Capítulo 3
Introducción al lenguaje C#

1. La sintaxis

1.1 Los identificadores

Los identificadores son los nombres que se asignan a las clases y a sus miembros. Un identificador debe estar compuesto por una única palabra. Ésta debe empezar por una letra o un carácter underscore (_). Los identificadores pueden tener letras mayúsculas o minúsculas, pero como el lenguaje C# es sensible a mayúsculas/minúsculas, éstas se deben respetar para que la referencia al identificador sea correcta: `miIdentificador` es diferente a `MiIdentificador`.

1.2 Las palabras clave

Las palabras clave son los nombres que reserva el lenguaje C#. Se interpretan por el compilador y, por tanto, no se pueden usar como identificadores. Estas palabras clave se diferencian en el editor de texto de Visual Studio utilizando el color azul (con los argumentos de apariencia predeterminados).

Si necesita utilizar una palabra clave como un identificador para un miembro, es necesario utilizar un prefijo con el nombre del identificador: el carácter @. La siguiente sintaxis es errónea y el compilador no la podrá ejecutar:

```
private bool lock;
```

Si utilizamos el prefijo @ con el miembro `lock`, el compilador considera que es un identificador y no una palabra clave:

```
private bool @lock;
```

El carácter @ también puede servir como prefijo de los identificadores que no tienen ningún conflicto con las palabras claves, de esta manera `@miIdentificador` se interpretará de la misma manera que `miIdentificador`.

A continuación, se muestra una lista de las palabras clave del lenguaje C#. Se explicarán, en parte, a lo largo del libro:

abstract	add	as	ascending	async
await	base	bool	break	by
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	descending	do	double	dynamic
else	equals	enum	event	explicit
extern	false	file	finally	fixed
float	for	foreach	from	get
global	goto	group	if	implicit
in	int	interface	internal	into
is	join	let	lock	long
nameof	namespace	new	null	object
on	operator	orderby	out	override
params	partial	private	protected	public
readonly	ref	remove	required	return
sbyte	sealed	select	set	short

© Editions ENI - All rights reserved

sizeof	stackalloc	statics	string	struct
switch	this	throw	true	try
typeof	uint	ulong	unchecked	unsafe
ushort	using	value	var	virtual
volatile	void	where	while	yield

1.3 Las reglas de puntuación

El objetivo de las reglas de puntuación es separar las instrucciones del programa de manera lógica, comprensible por el ser humano e interpretable por el compilador.

Cualquier instrucción debe terminar con un punto y coma `;`. Si se olvida al final de la instrucción, el compilador devuelve un error de sintaxis. Sin embargo, la ventaja es poder escribir una instrucción en varias líneas:

```
int i = 5
        + 2;
```

El punto . después de un identificador permite acceder a los miembros de un objeto. A través de IntelliSense, Visual Studio muestra la lista de miembros disponibles, tan pronto como se añade el punto a un objeto:

```
miObjeto.miPropiedad
```

Las llaves `{` y `}` se usan para agrupar varias instrucciones dentro de un bloque de control o de un método. Indican dónde comienzan las instrucciones y dónde terminan:

```
class Program
{
}
```

Los paréntesis (y) se usan para declarar o para llamar a métodos. Pueden contener argumentos después de la declaración del método. Los argumentos de un método se separan con una coma `,`:

```
miObjeto.miMetodo(Parametro1, Parametro2);
```

Los paréntesis también se utilizan para agrupar las instrucciones de la misma manera que para una operación matemática.

Los corchetes `[` y `]` permiten acceder a los elementos de un array o, si la clase contiene una propiedad para indexar, a los elementos de una clase. Por ejemplo, si la clase `miObjeto` es un array de valores de tipo `string`, para acceder a su primer elemento, la sintaxis sería la siguiente:

```
string s = miObjeto[0];
```

Los elementos de los arrays se indexan comenzando por 0.

1.4 Los operadores

1.4.1 Los operadores de cálculo

Los operadores de cálculo permiten, como en matemáticas, realizar operaciones.

La adición se realiza con el operador `+`:

```
i = 5 + 2;          // i = 7
```

La sustracción se realiza con el operador `-`:

```
i = 5 - 2;          // i = 3
```

La multiplicación se realiza con el operador `*`:

```
i = 5 * 2;          // i = 10
```

La división se realiza con el operador `/`:

```
i = 6 / 2;          // i = 3
```

El modulo se realiza con el operador `%`:

```
i = 5 % 2;          // i = 1
```

© Editions ENI - All rights reserved

1.4.2 Los operadores de asignación

Los operadores de asignación permiten asignar un valor a una variable. El operador más utilizado es el carácter =:

```
i = x;
```

También es posible realizar una asignación y un cálculo al mismo tiempo, combinando dos operadores:

```
i += 1;
```

usando el operador +=, hay una asignación a la variable de su propio valor, sumando el valor de la derecha del operador. Esta instrucción es equivalente a la siguiente:

```
i = i + 1;
```

La combinación de operadores de cálculo y de asignación es posible para todos los operadores:

```
int i = 5;
i += 2;          // i = 7
i -= 2;          // i = 5
i *= 2;          // i = 10
i /= 2;          // i = 5
i %= 2;          // i = 1
```

1.4.3 Los operadores de comparación

Los operadores de comparación se utilizan fundamentalmente para tomar decisiones dentro de instrucciones de control.

El operador == determina si dos variables son iguales:

```
x == y           // devuelve true si x igual a y
```

El operador != determina si dos variables son diferentes:

```
x != y           // devuelve true si x es diferente de y
```

El operador > determina si la variable de la izquierda es estrictamente superior a la variable de la derecha:

```
x > y            // devuelve true si x es superior a y
```

El operador `>=` determina si la variable de la izquierda es superior o igual a la variable de la derecha:

```
x >= y          // devuelve true si x es superior o igual a y
```

El operador `<` determina si la variable de la izquierda es estrictamente inferior a la variable de la derecha:

```
x < y           // devuelve true si x es inferior a y
```

El operador `<=` determina si la variable de la izquierda es inferior o igual a la variable de la derecha:

```
x <= y          // devuelve true si x es inferior o igual a y
```

El operador (y palabra clave) `is` permite determinar el tipo de un objeto:

```
x is int        // devuelve true si x es del tipo int
```

El filtrado por motivo permite verificar si un valor corresponde a un motivo. Se trata de utilizar el operador `is` para definir el motivo que puede reemplazarse en instrucciones condicionales:

```
o is DateTime d // devuelve true si o es de tipo DateTime
```

La variable `o` es automáticamente convertida en el tipo testeado y almacenada en la nueva variable `d`, que se puede utilizar de manera clásica.

También es posible combinar los operadores de comparación con los operadores lógicos. El operador `&&` permite especificar un AND lógico, mientras que el operador `||` especifica un OR lógico. Las diferentes expresiones se pueden combinar con ayuda de los paréntesis para modificar el orden de interpretación:

```
(x >= 0 || x > 10) && (x <= 1 || x < 25)
```

1.5 La declaración de variables

La declaración de variables se realiza especificando su tipo y posteriormente indicando su identificador. La siguiente instrucción declara una variable llamada `s` de tipo `string`:

```
string s;
```

© Editions ENI - All rights reserved

La instrucción de declaración termina como cualquier instrucción, es decir, con un punto y coma.

Una variable se puede declarar e inicializar con la misma instrucción:

```
string s = "El valor de mi variable";
```

También es posible declarar e inicializar varias variables en una única instrucción, con la condición de que sean del mismo tipo. Las variables se separan con una coma:

```
bool b1 = true, b2 = false;
```

Una variable también se puede marcar con la palabra clave `const`, que especifica que el valor de la variable no se puede modificar durante la ejecución. Es una variable en modo de solo lectura:

```
const int i = 0;
```

Añadiendo la palabra clave `required` a una propiedad o a un campo, indica que los elementos que llaman al constructor tienen la obligación de inicializar estos valores.

```
required int i;
```

El ámbito de una variable declarada en el cuerpo de un método se limita a ese método, es decir, se elimina cuando finaliza la ejecución del método. Queda fuera de ámbito.

Las variables declaradas dentro de un bloque condicional o iterativo solo son accesibles dentro de este bloque. Para los bloques iterativos, la variable se elimina al final del bucle y se inicializa de nuevo en la siguiente iteración del bucle.

1.6 Las instrucciones de control

1.6.1 Las instrucciones condicionales

Las instrucciones condicionales permiten ejecutar una porción de código en función de comprobaciones realizadas sobre las variables de la aplicación. Existen dos tipos de estructuras condicionales: los bloques `if` y los bloques `switch`.

if, else y else if

Sintaxis general:

```
if (expresión)
{
       instrucciones
}
[else if (expresión)
{
       instrucciones
}]
[else
{
       instrucciones
}]
```

La instrucción `if` evalúa una expresión booleana y ejecuta el código si esta expresión es verdad (`true`). Su sintaxis es la siguiente:

```
if (x > 10)
{
   // Instrucciones que se ejecutan si x es superior a 10
}
```

La expresión a evaluar se inserta entre paréntesis después de la palabra clave `if` y debe tener un resultado de tipo booleano.

La instrucción `else` permite integrar el código que se ejecutará si la expresión que se evalúa en la instrucción `if` es falsa (`false`):

```
if (x > 10)
{
   // Instrucciones que se ejecutan si x es superior a 10
}
else
{
   // Instrucciones que se ejecutan si x es inferior o igual a 10
}
```

© Editions ENI - All rights reserved

La instrucción `else if` permite evaluar una nueva expresión cuando el resultado de la instrucción `if` es falsa. Puede haber varias instrucciones `else if` pero un bloque de decisión `if` siempre debe empezar por una instrucción `if`, seguida de una o varias instrucciones `else if` y terminar de manera no obligatoria por una instrucción `else`:

```
if (x > 10)
{
   // Instrucciones que se ejecutan si x es superior a 10
}
else if (x = 10)
{
   // Instrucciones que se ejecutan si x es igual a 10
}
else
{
   // Instrucciones que se ejecutan si x es inferior a 10
}
```

Las instrucciones dentro de la estructura de decisión se encierran entre llaves. Esto indica el inicio y el fin del bloque. En el caso en el que el bloque solo tenga una línea, se permite omitir las llaves (también válido para las demás instrucciones de control):

```
if (x > 10)
   x -= 10;
else if (x = 10)
   x -= 1;
else
   x += 10;
```

El operador de condición `null ?.` permite comprobar el valor `null` cuando se accede a los miembros de una clase. Esto limita el número de instrucciones condicionales y el código resulta más legible. Por ejemplo, para acceder a una propiedad encapsulada en un objeto y asegurarse de que ningún padre es `null` y, por tanto, que no se producirá ninguna excepción durante su acceso, deberíamos escribir una instrucción de este tipo:

```
string MajorVersion;
if(soft!=null && soft.Version!=null)
{
    MajorVersion = soft.Version.Major;
}
```