Bölüm 10

Statik ve Dinamik Öğeler

Statik ve Dinamik Öğeler

Bir sınıfta static sıfatıyla nitelendirilen öğelerdir. Örneğin,

```
static int x;
static double ÜcretHesapla(){ ... };
```

deyimleri, sırasıyla, `int` tipinden `static x` değişkeni ile `double` değer alan `static ÜcretHesapla()` metodunun bildirimidir.

Bir sınıfın `static` öğeleri, o sınıfa ait bir nesne yaratılmadan erişilebilen öğelerdir. Sınıfın herhangi bir nesnesiyle bağlantısı olsun istenmeyen öğelere `static` nitelemesini vermek ona her yerden erişilebilen bir globallik niteliği verir. Daha önce, C# dilinde *global değişken* ve *global metot* olmadığını söylemiş, public nitelemesiyle öğelere bir tür global işlevsellik kazandırılabildiğini belirtmiştik. Öğelere globallik kazandırmanın ikinci etkin bir yolu `static` nitelemesidir. Statik öğeler, sınıfın bir nesnesi içinde olmadığından, onlara programın her yerinden erişilebilir. Örneğin, `Main()` metodu daima `static` nitelemesini alır. O nedenle, ait olduğu sınıfın bir nesnesi yaratılmadan doğrudan çalışabilir. Sınıfa ait bir ya da daha çok nesne yaratıldığında, bu nesneler içinde yapılan işler static öğeleri etkilemez. Bunun nedeni, statik bir öğeye, o sınıfa ait bir nesne yaratılmadan ana bellekte bir ve yalnız bir tane bellek adresi ayrılmasıdır. Nesnelerin yaratılıp yokedilmesi, ya da nesneler içindeki değişimler onları etkilemez. Dolayısıyla, örneğin, `static` bir değişkenin belli bir anda yalnız bir tane değeri olabilir. Daha önemlisi, ana bellekte nesneye bağlı olmadan birer adresleri olduğu için, static öğelere nesne yaratılmadan erişilebilir.

Statik olmayan öğelere *dinamik* öğeler diyeceğiz. Öndeğer (default) olarak, bildirim sırasında `static` nitelemesini almayan her öğe *dinamiktir*. Örneğin,

```
long y;
float Topla(a,b){ return a+b};
```

deyimleri, sırasıyla, `long` tipinden `dinamik x` değişkeni ile `float` değer alan dinamik `Topla()` metodunun bildirimidir.

Bir sınıfın dinamik öğelerine, o sınıfın yaratılan her bir nesnesi için, ana bellekte birer adres ayrılır. Başka bir

deyişle, bir sınıfın üç ayrı nesnesi yaratılmışsa, sınıftaki dinamik bir öğeye her nesnede bir tane olmak üzere üç ayrı bellek adresi ayrılır. Dolayısıyla, dinamik değişkenin yaratılan her nesnede ayrı bir değeri vardır. Ama, onların var olabilmeleri için, önce sınıfa ait nesnenin yaratılması gerekir.

Neden Nesne Yönelimli Programlama?

Bu soruya yanıt vermeden önce aşağıdaki bir dizi programı inceleyeceğiz. İncelemeyi bitirince, sorunun yanıtı kendiliğinden ortaya çıkacaktır. Örnek programların hepsi aynı işi yapmaktadırlar: Konsoldan brüt geliri okur, hesapladığı gelir vergisini konsola yazar.

İlk program, sözkonusu işi yapan en basit, en kısa ve en kolay programdır. Ama, yapısal programlama açısından asla tercih edilemeyecek kadar kötü bir programdır.

Program01.cs

```
using System;

namespace ErişimKısıtları
{
    class Program01
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Brüt Geliri giriniz :");
            Console.WriteLine(Decimal.Parse(Console.ReadLine())*40/100);
        }
    }
}
```

Çıktı

```
Brüt Geliri giriniz :
123456,789
49382,7156
```

Şimdi programımıza konsoldan veri okuyan, hesap yapan ve konsola veri yazan `VeriOku()`, `Hesapla()` ve `VeriYaz()` adlı üç ayrı metot ekleyelim. Ama öyle yapalım ki, `Main()` metodu `VeriYaz()` metodunu çağırsın. `VeriYaz()` metodu `Hesapla()` metodunu çağırsın. `Hesapla()` metodu `VeriOku()` metodunu çağırsın. Böylece arka arkaya dört metot çalışacaktır. Bir metodun başka bir metodu çağırması, matematikteki bileşik fonksiyon (fonksiyon fonksiyonu) kavramından başka bir şey değildir. İçteki fonksiyon, dıştaki fonksiyonun değişkenidir.

Program02.cs

```
using System;

namespace ErişimKısıtları
{
    class Program02
    {
        decimal VeriOku()
        {
            Console.WriteLine("Brüt Geliri giriniz :");
```

```
            return Decimal.Parse(Console.ReadLine());
        }

        decimal Hesapla()
        {
            return VeriOku() * 40 / 100;
        }

        void VeriYaz()
        {

            Console.WriteLine(Hesapla());
        }

        static void Main(string[] args)
        {
            VeriYaz();
        }
    }
}
```

Program derlenince aşağıdaki hata iletisini gönderir.

```
Error   1       An object reference is required for the non-static field, method, or property ...
```

Bu ileti bize, fonksiyonlara erişilemediğini, onların hangi nesnede olduklarını işaret eden referansların (işaretçi) verilmesi gerektiğini söylemektedir. Bu mesajı alınca şu iki yoldan birisini izleyebiliriz.

1. Metotlarımızı `static` yaparız.

2. Metotlarımızı içeren sınıftan bir nesne yaratırız (instantiate).

Her iki yöntemi ayrı ayrı deneyelim. Birincisi kolaydır. Metot bidirimlerinde her üçünü `static` sıfatı ile niteleyelim:

Program03.cs

```
using System;

namespace ErişimKısıtları
{
    class Program03
    {
        static decimal VeriOku()
        {
            Console.WriteLine("Brüt Geliri giriniz :");
            return Decimal.Parse(Console.ReadLine());
        }

        static decimal Hesapla()
        {
            return VeriOku() * 40 / 100;
        }

        static void VeriYaz()
```

```
        {

            Console.WriteLine(Hesapla());
        }

        static void Main(string[] args)
        {
            VeriYaz();
        }
    }
}
```

Çıktı

```
Brüt Geliri giriniz :
123456,789
49382,7156
```

Şimdi de ikinci yöntemi deneyelim. `Program03` sınıfından `p` adlı bir nesne yaratalım. Bunu yapmak için

```
Sınıf_adı nesne_adı = new Sınıf_adı();
```

nesne yaratıcı (instantiate) deyimini kullanıyoruz. Örneğimizde, nesne_adı p olacaktır.

Program04.cs

```
using System;

namespace ErişimKısıtları
{
    class Program04
    {
        decimal VeriOku()
        {
            Console.WriteLine("Brüt Geliri giriniz :");
            return Decimal.Parse(Console.ReadLine());
        }

        decimal Hesapla()
        {
            return VeriOku() * 40 / 100;
        }

        void VeriYaz()
        {

            Console.WriteLine(Hesapla());
        }

        static void Main(string[] args)
        {
            Program04 p = new Program04();
            p.VeriYaz();
        }
    }
}
```

Çıktı

```
Brüt Geliri giriniz :
123456,789
49382,7156
```

İpucu

Buradaki `VeriOku()`, `Hesapla()`, `VeriYaz()` metotlarının önüne `static` nitelemesi konulmadığı için, doğal olarak (default) *dinamik* olurlar; yani static değildirler. Dolayısıyla, ancak sınıfa ait bir nesne yaratılınca o nesne içinde ana bellekte kendilerine birer yer ayrılır. Üstlendikleri işlevlerini o nesne içinde yaparlar. Böyle olduğunu görmek için, `p` ve `q` adlarıyla iki farklı nesne yaratalım ve o nesneleredeki metotlarımızı farklı verilerle çalıştıralım. `p` ve `q` nesnelerinden gelen çıktıların farklı olduğunu görebiliriz.

Program05.cs

```
using System;

namespace ErişimKısıtları
{
    class Program05
    {
        decimal VeriOku()
        {
            Console.WriteLine("Brüt Geliri giriniz :");
            return Decimal.Parse(Console.ReadLine());
        }

        decimal Hesapla()
        {
            return VeriOku() * 40 / 100;
        }

        void VeriYaz()
        {

            Console.WriteLine(Hesapla());
        }

        static void Main(string[] args)
        {
            Program05 p = new Program05();
            Program05 q = new Program05();

            p.VeriYaz();
            q.VeriYaz();
        }
    }
}
```

Çıktı

```
Brüt Geliri giriniz :
123456,789
49382,7156
Brüt Geliri giriniz :
2000
800
```

Şimdi başka bir durumu deneyelim. Sınıfta bildirimi yapılan metotların hepsini satic yapalım ve sınıftan bir nesne yaratmayı deneyelim.

Program06.cs

```
using System;

namespace ErişimKısıtları
{
    class Program06
    {
        static decimal VeriOku()
        {
            Console.WriteLine("Brüt Geliri giriniz :");
            return Decimal.Parse(Console.ReadLine());
        }

        static decimal Hesapla()
        {
            return VeriOku() * 40 / 100;
        }

        static void VeriYaz()
        {

            Console.WriteLine(Hesapla());
        }

        static void Main(string[] args)
        {
            Program06 p = new Program06();
            p.VeriYaz();

        }
    }
}
```

Bu programı derlemek istersek şu hata mesajını verecektir:

```
Error 1 Member 'ErişimKısıtları.Program06.VeriYaz()' cannot be accessed with an instance reference; qualify it with a type name instead...
```

Mesaj bize şunu söylüyor. p referanslı (p nin işaret ettiği) bir nesne yaratılmıştır ve p işaretçisi yaratılan bu nesnenin adresini işaret etmektedir. Sınıfın static öğelerinin bellek adreslerini göstermez. Dolayısıyla, p referansı ile static `VeriYaz()` metoduna erişilemez. Çünkü statik öğelere ana bellekte ayrılan yer p nin işaret ettiği nesne içinde değildir. p işaretçisi ancak yaratılan nesne içindeki dinamik öğeleri gösterebilir.

İpucu

Şimdi sırayla aşağıdaki durumları ayrı ayrı deneyelim:

Metotların üçü de dinamik ise, program derlenir ve çalışır.

Metotların üçü de static ise, en sondaki `p.VeriYaz()` deyimi yerine `VeriYaz()` ya da `Program06.VeriYaz()` yazılırsa program derlenir ve çalışır. Bu durumda yaratılan nesne işleme girmez.

`VeriYaz()` ve `Hesapla()` metotları dinamik, `VeriOku()` metodu statik ise program derlenir ve çalışır.

`VeriYaz()` dinamik, `Hesapla()` static, `VeriOku()` dinamik ise program derlenemez.

`VeriYaz()` metodu static yapılınca, öteki ikisi ne olursa olsun, program derlenemez.

Şaşırtıcı görünen bu durumun basit bir nedeni vardır. O nedeni artık biliyoruz. Dinamik öğeye referans ile erişilir. VeriYaz() metodu dinamik yapılınca, p referansı onun yerini işaret eder. VeriYaz() metoduna erişilince, artık kontrol p nin elinden çıkar, VeriYaz() metoduna geçer. VeriYaz() metodu Hesapla() metodunu çağırıyor. Eğer, `Hesapla()` dinamik ise, `VeriYaz()` metodunun bulunduğu nesne içindedir ve oraya erişebilir. Eğer `Hesapla()` statik ise, zaten ona erişmek mümkündür. Hesapla `VeriOku()` metodunu çağırıyor. Bu metot statik ise, yeri bellidir ve ona erişir. Ama VeriOku() dinamik ise, onun bulunduğu nesneyi Hesapla() metodu göremez.

Çıktı

```
Brüt Geliri giriniz :
123456,789
49382,7156
Brüt Geliri giriniz :
2000
800
```

Erişimkısıtları.cs

```
using System;

namespace ErişimKısıtları
{
    class Program01
    {
        static void Main(string[] args)
        {
            Vergiler.VergiYaz();
        }
    }
}

class Vergiler
```

```
{
    static decimal brütGelir;
    const byte vergiYüzdesi = 40;

    public static decimal GelirVergisiHesapla(decimal x)
    {
        decimal y;
        brütGelir = x;
        y = x * vergiYüzdesi/100;
        return y;
    }

    public static void VergiYaz()
    {
        string s;
        Console.WriteLine("Brüt Geliri giriniz :");
        s = Console.ReadLine();
        Vergiler.brütGelir = Decimal.Parse(s);
        Console.WriteLine(GelirVergisiHesapla(Vergiler.brütGelir));
    }
}
```

Çıktı

```
Brüt Geliri giriniz :
3425,467
3425,467 nin vergisi 1370,1868 YTL dir.
```

Program02.cs

```
using System;

namespace ErişimKısıtları
{
    class Program01
    {
        static void Main(string[] args)
        {
            Vergiler.VergiYaz();
        }
    }
}


class Vergiler
{
    static decimal brütGelir;
    const byte vergiYüzdesi = 40;

    public static decimal GelirVergisiHesapla(decimal x)
    {
        decimal y;
        brütGelir = x;
        y = x * vergiYüzdesi/100;
```

```
        return y;
    }

    public static void VergiYaz()
    {
        string s;
        decimal vergi;
        Console.WriteLine("Brüt Geliri giriniz :");
        s = Console.ReadLine();
        Vergiler.brütGelir = Decimal.Parse(s);
        vergi = GelirVergisiHesapla(Vergiler.brütGelir);
        Console.WriteLine("{0} nin vergisi {1} YTL dir." , brütGelir,
vergi);
    }

}
```

Çıktı

```
Brüt Geliri giriniz :
3000
3000 nin vergisi 1200 YTL dir.
```

Program03.cs

```
using System;

namespace ErişimKısıtları
{
    class Program01
    {
        static void Main(string[] args)
        {
            Hesap.VergiYaz();
        }
    }
}


class Vergiler
{
    public static decimal brütGelir;
    const byte vergiYüzdesi = 40;

    public static decimal GelirVergisiHesapla(decimal x)
    {
        decimal y;
        brütGelir = x;
        y = x * vergiYüzdesi/100;
        return y;
    }
}

class Hesap
```

```
{
    static decimal vergi;
    public static void VergiYaz()
    {
        string s;

        Console.WriteLine("Brüt Geliri giriniz :");
        s = Console.ReadLine();
        Vergiler.brütGelir = Decimal.Parse(s);
        vergi = Vergiler.GelirVergisiHesapla(Vergiler.brütGelir);
        Console.WriteLine("{0} nin vergisi {1} YTL dir." ,
Vergiler.brütGelir, vergi);
    }
}
```

Çıktı

```
Brüt Geliri giriniz :
3000
3000 nin vergisi 1200 YTL dir.
```