3.9 Statik Üyeler

İçinde bulunduğu sınıftan nesne oluşturulmadan ya da hiç bir nesneye referans olmadan kullanılabilen üyeler `static` olarak nitelendirilir.

Nesne yaratmadan bir sınıfın değişkeninin veya metodunun kullanılması için o değişkenin ve metodun `static` olması gerekir.

3.9.1 Hazır Static Metodlar

C# içinde `new` ile üretilmeden direkt olarak kullanılan üyeler statictir. Örnek:

`Math` kütüphanesi içindeki `Sqrt` metotu direkt kullanılır

```
Double sayi = Math.Sqrt(55);
```

Main metodu da static tenımlanır. Değilse main'i kim oluşturacaktı? Bir NYP uygulamasında en az bir üye static olmalıdır ve bu üye ilk çalışan bölüm olmalıdır.

Dinamik sınıflar ihtiyaç durumunda belleğe yüklenir (new). Eğer bir sınıfı veya sınıf içindeki üyeyi çok sık kullanıyorsak static olarak tanımlayabiliriz. Ama static üyelerin yazılım çalıştığı sürece bellekte yer kapladığını unutmayın. Dinamik sınıflar ise GC tarafından kullanılmıyorsa yok edilirler.

Eğer bir sınıftan çok fazla sayıda nesneye ihtiyaç varsa stativ değil dinamic tanımlanır.

Örnek Static Kullanımı:

```
using System;

namespace yEmre
{
    class oyuncu
    {
        private string ad;
        private string mevki;
        private int savunma;
        private int atak;
        private int topSurme;
        public static int maxYetenekPuani=100;

        public oyuncu(string adi, string mevkisi)
        {
            ad = adi;
```

```
            mevki = mevkisi;
        }

        public void setYetenek(int sav, int atk, int ts)
        {
            savunma = sav;
            atak = atk;
            topSurme = ts;
            degerBelirle();
        }
    }

    class Futbol
    {
        static void Main(string[] args)
        {
            oyuncu oyuncu1 = new oyuncu("Emir", "Santrafor");
            oyuncu1.setYetenek(45, 79, 81);

            Console.WriteLine("Oyuncu geliştirilebilir atak yeteneği="
    + (oyuncu.maxYetenekPuani-oyuncu1.Atak) );

            Console.ReadKey();
        }
    }
}
```

3.10 İsim Uzayı (Namespace)

C#, birbiriyle ilişkili ve birbirine bağımlı sınıfları birlikte gruplamak için isim uzaylarını kullanır. İsim uzayı, sınıf kümelerini tanımlamak için kullanılan bir anahtar sözcüktür. İsim uzaylarındaki sınıf kümelerinin paketlenmesi, temiz bir kod yazmamıza yardımcı olur. İki sınıf aynı isim uzayındaysa, birbirlerini tanıyacaklardır.

Programlama dillerinde bir takım **hazır kütüphaneler** mevcuttur.

Bu kütüphanelerin bazıları standarttır. Bazıları ise programcılar tarafından oluşturulmuştur.

Bu konuda Microsoft'un geliştirdiği .NET Framework alt yapısı kullanılmaktadır. Bu alt yapı Microsoft'un geliştirdiği programlama dillerini kullanan programcılara temel türler ve sınıflar sunar.

```
using System;     // isim uzayı çağırmak
namespace yEmre   // yazdığımız sınıfın isim uzayı
{
    class oyuncu
    {
```

```
        private string ad;
…
```

İsim uzayları (Namespaces) C# dilinde **using** anahtar sözcüğü ile kullanılır.

```
using İsim_Uzay_İsmi;
```

Yazılımcı kod organizasyonunda isim uzaylarını (namespaces) kullanır. Yazılan sınıflar için de bir isim uzayı (namespace) tanımlanarak kaynak kod istenilen şekilde organize edilebilir.

3.10.1 İsimuzaylarını İçiçe Kullanma

İçiçe (nested) kullanılan isim uzayları (namespaces) kodları daha rahat organize etmeye yardımcı olabilir.

İçiçe isim uzayları ile hiyerarşik düzende kod blokları yazılabilir.

İçiçe isim uzayları programa eklenirken de "using" anahtar sözcüğü kullanılır. Bir isim uzayı kullanılmak istendiğinde tam yolu belirtilmelidir.

```
using System;
namespace ust
{
    namespace yunusEmre
    {
        class kaleci
        {
            private string ad;
            private double deger;
            …
        }

        namespace yEmre
    {
        class oyuncu
        {
            private string ad;
            private string mevki;
            …
        }
    }
}
```

Yukarıdaki isim uzaylarını dâhil etmek için aşağıdaki kodlar kullanılabilir:

```
using ust;
using ust.yunusEmre;
```

```
using ust.yEmre;
```

3.10.2 Projeye Yeni Bir Sınıf Ekleme

Solution Explorer penceresinde proje adının üzerinde sağ tuş Add -> New Item -> Class komutu ile projeye yeni bir sınıf dosyası eklenebilir.

3.10.3 "dll" Dosya Yazma

Dinamik Bağlantı Kitaplığı (DLL-Dynamic Link Library), birden fazla program tarafından kullanılabilen işlevler ve kodlar içeren bir kitaplıktır. Bir DLL dosyası oluşturduktan sonra onu birçok uygulamada kullanabiliriz. Bunu için referans oluşturularak DLL dosyası içe aktarılır.

".dll" dosyaları ve ".exe" dosyaları yürütülebilir program modülleridir, ancak DLL dosyaları doğrudan çalıştırılmazlar.

Visual Studio'da yeni bir **Class Library** projesi oluşturularak dll yazılabilir:

- Açılan projede class özellik ve metotlarıyla oluşturulur.
- Proje derlenince, 'bin/debug' dizininde bir 'oyuncu.dll' dosyası üretilir.
- Yeni açacağınız bir projede Solution Explorer üzerinden Add -> Project Reference -> Browse komutu ile oluşturulan "dll" dosya seçilerek referanslara eklenir.
- Yeni projede; using satırı ile dll isim uzayı eklenirek dll kullanılır.

Dll Dosya Kodları:

```
using System;

namespace oyuncular
{
    public class oyuncu
    {
        private string ad;
        private string mevki;
        private int savunma;
        private int atak;
        private int topSurme;
        private double deger;
        public static int maxYetenekPuani = 100;

        public oyuncu(string adi, string mevkisi)
        {
            ad = adi;
```

```
        mevki = mevkisi;
    }

    public int Atak
    {
        set
        {
            if (value < 25)
                atak = 25;
            else
                atak = value;
            degerBelirle();
        }
        get
        {
            return atak;
        }
    }
    public void setYetenek(int sav, int atk, int ts)
    {
        savunma = sav;
        atak = atk;
        topSurme = ts;
        degerBelirle();
    }

    #region metotlar
    public void calis(string konu)
    {
        if (konu == "savunma") savunma += 1;
        if (konu == "atak") atak += 1;
        if (konu == "topSurme") savunma += 1;
        degerBelirle();
    }
    public void degerBelirle()
    {
        deger = savunma * 0.3 + atak * 0.4 + topSurme * 0.5;
    }

    public double getDeger()
    {
        return deger;
    }

    ~oyuncu()
    {
        Console.Write("Nesne ömrünü tamamladı.");
    }
    #endregion
```

```
        }
    }
```

Dll Dosyayı Kullanan Uygulama Kodları:

```
using System;
using oyuncular;

namespace oyuncuUygulama
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Oyuncu İsmi?");
            string isim = Console.ReadLine();
            //oyuncular.oyuncu o1 = new oyuncular.oyuncu();
            oyuncu o1 = new oyuncu(isim, "ORTASAHA");
            o1.setYetenek(89, 91, 80);
            Console.WriteLine(o1.getDeger()+ "M$");
            o1.calis("atak");
            Console.WriteLine(o1.getDeger() + "M$");
            Console.ReadKey();

        }
    }
}
```