Bölüm 27

string'den sayıya dönüşüm

Convert Sınıfı
Parse() Metodu
Biçemli Stringe Dönüşen Sayıların Ters Dönüşümü
CultureInfo sınıfı
Sayı Stilleri Listesi (Number Styles)

Önceki bölümlerde sayıları istenen biçemde stringe dönüştürmeyi öğrendik. Programcılıkta, bu işin tersine de sık sık gerekseme duyarız. Herhangi bir biçemde stringe dönüştürülmüş sayıyı tekrar sayının aslına dönüştürmek gerekebilir. Bu işi yapacak yöntemlere sahibiz.

String tipinin giriş/çıkış işlemlerinde yaşamsal rol oynadığını biliyoruz. Bilgisayara girişlerin ve çıkışların hemen hepsi string tipindendir. Örneğin, klavyeden yaptığımız bütün girişler string tipindendir. Sistem giriş biriminden aldığı stringi içeride olması gereken veri tipine dönüştürür. Bunun tersi de doğrudur. Örneğin, bilgisayardan gelen bir sayısal çıktı, konsola veya yazıcıya rakamlardan oluşan bir string olarak gider. Çıkışta sayıdan stringe dönüşüm gene sistem tarafından yapılır. Burada sistem derken, işletim sistemi derleyici ve kaynak programdan oluşan bütünü kasdediyoruz.

Öyleyse girdi/çıktı işlemlerinde bir veri tipinden başka bir veri tipine dönüşüm söz konusudur. Daha önce sayısal veri tiplerinin birinden ötekine *istemsiz* (implicit) veya *istemli* (explicit) dönüşümler (casting) yapmıştık. Ama şimdi sözünü ettiğimiz dönüşüm bir 'casting' olayı değildir. Ondan çok farklıdır. Bu dönüşüm sayıdan sayıya değil, sayıdan stringe ve stringden sayıya yapılan ters dönüşümlerdir.

Özellikle şunu da belirtmeliyiz. Sayılar ile stringler arasındaki bu dönüşüm, başka veri tiplerine uygulanamaz.

Convert sınıfı

`.NET Framework` System namespace içinde `Convert` adlı bir sınıf sunar. Bu sınıf çok sayıda static dönüşüm metodu içerir. Metotlar `static` olduğu için, onlar sınıfa ait bir nesne yaratmadan doğrudan kullanılabilirler. Böyle oluşu, programcının işini kolaylaştırır. Dönüşümler için, C# sayısal sınıfları ile .NET sınıfları arasındaki ilişkiyi anımsayalım.

C# Veri Tipi	.NET Yapı Adı
bool	Boolean
byte	Byte
decimal	Decimal
double	Double
float	Single
int	Int32
long	Int64
sbyte	SByte
short	Int16
uint	UInt32
ulong	UInt64
ushort	UInt16

Convert sınıfının metotlarından bazıları şunlardır:

```
ToBoolean, ToByte, ToChar, ToDateTime, ToDecimal, ToDouble, ToInt16,
ToInt32, ToInt64, ToSByte, ToSingle, ToString, ToUInt16, ToUInt32,
ToUInt64
```

Bu metotların işlevleri adlarından bellidir. Örneğin,

Convert.ToInt32	metodu stringe dönüşmüş tamsayıyı, tamsayı aslına dönüştürür (Int32).
Convert.ToDecimal	metodu stringe dönüşmüş kesirlisayıyı, kesirlisayı aslına dönüştürür.
Convert.ToBoolean	metodu "true" ve "false" stringlerini, sırasıyla True ve False boolean değerlerine dönüştürür. "true" ve "false" stringlerinin büyük ya da küçük harfle yazılmış olmaları sonuca etkimez.

```
using System;
class TestIntegerCustomFormats
{
```

```
    static void Main()
    {
        // Stringi tamsayıya dönüştür
        int stringToInt = Convert.ToInt32("12345");
        Console.WriteLine(stringToInt); // Çıktı: 12345

        // Stringi kesirli sayıya dönüştür
        decimal stringToDecimal = Convert.ToDecimal("12345,6789");
        Console.WriteLine(stringToDecimal);  // Çıktı : 12345,6789

        /* "true" stringini boolean'a dönüştür
           "true" nun büyük/küçük harfle yazılması
           sonucu etkilemez
       */
        bool stringToTrue = Convert.ToBoolean("true");
        Console.WriteLine(stringToTrue); // Çıktı : True

        /* "false" stringini boolean'a dönüştür
           "false" 'in büyük/küçük harfle yazılması
           sonucu etkilemez
       */
        bool stringToFalse = Convert.ToBoolean("FALse");
        Console.WriteLine(stringToFalse); // Çıktı : False
    }
}
```

Kesirli sayıları tamsayılara dönüştürmek için `casting` yerine `ToInt32()` metodu kullanılabilir. Yalnız bir farklılığa dikkat etmek gerekir. Casting sayının kesir kısmını atar, tamsayı kısmını alır. ToInt32() metodu kesirli sayıyı en yakın tamsayıya yuvarlar. Aşağıdaki program bu ayrımı göstermektedir.

```
using System;
class Dönüşümler
{
    static void Main()
    {
        decimal ilkSayı = 17.5M;
        Console.WriteLine(ilkSayı);  // Çıktı: 17,5
        int tamSayıya = Convert.ToInt32(ilkSayı);
        Console.WriteLine(tamSayıya); // Çıktı: 18
        int castEdilen = (int)ilkSayı;
        Console.WriteLine(castEdilen); // Çıktı: 17
    }
```

Parse() Metodu

`Parse()` metodu `string` olarak yazılan sayıları aslına dönüştürür. Bir bakıma `ToString()` metodunun tersidir.

Bütün sayı sınıflarının birer tane `ToString()` metodu olduğunu ve bunların herbirinin dörder tane overload edilmiş sürümlerinin varlığını biliyoruz. Bunları kullanarak, sayıları istediğimiz biçemde stringe dönüştürebiliyoruz.

Benzer olarak bütün sayı sınıflarının birer tane `Parse()` metodu vardır. Bunların da *aşkın yüklenmiş* (`overloaded`) dörder tane sürümleri vardır.

`ToString()` metodunun yaptığı işin tersini yapan `Parse()` metodunun, en azından `ToString()` metodu ile stringe dönüşen sayıları, bir ters dönüşüm gibi, kendi özlerine dönüştürmesini umut etmeliyiz. Gerçekten, `Parse()` metodu bu ters dönüşümleri yapabilecek yeteneklere sahiptir. Bu yeteneklerini, aşağıda bazı örnekler üzerinde göreceğiz.

Parse Metotları

- Parse (String)
- Parse (String, NumberStyles)
- Parse (String, IFormatProvider)
- Parse (String, NumberStyles, IFormatProvider)

Birincisini kullanmayı daha ilk bölümlerde öğrendik. Stringe dönüşmüş `int` tipi bir tamsayıyı kendi özüne dönüştürmek için

```
int.Parse(s);
```

dönüşümünü kullanırız. Aşağıdaki program bunun nasıl yapıldığını göstermektedir.

Parse01.cs

```
using System;

namespace  StringdenSayıyaDönüşüm
{
    class Parse01
    {
        static void Main(string[] args)
        {
            string s = "12345";
            int n = int.Parse(s);
            n++;
            Console.WriteLine(n);// Çıktı: 12346
        }
    }
}
```

Aşağıdaki program 123 sayısını önce `ToString()` metodu ile "123" stringine, sonra `Parse()` metodu ile tekrar 123 sayısına dönüştürüyor.

IntParse.cs

```
using System;
using System.Globalization;
using System.Threading;

namespace Dönüşümler
{
    class IntParse
```

```
    {
        static void Main(string[] args)
        {
            int n = 123;
            string s = n.ToString();     // s = "123"
            n = Int32.Parse(s);          // n =  123
            Console.WriteLine(n);        // 123
        }
    }
}
```

Biçemli Stringe Dönüşen Sayıların Ters Dönüşümü

Sayının “ $12.345,6789” biçiminde stringe dönüşmüşünü bizler hemen algılarız. Ama bilgisayarın bunu algılamasını umamayız. Ona, $ işaretinin önünde ve ardında boşluklar olduğunu, sayının binlik gruplara ayrıldığını ve kesir ayracının var olduğunu bildirmeliyiz. `Parse()` metodu o zaman stringi algılar ve onu sayının aslına geri dönüştürebilir. `Parse()` metodu bu iş için yeterince araçlara sahiptir. Böyle durumlarda `Parse()` metodunun öteki sürümlerini kullanırız. O metotların nasıl kullanıldığa dair birkaç örnek vereceğiz.

Aşağıdaki program, binlik basamaklarına ayrılmış bir sayıyı çözümleyip aslına dönüştürüyor.

DoubleParse01.cs

```
using System;
using System.Globalization;
using System.Threading;

namespace Metotlar
{
    class DoubleParse01
    {
        static void Main(string[] args)
        {
            string s = "12.345.678" ;

            // Binlik gruplarına ayrılmış sayıyı aslına dönüştür
            int n = int.Parse(s, NumberStyles.AllowThousands); //
12345678
            Console.WriteLine(n);  //Çıktı: 12345678
        }
    }
}
```

Aşağıdaki program, kesir ayracı içeren bir stringi çözümleyip asıl sayıya dönüştürüyor.

DoubleParse.cs

```
using System;
using System.Globalization;
using System.Threading;
```

```
namespace Metotlar
{
    class DoubleParse
    {
        static void Main(string[] args)
        {
            string s = "12345,678" ;

            // Kesir ayracı içeren stringi asıl sayıya dönüştür
            double flt = double.Parse(s, NumberStyles.AllowDecimalPoint);
            Console.WriteLine(flt); // Çıktı: 12345,678
        }
    }
}
```

Aşağıdaki program TL para birimi simgesini içeren stringi çözümleyip asıl sayıya dönüştürüyor.

Cparse.cs

```
using System;
using System.Globalization;
using System.Threading;

namespace Metotlar
{
    class cParse
    {
        static void Main(string[] args)
        {
            string s = "12345TL" ;

            // TL para birimi içeren stringi
            // asıl sayıya dönüştürüyor
            int n = int.Parse(s, NumberStyles.AllowCurrencySymbol);
            Console.WriteLine(n);  // Çıktı: 12345
        }
    }
}
```

Bazan string birden çok biçemleme öğesi içerebilir. Örneğin, “12345,67TL” stringi hem para birimini hem de kesir ayracı içeriyor. Bu durumda `OR` ya da `(|)` simgesini kullanarak her ikisini çözümlenmesini sağlayabiliriz. Aşağıdaki program o işi yapmaktadır.

ckParse.cs

```
using System;
using System.Globalization;
using System.Threading;

namespace Metotlar
{
    class ckParse
    {
        static void Main(string[] args)
        {
            string s = "12345,67TL";
```

```
            // Kesir ayracı ve para birimi içeren stringi
            // çözümleyip asıl sayıya dönüştürür
            float flt = float.Parse(s,
              NumberStyles.AllowDecimalPoint |
NumberStyles.AllowCurrencySymbol);
            Console.WriteLine(flt); // Çıktı : 12345,67
        }
    }
}
```

`CultureInfo` sınıfı

Farklı kültürlerde kesirli sayıların yazılışları farklıdır. Örneğin, ABD ve bazı avrupa ülkeleri kesirli sayı yazarken, ondalık kesir ayracı olarak nokta (.) ve yüzlükleri gruplamak için virgül (,) kullanır. Türkiye ve başka bazı ülkeler bunun tam tersini yaparlar; ondalık kesir ayracı olarak virgül (,) yüzlükleri gruplamak için nokta (.) kullanır. Dolayısıyla, bu farklı kültürlerde, örneğin 123.456 sayısı çok farklı anlamlara sahiptir. Bu nedenle, birçok bilgisayar programının yaptığı gibi, C# dili de bu ayrımı yapacak yeteneğe sahiptir. Programcının hangi kültüre göre sayıyı yazdırmak istediğini doğru sözdizimiyle ifade etmesi yeterlidir. Bunun için `CultureInfo` sınıfı kullanılır.

Aşağıdaki program `s="123,456"` biçimindeki bir stringi ABD sistemine göre tamsayıya çevirmek istemektedir. Ancak derleyici string içindeki (,) ün hangi sisteme göre yazıldığına karar veremez; verirse istenmedik yanlışlar doğar. O nedenle, hata iletisi verir.

Parse02.cs

```
using System;
using System.Globalization;

namespace Methods
{
    class Parsing
    {
        static void Main(string[] args)
        {
            CultureInfo vCultureInfo = new CultureInfo("en-US");
            string s = "123,456";
            int n = int.Parse(s, vCultureInfo);
            Console.WriteLine(n);
            // System.Format hatası.
        }
    }
}
```

Hata iletisi:

```
Unhandled Exception: System.FormatException: Input string was not in a correct format.
  at System.Number.StringToNumber(String str, NumberStyles options, NumberBuffer& number, NumberFormatInfo info,
Boolean parseDecimal)
  at System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInfo info)
  at Methods.Parsing.Main(String[] args) in C:\vsProjects\Metotlar\Metotlar\Parse02.cs:line 14
```

stringi sayıya çevirirken, programcının string içindeki virgülün (,) yüzlükleri ayıran ayraç olduğunu özenle belirtmesi gerekir. Bunun için

```
int n = int.Parse(s, vCultureInfo);
```

deyiminin yerine

```
int n = int.Parse(s, NumberStyles.AllowThousands, vCultureInfo);
```

deyimini yazmak yetecektir. Deneyerek sonucu görünüz.

Alıştırmalar

Aşağıdaki iki program, farklı stringleri çözümleyip sayıya dönüştürmektedir. Programı satır satır inceleyip, deyimlerin yaptığı işi anlayınız.

ParseMetodu

```
using System;
using System.Text;
using System.Globalization;

public sealed class ParseMetodu
{
    static void Main()
    {
        // stringi hex imiş gibi çözümle, decimal olarak yaz
        String num = "b";
        int val = int.Parse(num, NumberStyles.HexNumber);
        Console.WriteLine("{0} hex = {1}  decimal.", num, val);

        // işaret öntakısını algılayarak ama tekrarlayan
        //boşlukları ihmal ederek stringi çözümle
        num = "    -72   ";
        val = int.Parse(num, NumberStyles.AllowLeadingSign |
            NumberStyles.AllowLeadingWhite |
NumberStyles.AllowTrailingWhite);
        Console.WriteLine("'{0}' çözümlenince '{1}' oldu.", num, val);

        // parantezleri algılayarak ama tekrarlayan boşlukları
        //ihmal ederk stringi çözümle
        num = "    (86)   ";
        val = int.Parse(num, NumberStyles.AllowParentheses |
NumberStyles.AllowLeadingSign | NumberStyles.AllowLeadingWhite |
NumberStyles.AllowTrailingWhite);
        Console.WriteLine("'{0}' çözümlenince '{1}' oldu.", num, val);
    }
}
```

Çıktı

```
b hex = 11  decimal.
'   -72   ' çözümlenince '-72' oldu.
```

```
'   (86)   ' çözümlenince '-86' oldu.
```

```
using System;
using System.Globalization;
using System.Threading;

namespace Metotlar
{
    class DoubleParse
    {
        static void Main(string[] args)
        {
            // en-US kültür threadini etkin yap.
            Thread.CurrentThread.CurrentCulture =
CultureInfo.CreateSpecificCulture("en-US");

            string value;
            NumberStyles styles;

            // Yalnızca AllowExponent flagı ile üstel notasyonu çözümle
            value = "-5.072E+02";
            styles = NumberStyles.AllowExponent;
            ShowNumericValue(value, styles);

            // AllowExponent flagı ile Number flagı
            // kullanarak üstel notasyonu çözümle
            styles = NumberStyles.AllowExponent | NumberStyles.Number;
            ShowNumericValue(value, styles);

            // $ simgesi önünde ve arkasında tekrarlanan
            // boşluklar olan stringi çözümle
            value = " $ 3,841.7561  ";
            styles = NumberStyles.Number |
NumberStyles.AllowCurrencySymbol;
            ShowNumericValue(value, styles);

            // Binlik gruplarına ayrılmış ve kesir ayracı olan stringi
çözümle
            value = "(8,245.73)";
            styles = NumberStyles.AllowParentheses |
NumberStyles.AllowTrailingSign |
                     NumberStyles.Float;
            ShowNumericValue(value, styles);

            styles = NumberStyles.AllowParentheses |
NumberStyles.AllowTrailingSign |
                     NumberStyles.Float | NumberStyles.AllowThousands;
            ShowNumericValue(value, styles);
        }


        private static void ShowNumericValue(string value, NumberStyles
styles)
        {
            double number;
            try
            {
```

```
                number = Double.Parse(value, styles);
                Console.WriteLine("Converted '{0}' using {1} to {2}.",
                                  value, styles.ToString(), number);
            }
            catch (FormatException)
            {
                Console.WriteLine("Unable to parse '{0}' with styles
{1}.",
                                  value, styles.ToString());
            }
            Console.WriteLine();
        }
    }
}
```

Çıktı

```
/Unable to parse '-5.072E+02' with styles AllowExponent.
Converted '-5.072E+02' using AllowTrailingSign, AllowThousands, Float to -507.2.
Converted ' $ 3,841.7561  ' using Number, AllowCurrencySymbol to 3841.7561.
Unable to parse '(8,245.73)' with styles AllowTrailingSign, AllowParentheses, Float.
Converted '(8,245.73)' using AllowTrailingSign, AllowParentheses, AllowThousands
, Float to -8245.73.
```

Sayı Stilleri Listesi (Number Styles) :

C# Sayı Stili	Açıklama	Örnek
AllowCurrencySymbol	String para birimi simgesi içerebilir	"15000TL"
AllowDecimalPoint	String bir tek kesir ayracı içerebilir	"15000,00"
AllowExponent	String üstel biçimde olabilir	"1.5E3"
AllowHexSpecifier	String yalnızca hex biçimindedir	"5DC"
AllowLeadingSign	Stringin önünde + veya – işareti olabilir	"-15000"
AllowLeadingWhite	Stringin önünde boş karakter ya da kontrol karakterleri olabilir (Unicode characters 9 to 13).	" 15000"
AllowParentheses	String () içindeyse, sayıyı negatif sayı yap	"(15000)"
AllowThousands	String binlik gruplara ayrılabilir	"1.5000"
AllowTrailingSign	Stringin sonunda + veya – simgesi olabilir	"15000-"
AllowTrailingWhite	String tekrarlanan boş karakter içerebilir	"15000 "

Any	Yukarıdaki "Allow" stillerinin karması olabilir (hexedecimal hariç)	"1.5E3-"
Currency	Hex ve üstel dışındaki stringler, yukarıdakilerin bileşimi biçiminde olabilir	"(£1,5000.00)"
Float	String AllowDecimalPoint, AllowExponent, AllowLeadingSign, AllowLeadingWhite ve AllowTrailingWhite bileşimi olabilir	"-1,5000.00"
HexNumber	String AllowHexSpecifier, AllowLeadingWhite ve AllowTrailingWhite bileşimi olabilir	" 5DC "
Integer	String AllowLeadingSign, AllowLeadingWhite ve AllowTrailingWhite bileşimi olabilir	" -15000"
None	String hiç bir biçem içermiyor olabilir	"15000"
Number	String AllowDecimalPoint, AllowLeadingSign, AllowLeadingWhite, AllowThousands, AllowTrailingSign ve AllowTrailingWhite bileşimi olabilir.	" -1,5000.00

Kültürler ve Bölgeler

Farklı ülke ve kültürlerde sayıların yazılışı, tarih'in yazılışı, alfabetik sıralamanın yapılışı gibi eylemler birbirlerinden farklıdır. Hatta bazı alfabelerde yazı sağdan sola doğru yazılır. Farklı ülkeler farklı takvimler kullanır. Örneğin, `GregorianCalendar`, `HebrewCalendar`, `JapaneseCalendar`. C# dili farklı kültürlerin 'locale' (yöresel veya yerel) denilen farklı gereksemelerine yanıt verebilme yeteneğine sahiptir. Bu yeteneğini, `System.Globalization` aduzayı (namespace) ile yapmaktadır. Bu aduzayı içinde, bütün yörelerin isteklerine yanıt verebilecek sınıflar ve sınıflar içinde özelikler ve metotlar vardır. Çok geniş bir alanı dolduran bu konuyu sistematik incelemeye gitmeyeceğiz. Örneklerle, tarih ve sayı yazdırmada yöresel kullanışlara nasıl geçilebildiğini anlatmakla yetineceğiz. Konuyu sistematik incelemek isteyenler msdn web sitesinde .NET kütüphanesine bakabilirler.

Namespace `System.Globalization`

Dünyamız farklı bölgelere ve farklı kültürlere ayrılmıştır. `System.Globalization` namespace içinde bu farklılıkları gözetecek yaklaşık 30 sınıf vardır. Programcı gerekseme duyduğunda onları msdn web sitesinden görebilir. Konuyu çok dağıtmamak için bu aduzayının sistematik incelemesine girmeyeceğiz. Ancak, şimdiye kadar yaptığımız öneklere ek olarak, bundan sonraki bölümde, çeşitli kültürlere göre tarih yazdırma yöntemlerini görecek ve yeterince örnek yapacağız.

Class CultureInfo

`System.Globalization` namespace içinde bir sınıftır. Yöresel (locale) diye adlandırılan kültüre dayalı bilgileri tutar. Kültürün adı, yazı sistemi, kullanılan takvim, tarih'in yazılış biçemi, string'lerin sıralanma (sorting) yöntemi gibi şeyler tutulan bilgiler arasındadır. Dil, alt-dil grubu, ülke/bölge, takvim gibi kültüre dayalı biçemleri o yöre halklarının alıştığı gibi kullanmak elbette çok önemlidir. Farklı ülke ve kültürlerde kesirli sayıların yazılışı, tarihin yazılışı, alfabetik sıralamanın yapılışı gibi eylemler birbirlerinden farklıdır. Bazı alfabelerde yazı sağdan sola doğru yazılır. Örnekse, Arapça'nın 20 ye yakın ülke/bölge farklılığı, İngilizce'nin 10 dan fazla ülke/bölge farklılığı vardır. `CultureInfo` sınıfı bize bu bilgileri verir.

Aşağıdaki kısaltılmış tablo bu konuda bir fikir verebilir. Tablonun tamamı için msdn web sitesine bakılabilir.

Kısa Adı	Kodu	Dil
" " Boş string	0x007F	Invariant culture
ar	0x0001	Arabic
ar-EG	0x0C01	Arabic (Egypt)
en	0x0009	English
en-GB	0x0809	English (United Kingdom)
en-US	0x0409	English (United States)
es	0x000A	Spanish
es-VE	0x200A	Spanish-Venezuela
de	0x0007	German
de-AT	0x0C07	German-Austria
tr	0x001F	Turkish
tr-TR	0x041F	Turkish (Turkey)

CultureAndRegionInfoBuilder Sınıfı

Bir kültüre dayalı alt kültürleri belirler, yenilerini tanımlar. Gereksiz yer kaplayacakları için, bütün alt-kültürlere ait bilgiler çalışan her işletim sisteminde faal değildir. Kullanılmak istenen alt kültür sisteme yüklenebilir.

Thread Sınıfı

`System.Threading` aduzayı içinde bir sınıftır. Thread yaratır ve kontrol eder, önceliğini belirler, statüsünü tutar.

.NET birden çok işi eşzamanlı yapabilir. Farklı işlerin birbirleriyle karışmadan farklı yollardan işleme girdiğini düşününüz. Bu yollara `thread` denir.