Future Developer Week

Javier San Juan Cervera

Contenido

Día 5: Bucles. Programación Orientada a Objetos

- Comentarios en el código
- Bucles
- Principios de la Programación Orientada a Objetos
- Tutoriales

Comentarios en el código

Comentarios

- En ocasiones, resulta útil poner comentarios en el código, que son **ignorados por el compilador**, pero **sirven de indicación al programador** sobre qué hace un bloque de sentencias.
- Podríamos pensar, *“Pero si yo soy el programador de la aplicación, ¡ya sé lo que hace mi programa! ¿Por qué iba a necesitar poner comentarios que lo expliquen?”*. Porque nuestro código podría ser leído por **otros programadores**, o porque podríamos **volver a consultar nuestro programa** meses o años después de haberlo escrito, y haber olvidado cómo estaba construido (¡ocurre más a menudo de lo que parece!).

Comentarios

- Un comentario comienza con el símbolo //, y continúa **hasta el final de la línea**. Por ejemplo:

```
// Calculamos la distancia entre las coordenadas (x1,y1) y (x2,y2)
double dx = x2 - x1;
double dy = y2 - y1;
double distancia = Math.Sqrt(dx*dx + dy*dy);
```

- Para cualquiera que recuerde el Teorema de Pitágoras, este código es obvio, pero quien no lo recuerde podría tener problemas para entenderlo. El comentario deja claro qué es lo que hacen las sentencias.

Comentarios

- Si la finalidad de una sentencia es muy obvia, no pondremos ningún comentario. Por ejemplo, no haremos esto:

```
// Incrementamos en 1 el valor de la variable a
a = a + 1;
```

- Ya que es evidente lo que hace la sentencia con sólo mirar el código.

Comentarios

- También se utilizan mucho los comentarios para **eliminar temporalmente una sentencia**.
- Por ejemplo, supongamos que en un momento determinado queremos que deje de ejecutarse una determinada sentencia, pero en el futuro vamos a volver a incluirla.
- Comentar la línea nos permite volver a poner la sentencia en el futuro (eliminando el símbolo `//`) sin tener que volver a escribirla.

Comentarios

- Existen también otro tipo de comentarios que se pueden extender **varias líneas**. Si comenzamos el comentario con el símbolo `/*` en lugar de `//`, el comentario continuará hasta que aparezca el símbolo `*/`, incluso si éste está varias líneas más adelante:

```
/* Esto es un comentario
   de varias líneas */
```

Bucles

Bucles

- Ya hemos aprendido que si escribimos varias sentencias una detrás de otra, éstas se ejecutan en orden, y que si metemos un bloque de sentencias dentro de un `if` o un `else`, dichas sentencias se ejecutarán en función de una determinada condición.
- También podemos hacer que un bloque de sentencias **se ejecute varias veces**. Esto es posible gracias a los **bucles**.
- Existen varios tipos de bucles en C#: `while`, `do`, `for` y `foreach`.

While

- Vamos a recordar brevemente para qué servía `if`: Si la condición entre paréntesis se cumple, se ejecutan las sentencias del bloque.
- Cuando el `if` se ejecuta, **la comprobación se realiza una sola vez**: si la expresión es cierta, se ejecuta el bloque; si no, se ejecuta el bloque `else` (si es que existe). Después, continúa la ejecución del programa.

While

- El bucle `while` es similar, sólo que tras ejecutar el bloque de sentencias, se **vuelve a comprobar la condición**, y si sigue siendo cierta, **se vuelven a ejecutar las sentencias**.
- Esto **continúa** de esta forma **hasta que la condición pasa a ser falsa**.

While

- Veamos un ejemplo de bucle `while` que escribe los números enteros de 1 a 10:

```
int numero = 1;
while (numero <= 10)
{
    Console.WriteLine(numero);
    numero = numero + 1;
}
```

While

- Cuando escribimos un bucle, hay que asegurarse de que la condición va a ser falsa en algún momento, ya que si no el bucle **se ejecutará infinitamente**.
- ¿Por qué es incorrecto este código?

```
int numero = 1;
while (numero != 0)
{
    Console.WriteLine(numero);
    numero = numero + 1;
}
```

Do

- Debemos tener en cuenta que, si la condición del `while` no es cierta la primera vez, las sentencias del bucle **no se ejecutarán nunca**.
- Existe otro tipo de bucle, llamado `do`, que garantiza que el bucle se ejecute al menos una vez, **incluso si la primera vez la condición es falsa**.

Do

- Veamos cómo imprimir lo números enteros del 1 al 10 con un bucle `do`:

```
int numero = 1;
do
{
    Console.WriteLine(numero);
    numero = numero + 1;
} while (numero <= 10);
```

For

- Vamos a empezar viendo cómo escribir los números enteros de 1 a 10 con este tipo de bucle:

```
for (int numero = 1; numero <= 10; numero = numero + 1)
{
    Console.WriteLine(numero);
}
```

For

- En los otros bucles, teníamos que encargarnos de dar el valor inicial a la variable utilizada en la condición (llamada **variable de control**) antes del bucle, y escribir el cambio de valor de la variable dentro del mismo.
- El bucle for nos permite escribir un código más limpio, poniendo la inicialización y actualización de la variable de control en la cabecera del bucle:

```
for (inicialización; condición; actualización)
```

Foreach

- Si quisiéramos, por ejemplo, imprimir todos los elementos de un array con un bucle for, tendríamos que hacer lo siguiente:

Ésta es otra forma de crear un array

```
int[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
for (int indice = 0; indice < array.Length; indice = indice + 1)
{
    Console.WriteLine(array[indice]);
}
```

Foreach

- Cuando lo que queremos es obtener uno a uno todos los elementos de una colección, podemos utilizar el bucle `foreach`:

```
int[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
foreach (int numero in array)
{
    Console.WriteLine(numero);
}
```

Principios de la Programación Orientada a Objetos

Programación estructurada

La forma de escribir las sentencias de código que hemos ido viendo hasta ahora responde al paradigma de la programación estructurada, que utiliza tres estructuras:

- **Secuencia** (las sentencias se ejecutan en el orden en que aparecen).
- **Selección** (se puede elegir si ejecutar un bloque de sentencias con un `if`).
- **Iteración** (se puede ejecutar un bloque de sentencias varias veces con `while`, `do`, `for` y `foreach`).

Programación procedimental

- Como una ampliación de la programación estructurada, existe la **programación procedimental**, que agrupa el código en unidades reutilizables llamadas **funciones**.
- Las funciones realizan operaciones con datos, mientras que las **variables** nos permiten almacenar dichos datos.

Programación Orientada a Objetos

- También existe el paradigma de la Programación Orientada a Objetos, basado en la programación procedimental, pero que introduce el concepto de **objeto**.
- Un objeto es una entidad que contiene una serie de **datos** y define las **operaciones** que se pueden hacer sobre ellos.

Programación Orientada a Objetos

- Para entenderlo mejor, vamos a volver al tema de las **estructuras**.
- Una estructura es un tipo de dato que permite almacenar **propiedades**, que son un conjunto de datos cuya finalidad está relacionada (por ejemplo, el texto, imagen, y opciones de respuesta de una pregunta).
- Si además de las propiedades, la estructura contiene funciones que operan con esas propiedades, a la estructura se le denomina **clase**, y a las variables que cree del tipo de la clase, **objetos** (o **instancias**).

Programación Orientada a Objetos

- Las funciones de un objeto tienen algunas características especiales, y se suelen denominar **métodos**.
- Hemos dicho que el propósito de los métodos es operar con las propiedades del objeto. Una de las características especiales de los métodos es que cuando no son estáticos, tienen un parámetro especial llamado **this**.
- El parámetro this no hay que especificarlo entre paréntesis en la definición, sino que compilador lo coloca automáticamente (igual que cuando llamamos una función, coloca automáticamente el contexto si no lo ponemos nosotros).

Programación Orientada a Objetos

Por ejemplo:

```
class Button
{
	public bool Visible;

	public void Show()
	{
		this.Visible = true;
	}

	public void Hide()
	{
		this.Visible = false;
	}
}
```

Programación Orientada a Objetos

- Cuando llamamos a una función, decíamos que hay que poner el contexto de la función antes del nombre, separados por un punto:

```
Console.WriteLine(“¡Hola, mundo!”);
```

- En un método, el contexto no es el nombre de la clase en que está definido, sino el nombre de un objeto de esa clase:

```
button1.Hide();
```

- El parámetro **this** recibe como valor el objeto **button1**.

Programación Orientada a Objetos

- Decíamos que la programación estructurada se basaba en tres principios: secuencia, selección, iteración.
- La programación orientada a objetos añade otros tres principios fundamentales:
  - **Encapsulación**.
  - **Herencia**.
  - **Polimorfismo**.

Encapsulación

- La encapsulación permite a una clase especificar qué propiedades y métodos son accesibles desde fuera de la clase, y cuáles son accesibles sólo desde dentro. Es decir, especifica su **visibilidad**.
- Habíamos hablado anteriormente de esto. Si recordamos, la visilibidad de un método (y también las de una propiedad) podía ser:
  - **private**: La propiedad o método sólo puede ser utilizada dentro de la clase.
  - **protected**: La propiedad o método sólo puede ser utilizada dentro de la clase o en una de sus clases derivadas (hablaremos de esto en la diapositiva siguiente).
  - **public**: La propiedad o método puede ser utilizada en cualquier punto del programa.

Herencia

- Una característica de las clases es que, en su definición, podemos indicarle que hereden todas las propiedades y métodos que están definidos en otra clase:

```
class Button : Control
{
    // ...
}
```

- Diremos que Control es la **clase padre o base**, y que Button es la **clase hija o derivada**.

Polimorfismo

- Polimorfismo es la capacidad de definir un método en varios contextos, y que el programa elija cuando se llama al método cuál de todas las versiones definidas debe ser llamada en función del contexto.
- Por ejemplo, tanto la clase `Button` como la clase `Label` tienen un método `Hide`. La versión del método a la que se llama depende del objeto que se ponga como contexto:

```
button1.Hide();    // Se llama a la definición de la clase Button
label1.Hide();     // Se llama a la definición de la clase Label
```

Tutoriales

Tutoriales

- Ahora que hemos sido capaces de terminar nuestra primera aplicación en C#, podríamos querer indagar un poco más en el aprendizaje del lenguaje (¡nos quedan muchas cosas por ver!).
- Existen multitud de sitios en Internet donde podemos encontrar información sobre el lenguaje C#.
- Vamos a ver aquí un par de sitios interesantes.

Tutoriales de Microsoft

Microsoft nos ofrece muy buena documentación sobre su lenguaje C# (¡y en castellano!). Podemos encontrar algunos tutoriales sencillo en la siguiente página:

Tutoriales de C#

VISUALCSHARPTUTORIALS.COM

El sitio visualcsharptutorials.com nos ofrece varios tutoriales interesantes sobre el lenguaje (en inglés). Especialmente interesantes son los de **Windows Forms:**

Windows Forms Tutorials