ARAYÜZ, YAPILAR, SABİT LİSTELERİ

Arayüz Kavramı

C# programlama dilinde bir sınıfın miras alabileceği maksimum sınıf sayısı 1' dir. Hazırlanan programların daha anlaşılabilir olması ve kalıtım karmaşasının önüne geçilebilmesi adına miras alma işleminde böyle bir sınırlama söz konusudur.

Birden çok kaynaktan miras alınabilmesini mümkün kılan yapılar olan arayüzler de tıpkı sınıflar gibi isim uzayları içerisinde yer almaktadırlar. Her ne kadar bir zorunluluk olmasa da bir programlama geleneği olarak arayüz isimleri "**I**" karakteri ile başlayacak biçimde belirlenir.

```
using System;

namespace Uzay1
{

    public interface ISinif1
    { // arayuz tanimlamasi
        string ozellik1 { get; set; }
        string ozellik2 { get; }
        string ozellik3 { set; }

        int fonksiyon1();
        int fonksiyon2(int arg1);
        void fonksiyon3(int arg2);
        void fonksiyon4();
    }

    public class Sinif1:ISinif1
    { // sinif tanimlamasi
        // Sinif1, ISinif1 arayuzunden miras alir yani bu
        // arayuzdeki tum uyeleri gerceklemek zorundadir

        // miras alinan arayuzde bulunmayan ancak miras alan
        // sinifta yer alan uyeler de bulunabilir
        private string degisken1;
        private string degisken2;
        private string degisken3;

        public Sinif1()
        { // yapici metot
            degisken1 = "degisken_1";
            degisken2 = "degisken_2";
            degisken3 = "degisken_3";
        }

        public string ozellik1
        { // ozellik1' in gerceklemesi

            // bu sinifin bir nesnesinin ozellik1 alt alaninin
            // dondurecegi deger burada belirlenir
            get { return degisken1; }

            // bu sinifin bir nesnesinin ozellik1 alt alanina
            // atama yapildiginda atanan degerin nerede tutulacagi
            set { degisken1 = value; }
        }
```

Çıktı:

```
degisken_1
yeni_deger_1
degisken_2
nesne1.fonksiyon1() --> 1
nesne1.fonksiyon2(60) --> 60
35
fonksiyon_4
```

(devam)

```
        public string ozellik2
        { // sadece okunabilir ozellik
            get { return degisken2; }
        }

        public string ozellik3
        { // sadece yazilabilir ozellik
            set { degisken3 = value; }
        }

        // fonksiyonlar da arayuzde prototipleri (imzalari)
        // belirtildigi sekilde tanimlanmalidir
        public int fonksiyon1()
        {
            return 1;
        }

        public int fonksiyon2(int arg1)
        {
            return arg1;
        }

        public void fonksiyon3(int arg2)
        {
            Console.WriteLine(arg2.ToString());
        }

        public void fonksiyon4()
        {
            Console.WriteLine("fonksiyon_4");
        }
    }

    class Sinif2
    {
        static void Main()
        {
            Sinif1 nesne1 = new Sinif1();
            Console.WriteLine("{0}", nesne1.ozellik1);
            nesne1.ozellik1 = "yeni_deger_1";
            Console.WriteLine("{0}", nesne1.ozellik1);

            Console.WriteLine("{0}", nesne1.ozellik2);
            // nesne1.ozellik2 = "yeni_deger"; // hata: sadece okunabilir

            nesne1.ozellik3 = "yeni_deger_2";
            // Console.WriteLine("{0}", nesne1.ozellik3); // hata: sadece deger alabilir

            Console.WriteLine("nesne1.fonksiyon1() --> {0}", nesne1.fonksiyon1());
            Console.WriteLine("nesne1.fonksiyon2(60) --> {0}", nesne1.fonksiyon2(60));
            nesne1.fonksiyon3(35);
            nesne1.fonksiyon4();
            Console.ReadLine();
        }
    }
}
```

Yukarıda verilen programda, **ISinif1** isminde bir arayüz tanımlanmıştır. Bu arayüz içerisinde 7 farklı üye yer almaktadır:

- **ozellik1** → Karakter dizisi tipinde bir özelliktir (*property*). **ISinif1** arayüzünden miras alan sınıfların nesnelerinin "**ozellik1**" adında birer alt alanı olacaktır. Bu alt alana atama yapıldığında, atanan değerin ilgili sınıf içerisinde tanımlanmış olan *string* tipindeki değişkenlerden hangisinde tutulacağı, sınıf içerisinde tanımlanacaktır.

Benzer biçimde, bu alt alanın değeri elde edilmek istendiğinde hangi değerin döndürüleceği de ilgili sınıf içerisinde verilecektir.

- **ozellik2 → ozellik1'** den tek farkı, sadece okunabilir (değeri alınabilir) olmasıdır. Bu özelliğe atama yapmak mümkün değildir.
- **ozellik3 → ozellik1'** den tek farkı, sadece değer atamasında kullanılabilir olmasdır. Bu özellik aracılığı ile değer okuması yapmak mümkün değildir.
- **fonksiyon1** → Bu arayüzden miras alan sınıfların; "fonksiyon1" isminde, argüman almayan ve *int* tipinde değer döndüren bir fonksiyon tanımlamak zorunda olduklarını belirtir.
- **fonksiyon2** → Bu arayüzden miras alan sınıfların; "fonksiyon2" isminde, *int* tipinde tek bir argüman alan ve *int* tipinde değer döndüren bir fonksiyon tanımlamak zorunda olduklarını belirtir.
- **fonksiyon3** → Bu arayüzden miras alan sınıfların; "fonksiyon3" isminde, *int* tipinde tek bir argüman alan ve değer döndürmeyen bir fonksiyon tanımlamak zorunda olduklarını belirtir.
- **fonksiyon4** → Bu arayüzden miras alan sınıfların; "fonksiyon4" isminde, argüman almayan ve değer döndürmeyen bir fonksiyon tanımlamak zorunda olduklarını belirtir.

Görüldüğü gibi, arayüz içerisinde gerçekleştirim (sözgelimi bir fonksiyonun içerisini doldurarak ne yapacağını belirtmek) söz konusu değildir. Arayüzler sadece birer rehberdir. Arayüzler sayesinde büyük çaplı programların geliştirilmesi ve anlaşılabilirliği kolaylaşacak, üzerinde çalışılan projenin tasarım haritası rahatlıkla görülebilecektir.

Bir arayüz içerisinde yer alan tüm üyelerin erişim belirleyicisi **public** olarak kabul edilir ve bu üyelerin önüne erişim belirleyicisi koyulmaması gerekir. Bu üyeler, **static** erişimli de olamazlar. Bir arayüzden miras alan tüm sınıflar, bu arayüzde yer alan tüm üyeleri kullanmak zorundadırlar.

Programa geri döndüğümüzde, **ISinif1** arayüzünden miras alan **Sinif1** sınıfını görmekteyiz. Bir arayüzden miras alan bir sınıf içerisinde, bu arayüzde tanımlanmayan üyelerin bulunması mümkündür. Nitekim, **Sinif1** sınıfında üye olarak, **ISinif1** arayüzü içerisinde tanımlanmamış olan, *string* tipinde ve **private** erişimli olan **deger1**, **deger2** ve **deger3** değişkenleri yer

almaktadır. **Sinif1** sınıfının argüman almayan yapıcı metodunda ise bu değişkenlerin ilk değerlerinin belirlenmesi işlemi yürütülmektedir.

Sinif1 sınıfı içerisinde, **ISinif1** arayüzünden miras alınan özelliklerin ve fonksiyonların gerçekleştirimleri de yer almaktadır.

ISinif1 arayüzünün bir üyesi olan **ozellik1** özelliği *string* tipinden olup, **get** ve **set** metodları ile kullanılması gerekmektedir. Yani **Sinif1** sınıfı **ISinif1** arayüzünden miras aldığı için **ozellik1** özelliğini gerçekleyecek, bunu yaparken de **ozellik1**' in değeri elde edilmek istendiğinde bu değerin nereden temin edileceğini ve **ozellik1**' e değer atanmak istendiğinde ise bu değerin nerede saklanacağını açıkça belirtmek durumundadır. **ozellik1**' in gerçeklenmesine bakalım:

```
get { return degisken1; }
```

get metodunun gerçekleştiriminde yer alan ifade, "Bu sınıfın nesnesi yaratılarak bu nesnenin **ozellik1** alt alanının değeri elde edilmek istendiğinde **degisken1** tarafından tutulan değer ne ise onu döndür." anlamına gelmektedir. Buradaki dönüş değerinin bir değişkende tutuluyor olması şart değildir, **return** deyiminden sonra doğrudan değer de yazılabilir:

```
get { return "karakter_dizisi"; }
```

ozellik1' in gerçeklenmesinde **set** metodu da zorunlu kılınmıştır.

```
set { degisken1 = value; }
```

set metodunun gerçekleştiriminde yer alan ifade ise, "Bu sınıfın nesnesi yaratılarak bu nesnenin **ozellik1** alt alanına değer atanmak istendiğinde atanan bu değeri **degisken1** değişkeninde tut." anlamını taşımaktadır.

ozellik2 özelliğinin gerçekleştiriminde sadece **get** metodu bulunduğu için, bu özellik aracılığı ile sadece değer alınabilir; bu özellik, herhangi bir değişkene değer atamada kullanılamaz. **ozellik3** özelliği ise sadece **set** metodunu içerdiğinden bu özellik ile değer alma işlemi yapılamaz, sadece değer atama işlemi yapılabilir.

Örneklerden anlaşılacağı gibi, arayüzler vasıtası ile sınıflara miras olarak aktarılan özellikler, **dış dünya ile sınıf değişkenleri arasında elçi** konumundadırlar. Bu özelliklerin kendileri değer tutmamakta; sınıfta yer alan değişkenlere değer atanmasında, bu değişkenlerin değerlerinin ya da kimi zaman da bazı sabit değerlerin döndürülmesinde aracı rol üstlenmektedirler.

Sinif1 sınıfında, **ISinif1** araüzünden miras yolu ile alınan **fonksiyon1**, **fonksiyon2**, **fonksiyon3** ve **fonksiyon4** fonksiyonlarının da argüman alma ve değer döndürme biçimlerine uygun bir biçimde gerçeklendikleri görülmektedir. Tıpkı özelliklerde olduğu gibi, arayüzden miras yolu ile alınan fonksiyonların da miras alan sınıf içerisinde gerçeklenmeleri şarttır.

```
using System;

namespace Uzay1
{

    public interface ISinif1
    { // ISinif1 arayuzu
        int ozellik1 { get; set; }
        int fonksiyon1();
    }

    public interface ISinif2
    { // ISinif2 arayuzu
        string ozellik2 { get; set; }
        string fonksiyon2();
    }

    public class Sinif1:ISinif1, ISinif2
    { // iki arayuzden birden kalitlayan sinif
        public int i1;
        public string s1;

        public Sinif1()
        {
            i1 = 9;
            s1 = "dokuz";
        }

        public int ozellik1
        {
            get { return i1; }
            set { i1 = value; }
        }

        public string ozellik2
        {
            get { return s1; }
            set { s1 = value; }
        }

        public int fonksiyon1()
        {
            return 5;
        }
```

Çıktı:

```
9
77
dokuz
yetmisyedi
nesne1.fonksiyon1() --> 5
nesne1.fonksiyon2() --> bes
```

(devam)

```
        public string fonksiyon2()
        {
            return "bes";
        }

    }

    class Sinif2
    {
        static void Main()
        {
            Sinif1 nesne1 = new Sinif1();
            Console.WriteLine("{0}", nesne1.ozellik1);
            nesne1.ozellik1 = 77;
            Console.WriteLine("{0}", nesne1.ozellik1);

            Console.WriteLine("{0}", nesne1.ozellik2);
            nesne1.s1 = "yetmisyedi";
            Console.WriteLine("{0}", nesne1.ozellik2);


            Console.WriteLine("nesne1.fonksiyon1() --> {0}", nesne1.fonksiyon1());
            Console.WriteLine("nesne1.fonksiyon2() --> {0}", nesne1.fonksiyon2());
            Console.ReadLine();
        }
    }
}
```

C# programlama dilinde bir sınıfın birden çok sınıftan miras alması mümkün değilken, bir sınıfın birden fazla arayüzden miras alması mümkün kılınmıştır. Yukarıdaki programda **Sinif1** sınıfı, hem **ISinif1** arayüzünden, hem de **ISinif2** arayüzünden miras almaktadır. Bu durumda, her iki arayüzde bulunan üyelerin de **Sinif2** içerisinde gerçeklenmesi gerekmektedir.

Arayüzlerin birbirinden miras almaları da söz konusudur. Bir arayüz, miras aldığı arayüzlerde bulunan üyeleri de kendi içerisinde bulundurmaktadır:

```
using System;

namespace Uzay1
{
    public interface ISinif1
    { // ISinif1 arayuzu
        int ozellik1 { get; set; }
        int fonksiyon1();
    }

    public interface ISinif2
    { // ISinif2 arayuzu
        string ozellik2 { get; set; }
        string fonksiyon2();
    }

    public interface ISinif3 : ISinif1, ISinif2
    { // ISinif1 ve ISinif2' den miras alan ISinif3 arayuzu
        string fonksiyon3();
    }
```

(devam)

```
    public class Sinif1 : ISinif3
    { // ISinif3 arayuzunden kalitlayan sinif
        public int i1;
        public string s1;

        public Sinif1()
        {
            i1 = 9;
            s1 = "dokuz";
        }

        public int ozellik1
        {
            get { return i1; }
            set { i1 = value; }
        }

        public string ozellik2
        {
            get { return s1; }
            set { s1 = value; }
        }

        public int fonksiyon1()
        {
            return 5;
        }

        public string fonksiyon2()
        {
            return "bes";
        }

        public string fonksiyon3()
        {
            return "bir";
        }
    }

    class Sinif2
    {
        static void Main()
        {
            Sinif1 nesne1 = new Sinif1();
            Console.WriteLine("{0}", nesne1.ozellik1);
            nesne1.ozellik1 = 77;
            Console.WriteLine("{0}", nesne1.ozellik1);

            Console.WriteLine("{0}", nesne1.ozellik2);
            nesne1.s1 = "yetmisyedi";
            Console.WriteLine("{0}", nesne1.ozellik2);


            Console.WriteLine("nesne1.fonksiyon1() --> {0}", nesne1.fonksiyon1());
            Console.WriteLine("nesne1.fonksiyon2() --> {0}", nesne1.fonksiyon2());
            Console.WriteLine("nesne1.fonksiyon3() --> {0}", nesne1.fonksiyon3());
            Console.ReadLine();
        }
    }
}
```

Yukarıdaki örnekte **ISinif1** ve **ISinif2**, herhangi bir kaynaktan miras almayan başlı başına iki ayrı arayüzdür. **ISinif3** arayüzü ise hem **ISinif1**, hem de **ISinif2** arayüzünden miras almaktadır. Bu durumda **ISinif3** arayüzünden miras alan tüm sınıflar; hem **ISinif1** arayüzünün üyelerini, hem **ISinif2** arayüzünün üyelerini, hem de **ISinif1**' de ve **ISinif2**' de yer almayan ancak **ISinif3**' te bulunan üyeleri gerçeklemek zorundadırlar. Verilen örnekte

ISinif1 arayüzü içerisinde **ozellik1** özelliğine ve **fonksiyon1** fonksiyonuna ait imzalar (prototipler), **ISinif2** arayüzü içerisinde **ozellik2** özelliğine ve **fonksiyon2** fonksiyonuna ait imzalar, **ISinif3** arayüzü içerisinde ise **fonksiyon3** fonksiyonuna ait imza yer almaktadır. **Sinif1** sınıfı ise **ISinif3** arayüzünden miras almaktadır. Bu miras ilişkisinin sonucu olarak **Sinif1** sınıfı; **ozellik1** ve **ozellik2** özellikleri ile **fonksiyon1**, **fonksiyon2** ve **fonksiyon3** fonksiyonlarını gerçeklemek zorundadır.

Programın anlaşılabilirliğini artırmak adına, arayüzleri ve sınıfları ayrı ayrı kaynak kod dosyalarına koymak da tercih edilebilecek bir yaklaşımdır. Kaynak kod dosyalarının aynı proje kapsamında olmaları ve sınıflarla arayüzlerin aynı isim uzayı içerisinde yer almaları şarttır:

kod1.cs

```
namespace Uzay1
{
    public interface ISinif1
    {
        string ozellik1 { get; set; }
        string ozellik2 { get; }
        string ozellik3 { set; }

        int fonksiyon1();
        int fonksiyon2(int arg1);
        void fonksiyon3(int arg2);
        void fonksiyon4();
    }
}
```

kod2.cs

```
using System;

namespace Uzay1
{
    public class Sinif1 : ISinif1
    {
        private string degisken1;
        private string degisken2;
        private string degisken3;

        public Sinif1()
        {
            degisken1 = "degisken_1";
            degisken2 = "degisken_2";
            degisken3 = "degisken_3";
        }

        public string ozellik1
        {
            get { return degisken1; }
            set { degisken1 = value; }
        }

        public string ozellik2
        {
            get { return degisken2; }
        }

        public string ozellik3
        {
            set { degisken3 = value; }
        }

        public int fonksiyon1()
        {
            return 1;
        }

        public int fonksiyon2(int arg1)
        {
            return arg1;
        }

        public void fonksiyon3(int arg2)
        {
            Console.WriteLine(arg2.ToString());
        }

        public void fonksiyon4()
        {
            Console.WriteLine("fonksiyon_4");
        }
    }

    class Sinif2
    {
        static void Main()
        {
            Sinif1 nesne1 = new Sinif1();
            Console.WriteLine("{0}", nesne1.ozellik1);
            nesne1.ozellik1 = "yeni_deger_1";
            Console.WriteLine("{0}", nesne1.ozellik1);
            Console.WriteLine("{0}", nesne1.ozellik2);
            nesne1.ozellik3 = "yeni_deger_2";
            Console.WriteLine("nesne1.fonksiyon1() --> {0}", nesne1.fonksiyon1());
            Console.WriteLine("nesne1.fonksiyon2(60) --> {0}", nesne1.fonksiyon2(60));
            nesne1.fonksiyon3(35);
            nesne1.fonksiyon4();
            Console.ReadLine();
        }
    }
}
```

Yapılar (*Structs*)

Yapıların kullanım amaçları sınıflardan farklı olsa da, sözdizimsel olarak sınıflara benzerlik göstermektedirler. Yapılar, **değer** türündendir yani bir yapıdan türetilen her yapı örneği, kendi verisini beraberinde bulundurur.

```
using System;

namespace Uzay1
{
    struct Dikdortgen
    {
        // en bilgisini sunda saklayalim:
        private int gizli_en;

        public int en
        { // en bilglisne erisim saglayan ozellik (araci)
            get { return gizli_en; }
            set { gizli_en = value; }
        }

        // boy bilgisini sunda saklayalim:
        private int gizli_boy;

        public int boy
        { // boy bilglisne erisim saglayan ozellik (araci)
            get { return gizli_boy; }
            set { gizli_boy = value; }
        }

        public Dikdortgen(int arg_en, int arg_boy)
        { // 2 argumanli yapici metot
            gizli_en = arg_en;
            gizli_boy = arg_boy;
        }

        public Dikdortgen Buyut(Dikdortgen dd)
        { // sinifin ornekleri uzerinde kullanilabilir
            Dikdortgen dikdortgen1 = new Dikdortgen();
            dikdortgen1.en = en + dd.en;
            dikdortgen1.boy = boy + dd.boy;
            return dikdortgen1;
        }
    }

    class Sinif1
    {
        static void Main()
        {
            // Dikdortgen yapisinin bir ornegini olustur
            Dikdortgen d1 = new Dikdortgen();
            d1.en = 2;
            d1.boy = 7;
            Console.WriteLine("d1:  {0}:{1}", d1.en, d1.boy);

            Dikdortgen d2 = new Dikdortgen(5, 6);
            Console.WriteLine("d2:  {0}:{1}", d2.en, d2.boy);

            Dikdortgen d3 = d1.Buyut(d2);
            Console.WriteLine("d3:  {0}:{1}", d3.en, d3.boy);

            // paylasma YOK, kopyalama VAR
            Dikdortgen d4 = d3;
            d4.en = 789; // d3 ile d4 iki ayri ornek
            Console.WriteLine("\nd3.en:  {0}", d3.en);
            Console.WriteLine("d4.en:  {0}", d4.en);
            Console.ReadLine();
        }
    }
}
```

Çıktı:

```
d1:  2:7
d2:  5:6
d3:  7:13

d3.en:  7
d4.en:  789
```

Yukarıdaki örnekte, **Dikdortgen** isminde bir yapı tanımlanmıştır. İçerisinde, bu yapı ile temsil edilecek olan dikdörtgenlerin en ve boy bilgilerini tutmak için kullanılacak olan **gizli_en** ve **gizli_boy** değişkenleri ile bu değişkenlere erişimde aracı görevi üstlenen **en** ve **boy** özellikleri vardır. Ayrıca bu yapının 2 argüman alan bir yapıcı metodu vardır ve bu yapıcı metodun görevi, **gizli_en** ve **gizli_boy** değişkenlerine ilk değerlerini atamaktır. Yapı içerisinde yer alan ve dönüş değeri **Dikdortgen** tipinden bir yapı olan **Buyut** fonksiyonu da gene **dikdortgen** tipinden bir argüman almaktadır. Bu fonksiyon, içerisinde yer aldığı yapının bir örneği yaratıldığı zaman bu örneğin en ve boy değerleri ile, argüman olarak aldığı **Dikdortgen** tipinden yapının en ve boy değerlerini eşleştirmeli bir biçimde toplayarak elde ettiği yeni en ve boy değerlerine sahip bir **Dikdortgen** yapısı örneği yaratmakta ve bu örneği döndürmektedir.

Main fonksiyonunun içerisindeki son 5 satır incelendiğinde, **Dikdortgen** yapısının **d4** isminde yeni bir örneğinin yaratıldığı ve buna **d3**' ün atandığı görülmektedir. Bu noktada yapılar ile sınıflar birbiriyle karıştırılmamalıdır. Yapılar birer **değer** türü olduğundan, böylesi bir atamada **Dikdortgen** yapısına ait yeni bir örneğin tutulabileceği bir bellek bölgesi tahsis edilecek, bu bölgenin başlangıç adresine **d4** takma adı verilecek ve **d3** ile gösterilen bölgedeki değerler birebir **d4** ile gösterilen bölgeye kopyalanacaktır. **d3** ile **d4**, iki farklı bellek bölgesini temsil eden takma adlar olduğundan, **d4** üzerinden yapılacak olan değişiklikler, **d3** ile gösterilen bölgede herhangi bir etki yaratmayacaktır.

Değer ve referans türleri arasındaki farkın daha net bir biçimde kavranabilmesi için aşağıdaki örnek programın incelenmesi yararlı olacaktır:

```
using System;

namespace Uzay1
{
    public struct Yapi
    {
        public int a;
        public int b;

        public Yapi(int arg1, int arg2)
        {
            a = arg1;
            b = arg2;
        }
    }
```

Çıktı:

```
1. ADIM:
(ornek1.a, ornek1.b) = (5, 9)
(nesne1.a, nesne1.b) = (5, 9)
2. ADIM:
(ornek1.a, ornek1.b) = (5, 9)
(nesne1.a, nesne1.b) = (5, 789)
```

(devam)

```
    public class Sinif
    {
        public int a;
        public int b;

        public Sinif(int arg1, int arg2)
        {
            a = arg1;
            b = arg2;
        }
    }

    public class Test
    {
        static void Main()
        {
            Yapi ornek1 = new Yapi(5, 9);
            Sinif nesne1 = new Sinif(5, 9);

            Console.WriteLine("1. ADIM:");
            Console.WriteLine("(ornek1.a, ornek1.b) = ({0}, {1})", ornek1.a, ornek1.b);
            Console.WriteLine("(nesne1.a, nesne1.b) = ({0}, {1})", nesne1.a, nesne1.b);

            Yapi ornek2 = ornek1;  // klonlama
            Sinif nesne2 = nesne1; // paylasma

            ornek2.b = 789;
            nesne2.b = 789;

            Console.WriteLine("2. ADIM:");
            Console.WriteLine("(ornek1.a, ornek1.b) = ({0}, {1})", ornek1.a, ornek1.b);
            Console.WriteLine("(nesne1.a, nesne1.b) = ({0}, {1})", nesne1.a, nesne1.b);

            Console.ReadLine();
        }
    }
}
```

Yukarıdaki örnekten de anlaşılacağı gibi atama durumunda, birer **değer türü** olan *yapı örnekleri* kendi verilerini kendileriyle beraber bulundururken, birer **referans türü** olan *sınıf nesneleri*nde bellekte tutulan aynı verinin farklı tutacaklar tarafından paylaşılması söz konusu olmaktadır.

Sabit Listeleri (*Enums*)

Sabit listeleri, programlama yaparken yazdığımız kodun daha anlaşılabilir olmasına katkıda bulunurlar. Sözgelimi yazdığımız bir programda sesin kısık, orta seviyeli ve yüksek olması için "1, 2, 3" gibi rakamlar yerine "Dusuk, Orta, Yuksek" gibi sabit listesi elemanları kullanıp bunları rakamlarla eşleştirmek programımızın anlaşılabilirliğini büyük ölçüde artıracaktır.

```
using System;

// sabit listesinin ilan edilmesi
public enum Ses : byte
{
    Dusuk = 1, // 1
    Orta, // 2
    Yuksek // 3
}

class Sinif1
{
    static void Main()
    {
        Sinif1 nesne1 = new Sinif1();

        // Acik donusum: int --> Ses
        nesne1.KullanicidanSabitListesiAl();

        // Ses listesinde ada gore gez
        nesne1.ListeUyeleriniAdaGoreSirala();

        // Ses listesinde degere gore gez
        nesne1.ListeUyeleriniDegereGoreSirala();

        Console.ReadLine();
    }

    public void KullanicidanSabitListesiAl()
    {
        Console.WriteLine("\n-----------------");
        Console.WriteLine("Ses Ayarlari:");
        Console.WriteLine("-----------------\n");

        Console.Write(@"
1 - Dusuk
2 - Orta
3 - Yuksek

Lutfen secim yapiniz (1, 2, ya da 3): ");

        string sesString = Console.ReadLine();
        int sesInt = Int32.Parse(sesString);

        // Acik donusum: int --> Ses
        Ses sesDegeri = (Ses)sesInt;

        Console.WriteLine();

        // Karar ver:
        switch (sesDegeri)
        {
            case Ses.Dusuk:
                Console.WriteLine("Ses kisildi.");
                break;
            case Ses.Orta:
                Console.WriteLine("Ses orta seviyede.");
                break;
            case Ses.Yuksek:
                Console.WriteLine("Ses acildi.");
                break;
        }

        Console.WriteLine();
    }

    // Ses listesinde ada gore gez
    public void ListeUyeleriniAdaGoreSirala()
    {
        Console.WriteLine("\n---------------------------- ");
        Console.WriteLine("Ada gore liste uyeleri:");
        Console.WriteLine("----------------------------\n");
```

(devam)

```
        // Ses listesinden uye adlarinin bir listesini al
        // sayisal degerleri elde et ve goster
        foreach (string ses in Enum.GetNames(typeof(Ses)))
        {
            Console.WriteLine("Ses Uyesi: {0}\n Deger: {1}",
                ses, (byte)Enum.Parse(typeof(Ses), ses));
        }
    }

    // Ses listesinde degere gore gez
    public void ListeUyeleriniDegereGoreSirala()
    {
        Console.WriteLine("\n----------------------------- ");
        Console.WriteLine("Degere gore liste uyeleri:");
        Console.WriteLine("-----------------------------\n");

        // Ses listesindeki tum sayisal degerleri al,
        // uye adlarini elde et ve goster
        foreach (byte deger in Enum.GetValues(typeof(Ses)))
        {
            Console.WriteLine("Ses Degeri: {0}\n Uye: {1}",
                deger, Enum.GetName(typeof(Ses), deger));
        }
    }
}
```

(yararlanılan kaynak: www.csharp-station.com)

Yukarıdaki örnek programda, **Ses** isminde bir sabit listesi tanımlanmıştır. Bu sabit listesi **byte** yapısından miras almakta olup, içerisindeki elemanlar 0 ile 255 arasında değişen değerleri temsil eden takma adlardır. Normalde bir sabit listesinde yer alan elemanların değerleri 0' dan başlar ve 1' er 1' er artırılır. Ancak eğer bu elemanlardan herhangi birinin değeri elle atanırsa, bu eleman için elle atanan değer geçerli olur ve bu elemandan sonraki elemanların değerleri de 1' er 1' er artırılarak belirlenir.

Programdaki **Sinif1** sınıfı içerisinde yer alan **Main** fonksiyonunda, bu sınıf içerisindeki diğer üç fonksiyon çağrılmıştır. Bu fonksiyonlardan **KullanicidanSabitListesiAl** fonksiyonu, kullanıcıdan bir tamsayı değeri alarak bunu **Ses** tipine dönüştürmekte ve **Ses** tipindeki bu değeri bir **switch-case** yapısı içerisinde karşılaştırarak eşitlik şartını sağlayan durum için ekrana ilgili değeri yazdırmaktadır. **ListeUyeleriniAdaGoreSirala** fonksiyonu **Ses** listesindeki üyeleri adlarına göre sıralı bir biçimde ekrana yazdırırken **ListeUyeleriniDegereGoreSirala** fonksiyonu da bu üyeleri sahip oldukları sayısal değerlerine göre sıralayarak ekrana yazdırmaktadır.