Bölüm 19

Arayüzler (interfaces)

Arayüz Nedir?
C# dilinde arayüz
Arayüz Bildirimi
Arayüz Kullanımı
Arayüzden Türetme

Arayüz Nedir?

Konuyu basitleştirmek için bir örnekle başlayalım. Aşağıdaki programı çözümleyince arayüzün işlevini ve nasıl kullanıldığını anlayacağız.

Arayüz01.cs

```
using System;

interface Idemo
{
    void Göster();
}

class SDemo : Idemo
{
    public void Göster()
    {
        Console.WriteLine("Idemo arayüzünü kullanıyorum");
    }

    public static void Main(string[] args)
    {
        SDemo d = new SDemo();
        d.Göster();
    }
}
```

Çözümleme:

```
interface Idemo
{
    void Göster();
}
```

bloku `Idemo` adlı bir arayüz bildirimidir. Arayüz bildirimi `interface` anahtar sözcüğü ile başlamış, hemen arkasından arayüzün `Idemo` adı yazılmıştır. Arayüz adının ilk harfi `'I'` ile başlamıştır. Arayüzün gövdesinde `void` değer alan `Göster()` adlı bir metot bildirimi vardır. Arayüz gövdesinde metodun yalnızca imzası (signature) yazılıdır, metodun *tanımı* yoktur. Bu metodun tanımı arayüzü kullanan (implement eden) `SDemo` sınıfı içinde yapılmışır:

```
    public void Göster()
    {
        Console.WriteLine("Idemo arayüzünü kullanıyorum");
    }
```

`Sdemo` sınıfı bildirimi başlarken yazılan

```
class SDemo : Idemo
```

deyimi, derleyiciye `SDemo` sınıfının `Idemo` arayüzünü kullanacağını (implement edeceğini) bildirir.

```
SDemo d = new SDemo();
```

deyimi Sdemo sınıfına ait d adlı bir nesne yaratır.

```
d.Göster();
```

deyimi, `Idemo` arayüzünde bildirilen ve `Sdemo` sınıfında tanımı yapılan `Göster()` metodunu çalıştırır.

An *interface* looks like a class, but has no implementation. The only thing it contains are definitions of *events*, *indexers*, *methods* and/or *properties*. The reason *interfaces* only provide definitions is because they are inherited by *classes* and *structs*, which must provide an implementation for each interface member defined.

C# dilinde `arayüz`

C# dilinde `arayüz` (interface) soyut bir veri tipidir. Sınıfa benzer şekilde bildirilir. Ancak içerdiği öğelerin tanımı değil, yalnızca imzaları vardır. Arayüzler öğe olarak şunları içerebilirler:

Özgen imzaları (signature of properties)
Metot imzaları (method signature)
Delegeler (delegates)
Olaylar (events)
İndeksçiler (indexers)

Arayüzler öğe olarak şunları içeremezler:

Alan (field)
Kurucu
Yokedici

Arayüz asla içerdiği öğeleri tanımlamaz; öyle olmasının basit bir nedeni vardır. Bir arayüz farklı sınıflar tarafından farklı amaçlarla kullanılabilir. Her sınıf, kullandığı (implement) arayüzün öğelerini kendi amacına göre tanımlar.

Arayüzler soyut yapılar olduğu için, doğrudan kullanılamazlar. Onları bir `nesne` (object) için yaptığımız gibi `referans` (işaret) edebiliriz. Bu referanslar ya `null` ya da arayüzü oluşturan `nesneyi` işaret ederler.

C# dilinde, en tepedeki `Object` sınıfı dışında, bir sınıfın ancak bir tane *atası* (taban, base) olabilir. Ama sınıflar ve arayüzler istenildiği kadar oğul üretebilirler. Bu olgu, C++ dilinde var olan, ama java ve C# dilinde olmayan *çoklu kalıtımın* (multiple inheritance) işlevinin daha iyi yapılmasını sağlar.

Bunu şöyle bir örnekle açıklayabiliriz. *Kuşlar* ve *Uçaklar* uçarlar, ama ataları aynı değildir. Onları ayrı ayrı birer sınıf olarak tanımlayabiliriz. Programcı, onları, diyelim ki, `Uçanlar` adlı bir üst sınıftan üretmek zorunda değildir. Ama her ikisi *Uçanlar* adlı bir arayüz oluşturabilir.

Arayüz bildirimi için başlık sözdizimi şöyledir:

```
public inteface IAd
{
}
```

Arayüz daima `public` erişim belirtecini gendeğer (default) olarak alır; öteki erişim nitelemelerini alamaz. O nedenle `public` nitelemesinin konulması ya da konulmaması sonucu değiştirmez. Başlıkta `interface` anahtar sözcüğünden sonra arayüzün adı yazılır. Arayüz adının ilk harfi daima `(I)` olmak zorundadır. Arayüzün gövdesine arayüz öğelerinin yalnızca imzaları (signature) yazılır; bu öğelerin tanımı o arayüzü kullanan (implement eden) sınıf içinde yapılır. Dolayısıyla, bir arayüzü kullanan sınıf, arayüzdeki bütün öğeleri tanımlamış olmalıdır. Aşağıdaki, yalnızca bir metot başlığı içeren basit bir arayüzdür.

```
interface Iaaa
{
    void bbb();
}
```

Bu arayüzün adı `Iaaa` 'dır. Arayüzün bir tek öğesi vardır: `bbb()` metodu. `void bbb()` ifadesi `bbb()` metodunun imzasıdır (signature). Anımsayacaksınız, metodun imzasında metodun adı, değer kümesinin veri tipi ve varsa parametreleri belirtilir. *Arayüz gövdesine* yalnızca metodun imzası yazılır; tanımı arayüzde değil, onu yaratacak sınıfta yapılır:

```
 class ccc : Iaaa
{
    // Arayüzdeki öğe(ler)in açık tanımı
    void Iaaa.bbb()
    {
        // Metot tanımı
    }
}
```

Tabii, arayüzü çağıracak `Main()` metodu herhangi bir sınıfta tanımlanabilir:

```
class Uygulama
{
    static void Main()
    {
        // Arayüzün bir nesnesini yaratır
        Iaaa obj = new ccc();
```

```
        // Arayüzün metodunu çağır
        obj.bbb();
    }
}
```

Bütün bunları bir araya getirirsek, şu program ortaya çıkar:

Arayüz02.cs

```
using System;

interface Iaaa
{
    void bbb();
}

class ccc : Iaaa
{
    // Arayüzdeki öğe(ler)in açık tanımı
    void Iaaa.bbb()
    {
        Console.WriteLine("bbb() metodu arayüzden çağrıldı.");
    }
}

class Uygulama
{
    static void Main()
    {
        // Arayüzün bir nesnesini yaratır
        Iaaa obj = new ccc();

        // Arayüzün metodunu çağır
        obj.bbb();
    }
}
```

Çıktı

```
bbb() metodu arayüzden çağrıldı.
```

Bu basit programı satır satır çözümlersek, arayüzlerin yapılışını ve kullanımını öğrenmiş olacağız. Bu örnek için yaptığımız gözlemleri, bütün arayüzlere genelleştirebiliriz.

interface Iaaa

Arayüzün imzasıdır. Bildirim *interface* anahtar sözcüğü ile başlar, adıyla devam eder.

Arayüze erişim belirteci konmadı. Arayüzün gendeğer (default) erişim belirteci *public* 'dir; niteleme konulmadığında otomatik olarak *public* nitelemesini alır. Başka niteleme yazılırsa derleme hatası doğar.

Arayüzün adının ilk harfi `'I'` dır. Bu harf, derleyiciye sözkonusu ad'ın bir arayüz adı olduğunu bildirir. `'I'` harfi yazılmadığında derleme hatası doğmaz, ama bunu yazmak iyi bir alışkanlıktır. Iaaa gibi bir ad görünce, onun bir arayüz olduğunu hemen anlarız.

void bbb();

Arayüzün öğe(ler)i yalnızca o öğe(ler)in imzasını taşır. Öğelerin tanımları arayüzü kullanan (implement eden) sınıfta yapılır.

```
class ccc : Iaaa
```

ccc sınıfı Iaaa arayüzünü kullanan (implement eden) sınıftır; görünüşte *ccc* sınıfı *Iaaa* arayüzünden türemiştir. Ama *Iaaa* arayüzünün öğelerinin tanımlandığı yer *ccc* sınıfıdır.

```
Iaaa obj = new ccc();
```

Herhangi bir sınıftaki `Main()` metodu içinde arayüzün bir nesnesi (object) yaratılabilir. Bu referans ile arayüze ait öğe(ler) çağrılabilir.

ccc sınıfının bir nesnesi içinden *Iaaa* arayüzünün öğelerine erişilemez.

Şimdi şu denemeyi yapalım. ccc sınıfı içindeki

```
void Iaaa.bbb()
```

imzasını

```
void bbb()
```

biçiminde yazalım ve derlemeyi deneyelim. Derleyici şu hata iletisini gönderir:

```
Error 1 'ccc' does not implement interface member 'Iaaa.bbb()'. 'ccc.bbb()' cannot implement an interface member because it is not public...
```

Şimdi de aynı metodu public nitelemesini verelim:

```
public void bbb()
```

Bu kez programın hatasız derlendiğini göreceğiz.

Aşağıdaki örnek bir noktanın koordinatlarını gösterecek iki özgen (property) içermektedir. Programı satır satır çözümleyiniz.

Arayüz03.cs

```
using System;

interface Inokta // Interface oluşturma
{
    // Özgenlerin imzaları
    int x
    {
        get;
        set;
    }

    int y
    {
        get;
        set;
    }
```

```
}
```

Arayüzü implement eden sınıf:

```
class Nokta : INokta
{
    // Alanlar (Fields):
    private int _x;
    private int _y;

    // Kurucular (Constructor):
    public Nokta(int x, int y)
    {
        _x = x;
        _y = y;
    }

    // Özgenlerin oluşumu (Property implementation):
    public int x
    {
        get
        {
            return _x;
        }

        set
        {
            _x = value;
        }
    }

    public int y
    {
        get
        {
            return _y;
        }
        set
        {
            _y = value;
        }
    }
}

class Uygulama
{
    static void NoktaYaz(INokta p)
    {
        Console.WriteLine("x={0}, y={1}", p.x, p.y);
    }

    static void Main()
    {
```

```
        Nokta p = new Nokta(2, 3);
        Console.Write("Noktanın koordinatları: ");
        NoktaYaz(p);
    }
}
```

Çıktı

```
Noktanın koordinatları: x=2, y=3
```

Arayüzler için şu kurallar geçerlidir:

Bir arayüzün dendeğer erişim belirteci public 'dir.
Bir arayüz'ün bütün öğelerinin erişim belirteci public 'dir.
Bir arayüz, bir yapı(struct)'dan veya bir sınıf(class)'tan kalıtımla türetilemez.
Bir arayüz, başka bir arayüzden ya da başka arayüzlerden kalıtımsal olarak türetilebilir.
Arayüz öğeleri static nitelemesini alamaz.
Arayüzü kullanan bir sınıf, arayüzün bütün öğelerinin açık tanımlarını vermek zorundadır.
Arayüzü kullanan sınıfta, arayüzün bütün öğeleri public erişim belirtecini alır.
Arayüz, nesnelere erişimi standart biçime sokar; nesnenin iç yapısını bilmeden nesneyi kullanma olanağı yaratır.

Arayüzden Arayüz Türetme

Bazan arayüzden arayüz türetmemiz gerekebilir. Bu iş, sınıftan sınıf türetmek kadar kolaydır. Aşağıdaki program bu işi yapmaktadır.

```
using System;

interface IAnaInterface
{
    void f();
}

interface IOğulInterface : IAnaInterface
{
    void g();
}

class aaaSınıf : IOğulInterface
{
    static void Main()
    {
        aaaSınıf iImp = new aaaSınıf();
        iImp.g();
        iImp.f();
    }

    public void g()
    {
        Console.WriteLine("g() metodu çağrıldı.");
    }

    public void f()
```

```
    {
        Console.WriteLine("f() metodu çağrıldı.");
    }
}
```

Okuma Parçası

Kullanıcı Arayüzü

Kullanılan herhangi bir alet için kullanıcıya bazı kolaylıklar sağlanır. Örneğin, bir fotoğraf makinasının bir televizyonun, bir otomobilin, bir uçağın, bir geminin yönetilmesi sırasında, kullanıcı, kullandığı aletin içindeki onlarca ya da on binlerce parçanın nasıl çalıştığını, o parçalar arasındaki etkileşimin nasıl yapıldığını bilmek zorunda değildir. O, yalnızca belirli işleri yaptıran düğmelere basmak, kolları çevirmek, pedallara basmak gibi kolay işleri yapar. Bir otomobil sürücüsünü alalım. Sürücü kontak anahtarını çevirince motor çalışır. Motorun çalışması için yakıt deposunun, yakıt pompasının, karbüratörün, silindirlerin, ateşleme sisteminin nasıl çalıştıklarını, kendi aralarında nasıl bir uyum sağladıklarını bilmeyebilir. Vites değiştirince aktarma organlarındaki dişlilerin nasıl yer değiştirdiğini, gaz pedalına basınca otomobilin neden hızlandığını, frene basınca aracın neden durduğunu bilmeyebilir. Bir düğmeye basmak, bir kolu çevirmek, arka arkaya bir çok işin sırayla ve önceden tasarlanmış bir düzen içinde yapılmasına neden olur.

Aletle insan arasında bu iletişimi kuran şey bir kullanıcı arayüzüdür. Bazı arayüzler gerçek anlamda kullanıcı dostudur, kolay öğrenilir, kolay kullanılırlar. Bazıları daha zor olabilir. Örneğin, ilk otomobillerde kontak anahtarı ve pistonlara ilk hareketi veren elektrikli mekanizma yoktu. Motoru çalıştırmak için, kullanıcının, kol kuvvetiyle pistonları harekete geçirmesi gerekiyordu. Benzer olarak bazı otomobillerde camlar bir düğmeyle açılıp kapanır, bazılarında ise mekanik bir kolla yapılır. Camı açan düğme ve kol birer kullanıcı arayüzüdür. Arayüzlerin bazısı çok kolay kullanılır, bazısı daha zor kullanılır.

Bilgisayar programları için de benzer şeyi düşünebiliriz. Yeni kuşak programlarda, grafiksel kullanıcı arayüzü (Graphical User Interface) denilen bir görüntü ekrana gelir. Örneğin, *Windows İşletim Sistemi* açılınca önümüze gelen ekran bir arayüz görüntüsüdür. *MSWord* programını ya da *Excel* programını açtığımızda önümüze gelen görüntüler arayüzlerin görüntüleridir. Ekrandaki göstergeden seçilen bir düğmeye basınca, kullanıcının asla görmediği program parçaları belli bir sıra ve uyum içinde geri planda çalışır ve kullanıcının istediğini yapar. Bu programlar için, geride çalışan modüllere hiç dokunmadan, farklı arayüzler hazırlanabilir. Mekanik aletlerde olduğu gibi, bilgisayar programlarında da bazı arayüzler gerçek anlamda kullanıcı dostudurlar, kolay öğrenilirler, kolay kullanılırlar. Bazıları ise daha az kullanıcı dostu olabilirler. Kullanıcı arayüzünün iyi ya da kötü oluşu, programı oluşturan modüllerle ilgili değildir. Aynı program için, çok iyi bir arayüz hazırlanabileceği gibi, kötü bir arayüz de hazırlanabilir.

Bilgisayar programları için çok çeşitli arayüzler vardır. Bunlar kendi aralarında Grafiksel Kullanıcı Arayüzleri (GUI- Graphical User Interfaces), Web Tabanlı Arayüzler (WUI- Web User Interfaces), Komut Arayüzleri (Command Line Interfaces), Dokunmatik Arayüzler (Touch Interfaces), Ses Arayüzleri (Voice Interfaces), Uygulama Programı Arayüzü (API- Application Program Interface), vb. sınıflara ayrılır.