Método Parse

El método "parse", entre sus múltiples funciones, nos permite convertir caracteres númericos a datos numéricos, es decir, convertir un número almacenado como String a un dato del tipo int, double u otro según se requiera.

Esto es muy útil a la hora de ingresar números desde el teclado, ya que Java toma los datos capturados por el teclado como cadenas de caracteres con las que no se pueden realizar operaciones matemáticas, y en muchas ocaciones necesitamos leer números para realizar operaciones con ellos.

■ Convertir un dato numérico almacenado en un String usando el método "parse" se hace de la siguiente forma:

Conversión de datos en C# (I): ToString y Parse

Cada vez que hacemos una aplicación una de nuestras principales labores es darle formato a los datos, para que la aplicación saque las cosas por pantalla como lo debe hacer. Porque una aplicación tiene que ser bonita y tener una interfaz de usuario agradable, y esto, muchas veces, puede resultar un problema para el programador. C# nos provee muchas técnicas para evitarnos un dolor de cabeza, muy sencillas de usar.

Los métodos ToString() y Parse()

Seguramente los conozcas y los usases en el pasado. Veamos:

```
bool booleanvariable = true;
string stringvariable = booleanvariable.ToString();
```

El método ToString simplemente convierte una variable de cualquier tipo en un objeto del tipo String. En el ejemplo anterior asignamos el valor true a la variable booleanvariable y inmediatamente después llamamos al método ToString sobre esa variable. Lo que obtenemos es un string que contiene la cadena “True”.

```
string stringvar = "false";
bool boolvar = bool.Parse(stringvar);
```

Utilizamos el método Parse, el String stringvar que contiene false será parseado a la variable booleana boolvar. Sin embargo Parse es unsafe, y puede derivar en una excepción dependiendo del argumento que le entre. A modo de ejemplo, el siguiente código produciría una excepción FormatException:

```
string stringvar = "fasel";
bool boolvar = bool.Parse(stringvar);
```

Podemos solventarlo muy fácilmente utilizando el método Tryparse que nos devolverá un valor booleano en función de si la conversión fue exitosa:

```
string stringvar = "fasel";

bool boolvar = false;
if (!bool.TryParse(stringvar, out boolvar))
{
   boolvar = false; // Valor por defecto
}
```

Sin embargo, aunque todo parece ahora perfecto, nos encontramos con un pequeño problema. El problema es que no podemos distinguir mediante la variable si la conversión ha sido correcta o incorrecta. Para ello usaremos un nullable bool, que nos permitirá no asignar la variable ni a true ni a false, si no a null.

```
string stringvar = "fasel";
```

```
bool? boolvar = null;
bool result;
if (!bool.TryParse(stringvar, out result))
{
   boolvar = null; // Valor por defecto
}
else
{
   boolvar = (bool)result;
}
```

El uso de los nullable es muy controvertido, hasta el punto que mucha gente desaconseja su uso. En este ejemplo, si la conversión no fue buena (que no lo fue) boolvar será null, mientras que si fue buena, boolvartomará el valor de la conversión. Hay que tener en cuenta que esto hace el código más complejo, ya que a partir de ahora tendremos que preocuparnos de si boolvar contiene null o por el contrario está inicializada, y esto no es agradable.

```
bool x = (bool)boolvar;
bool y = boolvar.Value;
```

En estos dos casos, el compilador deja pasar como bueno el código, sin embargo, si boolvar, siendo unnullable int, contiene null, se producirá una Excepción InvalidOperationException en tiempo de ejecución. Si aún sabiendo la complejidad que pueden aportar los nullables al mantenimiento del código queremos seguir usándolos, una buena manera es haciendo las asignaciones de la siguiente manera:

```
bool x = boolvar ?? false;
```

En este caso la variable x cogería false si boolvar contiene null, sería algo así como el valor por defecto. Si no es null cogería el valor de boolvar. Pero la pregunta del millón es ¿Nos acordaremos de hacerlo cada vez que tengamos que trabajar con el valor de boolvar?

Conversión de Datos en C# (II): ToString y Format Providers

En este capítulo vamos a jugar un poco con el método ToString y la interfaz IFormattable, que nos proporciona .NET para poder dar formato a nuestros datos con facilidad. Antes de nada, recordemos que tenemos que usarSystem.Globalization para poder trabajar con cualquier clase que implemente Iformattable. Veamos:

```
using System;
using System.Globalization;

namespace ConsoleApplication3
{
   class Program
   {
      static void Main(string[] args)
      {
         // Nuevo objeto de la clase NumberFormatInfo, que hereda de IFormatProvider
         NumberFormatInfo formatonumero = new NumberFormatInfo();

         // Ajustamos algunas propiedades de formatonumero
         formatonumero.CurrencySymbol = "$";

         // Instanciamos un int y tiramos un ToString por consola
         int dinero = 5000;
         Console.WriteLine(dinero.ToString("C", formatonumero));
```

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

En este caso queremos imprimir un valor monetario en dólares por pantalla, y para ello hemos utilizado el método ToString al cual le entran dos argumentos. El primero de ellos, string format, en este caso es C, que significa que queremos que nos haga la conversión como si de una moneda se tratase. El segundo parámetro es un objeto heredado del tipo IFormatProvider, y este objeto establece que el CurrencySymbol es el símbolo del dólar. Así que por pantalla obtendremos:

```
$5,000.00
```

Culturas y formato

Normalmente no necesitamos crear nosotros nuestro propio objeto ya que la clase CultureInfo suele ser suficiente para cualquier propósito general. En este caso, vamos a usar ToString y CultureInfo para formatear un número según las diferentes culturas que rondan por este nuestro mundo. Vamos allá:

```
using System;
using System.Globalization;

namespace ConsoleApplication3
{
    class Program
    {
        static void Main(string[] args)
        {
            // Compatibilidad ASCII para el caracter €
            Console.OutputEncoding = System.Text.Encoding.UTF8;

            // Instanciamos un int
            int dinero = 5000;

            // Creamos un objeto de la clase CultureInfo con la cultura española y sacamos por consola
            CultureInfo spainCulture = CultureInfo.GetCultureInfo("es-ES");
            Console.WriteLine("Cultura española:");
            Console.WriteLine(dinero.ToString("C", spainCulture));

            // Creamos un objeto de la clase CultureInfo con la cultura estadouidense y sacamos por consola
            CultureInfo usaCulture = CultureInfo.GetCultureInfo("en-US");
            Console.WriteLine("Cultura estadounidense:");
            Console.WriteLine(dinero.ToString("C", usaCulture));

            // Creamos un objeto de la clase CultureInfo con la cultura mexicana y sacamos por consola
            CultureInfo venezuelaCulture = CultureInfo.GetCultureInfo("es-VE");
            Console.WriteLine("Cultura venezolana:");
            Console.WriteLine(dinero.ToString("C", venezuelaCulture));

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

Lo cual producirá la siguiente salida por consola:

```
Cultura española:
5.000,00 €
Cultura estadounidense:
$5,000.00
Cultura venezolana:
Bs.F. 5.000,00
```

En primer lugar, remarcar para los despistados, que no estamos haciendo conversión matemática de divisas, si no conversión de cadenas y formato a una salida determinada. Así vemos como cada país lleva su moneda actual, además de darnos cuenta de cómo en el caso de Venezuela y España usamos la coma para el decimal y los puntos para el separador de miles mientras que en estados unidos usan la coma para el separador de miles y el punto para el decimal.

Culturas en fechas

Podemos usar exactamente el mismo ejemplo anterior para mostrar una fecha según la cultura:

```
using System;
using System.Globalization;

namespace ConsoleApplication3
{
   class Program
   {
      static void Main(string[] args)
      {

         DateTime aDateTime = new DateTime(1992, 01, 31, 16, 0, 0); // 31 de Enero de 1992 a las
16:00:00

         // Creamos un objeto de la clase CultureInfo con la cultura española y sacamos por consola
         CultureInfo spainCulture = CultureInfo.GetCultureInfo("es-ES");
         Console.WriteLine("Cultura española:");
         Console.WriteLine(aDateTime.ToString(spainCulture));

         // Creamos un objeto de la clase CultureInfo con la cultura estadouidense y sacamos por consola
         CultureInfo usaCulture = CultureInfo.GetCultureInfo("en-US");
         Console.WriteLine("Cultura estadounidense:");
         Console.WriteLine(aDateTime.ToString(usaCulture));

         // Creamos un objeto de la clase CultureInfo con la cultura mexicana y sacamos por consola
         CultureInfo venezuelaCulture = CultureInfo.GetCultureInfo("es-VE");
         Console.WriteLine("Cultura venezolana:");
         Console.WriteLine(aDateTime.ToString(venezuelaCulture));

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

Con la salida por consola siguiente:

```
Cultura española:
31/01/1992 16:00:00
Cultura estadounidense:
1/31/1992 4:00:00 PM
Cultura venezolana:
31-01-1992 04:00:00 p.m.
```

Pero lo más normal si queremos desarrollar una aplicación es que se muestre en la cultura que tenga el usuario predefinida en el sistema, es decir, en la cultura configurada en el dispositivo. Para ello usaremosCultureInfo.CurrentCulture, que representará la cultura del usuario:

```
using System;
using System.Globalization;

namespace ConsoleApplication3
{
   class Program
   {
      static void Main(string[] args)
      {


         DateTime aDateTime = new DateTime(1992, 01, 31, 16, 0, 0); // 31 de Enero de 1992 a las
16:00:00

         // Creamos un objeto de la clase CultureInfo con la cultura actual del sistema y sacamos por
consola
         CultureInfo currentCulture = CultureInfo.CurrentCulture;
         Console.WriteLine(string.Format("Cultura {0}:", currentCulture.DisplayName));
         Console.WriteLine(aDateTime.ToString(currentCulture));

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

En mi caso, esto produce la salida siguiente:

```
Cultura Español (España):
31/01/1992 16:00:00
```

En vuestro caso, si tenéis el sistema en otra región producirá la salida correspondiente a esa región.
Por último vamos a ver cono formatear fechas y horas usando ToString para obtener solamente un dato determinado. Para ello usaremos culture.DateTimeFormat, que contiene strings con el formateo de fechas para la cultura del objeto. Mejor con un ejemplo:

```
using System;
using System.Globalization;

namespace ConsoleApplication3
{
   class Program
   {
      static void Main(string[] args)
      {

         DateTime aDateTime = new DateTime(1992, 01, 31, 16, 0, 0); // 31 de Enero de 1992 a las
16:00:00

         // Creamos un objeto de la clase CultureInfo con la cultura actual del sistema y sacamos por
consola
         CultureInfo currentCulture = CultureInfo.CurrentCulture;
         Console.WriteLine(string.Format("Cultura {0} (Solo fecha corta):", currentCulture.DisplayName));
         Console.WriteLine(aDateTime.ToString(currentCulture.DateTimeFormat.ShortDatePattern,
currentCulture));
         Console.WriteLine();

         currentCulture = CultureInfo.GetCultureInfo("en-US");
```

```
            Console.WriteLine(string.Format("Cultura {0} (Solo fecha corta):", currentCulture.DisplayName));
            Console.WriteLine(aDateTime.ToString(currentCulture.DateTimeFormat.ShortDatePattern,
currentCulture));
            Console.WriteLine();
            currentCulture = CultureInfo.CurrentCulture;
            Console.WriteLine(string.Format("Cultura {0} (Solo hora):", currentCulture.DisplayName));
            Console.WriteLine(aDateTime.ToString(currentCulture.DateTimeFormat.ShortTimePattern,
currentCulture));
            Console.WriteLine();

            currentCulture = CultureInfo.GetCultureInfo("en-US");
            Console.WriteLine(string.Format("Cultura {0} (Solo hora):", currentCulture.DisplayName));
            Console.WriteLine(aDateTime.ToString(currentCulture.DateTimeFormat.ShortTimePattern,
currentCulture));
            Console.WriteLine();

            currentCulture = CultureInfo.CurrentCulture;
            Console.WriteLine(string.Format("Cultura {0} (Solo día y mes largos):",
currentCulture.DisplayName));
            Console.WriteLine(aDateTime.ToString(currentCulture.DateTimeFormat.MonthDayPattern,
currentCulture));
            Console.WriteLine();

            currentCulture = CultureInfo.GetCultureInfo("en-US");
            Console.WriteLine(string.Format("Cultura {0} (Solo día y mes largos):",
currentCulture.DisplayName));
            Console.WriteLine(aDateTime.ToString(currentCulture.DateTimeFormat.MonthDayPattern,
currentCulture));
            Console.WriteLine();

            currentCulture = CultureInfo.CurrentCulture;
            Console.WriteLine(string.Format("Cultura {0} (Fecha larga):", currentCulture.DisplayName));
            Console.WriteLine(aDateTime.ToString(currentCulture.DateTimeFormat.LongDatePattern,
currentCulture));
            Console.WriteLine();

            currentCulture = CultureInfo.GetCultureInfo("en-US");
            Console.WriteLine(string.Format("Cultura {0} (Fecha larga):", currentCulture.DisplayName));
            Console.WriteLine(aDateTime.ToString(currentCulture.DateTimeFormat.LongDatePattern,
currentCulture));
            Console.WriteLine();

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

Esto producirá la siguiente salida, que variará dependiendo de vuestra cultura:

```
Cultura Español (España) (Solo fecha corta):
31/01/1992

Cultura Inglés (Estados Unidos) (Solo fecha corta):
1/31/1992

Cultura Español (España) (Solo hora):
16:00

Cultura Inglés (Estados Unidos) (Solo hora):
4:00 PM
Cultura Español (España) (Solo día y mes largos):
```

31 de enero

Cultura Inglés (Estados Unidos) (Solo día y mes largos):
January 31

Cultura Español (España) (Fecha larga):
viernes, 31 de enero de 1992

Cultura Inglés (Estados Unidos) (Fecha larga):
Friday, January 31, 1992