INAP | CAMPUS VIRTUAL

Programación II

Ministerio de Modernización
Presidencia de la Nación

Módulo II

Programación II

Módulo II

Módulo II:

- **Fecha y hora. Temporización.**
- **Lectura de directorios.**
- **El entorno. Llamadas al sistema.**
- **Datos sobre el entorno.**

Fecha y hora. Temporización

Desde C#, tenemos la posibilidad de manejar **fechas y horas** con facilidad. Para ello, tenemos el tipo de datos DateTime. Por ejemplo, podemos hallar la fecha (y hora) actual con:

```
DateTime fecha = DateTime.Now;
```

Dentro de ese tipo de datos DateTime, tenemos las herramientas para saber el día (Day), el mes (Month) o el año (Year) de una fecha, entre otros. También podemos calcular otras fechas sumando a la actual una cierta cantidad de segundos (AddSeconds), días (AddDays), etc. Un ejemplo básico de su uso sería:

```
using System;

public class Ejemplo_12_01a
{
    public static void Main()
    {
        DateTime fecha = DateTime.Now;
        Console.WriteLine("Hoy es {0} del mes {1} de {2}",
            fecha.Day, fecha.Month, fecha.Year);
        DateTime manyana = fecha.AddDays(1);
        Console.WriteLine("Mañana será {0}",
            manyana.Day);
    }
}
```

Algunos de las propiedades más útiles de este tipo de datos son: Now (fecha y hora actual de este equipo), Today (fecha actual); Day (día del mes), Month (número de mes), Year (año); Hour (hora), Minute (minutos), Second (segundos), Millisecond (milisegundos); DayOfWeek (día de la semana: su nombre en inglés, que se puede convertir en un número del 0-domingo- al 6-sábado- si se fuerza su tipo a entero, anteponiéndole "(int)"); DayOfYear (día del año).

Para calcular nuevas fechas, podemos usar métodos que generan un nuevo DateTime, como: AddDays (que aparece en el ejemplo anterior), AddHours, AddMilliseconds, AddMinutes, AddMonths, AddSeconds, AddHours, o bien un Add más genérico (para sumar una fecha a otra) y un Subtract también genérico (para restar una fecha de otra).

Cuando restamos dos fechas, obtenemos un dato de tipo "intervalo de tiempo" (TimeSpan), del que podemos saber detalles como la cantidad de días (sin decimales, Days, o con decimales, TotalDays), como se ve en este ejemplo:

```
using System;

public class Ejemplo_12_01b
{
    public static void Main()
    {
        DateTime fechaActual = DateTime.Now;
        DateTime fechaNacimiento = new DateTime(1990, 9, 18);

        TimeSpan diferencia =
            fechaActual.Subtract(fechaNacimiento) ;

        Console.WriteLine("Han pasado {0} días",
            diferencia.Days);

        Console.WriteLine("Si lo quieres con decimales: {0} días",
            diferencia.TotalDays);

        Console.WriteLine("Y si quieres más detalles: {0}",
            diferencia);
    }
}
```

El resultado de este programa sería algo como

```
Han pasado 8886 días
Si lo quieres con decimales: 8886,41919806277 días
Y si quieres más detalles: 8886.10:03:38.7126236
```

También podemos hacer una **pausa** en la ejecución: Si necesitamos que nuestro programa se detenga una cierta cantidad de tiempo, no hace falta que usemos un "while" que compruebe la hora continuamente, sino que podemos "bloquear" (Sleep) el subproceso (o hilo, "Thread") que representa nuestro programa una cierta cantidad de milésimas de segundo con: Thread.Sleep(5000);

Este método pertenece a System.Threading, que deberíamos incluir en nuestro apartado de"using", o bien usar la llamada completa:

```
using System;
using System.Threading;

public class Ejemplo_12_01c
{
    public static void Main()
    {
        DateTime fecha = DateTime.Now;
        Console.WriteLine("Son las {0}:{1}:{2}",
            fecha.Hour, fecha.Minute, fecha.Second);
        Console.WriteLine("Vamos a esperar 3 segundos...");
        Thread.Sleep(3000);
        fecha = DateTime.Now;
        Console.WriteLine("Ahora son las {0}:{1}:{2}",
            fecha.Hour, fecha.Minute, fecha.Second);
    }
}
```

Lectura de directorios

Si queremos analizar el contenido de un directorio, podemos emplear las clases Directory y DirectoryInfo.

La clase Directory contiene métodos para crear un directorio (CreateDirectory), borrarlo (Delete), moverlo (Move), comprobar si existe (Exists), etc. Por ejemplo, podríamos crear un directorio con:

```
using System.IO;

public class Ejemplo_12_03a
{
    public static void Main()
    {
        string miDirectorio = @"d:\datos";
        if (!Directory.Exists(miDirectorio))
            Directory.CreateDirectory(miDirectorio);
    }
}
```

(la clase Directory está declarada en el espacio de nombres System.IO, por lo que deberemos añadirlo entre los "using" de nuestro programa).

También tenemos un método "GetFiles" que nos permite obtener la lista de ficheros que contiene un directorio. Así, podríamos listar todo el contenido de un directorio con:

```
using System;
using System.IO;

public class Ejemplo_12_03b
{
    public static void Main()
    {
        string miDirectorio = @"c:\";
        string[] listaFicheros;

        listaFicheros = Directory.GetFiles(miDirectorio);
        foreach(string fichero in listaFicheros)
            Console.WriteLine(fichero);
    }
}
```

Se puede añadir un segundo parámetro a "GetFiles", que sería el patrón que deben seguir los nombres de los ficheros que buscamos, como "*.txt". Por ejemplo, podríamos saber todos los fuentes de C# ("*.cs") de la carpeta actual (".") con:

```
using System;
using System.IO;

public class Ejemplo_12_03c
{
    public static void Main()
    {
        string[] listaFicheros = Directory.GetFiles(".", "*.cs");
        foreach(string fichero in listaFicheros)
            Console.WriteLine(fichero);
    }
}
```

Si necesitamos más detalles, la clase DirectoryInfo permite obtener información sobre fechas de creación, modificación y acceso, y, de forma análoga, FileInfo nos permite conseguir información similar sobre un fichero. Podríamos usar estas dos clases para ampliar el ejemplo anterior, y que no sólo muestre el nombre de cada fichero, sino otros detalles adicionales como el tamaño y la fecha de creación:

```
using System;
using System.IO;

public class Ejemplo_12_03d
{
    public static void Main()
    {
        string miDirectorio = @"c:\";
        DirectoryInfo dir = new DirectoryInfo(miDirectorio);
        FileInfo[] infoFicheros = dir.GetFiles();
        foreach (FileInfo infoUnFich in infoFicheros)
        {
            Console.WriteLine("{0}, de tamaño {1}, creado {2}",
                infoUnFich.Name,
                infoUnFich.Length,
                infoUnFich.CreationTime);
        }

    }
}
```

que escribiría cosas como

```
hiberfil.sys, de tamaño 6775930880, creado 15/07/2013 17:48:07
```

El entorno. Llamadas al sistema

Si hay algo que no sepamos o podamos hacer, pero que alguna utilidad del sistema operativo sí es capaz de hacer por nosotros, podemos hacer que ella trabaje por nosotros. La forma de llamar a otras órdenes del sistema operativo (incluso programas externos de casi cualquier tipo) es creando un nuevo proceso con "Process.Start". Por ejemplo, podríamos lanzar el bloc de notas de Windows con:

```
Process proc = Process.Start("notepad.exe");
```

En los actuales sistemas operativos multitarea se da por sentado que no es necesario esperar a que termine otra la tarea, sino que nuestro programa puede proseguir. Si aun así, queremos esperar a que se complete la otra tarea, lo conseguiríamos con "WaitForExit", añadiendo esta segunda línea:

```
proc.WaitForExit();
```

Un programa completo que lanzara el bloc de notas y que esperase a que se cerrase podría ser:

```
using System;
using System.Diagnostics;

public class Ejemplo_12_04a
{
    public static void Main()
    {
        Console.WriteLine("Lanzando el bloc de notas...");

        Process proc = Process.Start("notepad.exe");
        Console.WriteLine("Esperando a que se cierre...");
        proc.WaitForExit();
        Console.WriteLine("Terminado!");
    }
}
```

Datos sobre "el entorno"

La clase "Environment" nos sirve para acceder a información sobre el sistema: unidades de disco disponibles, directorio actual, versión del sistema operativo y de la plataforma .Net, nombre de usuario y máquina, carácter o caracteres que se usan para avanzar de línea, etc:

```
using System;
using System.Diagnostics;

public class Ejemplo_12_05a
{
    public static void Main()
    {
        string avanceLinea = Environment.NewLine;
        Console.WriteLine("Directorio actual: {0}",
            Environment.CurrentDirectory);
        Console.WriteLine("Nombre de la máquina: {0}",
            Environment.MachineName);
        Console.WriteLine("Nombre de usuario: {0}", Environment.UserName);
        Console.WriteLine("Dominio: {0}", Environment.UserDomainName);
        Console.WriteLine("Código de salida del programa anterior: {0}",
            Environment.ExitCode);
        Console.WriteLine("Linea de comandos: {0}", Environment.CommandLine);
        Console.WriteLine("Versión del S.O.: {0}",
            System.Convert.ToString(Environment.OSVersion));
        Console.WriteLine("Version de .Net: {0}",
            Environment.Version.ToString());
        String[] discos = Environment.GetLogicalDrives();
            Console.WriteLine("Unidades lógicas: {0}",
                String.Join(", ", discos));
            Console.WriteLine("Carpeta de sistema: {0}",
                Environment.GetFolderPath(Environment.SpecialFolder.System));
    }
}
```