Bölüm 22

Boxing – Unboxing

Kutulama (boxing) Nedir?
Kutulama Nasıl Yapılır?
Kutu Nasıl Açılır?

Kutulama Nedir?

C# dilinde veri tiplerinin *değer-tipleri (value types)* ve *referans-tipleri (reference type)* olmak üzere ikiye ayrıldığını biliyoruz.

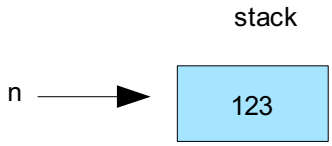
Değer tiplerinin referans tipine dönüşmesine *kutulama (boxing)*, referans tiplerin değer tipine dönüşmesine ise *kutu-açma (unboxing)* denir. Bu dönüşümler, daha önce veri tipleri arasında yaptığımız istemsiz (implicit) ve istemli (explicit) dönüşümlerin özel bir durumudur; yani tiplerden birinin nesne (object) olduğu durumdur.

Değer tiplerinin referans tipine dönüştürülmesiyle, nesne üzerinde yapılan bazı işlemlerin dönüşen değer tipine de uygulanabilmesi olanağı doğar. O nedenle, kutulama ve kutu-açma eylemleri C# dilinde önemlidirler.

C# dili, değer tiplerini *stack* içinde, referans tiplerini *heap* içinde tutar. Örneğin,

```
int n = 123;
```

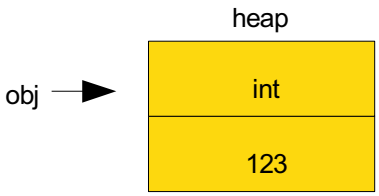
deyimi, n değişkeni için *stack* içinde bir adres ayırır, 123 değeri o adrese yerleşir, `n` ise o adresi işaret eden referanstır.



Öte yandan

```
object obj = n;  // istemsiz kutulama (implicit boxing)
```

deyimi istemsiz (implicit) bir dönüşümdür. Deyim `object` tipinden bir nesne yaratır ve `123` sayısını o nesne içine yerleştirir. `obj` o nesneyi işaret eden referanstır. `obj` referans tipi olduğu için, onun işaret ettiği adres *heap* içindedir. Bu eylem 123 sayısını bir kutuya koymaya benzer. "`Boxing`" kutulama, paketleme anlamındadır.



Kutulama (boxing) ve kutu-açma (unboxing) eylemleri değer tiplerinin nesne imiş gibi işlem görmesini sağlar. Bir değer tipini kutulayınca (*boxing)* edince `object` sınıfının bir nesnesi içine gömülür (paketlenir). Böyle olunca, söz konusu değer tipi *heap* içinde depo edilmiş olur. Daha önemlisi, o değer artık bir nesne gibi kullanılabilir.

Kutu-açma *(unboxing)* ise tersini yapar. Nesne içine gömülen değeri nesnenin dışına alır:

```
n = (int)obj;  // kutu-açma (unboxing)
```

Bu istemli (explicit) bir dönüşümdür; daha önce yaptığımız 'casting' eylemidir. `obj` referansının işaret ettiği nesne içindeki değeri `int` tipine dönüştürmektedir.

Kutulama (boxing) ve kutu-açma (*unboxing)* dönüşümleri performans azaltıcı eylemlerdir. Çünkü, *boxing* yaparken heap içinde bir nesne yaratılıyor, yani ana bellekte bir yer ayrılıyor ve *stack* 'taki değer oraya kopyalanıyor.

Kutu-açma (unboxing) eylemi biraz daha az olmakla birlikte, nesneden değere dönüşüm (casting) için bir zaman harcar ve performansı düşürür. C# 2.0 sürümünden sonra bu sorunu aşan yöntemler getirilmiştir. Özellikle `System.Collections.Generic` adyeri (namespace), *boxing* ve *unboxing* yapmaya gerek bırakmayan çok sayıda sınıf, metot ve arayüz tanımlamıştır. Performansın önemli olduğu zamanlarda, o adyerine başvurulması gerekir. C# dilinin ilk sürümlerinde önemli bir araç olarak kullanılan *boxing* ve *unboxing* eylemlerini, performansın önemli olmadığı durumlarda başvurulan kolay yöntemler olduğu için bilmekte yarar vardır. Bu eylemleri birkaç örnekle göstereceğiz.

Değer Tipleri (Value Types)

Değer tipleri ilkel tiplerdir, doğrudan FCL'e gönderilir. Örneğin *Int32* tipi *System.Int32* ye gider, *double* tipi `System.Double` ' a gider. Bütün değer tipleri *stack* içinde depolanır. Bütün değer tipleri

`System.ValueType` sınıfından türerler. `System.ValueType` 'dan tütetilen bütün *yapılar* ve *enumerated* (numaralanmış) tipler *stack* içinde yaratılırlar.

Referans Tipleri (Reference Types)

Bütün referans tiplere *heap* içinde yer ayrılır. Bütün sınıflar referans tipidir. `new` operatörü referansların işaret edeceği adresleri *heap* içinde ayırır.

Boxing01.cs

```
class TestBoxing
{
    static void Main()
    {
        int n = 123;
        object obj = n;  // istemsiz kutulama (implicit boxing)

        n = 456;         // n 'nin değerini değiştir

        System.Console.WriteLine("Değer-tipi değeri = {0}", n);
        System.Console.WriteLine("nesne tipi değeri = {0}", obj);
    }
}
```

Çıktı

```
Değer-tipi değeri = 456

nesne tipi değeri = 123
```

Bu programı çözümleyelim.

```
int n = 123 ;
```

deyimi n değişkenine *stack* içinde bir adres ayırıyor ve o adrese 123 sayısını yerleştiriyor.

```
object obj = n;
```

deyimi istemsiz kutulama yaparak *heap* içinde *obj* referansı ile işaret edilen bir nesne yaratıyor ve oraya 123 değerini yerleştiriyor. Bu aşamada hem *stack* 'ta hem *heap* 'te 123 tutuluyor.

```
n = 456;
```

atama deyimi, `n` 'nin *stack* içindeki adresine 456 değerini yazıyor. Bu atamadan sonra, *stack* 'ta 123 silinip, yerine 456 konulmuş oluyor.

Son iki satır *stack* 'taki ve *heap* 'teki verileri konsola yazıyor. Bunların farklı oluşu, kutulama eyleminin 123 sayısına ana bellekte farklı bir yer ayırdığını gösterir. Eğer farklı adres ayırmasaydı, son iki satır aynı çıktıyı verirdi.

Veri tipi değişse de sonuç aynıdır. Programdaki `int` yerine `string` konulursa, aşağıdaki gibi benzer sonuç elde edilir.

Boxing02.cs

```
class TestBoxing
{
    static void Main()
    {
        string s = "Ankara" ;
        object obj = s;  // istemsiz kutulama (implicit boxing)

        s = "İstanbul";  // s 'nin değerini değiştir

        System.Console.WriteLine("Değer-tipi değeri (stack 'taki değer) =
{0}", s);
        System.Console.WriteLine("Nesne tipi değeri (heap 'teki  değer) =
{0}", obj);
    }
}
```

Çıktı

```
Değer-tipi değeri (stack 'taki değer) = İstanbul
Nesne tipi değeri (heap 'teki  değer) = Ankara
```

Örnekler

Bütün değer-tipleri System.ValueType sınıfından türerler, dedik. Başka bir deyişle, System.ValueType sınıfı bütün değer-tiplerinin atasıdır, onların taban sınıfıdır. Bunun ne anlama geldiğini örnekler üzerinde açıklayalım. Yukarıdaki programda int n = 123 yerine

```
System.ValueType n = 123 ;
```

koyalım. Programı derleyip koşturursak, sonucun aynı olduğunu görebiliriz. Peki ama böyle olması ne anlama gelir? Yukarıdaki deyim

```
System.Int32  n = 123 ;
```

deyimine denk midir? Denk olup olmadığını, bu deyimi programda yazarak deneyebiliriz. Gerçekten int n = 123 deyimi yerine System.ValueType n = 123 deyimini yazdığımızda da aynı sonucu elde edeceğimizi deneyerek görebiliriz. Buradan şu sonuç çıkıyor:

```
int n = 123 ;
System.ValueType n = 123 ;
System.Int32 n = 123 ;
```

deyimleri denktir. `System.ValueType`, temsil ettiği veri tipiyle ilgili işlemlere girebilir. Örneğin,

```
int n = 5 ; n++ ;
```

deyimi

```
System.ValueType n = 5 ; n++ ;
```

deyimine denktir.

Şimdi aklımıza gelmesi gereken bir soru var. `System.ValueType n = 123` deyimi neden `System.Int32 n = 123` deyimine denktir? Başka bir deyişle, neden `System.Int16 n = 123` ya da `System.Int64 n = 123` deyimine denk değildir? Bu sorunun yanıtı şudur: `int n = 123` deyimi yazıldığında, derleyici `n` sayısını default olarak `Int32` 'ye gönderir. Her veri tipi için

derleyicinin belirlediği bir *başlangıç-tipi* vardır. Tamsayılar için bu tip `Int32` dir. Gerçekten böyle olduğunu görmek isterseniz, `GetType()` metodunu kullanabilirsiniz. Aşağıdaki program bazı veri tiplerinin başlangıç tiplerini yazmaktadır.

Boxing03.cs

```
using System;
class TestBoxing
{
    static void Main()
    {
        System.ValueType n = 123 ;
        Console.WriteLine(n.GetType()); // System.Int32
        //-------------------------------------------------------
        System.ValueType r = 23.45;
        Console.WriteLine(r.GetType()); // System.Double
        //-------------------------------------------------------
        System.ValueType x = 23.45F;
        Console.WriteLine(x.GetType()); // System.Single
        //-------------------------------------------------------
        System.ValueType y = 2U;
        Console.WriteLine(y.GetType()); // System.UInt32
        //-------------------------------------------------------
        System.ValueType b = false;
        Console.WriteLine(b.GetType()); // System.Boolean
        //-------------------------------------------------------
    }
}
```

System.ValueType 'dan object tipine istemsiz (implicit) dönüşümü C# kendiliğinden yapar. Bu demektir ki, istemsiz kutulama (implicit boxing) eylemi kendiliğinden olur. Ama istenirse, istemli (explicit) dönüşüm; yani istemli kutulama (explicit boxing) de yapılabilir. Aşağıdaki program bunu yapmaktadır.

Boxing04.cs

```
using System;

class TestBoxing
{
    static void Main()
    {
        Int32 x = 10;
        object obj1 = x;  // istemsiz kutulama (implicit boxing)
        Console.WriteLine(" obj1 nesnesi = {0}", obj1); // Çıktı : 10
        //-------------------------------------------------------------
        Int32 y = 10;
        object obj2 = (object)y; // istemli kutulama (Explicit Boxing)
        Console.WriteLine(" obj2 nesnesi  = {0}", obj2); // Çıktı : 10
    }
}
```

Kutudan başka tip çıkar mı?

Kutulanan bir değer, başka bir tipe dönüşmüş olarak açılabilir mi? Örneğin, *Int32* olarak kutulanmış bir veriyi kutudan *Int64* olarak geri alabilir miyiz? Bu sorunun yanıtı "hayır" dır. Veri hangi tip olarak kutulanmışsa, kutu açılınca aynı tip olarak çıkar.

Aşağıdaki program bunu östermektedir. Program derlenir, ama koşmaz, *RunTime* hatası doğar. Çünkü `Int32` tipi kutulanıyor (boxing), ama kutuyu açarken (unboxing) içeriğinin `Int64` tipinden olması isteniyor. Buna izin verilmez. Kutuya hangi tip konulduysa, kutu açılınca aynı tip çıkacaktır.

Boxing05.cs

```
using System;

class TestBoxing
{
    static void Main()
    {
        Int32 x = 123;     // Int32 tip değişken bildirimi
        Int64 y ;          // Int64 tip değişken bildirimi
        object obj = x;    // istemsiz Kutulama (Implicit Boxing)
        y = (Int64)obj;    // hata !
        Console.WriteLine("y={0}", y);
    }
}
```

Uyarı mesajı:

```
Unhandled Exception: System.InvalidCastException: Specified cast is not valid.
```

Tabii, şimdiye dek öğrendiklerimizle bu sorunun üstesinden nasıl gelebileceğimizi biliyoruz. Önce kutu-açma (unboxing) eylemi verinin asıl tipi olan *Int32* tipine yapılır. Sonra kutudan çıkan değere istemli (explicit) dönüşüm (casting) uygulayabiliriz.

Boxing06.cs

```
using System;

class TestBoxing
{
    static void Main()
    {
        Int32 x = 123;     // Int32 tip değişken bildirimi
        Int64 y ;          // Int64 tip değişken bildirimi
        object obj = x;    // istemsiz Kutulama (Implicit Boxing)
        y = (Int64)(Int32)obj; // double tipine istemli dönüşüm (casting)
        Console.WriteLine("y={0}", y);
    }
}
```

Kutuyu istemli dönüşümle açabilir miyiz?

Şimdi şunu deneyelim. Kutularken istemsiz ve istemli dönüşümler yaptık. Acaba kutuyu açarken de istersek istemsiz istersek istemli dönüşüm yapabilir miyiz? Zaten yukarıdaki örneklerde hep istemli dönüşümle kutuyu açtık; yani sorunun birisinin yanıtını biliyoruz: Kutu açarken istemli dönüşüm yapılabiliyor. Öyleyse, şimdi kutunun *istemsiz (implicit)* dönüşümle açılıp açılmayacağını deneyebiliriz. Aşağıdaki programı derlemeyi deneyelim.

Boxing07.cs

```
using System;

class TestBoxing
{
    static void Main()
    {
        Int32 x = 123;    // Int32 tip değişken bildirimi
        object obj = x;   // istemli Kutulama (Implicit Boxing)
        x = obj;          // istemli kutu-açma (implicit unboxing)
        Console.WriteLine("y={0}", x);
    }
}
```

Derleyici şu hata iletisini gönderecektir:

```
Error    1    Cannot implicitly convert type 'object' to 'int'. An explicit conversion exists (are you missing a cast?)...
```

Derleyicimiz, *nesne* tipinin *int* tipine istemsiz dönüşemeyeceğini, istemli dönüşüm gerektiğini söylüyor. *int* tipi yerine *string, bool* vb tipleri koyarsanız, gene benzer hata iletisini alırsınız. Demek ki, nesne tipinden herhangi bir `SystemValueType` tipine istemsiz dönüşüm yapılamaz.

Aşağıdaki program generic ve non-generic tiplerin hızlarını karşılaştırmaktadır.

Boxing08.cs

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

class Deneme
{
    static void Main(string[] args)
    {
        Deneme denek = new Deneme();
        denek.GenericsTest();
        denek.ArrayListTest();
    }

    public void GenericsTest()
    {
        int total = 0;
        List<int> li = new List<int>();
        for (int i = 0; i < 20; i++)
            li.Add(i);
        for (int i = 0; i < 10000000; i++)
            foreach (int el in li)
                total += el;
```

```
        Console.WriteLine(total);
    }

    public void ArrayListTest()
    {
        int total = 0;
        ArrayList al = new ArrayList();
        for (int i = 0; i < 20; i++)
             al.Add(i);
        for (int i = 0; i < 1000000; i++)
            foreach (int el in al)
                 total += el;
        Console.WriteLine(total);
    }
}
```

Alıştırmalar

Aşağıdaki programları koşturmadan, çıktılarını tahmin ediniz.

Boxing09.cs

```
using System;
class Test
{
    static void Main()
    {
        Console.WriteLine(7.ToString());
    }
}
```

Boxing10.cs

```
using System;
class Test
{
    static void Main()
    {
        int i = 1;
        object o = i;         // boxing
        Console.WriteLine(o);

        int j = (int)o;       // unboxing
        Console.WriteLine(j);

    }
}
```