TEMA Nº 3. VARIABLES Y CÁLCULOS

En este capítulo se conocerán:
• Los tipos de variables numéricas;
• Cómo declarar variables;
• La instrucción de asignación;
• Los operadores aritméticos;
• El uso de números con etiquetas y cuadros de texto;
• Los fundamentos del uso de cadenas de caracteres.

INTRODUCCIÓN

En casi todos los programas se utilizan números de un tipo u otro; por ejemplo, para dibujar imágenes mediante el uso de coordenadas en la pantalla, para controlar trayectorias de vuelos espaciales, o para calcular sueldos y deducciones fiscales. En este capítulo veremos los dos tipos básicos de números:

- los números sin decimales, conocidos como enteros en matemáticas, y como el tipo **int** en C#;
- los números con "punto decimal", conocidos como "reales" en matemáticas, y como el tipo **double** en C#. El término general para los números con punto decimal en computación es números de *punto flotante*.

En el capítulo previo utilizamos valores para producir gráficos en pantalla, pero para realizar programas más sofisticados necesitamos introducir el concepto de una variable: un tipo de caja de almacenamiento que se utiliza para recordar valores, de forma que éstos puedan utilizarse o modificarse más adelante en el programa.

Algunos de los casos en los que se utilizan números **int** son para representar o calcular:

- el número de estudiantes que hay en una clase;
- el número de píxeles que conforman una pantalla;
- el número de copias de un libro que se han vendido hasta cierto momento;

En cuanto a las situaciones que exigen el uso de números **double** podemos señalar:
- mi altura en metros;
- la masa de un átomo en gramos;

- el promedio de los enteros 3 y 4.

Sin embargo, algunas veces el tipo de número a utilizar no es obvio; por ejemplo, si queremos tener una variable para almacenar la calificación de un examen, ¿debemos emplear un número **double** o**int**? No podemos determinar la respuesta con base en lo que sabemos; debemos contar con más detalles.

Por ejemplo, podemos preguntar a la persona que se encarga de calificar si redondea al número entero más cercano, o si utiliza números decimales. En consecuencia, la elección entre **int** y **double** se determina a partir de cada problema en particular.

NATURALEZA DEL INT

Cuando se utiliza un número **int** en C#, puede tratarse de un número entero en el rango de 22,147,483,648 a 12,147,483,647, o aproximadamente de 22,000,000,000 a 12,000,000,000. Todos los cálculos con números **int** son precisos, en cuanto a que toda la información en el número se preserva sin errores.

NATURALEZA DEL DOUBLE

Cuando se utilizamos un número **double** en C# su valor puede estar entre 21.79 310308 y 11.79 310308. En términos no tan matemáticos, el mayor valor es 179 seguido de 306 ceros; ¡sin duda un valor extremadamente grande! Los números se guardan con una precisión aproximada de 15 dígitos.

El principal detalle respecto de las cantidades **double** estriba en que, en casi todos los casos, éstas se guardan en forma aproximada. Para comprender mejor esta característica, realice la siguiente operación en una calculadora:

7 / 3
Si utilizamos siete dígitos (por ejemplo) la respuesta es 2.333333, pero sabemos que una respuesta más exacta sería: 2.33333333333333333
Y aun así, ¡ésa no es la respuesta exacta!

En resumen, como las cantidades **double** se almacenan utilizando un número limitado de dígitos, pueden acumularse pequeños errores en el extremo menos significativo. Para muchos cálculos (por ejemplo, calificaciones de exámenes) esto no es importante, pero para aquellos relacionados con digamos, el diseño de una nave espacial, cualquier diferencia podría ser relevante. Sin embargo, el

rango de precisión de los números **double** es tan amplio que es posible emplearlos sin problemas en los cálculos de todos los días.

Para escribir valores **double** muy grandes (o muy pequeños) se requieren grandes secuencias de ceros. Para simplificar esto podemos usar la notación "científica" o "exponencial", con **e** o **E**, como en el siguiente ejemplo:
double valorGrande = 12.3E+23; lo cual representa 12.3 multiplicado por 10123. Esta característica se utiliza principalmente en programas matemáticos o científicos.

DECLARACIÓN DE VARIABLES

Una vez elegido el tipo de variables, se necesita darles un nombre. Se pueden imaginar como cajas de almacenamiento con un nombre en su exterior y un número (valor) en su interior.

El valor puede cambiar a medida que el programa realiza su secuencia de operaciones, pero el nombre es fijo. El programador tiene la libertad de elegir los nombres, y es recomendable escoger aquellos que sean significativos y no crípticos. A pesar de esa libertad, al igual que en casi todos los lenguajes de programación, en C# hay ciertas reglas que se deben seguir. Por ejemplo, los nombres:

- deben empezar con una letra (de la **A** a la **Z** o de la **a** a la **z**);
- pueden contener cualquier cantidad de letras o dígitos (un dígito es cualquier número del 0 al 9);
- pueden contener el guión bajo '_';
- pueden tener hasta 255 caracteres de longitud.

Es necesario que se tenga en cuenta que C# es sensible al uso de mayúsculas y minúsculas. Por ejemplo, si se declara una variable llamada **ancho**, no podrá referirse nunca a ella como **Ancho** o **ANCHO**, ya que el uso de las mayúsculas y minúsculas es distinto en cada caso. Éstas son las reglas de C#, y deben obedecerlas.

Pero también hay un estilo de C#, una forma de usar las reglas que se implementa cuando el nombre de una variable consta de varias palabras: las reglas no permiten separar los nombres con espacios, así que en lugar de utilizar nombres cortos o guiones bajos, el estilo aceptado es poner en mayúscula la primera letra de cada palabra.

Hay otro lineamiento de estilo para decidir si se debe usar mayúscula en la primera letra de un nombre o no. En este capítulo se trabajará con variables que

sólo se utilizan dentro de un método (en vez de que varios métodos las compartan). Las variables de este tipo se conocen como *locales* y sólo se pueden emplear entre los caracteres **{** y **}** en donde se declaren. Volviendo a las convenciones de estilo, la metodología de C# dicta *no* poner en mayúscula la primera letra de las variables locales.

Más adelante veremos que otros tipos de nombres, como los de métodos y clases, empiezan por convención con letra mayúscula.

Por lo tanto, en vez de:
Alturadecaja
h
hob
altura_de_caja
usaremos:
alturadeCaja

He aquí algunos nombres permitidos:
cantidad
x
pago2003

y éstos son algunos nombres no permitidos (ilegales):
2001pago
_area
mi edad

También existen algunos nombres reservados para utilización exclusiva de C#, de manera que el programador no los puede reutilizar. Se denominan *palabras clave* o *palabras reservadas*, y ya se han visto varias de ellas:
private
int
new

El código siguiente corresponde a un programa de ejemplo llamado Área de rectángulo, mismo que analizaremos con detalle a continuación. Supongamos que las medidas de los lados del rectángulo que nos interesa están representadas en números enteros (**int**). Sólo hay un control en el formulario: un botón con el texto "Calcular" en su propiedad **Text**. Todo el código que agregaremos estará dentro del método **button1_Click**.

```
private void button1_Click(object sender, EventArgs e)
{
int área;
int longitud;
int ancho;
longitud = 20;
ancho = 10;
área = longitud * ancho;
MessageBox.Show("El área es: " + Convert.ToString(área));
}
```

La Figura muestra lo que se verá en pantalla al ejecutar el código. En el programa utilizamos tres variables **int**, que en un momento dado guardarán los datos de nuestro rectángulo.

Es necesario recordar que se puede elegir cualquier nombre para nuestras variables; sin embargo, optamos por utilizar nombres claros en vez de nombres cómicos o de una sola letra, que no resultan lo suficientemente claros.

Una vez elegidos los nombres debemos declararlos en el sistema de C#. Aunque esto parece tedioso al principio, el propósito de introducirlos radica en permitir que el compilador detecte errores al escribirlos en el código del programa. He aquí las declaraciones:

```
int área;
int longitud;
int ancho;
```

Al declarar variables anteponemos al nombre que elegimos el tipo que necesitamos (en las tres variables anteriores utilizamos el tipo **int**, de manera que cada variable contendrán un número entero).

También se podría utilizar como alternativa una sola línea de código, como ésta:

```
int longitud, ancho, área;
```

usando comas para separar cada nombre. Usted puede emplear el estilo de su preferencia; no obstante, se recomienda usar el primero, ya que nos permite insertar comentarios en cada nombre, en caso de ser necesario. Si usted opta por el segundo estilo, úselo para agrupar nombres relacionados.

Por ejemplo:

```
int alturaImagen, anchoImagen;
int miEdad;
```

en vez de:

```
int alturaImagen, anchoImagen, miEdad;
```

En casi todos los programas utilizaremos varios tipos, y en C# podemos mezclar las declaraciones, como en el siguiente ejemplo:

```
double alturaPersona;
int calificaciónExamen;
double salario;
```

Además es posible establecer el valor inicial de la variable al tiempo que se puede declarar, como en estos ejemplos:

```
double alturaPersona = 1.68;
int a = 3, b = 4;
int calificaciónExamen = 65;
int mejorCalificación = calificaciónExamen + 10;
```

Este es un buen estilo, pero sólo debe usarlo cuando realmente conozca el valor inicial. Si no suministra un valor inicial, C# considera la variable como *no asignada*, y un error de compilación le informará sobre ello si trata de usar su valor en el programa.

INSTRUCCIÓN DE ASIGNACIÓN

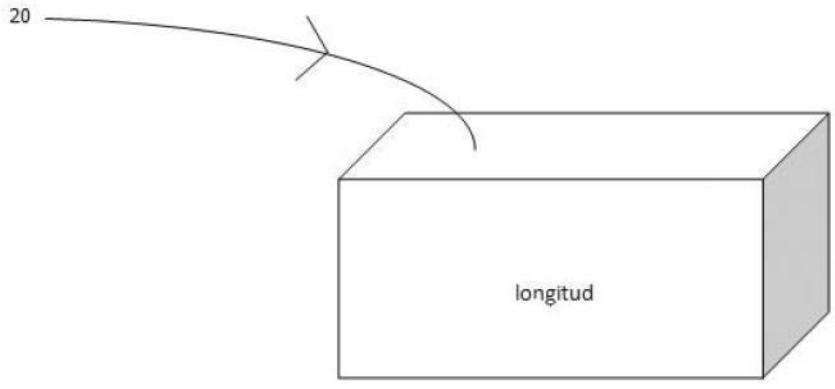
Una vez declaradas nuestras variables podemos colocar nuevos valores en ellas mediante la "instrucción de asignación", como en el siguiente ejemplo:

longitud = 20;

El proceso puede visualizarse como se ilustra en la figura siguiente. Decimos que "el valor 20 se ha asignado a la variable longitud", o que "longitud toma el valor de 20".

Nota:

- El flujo de los datos va de la derecha del signo = hacia la izquierda.
- Cualquier valor que haya tenido longitud antes será "sustituido" por 20. Las variables sólo tienen un valor: el actual. Para darle una idea de la velocidad de estas operaciones, considere que una asignación tarda menos de una millonésima de segundo en realizarse.



Asignación de un valor a una variable.

CÁLCULO Y OPERADORES

Veamos de nuevo nuestro programa del rectángulo, en el que se incluye la siguiente instrucción:
área = longitud * ancho;

La forma general de la instrucción de asignación es:
variable = expresión;

Una expresión puede tomar varias formas, como un solo número o como un cálculo. En nuestro ejemplo específico la secuencia de eventos es:

1. El carácter * hace que se multipliquen los valores almacenados en longitud y ancho, obteniéndose como resultado el valor 200.
2. El símbolo igual = hace que el número 200 se asigne a (se almacene en) área.

El carácter * es uno de varios "operadores" (se les llama así debido a que operan sobre los valores) y, al igual que en matemáticas, hay reglas para su uso.
Es importante comprender el flujo de los datos, ya que esto nos permite entender el significado de códigos como el siguiente:

```
int n = 10;
n = n + 1;
```

Lo que ocurre aquí es que la expresión que está al lado derecho del signo = se calcula utilizando el valor actual de n, con lo cual se obtiene 11. Después este valor se almacena en n, sustituyendo el valor anterior, que era 10. Hace algunos años se analizó una gran cantidad de programas, y se descubrió que las instrucciones de la forma:

```
algo = algo + 1;
```

estaban entre las más comunes. De hecho, C# cuenta con una versión abreviada de esta instrucción, llamada instrucción de *incremento*. Los operadores **++ y ––** realizan el incremento y el decremento (o resta de una unidad). Su uso más frecuente es en los ciclos. He aquí una forma de utilizar el operador ++:

```
n = 3;
n++; // ahora n vale 4
```

Respecto del signo =, lo importante es saber que no significa "es igual a" en el sentido algebraico.
Lo más correcto sería imaginar que significa "toma el valor de" o "recibe".

OPERADORES ARITMÉTICOS

En esta sección se presenta un conjunto básico de operadores: los aritméticos, similares a los botones de cualquier calculadora.

Operador	Significado
*	multiplicación
/	división
%	módulo
+	suma
-	resta

Es importante observar que se dividen los operadores en grupos para indicar su "precedencia", es decir, el orden en el que se realizan sus operaciones. Por lo tanto, la multiplicación, división y módulo (*****, **/** y **%**) se llevan a cabo antes que la suma y la resta (**+** y **–**). También se pueden usar paréntesis para agrupar los cálculos y forzarlos a llevarse a cabo en un orden específico. Si un cálculo incluye operadores de la misma precedencia, las operaciones se realizarán de izquierda a derecha. He aquí algunos ejemplos:

```
int i;
int n = 3;
double d;
i = n + 3;              // se convierte en 6
i = n * 4;              // se convierte en 12
i = 7 + 2 * 4;          // se convierte en 15
n = n * (n + 2) * 4;    // se convierte en 60
d = 3.5 / 2;            // se convierte en 1.75
n = 7 / 4;              // se convierte en 1
```

Es necesario recordar que las instrucciones forman una secuencia, la cual se ejecuta de arriba hacia abajo en la página. Siempre que se utilicen paréntesis, los elementos que éstos contengan se calcularán primero.

La multiplicación y la división se realizan antes de la suma y la resta. Por lo tanto:

3 + 2 * 4

se lleva a cabo como si se hubiera escrito así:

3 + (2 * 4)

Observe que, por cuestión de estilo, escribimos un espacio antes y después de un operador. Esto no es esencial, puesto que el programa se ejecutará de todas formas si se omiten los espacios. Sólo los utilizamos para que el programa sea más legible para el programador.

Ahora que se conocen las reglas. Se pueden ver a continuación algunas fórmulas matemáticas y su conversión a C#. Supongamos que todas las variables están declaradas como tipos **double**, y que su valor inicial ha sido establecido.

	Versión matemática	Versión de C#
1	$y = mx + c$	y = m * x + c;
2	$x = (a - b)(a + b)$	x = (a - b) * (a + b);
3	$y = 3[(a - b)(a + b)] - x$	y = 3 * ((a - b)*(a + b)) - x;
4	$y = 1 - \frac{2a}{3b}$	y = 1 - (2 * a) / (3 * b);

En el ejemplo 1 insertamos el símbolo de multiplicación. En C# **mx** se consideraría un nombre de variable.

En el ejemplo 2 necesitamos un signo de multiplicación explícito entre los paréntesis.

En el ejemplo 3 sustituimos los corchetes matemáticos por paréntesis.

En el ejemplo 4 podríamos haber cometido el error de usar esta versión incorrecta:

y = 1 - 2 * a / 3 * b;

Recuerde la regla según la cual los cálculos se realizan de izquierda a derecha cuando los operadores tienen igual precedencia. El problema tiene que ver con los operadores * y /. El orden de evaluación es como si hubiéramos utilizado:

y = 1 - (2 * a / 3) * b;

es decir, la **b** ahora está multiplicando en vez de dividir. La forma más simple de manejar los cálculos potencialmente confusos consiste en utilizar paréntesis adicionales; hacerlo no implica penalización alguna en términos de tamaño o velocidad del programa.

El uso de los operadores **+**, **–** y * es razonablemente intuitivo, pero la división es un poco más engañosa, ya que exige diferenciar entre los tipos **int** y **double**. En este sentido, lo importante es tomar en cuenta que:

- Cuando el operador / trabaja con dos números **double** o con una mezcla de **double** e **int** se produce un resultado **double**. Para fines de cálculo, cualquier valor **int** se considera como **double**. Así es como funciona la división en una calculadora de bolsillo.

- Cuando / trabaja con dos enteros se produce un resultado entero. El resultado se trunca, lo cual significa que se borran los dígitos que pudiera haber después del "punto decimal". Ésta *no* es la forma en que funcionan las calculadoras.

He aquí algunos ejemplos:

```
// división con valores double
double d;
d = 7.61 / 2.1; // se convierte en 3.7
d = 10.6 / 2; // se convierte en 5.3
```

En el primer caso la división se lleva a cabo de la manera esperada.
En el segundo el número **2** se trata como **2.0** (es decir, un **double**) y la división se realiza.
Sin embargo, la división con enteros es distinta:

```
//división con enteros
int i;
i = 10 / 5; // se convierte en 2
i = 13 / 5; // se convierte en 2
i = 33 / 44; // se convierte en 0
```

En el primer caso se espera una división con enteros; la respuesta exacta que se produce es **2**.
En el segundo caso el resultado también es **2**, debido a que se trunca el verdadero resultado.
En el tercer caso se trunca la respuesta "correcta" de **0.75**, con lo cual obtenemos **0**.

OPERADORES

Por último veremos el operador **%** (módulo). A menudo se utiliza junto con la división de enteros, ya que provee la parte del residuo. Su nombre proviene del término "módulo" que se utiliza en una rama de las matemáticas conocida como aritmética modular.

Anteriormente dijimos que los valores **double** se almacenan de manera aproximada, a diferencia de los enteros, que lo hacen de forma exacta. Entonces ¿cómo puede ser que **33/44** genere un resultado entero de **0**? ¿Acaso perder **0.75** significa que el cálculo no es preciso? La respuesta es que los enteros *sí* operan con exactitud, pero el resultado exacto está compuesto de dos partes: el cociente (es decir, el resultado principal) y el residuo. Por lo tanto, si dividimos **4** entre **3** obtenemos como resultado **1**, con un residuo de **1**. Esto es más exacto que **1.3333333**, etc.

En consecuencia, el operador **%** nos da el residuo como si se hubiera llevado a cabo una división.

He aquí algunos ejemplos:

```
int     i;
double d;
i = 12 % 4; // se convierte en 0
i = 13 % 4; // se convierte en 1
i = 15 % 4; // se convierte en 3
d = 14.9 % 3.9; // se convierte en 3.2 (se divide 3.2 veces)
```

Hasta ahora el uso más frecuente de **%** es con números **int**, pero cabe mencionar que también funciona con números **double**. Veamos un problema que involucra un resultado con residuo: convertir un número entero de centavos en dos cantidades: la cantidad de dólares y el número de centavos restantes. La solución es:

```
int centavos = 234;
int dólares, centavosRestantes;
```

dólares = centavos / 100; // se convierte en 2
centavosRestantes = centavos % 100; // se convierte en 34

UNIÓN DE CADENAS CON EL OPERADOR +

Hasta ahora hemos visto el uso de variables numéricas, pero también es muy importante el procesamiento de datos de texto. C# cuenta con el tipo de datos **string**; las variables **string** pueden guardar cualquier carácter. La longitud máxima de una cadena es de aproximadamente dos mil millones de caracteres; cantidad superior al tamaño de la RAM de las computadoras actuales.

El siguiente es un ejemplo del uso de cadenas:

```
string primerNombre = "Miguel";
string apellidoPaterno, nombreCompleto;
string saludo;
apellidoPaterno = "González";
nombreCompleto = primerNombre + apellidoPaterno;
saludo = "Saludos de " + nombreCompleto; //se convierte en "Saludos de Miguel González"
```

En el ejemplo anterior se declaran algunas variables **string** y se les asignan valores iniciales mediante el uso de comillas dobles. Después se utiliza la asignación, en la cual el valor de la cadena a la derecha del signo **=** se almacena en la variable utilizada a la izquierda del mismo, de manera similar a la asignación numérica (si se intenta utilizar una variable que no haya recibido un valor, C# nos informará que la variable no está asignada y el programa no se ejecutará).

Las siguientes líneas ilustran el uso del operador **+**, que (de igual manera que al sumar números) opera sobre las cadenas y las une extremo con extremo. A esto se le conoce como “concatenación”.

Después de la instrucción:

```
nombreCompleto = primerNombre + apellidoPaterno;
```

el valor de **nombreCompleto** es **Miguel González**.

Además hay un amplio rango de métodos de cadenas que proporcionan operaciones tales como búsqueda y modificación de cadenas.

Previamente se mencionó que el operador **/** considera los elementos que divide como números **double** si uno de ellos es **double**. El operador **+** trabaja de manera similar con las cadenas. Por ejemplo:

```
int i = 7;
string nombre = "a. avenida";
string s = i + nombre;
```

En este caso el operador **+** detecta que **nombre** es una variable **string** y convierte **i** en una cadena antes de unir ambas variables. Éste es un método abreviado conveniente para evitar la conversión explícita que veremos más adelante, pero puede resultar engañoso. Considere el siguiente código:

```
int i = 2, j = 3;
string s, nota = "La respuesta es: ";
s = nota + i + j;
```

¿Cuál es el valor de **s**? Las dos posibilidades son:

- **La respuesta es**: **23**, en donde ambos operadores **+** trabajan sobre cadenas.
- **La respuesta es**: **5**, en donde el segundo **+** suma números.

De hecho lo que ocurre es el primer caso. C# trabaja de izquierda a derecha. El primer **+** produce la cadena "La respuesta es: 2"; después, el segundo **+** agrega el **3** a la derecha. No obstante, si se coloca:

```
s = nota + (i + j);
```

Primero se calcula la operación **2 + 3**, obteniéndose **5**. Por último se lleva a cabo la unión de las cadenas.

CONVERSIÓN ENTRE CADENAS Y NÚMEROS

Uno de los usos más importantes del tipo de datos **string** son las operaciones de entrada y salida, en donde se procesan los datos que introduce el usuario y se despliegan los resultados en pantalla.

Muchos de los controles de la GUI de C# trabajan con cadenas de caracteres en vez de hacerlo con números, por lo cual es preciso que se aprenda a realizar conversiones entre números y cadenas.

La clase **Convert** proporciona varios métodos convenientes para ese propósito.

Para convertir una variable o cálculo (una expresión en general) se puede utilizar el método **ToString**. He aquí algunos ejemplos:

```
string s1, s2;
int num = 44;
double d=1.234;
s1 = Convert.ToString(num); // s1 es "44"
s2 = Convert.ToString(d); // s2 es "1.234"
```

Por lo general el nombre del método va precedido por el de un objeto con el que debe trabajar, pero aquí se suministra el objeto como un argumento entre paréntesis. Los métodos que funcionan de esta forma se denominan estáticos (**static**); cada vez que se utilicen se deberá identificar la clase a la que pertenecen. Éste es el motivo por el que se coloca **Convert** antes de **ToString**.

En el ejemplo anterior el método **ToString** se regresa una cadena que podemos almacenar en una variable, o utilizarla de alguna otra forma.

En el programa para calcular el área de un rectángulo se utiliza el operador **+** y el método **ToString** con un cuadro de mensaje desplegable. En vez de mostrar sólo el número, lo unimos a un mensaje:

```
MessageBox.Show("El área del rectángulo es: " + Convert.ToString(área));
```

El siguiente código no compila, ya que el método **Show** espera un valor **string** como parámetro:

```
MessageBox.Show(área); //NO - ¡no compilará!
```

Se debe utilizar:

```
MessageBox.Show(Convert.ToString(área));
```

Para complementar el método **ToString** se tienen los métodos **ToInt32** y **ToDouble**, los cuales convierten las cadenas de caracteres en números. Observe que no hay un método **ToInt**. La clase **Convert** está disponible para cualquier lenguaje que utilice el marco de trabajo (framework) .NET, y el nombre de clase a nivel de marco de trabajo para los elementos **int** en C# es **Int32** (enteros de 32 bits). He aquí algunos ejemplos:

```
double d;
int i;
```

```
string s1 = "12.3";
string s2 = "567";
d = Convert.ToDouble(s1);
i = Convert.ToInt32(s2);
```

Ahora que sabemos realizar conversiones de cadenas, podemos empezar a utilizar varios controles nuevos.

CUADROS DE TEXTO Y ETIQUETAS

En los programas en que hemos venido trabajando utilizamos instrucciones de asignación con el propósito de establecer valores iniciales para los cálculos; pero, en la práctica no conoceremos esos valores al escribir el programa, ya que el usuario los introducirá a medida que éste se vaya ejecutando.

En esta sección veremos el control **TextBox**, el cual permite que un usuario introduzca datos, y el control **Label** que se utiliza para desplegar información (por ejemplo, los resultados de un cálculo, o instrucciones para el usuario) en un formulario.

Como sabemos, para usar un cuadro de texto todo lo que tenemos que hacer es seleccionarlo en el cuadro de herramientas y colocarlo en un formulario. Estos controles tienen muchas propiedades, pero la principal es **Text**, que nos proporciona la cadena escrita por el usuario. Para acceder a esta propiedad utilizamos la ya conocida notación de "punto", como en el siguiente ejemplo:

```
string s;
s = textBox1.Text;
```

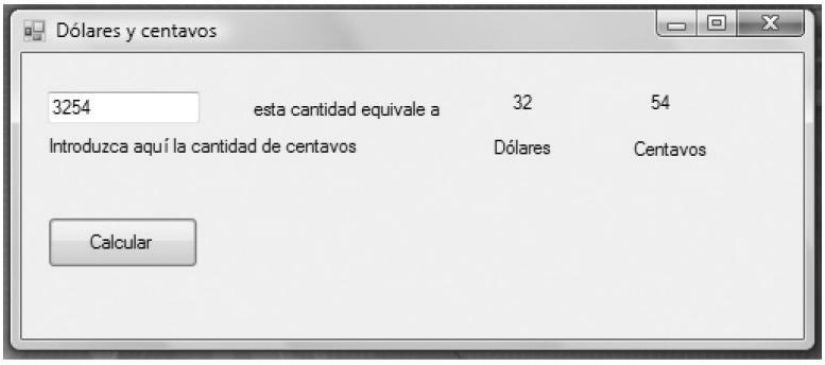
Es bastante común que el programador elimine el contenido de la propiedad **Text** del control en tiempo de diseño (mediante la ventana de propiedades), para que el usuario pueda escribir en un área en blanco.

Al igual que en el caso de los cuadros de texto, la principal propiedad del control **Label** (disponible también en el cuadro de herramientas) es **Text**, pues nos permite establecer la cadena que la etiqueta mostrará en pantalla. Podemos acceder a esta propiedad de la siguiente manera:

```
string s = "Alto";
label1.Text = s;
```

Algunas etiquetas se utilizan para mostrar mensajes de ayuda al usuario; por lo general establecemos su propiedad **Text** en tiempo de diseño mediante la ventana de propiedades. No es necesario que el texto que contienen cambie durante la ejecución del programa. Por otro lado, en el caso de las etiquetas que despliegan resultados hay que establecer su propiedad **Text** en tiempo de ejecución, como se muestra en el ejemplo anterior. El usuario puede sobrescribir los cuadros de texto, pero las etiquetas están protegidas.

En general, las clases tienen métodos y propiedades. Los métodos hacen que los objetos realicen acciones, mientras que las propiedades nos permiten acceder al estado actual de un objeto. He aquí:



un programa de ejemplo (Dólares y centavos), en el que una cantidad en centavos se convierte a dólares y centavos. Anteriormente en este capítulo vimos cómo usar los operadores **/** y **%**. En la Figura se muestra este programa en ejecución; en él se utiliza un cuadro de texto y varias etiquetas.

```
private void button1_Click(object sender, EventArgs e)
{
int centavos;
centavos = Convert.ToInt32(textBox1.Text);
dólaresEtiqueta.Text = Convert.ToString(centavos / 100);
centavosEtiqueta.Text = Convert.ToString(centavos % 100);
}
```

Los principales controles que se utilizan en este programa son:
• un botón para iniciar la conversión;
• un cuadro de texto en donde el usuario introduce una cantidad en centavos;
• dos etiquetas: para mostrar el número de dólares y el número de centavos.

Además hay tres etiquetas debajo del cuadro de texto y las dos etiquetas que muestran el resultado para ayudar al usuario a entender el formulario. Los valores de texto de las etiquetas son:

Introduzca aquí la cantidad de centavos
Dólares
Centavos

Como vimos anteriormente, es recomendable cambiar el nombre de los controles cuando hay más de una instancia del mismo tipo de control en un formulario. En este programa:

- hay un botón y un cuadro de texto, por lo que podemos dejar a estos controles el nombre que C# les asignó;
- hay dos etiquetas que muestran resultados. Como tener dos controles Etiqueta podría causar confusión, les damos un nombre específico a cada uno de ellos;
- al resto de las etiquetas se les asigna su propiedad de texto en tiempo de diseño, y el programa nunca las manipulará. Podemos dejar a estas etiquetas los nombres que C# les asignó.

He aquí un resumen de las principales propiedades de los controles.

Control	Propiedad	Valor
button1	Text	Calcular
textBox1	Text	(vacía)
dólaresEtiqueta	Text	(vacía)
centavosEtiqueta	Text	(vacía)

Recuerde que es conveniente cambiar el nombre de los controles tan pronto como los coloque en el formulario, antes de hacer doble clic para crear el código de cualquier evento.

Cuando el programa se ejecuta el usuario introduce un número en el cuadro de texto. Al hacer clic en el botón se lleva a cabo el cálculo y los resultados se colocan en las dos etiquetas. Sin embargo, hay que realizar conversiones de cadenas a números y viceversa. He aquí un extracto:

```
centavos = Convert.ToInt32(textBox1.Text);
```

dólaresEtiqueta.Text = Convert.ToString(centavos / 100);

El programa ilustra el uso de un cuadro de texto y de etiquetas para mostrar resultados que pueden cambiar, junto con mensajes que no se modifican.

CONVERSIÓN ENTRE NÚMEROS

Habrá ocasiones en que necesitaremos convertir valores numéricos de un tipo a otro. Los casos más comunes son la conversión de un **int** a un **double** y viceversa.

Veamos un ejemplo: tenemos nueve manzanas y queremos repartirlas de manera equitativa entre cuatro personas. Sin duda los valores **9** y **4** son enteros, pero la respuesta es un valor **double** (es decir, incluye decimales). Para resolver este problema debemos conocer algunos fundamentos sobre la conversión numérica.

Veamos primero algunos ejemplos de conversiones:

```
int i = 33;
double d = 3.9;
double d1;
d1 = i; // se convierte en 33.0
// o, de manera explícita:
d1= (double)i; // se convierte en 33.0
i = (int)d; // se convierte en 3
```

Los puntos a tomar en cuenta son:
• Asignar un **int** a un **double** no requiere programación adicional. Es un proceso seguro, ya que no se puede perder información; no hay posiciones decimales por los cuales preocuparse.
• Al asignar un **double** a un **int** pueden perderse los dígitos que suceden al punto decimal, ya que no caben en el entero. Debido a esta pérdida potencial de información, C# requiere que especifiquemos esta conversión de manera explícita. Podríamos utilizar la clase **Convert** para solucionar la situación, pero mejor usaremos otro método, conocido como conversión de tipos o *casting* (emplearemos también esta característica cuando veamos las herramientas más avanzadas de la programación orientada a objetos).
• Para convertir un **double** a la forma de un **int** debemos anteponer la palabra (**int**). En ese caso el valor se truncará al eliminar los dígitos que suceden al punto decimal.

• Cabe mencionar que podríamos usar una conversión explícita de tipos al convertir un **int** en un **double**, pero esto no es necesario.
Volviendo a nuestro ejemplo de las manzanas, podemos obtener una respuesta **double** si utilizamos las siguientes líneas de código:

```
int manzanas = 9; //u obtener el valor a partir de un cuadro de texto
int personas = 4; //u obtener el valor a partir de un cuadro de texto
MessageBox.Show("Cada persona recibe: " + Convert.ToString(
(double)manzanas / (double)personas));
```

Observe que (double)(manzanas / personas) produciría la respuesta incorrecta, ya que se realizaría una división entre enteros.

FUNDAMENTOS DE PROGRAMACIÓN

- Las variables tienen un nombre, y el programador puede elegir el que desee asignarles.
- Las variables tienen un tipo, y el programador puede elegir cuál de ellos utilizará.
- Las variables contienen un valor.
- El valor de una variable puede modificarse mediante una instrucción de asignación.

ERRORES COMUNES DE PROGRAMACIÓN

- Tener cuidado al escribir los nombres de las variables. Por ejemplo, en:

```
int círculo; // error de escritura
círculo = 20;
```

- la variable está mal escrita en la primera línea, ya que se utiliza un '1' (uno) en vez de una 'L' minúscula. En este caso el compilador de C# detectará que la variable de la segunda línea no está declarada. Otro error común es utilizar un cero en vez de una 'O' mayúscula.
- Es difícil detectar los errores de compilación al principio. Aunque el compilador de C# nos da una señal de la posición en la que cree que se encuentra el error, en realidad éste podría hallarse en una línea anterior.
- Los paréntesis deben estar balanceados, es decir, debe haber el mismo número de '**(**' que de '**)**'.
- Al utilizar números con la propiedad de texto de las etiquetas y los cuadros de texto, recuerde utilizar las herramientas de conversión de cadenas.

• Al multiplicar elementos se debe colocar el carácter * entre ellos, mientras que en matemáticas se omite este signo. Al dividir elementos, debemos recordar que:
– **int / int** nos da una respuesta **int**.
– **double / double** nos da una respuesta **double**.
– **int / double** y **double / int** nos dan una respuesta **double**.

RECOMENDACIONES PARA CODIFICAR

Para declarar variables indicamos su clase y su nombre; por ejemplo:
int miVariable;
string tuVariable = "¡Saludos a todos!";
• Los tipos de variables más útiles son **int**, **double** y **string**.
• Los principales operadores aritméticos son *, /, %, + y –.
• El operador + se utiliza para unir cadenas.
• Los operadores ++ y -- pueden emplearse para incrementar y decrementar.
• Podemos convertir números a cadenas con el método **Convert.ToString**.
• Podemos convertir cadenas a números con los métodos **Convert.ToInt32** y **Convert.ToDouble**.
• Si colocamos el operador de conversión (**int**) antes de un elemento **double**, éste se convierte en un entero.
• Si colocamos el operador de conversión (**double**) antes de un elemento **int**, éste se convierte en un valor **double**.

NUEVOS ELEMENTOS DEL LENGUAJE

int double string
• los operadores **+ - * / %**
• **++** y **--** para incremento y decremento
• **=** para asignación
• Conversión de tipos: la clase **Convert**, los operadores de conversión (**double**) e (**int**).

NUEVOS CARACTERÍSTICAS DEL IDE

• Los controles **TextBox** y **Label**, con sus propiedades **Text**.
• La posibilidad de cambiar el nombre de los controles.

RESUMEN

- Las variables se utilizan para contener (guardar) valores. Mantienen su valor hasta que éste es modificado de manera explícita (por ejemplo, mediante otra instrucción de asignación).
- Los operadores operan sobre valores.
- Las expresiones son cálculos que producen un valor. Pueden utilizar en una variedad de situaciones, incluyendo al lado derecho de una asignación, o como argumentos para invocar un método.