## 13.3. Enumeraciones

Cuando tenemos varias constantes, cuyos valores son números enteros, hasta ahora estamos dando los valores uno por uno, así:

```
const int LUNES = 0, MARTES = 1,
  MIERCOLES = 2, JUEVES = 3,
  VIERNES = 4, SABADO = 5,
  DOMINGO = 6;
```

Hay una forma alternativa de hacerlo, especialmente útil si son números enteros consecutivos. Se trata de **enumerarlos**:

```
  enum diasSemana { LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO,
      DOMINGO };
```

(Al igual que las constantes de cualquier otro tipo, se puede escribir en mayúsculas para recordar "de un vistazo" que son constantes, no variables)

La primera constante valdrá 0, y las demás irán aumentando de una en una, de modo que en nuestro caso valen:

```
   LUNES = 0, MARTES = 1, MIERCOLES = 2, JUEVES = 3, VIERNES = 4,
   SABADO = 5, DOMINGO = 6
```

Si queremos que los valores no sean exactamente estos, podemos dar valor a cualquiera de las contantes, y las siguientes irán aumentando de uno en uno. Por ejemplo, si escribimos

```
    enum diasSemana { LUNES=1, MARTES, MIERCOLES, JUEVES=6, VIERNES,
      SABADO=10, DOMINGO };
```

Ahora sus valores son:

```
   LUNES = 1, MARTES = 2, MIERCOLES = 3, JUEVES = 6, VIERNES = 7,
   SABADO = 10, DOMINGO = 11
```

Un ejemplo básico podría ser

```
/*----------------------------*/
/*  Ejemplo en C#             */
/*  enum.cs                   */
/*                            */
/*  Ejemplo de enumeraciones  */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class enumeraciones
{

  enum diasSemana { LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO,
```

```
      DOMINGO };

  public static void Main()
  {

    Console.Write("En la enumeracion, el miércoles tiene el valor: {0} ",
      diasSemana.MIERCOLES);
    Console.WriteLine("que equivale a: {0}",
      (int) diasSemana.MIERCOLES);

    const int LUNES = 0, MARTES = 1, MIERCOLES = 2, JUEVES = 3,
      VIERNES = 4, SABADO = 5, DOMINGO = 6;

    Console.WriteLine("En las constantes, el miércoles tiene el valor: {0}",
      MIERCOLES);

  }
}
```

y su resultado será:

```
En la enumeracion, el miércoles tiene el valor: MIERCOLES que equivale a: 2
En las constantes, el miércoles tiene el valor: 2
```

Nosotros hemos usado enumeraciones muchas veces hasta ahora, sin saber realmente que lo estábamos haciendo. Por ejemplo, el modo de apertura de un fichero (FileMode) es una enumeración, por lo que escribimos FileMode.Open. También son enumeraciones los códigos de color de la consola (como ConsoleColor.Red) y las teclas de la consola (como ConsoleKey.Escape).

Nota: las enumeraciones existen también en otros lenguajes como C y C++, pero la sintaxis es ligeramente distinta: en C# es necesario indicar el nombre de la enumeración cada vez que se usen sus valores (como en diasSemana.MIERCOLES), mientras que en C se usa sólo el valor (MIERCOLES).

**Ejercicios propuestos**

- **(13.3.1)** Crea una versión de la agenda con colores (ejercicio 12.2.2) en la que las opciones sean parte de una enumeración.

## *13.4. Propiedades*

Hasta ahora estábamos siguiendo la política de que los atributos de una clase sean privados, y se acceda a ellos a través de métodos "get" (para leer su valor) y "set" (para cambiarlo). En el caso de C#, existe una forma alternativa de conseguir el mismo efecto, empleando las llamadas "propiedades", que tienen una forma abreviada de escribir sus métodos "get" y "set":

```
/*--------------------------*/
/*  Ejemplo en C#           */
/*  propiedades.cs          */
/*                          */
/*  Ejemplo de propiedades  */
/*                          */
```

```
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class EjemploPropiedades
{

  // ------------------------

  // Un atributo convencional, privado
  private int altura = 0;

  // Para ocultar detalles, leemos su valor con un "get"
  public int GetAltura()
  {
    return altura;
  }

  // Y lo fijamos con un "set"
  public  void SetAltura(int nuevoValor)
  {
    altura = nuevoValor;
  }


  // ------------------------

  // Otro atributo convencional, privado
  private int anchura = 0;

  // Oculto mediante una "propiedad"
  public int Anchura
  {
    get
    {
      return anchura;
    }

    set
    {
      anchura = value;
    }
  }


  // ------------------------

  // El "Main" de prueba
  public static void Main()
  {

    EjemploPropiedades ejemplo
      = new EjemploPropiedades();

    ejemplo.SetAltura(5);
    Console.WriteLine("La altura es {0}",
      ejemplo.GetAltura() );
```

```
    ejemplo.Anchura = 6;
    Console.WriteLine("La anchura es {0}",
      ejemplo.Anchura );

  }
}
```

Al igual que ocurría con las enumeraciones, ya hemos usado "propiedades" anteriormente, sin saberlo: la longitud ("Length") de una cadena, el tamaño ("Length") y la posición actual ("Position") en un fichero, el título ("Title") de una ventana en consola, etc.

Una curiosidad: si una propiedad tiene un "get", pero no un "set", será una propiedad de sólo lectura, no podremos hacer cosas como "Anchura = 4", porque el programa no compilaría. De igual modo, se podría crear una propiedad de sólo escritura, definiendo su "set" pero no su "get".

**Ejercicios propuestos**

- **(13.4.1)** Añade al ejercicio de los trabajadores (6.7.1) una propiedad "nombre", con sus correspondientes "get" y "set".

## *13.5. Parámetros de salida (out)*

Hemos hablado de dos tipos de parámetros de una función: parámetros por valor (que no se pueden modificar) y parámetros por referencia ("ref", que sí se pueden modificar). Un uso habitual de los parámetros por referencia es devolver más de un valor a la salida de una función. Para ese uso, en C# existe otra alternativa: los parámetros de salida. Se indican con la palabra "out" en vez de "ref", y no exigen que las variables tengan un valor inicial:

```
public void ResolverEcuacionSegundoGrado(
          float a, float b, float c,
          out float x1, out float x2)
```

**Ejercicios propuestos**

- **(13.5.1)** Crea una nueva versión del ejercicio que resuelve ecuaciones de segundo grado (5.9.2.2), usando parámetros "out".

## *13.6. Introducción a las expresiones regulares.*

Las "expresiones regulares" permiten hacer comparaciones mucho más abstractas que si se usa un simple "IndexOf". Por ejemplo, podemos comprobar con una orden breve si todos los caracteres de una cadena son numéricos, o si empieza por mayúscula y el resto son minúsculas, etc.

Vamos a ver solamente un ejemplo con un caso habitual: comprobar si una cadena es numérica, alfabética o alfanumérica. Las ideas básicas son: