Nesne Yönelimli Programlama ve Kalıtım

Nesne Yönelimli Programlama

- Modern dillerin bir çoğunda nesneye yönelimli programlama tekniği desteklenmektedir. Bu teknik yazılım geliştirmeyi kısaltan ve sistematik hale getiren bir yapıdır.

- C# dili de bu tekniği tamamıyla desteklemektedir.

- Nesne yönelim tekniği, gerçek hayatı programlar için simule edecek yöntemlerin birleşimidir.

- Bu teknikte geliştirilmek istenen sistem parçalara ayrılır ve bu parçalar arasında ilişkiler kurulur. Parçalar hiyerarşik ya da bağımsız olabilir.

- Bağımsız bileşenler birbirleriyle haberleşerek etkileşimde bulunurlar.

Nesne yönelimli programlama kavramlarından bazıları şunlardır:

- 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 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.

- 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.

- Tüm nesneler birer sınıf örneğidir. Sınıflar nesnelerin özelliklerini belirlerler. Nesneler derleme ya da çalışma anında oluşturulabilir.

- Kalıtım ile nesneler birbirinden türetilebilir. Türeyen sınıf diğer sınıfın tüm özelliklerini ve kendine has özellikleri içerebilir. Kalıtım yolu ile türetilmiş sınıflar ile hiyerarşik sınıf organizasyonu gerçekleştirilebilir.

- Nesneye yönelimli programlama tekniğinde nesneler çok biçimli olabilir. Çok biçimlilik (polymorphism) kavramı türeme ile alakalıdır ve anlamı bir nesnenin farklı şekillerde davranabilmesidir.

Nesne Kavramı

- Gerçek dünyadaki varlığını bildiğimiz bir çok şey birer nesnedir. Nesne yönelimli programlama tekniğinde de sınıflar nesnelerin biçimlerini belirlerler. Oluşturulan nesneler sınıf türünden nesne olarak adlandırılır. Her nesne kendi içinde tutarlı bir yapıya sahiptir yani veriler arasında sıkı bir bağ bulunur ki bu nesne mantığının temelidir.

- Sınıflardan nesneler oluşturmak için new anahtar sözcüğü kullanılıyordu.
  - Sinif nesne = new Sinif();

- Sınıflar nesnelerin şeklini belirlerler. Yani nesnenin türünü tanımlarlar. Kısaca sınıflar bir tür bilgisidir.

Kalıtım(Inheritence)

- Kalıtım nesne yönelimli programlama tekniğinin en önemli özelliğidir. Kalıtım yolu ile sınıflar birbirinden türetilir. Türeyen sınıflar türedikleri sınıfın özelliklerini kalıtım yoluyla devralırlar ve kendisi de yeni özellikler tanımlayabilir. Türetme ile sınıflar arasında hiyerarşik bir yapı kurulabilir.

- Örnek vermek gerekirse dünya üzerinde yaşayan canlıları sınıflandırmak mümkün. Bu sınıflardan bir tanesi de hayvanlar olabilir. Kedi, Köpek, Kuş, Balık gibi bir çok hayvan türünden bahsedilebilir. Her türün kendine ait değişik özellikleri olabilir. Dolayısıyla her biri için değişik sınıfların tasarlanması gerekebilir.

- Fakat ortak bir takım özelliklerinin olması da kaçınılmazdır ve her biri için bağımsız sınıflar tasarlandığında bu benzerlikler her birisi için tekrarlanmak durumunda kalacaktır.

- Bu yüzden önce tüm hayvanlar için bir sınıf oluşturulup diğer kedi, köpek gibi sınıflar bu sınıfın devamı gibi tasarlanabilir. İşte temelde bir sınıf tanımlanıp diğer sınıfları bu sınıftan türeterek özelleştirmeye kalıtım yoluyla türetme adı verilir.

Örn:

```
class Program
    {
        static void Main(string[] args)
        {
            model1 oto1 = new model1();
            model2 oto2 = new model2();
            oto1.tur = "Sedan";
            oto1.silindir_sayisi = 4;
            oto1.subap_sayisi = 8;
            oto1.guc = 75;
            oto1.tork = 100;
            oto1.ozellikyaz();
            Console.WriteLine("********");
            oto2.model2_boy = 6;
            oto2.model2_agirlik = 900;
            oto2.model2_renk = "Kırmızı";
            oto2.ozellikyaz();
            Console.WriteLine("********");
            oto2.goster();
            Console.WriteLine("********");
            oto1.goster();
            Console.ReadLine();

        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication9
{
    class oto
    {
        protected double boy=5;
        protected double agirlik=800;
        protected string renk="Sarı";
        public void goster()
        {
            Console.WriteLine("Boy=" + boy);
            Console.WriteLine("Ağırlık=" + agirlik);
            Console.WriteLine("Renk=" + renk);
        }
    }
    class model1:oto
    {
        public string tur;
        public int silindir_sayisi;
        public int subap_sayisi;
        public int tork  ;
        public int guc ;
        public void ozellikyaz()
        {
  Console.WriteLine("Tür=" + tur);
  Console.WriteLine("Boy="+boy);
  Console.WriteLine("Ağırlık=" + agirlik);
  Console.WriteLine("Renk=" + renk);
  Console.WriteLine("Silindir Sayisi=" + silindir_sayisi);
  Console.WriteLine("Subap Sayısı=" + subap_sayisi);
  Console.WriteLine("Tork=" + tork);
  Console.WriteLine("Güç=" + guc);
        }
    }
```

```
class model2 : oto
    {
        public double model2_boy
        {
            get { return boy; }
            set { boy = value; }
        }
        public double model2_agirlik
        {
            get { return agirlik; }
            set { agirlik = value; }
        }
        public string model2_renk
        {
            get { return renk; }
            set { renk = value; }
        }
        public string tur="Hatchback";
        public int silindir_sayisi=8;
        public int subap_sayisi=16;
        public int tork=300;
        public int guc=210;
        public void ozellikyaz()
        {
          Console.WriteLine("Tür=" + tur);
          Console.WriteLine("Boy=" + model2_boy);
          Console.WriteLine("Ağırlık=" + model2_agirlik);
          Console.WriteLine("Renk=" + model2_renk);
          Console.WriteLine("Silindir Sayisi=" + silindir_sayisi);
          Console.WriteLine("Subap Sayısı=" + subap_sayisi);
          Console.WriteLine("Tork=" + tork);
          Console.WriteLine("Güç=" + guc);
        }
    }
```

Türetme

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.

```
using System;

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();
    }
}
```

- 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)

```
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.

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.

```
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();
    }
```

- 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.

Örn:

```
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);
        }
    }
```

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.

```
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.

Çok çeşitlilik ile ilgili olarak:

- 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.

- 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.

- 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.

- 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.

```
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 kısaca kendisini uygulayan sınıfların kesin olarak içereceği özellikleri be metotları belirler.

- Arayüzler kişisel uygulamalarda çok fazla kullanılmayabilir. Ancak bir grup tarafından geliştirilen projelerde ortak yapılar tanımlamak için arayüzlerden faydalanılır.

- .NET sınıf kitaplığı içinde de tanımlı bir çok arayüz bulunur. IDisposable arayüzü GC için Dispose() metodunun uygulanmasını sağlar. System.Collections isim alanında bulunan IEnumarable arayüzü tanımlanan sınıfların foreach döngü yapısı ile kullanılabilmesini sağlar.

- 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.

```
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.

```
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();
    }
}
```

Partial (Kısmi) Tipler

- Şu ana kadar tanımlanan tip bildirimleri tek bir proje dosyası içersinde bulunmaktaydı. Oysaki bir projede birden çok dosya bulunabilir.

- Sınıf, arayüz ve yapı tanımlamaları partial anahtar sözcüğü ile birden fazla dosyaya dağıtılabilir. Bütün dosyadaki tanımlamalar tek bir bildirimi göstermektedir. (isim alanları tanımlarken de benzer bir durum söz konusuydu.)

- **Ozellikler.cs**

- ```
  partial class Sinif
  {
      public int x;
      public int y;
  }
  ```

- **Metotlar.cs**

- ```
  partial class Sinif
  {
      public int Topla()
      {
      }

      public void EkranaYaz()
      {
      }
  }
  ```