamına gelir.

Bunu şöyle bir örnekle açıklayalım. Çaldığında farklı marka telefonlar farklı sesler çıkarırlar. Alışılmış telefonlar bir zil çalar. Yeni kuşak telefonlar ve cep telefonları sayısız farklı sesler verebilirler. Telekom şirketi bu telefonların nasıl ses çıkardığını bilmez ve o seslerle ilgilenmez. Telekomun görevi, aranan numaraya gerekli sinyali ulaştırmaktır. Bu işi bütün telefonlar için aynı yöntemle yapar; karşıdaki telefonun nasıl ses verdiğine bakmaz. Bu işi yapan bir ata (taban, base) sınıfı olduğunu varsayalım. Bunun her telefonda farklı bir oğulu (inherited) sınıfı var. O sınıflarda ses özeliği hepsinde farklı farklı düzenlenebilir. Bu bir tür polimorfizmdir.

Şimdi bunu nesne yönelimli programlamada sınıflar için düşünelim. Bir atadan oğul yaratıldığında, bazan, miras geçen öğeler üzerinde değişiklikler yapılması gerekir. Örneğin, bir alan farklı veri tutsun veya bir adı ve değer kümesi korunan bir metot farklı iş yapsın istenebilir. Bunun için, öğenin new ile nitelenmesi yeterlidir. Aşağıdaki örnek bunun nasıl yapılacağını göstermektedir.

Kalıtım05.cs

```
public class Ata
{
    public void BirMetot() { }
    public int birAlan;
    public int birÖzgen
    {
        get { return 0; }
    }
```

```
}

public class Oğul : Ata
{
    public new void BirMetot() { }
    public new int birAlan;
    public new int birÖzgen
    {
        get { return 0; }
    }
}
```

Aşağıdaki örnek konuyu biraz daha açmaktadır.

Kalıtım06.cs

```
using System;

public class Çizici
{
    public virtual void Çiz()
    {
        Console.WriteLine("Çizici");
    }
}

public class DoğruÇiz : Çizici
{
    public override void Çiz()
    {
        Console.WriteLine("Line.");
    }
}

public class DaireÇiz : Çizici
{
    public override void Çiz()
    {
        Console.WriteLine("Circle");
    }
}

public class KareÇiz : Çizici
{
    public override void Çiz()
    {
        Console.WriteLine("Kare");
    }
}


public class Uygulama
{
    public static int Main()
    {
        Çizici[] birÇizici = new Çizici[4];

        birÇizici[0] = new DoğruÇiz();
```

```
        birÇizici[1] = new DaireÇiz();
        birÇizici[2] = new KareÇiz();
        birÇizici[3] = new Çizici();

        foreach (Çizici sayaç in birÇizici)
        {
            sayaç.Çiz();
        }

        return 0;
    }
}
```

Alıştırmalar

Aşağıdaki programın deyimlrini çıktıları ile karşılaştırarak programı çözümleyiniz.

Kalıtım07.cs

```
using System;
class Anne
{
    public Anne()
    {
        Console.WriteLine("Anne kurucu");
    }
    public void Selam()
    {
        Console.WriteLine("Anne size 'Merhaba' dedi");
    }
    public void Konuş()
    {
        Console.WriteLine("Anne konuşuyor");
    }
    public virtual void Şarkı()
    {
        Console.WriteLine("Anne şarkı söylüyor");
    }
};

class Çocuk : Anne
{
    public Çocuk()
    {
        Console.WriteLine("Çocuk kurucu");
    }
    public new void Konuş()
    {
        Console.WriteLine("Çocuk konuşuyor");
    }
    public override void Şarkı()
    {
        Console.WriteLine("Çocuk şarkı söylüyor");
    }
};
```

```
public class Uygulama
{
    public static void Main()
    {
        Anne a1 = new Anne();
        a1.Konuş();
        a1.Şarkı();
        a1.Selam();
    }
}
```

Çıktı

```
Anne kurucu
Anne konuşuyor
Anne şarkı söylüyor
Anne size 'Merhaba' dedi
```