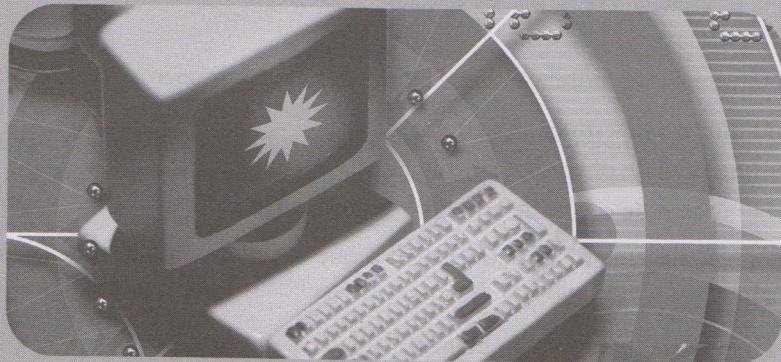


CAPÍTULO 4



Variables y cálculos

En este capítulo conoceremos:

- Los tipos de variables numéricas.
- Cómo declarar variables y constantes.
- La instrucción de asignación.
- Los operadores aritméticos.
- El uso de cuadros de mensaje y de entrada para las operaciones de entrada y salida.
- Los aspectos esenciales sobre las cadenas de texto.

● Introducción

En la mayoría de los programas se utilizan números de un tipo u otro; por ejemplo, para dibujar imágenes mediante coordenadas en la pantalla, controlar trayectorias de vuelos espaciales, así como calcular sueldos y deducciones fiscales.

En este capítulo veremos los dos tipos básicos de números:

- Los números enteros, conocidos como tales en las matemáticas y como tipo `int` en Java.
- Los números con “punto decimal”, conocidos como “reales” en las matemáticas y como tipo `double` en Java. El término general para los números con punto decimal en computación es *números de punto flotante*.

En el capítulo anterior utilizamos valores para producir gráficos en pantalla, pero para programas más sofisticados necesitamos introducir el concepto de variable: una especie de caja de almacenamiento que se utiliza para recordar valores, de forma que éstos se puedan utilizar o modificar más tarde en el programa.

Algunos casos indiscutibles en los que se utilizan números `int` son:

- El número de estudiantes en una clase.
- El número de píxeles en la pantalla.
- El número de copias de este libro vendidas hasta ahora.

Y también hay casos indiscutibles en los que se utilizan números `double`:

- Mi altura en metros.
- La masa de un átomo en gramos.
- El promedio de los números enteros 3 y 4.

Sin embargo, algunas veces el tipo no es obvio; por ejemplo, una variable para almacenar la calificación de un examen: ¿`double` o `int`? No podemos determinar la respuesta con lo que sabemos hasta el momento; debemos buscar más detalles. Por ejemplo, podemos preguntar a la persona que se encarga de calificar si redondea al número entero más cercano, o si utiliza lugares decimales. Por ende, la elección entre `int` y `double` se determina de acuerdo con el problema.

● La naturaleza de `int`

Cuando utilizamos un número `int` en Java, puede ser un número entero en el rango de -2147483648 a $+2147483647$, o aproximadamente de -2000000000 a $+2000000000$.

Todos los cálculos con números `int` son precisos en cuanto a que toda la información en el número se preserva con exactitud.

● La naturaleza de `double`

Cuando utilizamos un número `double` en Java, su valor puede estar entre -1.79×10^{308} y $+1.79 \times 10^{308}$.

En términos no tan matemáticos, el valor más grande 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 sobre las cantidades `double` es que en muchos casos se guardan en forma aproximada. 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:

2.333333333333333

Y aun así, ¡esta no es la respuesta exacta!

En resumen, como las cantidades `double` se almacenan en un número limitado de dígitos, se pueden acumular pequeños errores en el extremo menos significativo. Para muchos cálculos (por ejemplo, calificaciones de exámenes) esto no es importante, pero para cálculos relacionados con el diseño de un transbordador espacial esto sí podría ser significativo. Sin embargo, los números `double`

tienen tantos dígitos de precisión y un rango tan extenso que pueden utilizarse sin problemas en los cálculos relacionados con las cantidades que manejamos cotidianamente.

Para escribir valores `double` muy grandes (o muy pequeños) se requieren grandes secuencias de ceros. Para simplificar estas cifras podemos usar la notación “científica” o “exponencial” con `e` o `E`, como en el siguiente ejemplo:

```
double valorGrande = 12.3E+23;
```

lo cual representa a un 12.3 multiplicado por 10^{23} . Esta característica se utiliza principalmente en programas matemáticos o científicos.

● Declaración de variables

Una vez elegido el tipo de nuestras variables, necesitamos darles un nombre. Podemos imaginarlas 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 puede elegir los nombres, y recomendamos elegir nombres que sean significativos y no enigmáticos. Pero al igual que en la mayoría de los lenguajes de programación, hay ciertas reglas que debemos seguir. En Java, los nombres:

- Deben empezar con una letra (de la “A” a la “Z” o de la “a” a la “z”) o (lo que no es muy común) con un guión bajo “_”.
- Pueden contener cualquier cantidad de letras o dígitos (un dígito es del 0 al 9).
- Pueden contener el guión bajo “_”.
- Pueden tener hasta 255 caracteres de longitud.

Tome en cuenta que Java es susceptible al uso de mayúsculas y minúsculas. Por ejemplo, si usted declara una variable llamada `anchura`, no se puede referir a ella como `Anchura` o `ANCHURA`, ya que el uso de las mayúsculas y minúsculas es distinto en cada caso.

Éstas son las reglas de Java y debemos obedecerlas. Pero Java también tiene un estilo, una forma de usar las reglas que se implementa cuando una variable consta de varias palabras. Las reglas no permiten espacios en los nombres, por lo que en vez de utilizar nombres cortos o el guión bajo, el estilo aceptado para las variables es poner en mayúscula la primera letra de cada palabra dentro de un nombre.

Hay otro lineamiento de estilo para decidir si usar mayúscula en la primera letra de un nombre o no. Todas las variables deben empezar con una letra minúscula, mientras que los nombres de las clases (como veremos más adelante) empiezan por lo general con una letra mayúscula. Por lo tanto, en vez de:

```
Alturadecaja  
a  
adc  
altura_de_caja
```

usaremos:

```
alturaDeCaja  
  
cantidad  
x  
pago2003
```

He aquí algunos nombres permitidos:

y he aquí algunos nombres no permitidos (ilegales):

```
2001pago  
%área  
mi edad
```

También existen algunos nombres reservados que Java utiliza que el programador no puede reutilizar. Se denominan *palabras clave* o *palabras reservadas* en Java. Ya ha visto algunas de estas palabras clave:

```
private  
new  
int
```

El apéndice F incluye una lista completa.

PRÁCTICA DE AUTOEVALUACIÓN

- 4.1 ¿Cuáles de los siguientes nombres de variables locales están permitidos en Java y cuáles tienen el estilo correcto?

```
volumen  
ÁREA  
Longitud  
3lados  
lado1  
lonitud  
Misalario  
su salario  
tamañoPantalla  
screenSize
```

A continuación estudiaremos con detalle el siguiente programa de ejemplo, llamado **CalculoÁrea**. Este programa calcula el área de un rectángulo. Vamos a suponer que sus lados son cantidades **int**. El resultado se muestra en un cuadro de mensaje. La parte más importante se muestra con una pantalla.

```
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;  
  
public class CálculoÁrea extends JFrame  
    implements ActionListener {  
  
    private JButton botón;  
  
    public static void main(String[] args) {  
        CálculoÁrea marco = new CalculoArea();  
        marco.setSize(400, 300);  
        marco.createGUI();  
        marco.setVisible(true);  
    }  
}
```

```
private void createGUI() {  
    setDefaultCloseOperation(EXIT_ON_CLOSE);  
    Container ventana = getContentPane();  
    ventana.setLayout(new FlowLayout());  
  
    botón = new JButton("Haga clic");  
    ventana.add(botón);  
    botón.addActionListener(this);  
}  
  
public void actionPerformed(ActionEvent event) {  
    int área;  
    int longitud;  
    int anchura;  
    longitud = 20;  
    anchura = 10;  
    área = longitud * anchura;  
    JOptionPane.showMessageDialog(null,"El área es: " + área);  
}
```

La figura 4.1 muestra la ventana y el cuadro de mensaje que aparecen al ejecutar el programa.

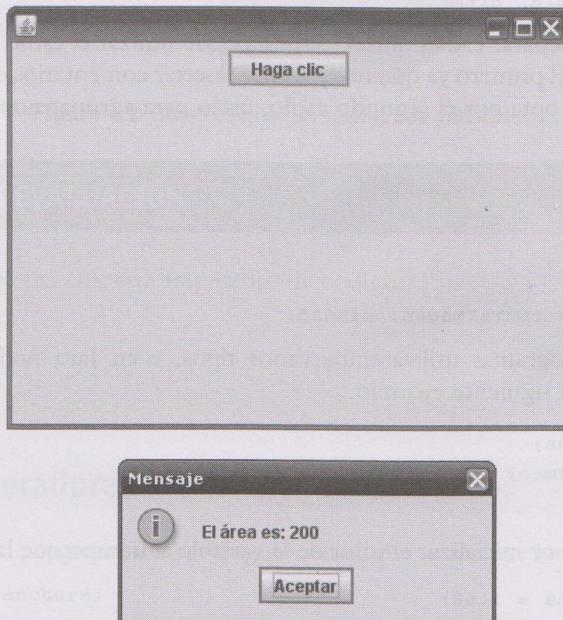


Figura 4.1 Pantalla de la salida de **CalculoArea** en donde se muestra la ventana y el cuadro de mensaje.

Para nuestros fines en este capítulo, el programa consta de dos partes. La primera es responsable de crear una ventana que contiene un solo botón. Esto será estándar para todos los programas de este capítulo. La parte que examinaremos con detalle tiene el siguiente encabezado:

```
public void actionPerformed(ActionEvent event) {
```

Esta sección se ejecuta cada vez que el usuario hace clic en el botón. De hecho, la sección se conoce como “método” y debemos colocar nuestras instrucciones entre sus llaves de apertura y de cierre.

En el programa utilizamos tres variables `int`, que en un momento dado guardarán los datos de nuestro rectángulo. Recuerde que podemos elegir cualquier nombre para nuestras variables, sin embargo optamos por utilizar nombres legibles en vez de nombres graciosos o de una sola letra (¡los nombres graciosos sólo son graciosos la primera vez que los vemos!).

Una vez elegidos los nombres, debemos declararlos en el sistema de Java. Aunque esto parece tedioso al principio, el objeto de introducirlos es permitir que el compilador detecte errores ortográficos en el código del programa en el que se utilicen estos nombres. He aquí las declaraciones:

```
int área;
int longitud;
int anchura;
```

Para 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 puede contener un número entero).

La siguiente línea puede ser una alternativa a las tres líneas anteriores de código:

```
int longitud, anchura, área;
```

aquí usamos comas para separar cada nombre. Usted puede utilizar el estilo de su preferencia, pero nosotros preferimos 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, use:

```
int alturaImagen, anchuraImagen;
int miEdad;
```

en vez de:

```
int alturaImagen, anchuraImagen, miEdad;
```

En la mayoría de los programas utilizaremos varios tipos, y en Java podemos entremezclar las declaraciones, como en el siguiente ejemplo:

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

Además, podemos optar por inicializar el valor de la variable al tiempo que la declaramos, como

```
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 conozca el valor inicial. Si no suministra un valor inicial para las variables declaradas dentro de un método, Java las considera como no inicializadas y un error de compilación le informará sobre ello si trata de usarlas en el programa.

● La 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;
```

Podemos visualizar el proceso como se muestra en la figura 4.2. Decimos que “el valor 20 se ha asignado a la variable `longitud`”, o que “`longitud` se vuelve 20”.

Nota:

- El movimiento de los datos es de la derecha del signo = hacia la izquierda.
- Cualquier valor que haya tenido `longitud` antes será “sustituido” por el 20. Las variables sólo tienen un valor: el actual. Y para darle una idea de la velocidad de estas operaciones, una asignación tarda menos de una millonésima parte de un segundo.

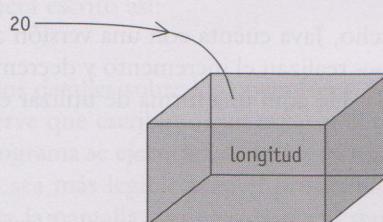


Figura 4.2 Cómo asignar un valor a una variable.

PRÁCTICA DE AUTOEVALUACIÓN

- 4.2 Explique el problema con este fragmento de código:

```
int a, b;  
a = b;  
b = 1;
```

● Cálculos y operadores

Volvamos de nuevo nuestro programa del rectángulo, en el que se incluye la siguiente instrucción:

```
área = longitud * anchura;
```

La forma general de la instrucción de asignación es:

```
variable = expresión;
```

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

1. El signo `*` hace que se multipliquen los valores almacenados en `longitud` y `anchura`, y se obtiene como resultado el valor 200.
2. El signo `=` hace que el 200 se asigne a (se almacene en) `área`.

El signo `*` 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 movimiento de los datos, ya que nos permite comprender el significado de código tal como:

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

Lo que ocurre aquí es que el lado derecho del signo `=` se calcula utilizando el valor actual de `n`, con lo cual se obtiene un 11. Después este valor se almacena en `n`, sustituyendo al valor anterior de 10.

Hace algunos años se estudiaba una gran cantidad de programas, y se descubrió que las instrucciones de la forma:

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

eran de las más comunes! De hecho, Java cuenta con una versión abreviada de esta instrucción de *incremento*. Los operadores `++` y `--` realizan el incremento y decremento (restar 1). Su uso más frecuente es en los ciclos (capítulo 8). He aquí una forma de utilizar el operador `++`:

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

Lo importante sobre el signo `=` es que no significa “es igual a” en el sentido algebraico. Debemos imaginar que significa “se convierte en” o “recibe”.

● Los operadores aritméticos

En esta sección le presentaremos un conjunto básico de operadores: los aritméticos, similares a los botones de su calculadora.

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

Observe que dividimos a los operadores en grupos para indicar su “precedencia”: el orden en el que se llevan a cabo. Por lo tanto, `*`, `/` y `%` se llevan a cabo antes que `+` y `-`. También podemos

usar paréntesis para agrupar los cálculos y forzarlos a que se calculen primero. Si un cálculo incluye operadores de la misma precedencia, se lleva a cabo 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
```

Recuerde que las instrucciones forman una secuencia, la cual se ejecuta de arriba hacia abajo en la página. Siempre que se utilicen paréntesis, los elementos dentro de ellos 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:

$3 + (2 * 4)$

Más adelante le explicaremos los detalles sobre los operadores / y %.

Por cuestión de estilo, observe que 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. Los utilizamos para que el programa sea más legible para el programador. Además, las líneas se pueden volver demasiado extensas para la pantalla. En este caso insertamos una nueva línea en una parte conveniente (aunque no a la mitad de un nombre) y aplicamos sangría a la segunda parte de la línea.

PRÁCTICA DE AUTOEVALUACIÓN

- 4.3 En el siguiente fragmento de código, ¿cuáles son los valores finales de las variables?

```
int a, b, c, d;
d = -8;
a = 1 * 2 + 3;
b = 1 + 2 * 3;
c = (1 + 2) * 3;
c = a + b;
d = -d;
```

Ahora conocemos las reglas. Pero aún hay obstáculos para el principiante. Veamos a continuación algunas fórmulas matemáticas y su conversión a Java. Vamos a suponer que todas las variables están declaradas como tipos `double` y que fueron inicializadas.

Versión de matemáticas	Versión de Java
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 Java, `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 con paréntesis.

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

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

Recuerde la regla de izquierda a derecha para los operadores de 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 es utilizar paréntesis adicionales; no hay ningún castigo en términos de tamaño o reducción en la velocidad del programa.

El uso de los operadores `+`, `-` y `*` es razonablemente intuitivo, pero la división es un poco más engañosa, ya que necesitamos diferenciar entre los tipos `int` y `double`. Los puntos importantes son:

- Cuando el operador `/` trabaja con dos números `double` o con una mezcla de `double` e `int`, se produce un resultado `double`. Tras bambalinas, cualquier valor `int` se considera como `double` para los fines del cálculo. Así es como funciona la división en una calculadora de bolsillo.
- Cuando el operador `/` trabaja con dos enteros, se produce un resultado entero. El resultado se trunca, lo cual significa que se borran los dígitos después del “punto decimal”. Esta **no** es la forma en que funciona una calculadora.

He aquí algunos ejemplos:

```
// división con valores double
double d;
d = 7.61 / 2.1;           // se convierte en 3.62
d = 10.6 / 2;             // se convierte en 5.3
```

En el primer caso, la división se lleva a cabo de la forma esperada. En el segundo caso, el número 2 se trata como `2.0` (es decir, un `double`) y la división se lleva a cabo.

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 la división con enteros da el resultado esperado. Se produce la respuesta exacta de 2. En el segundo caso el resultado es 2, debido a que se trunca el verdadero resultado. En el tercero caso se trunca la respuesta “correcta” de 0.75, con lo cual obtenemos un 0.

PRÁCTICAS DE AUTOEVALUACIÓN

- 4.4 Mi salario es de \$20,000 y estoy de acuerdo en darle a usted la mitad del mismo, para lo cual utilizaré el siguiente cálculo:

```
int mitad = 20000 * (1 / 2);
```

¿Cuánto recibirá usted?

- 4.5 Indique los valores con los que terminan **a**, **b**, **c** y **d** después de realizar los siguientes cálculos:

```
int a, b, c, d;
a = 7 / 3;
b = a * 4;
c = (a + 1) / 2;
d = c / 3;
```

● El operador %

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

Por ende, el operador % nos da el residuo como si se hubiera llevado a cabo una división. Ve 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 veces)
```

Calce mencionar que el operador % también funciona con números **double**, aunque el uso más frecuente de % es con tipos **int**. Veamos ahora un problema con un residuo: convertir un número

entero de centavos en dos cantidades —el número 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
```

PRÁCTICA DE AUTOEVALUACIÓN

- 4.6 Complete el siguiente fragmento de código. Agregue instrucciones de asignación para dividir `totalSegundos` en dos variables: `minutos` y `segundos`.

```
int totalSegundos = 307;
```

● 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. Java cuenta con el tipo de datos `String`; las variables `String` pueden guardar cualquier número de caracteres. He aquí un ejemplo del uso de cadenas:

```
String primerNombre = "Mike ";
String apellidoPaterno, nombreCompleto;
String saludo;
apellidoPaterno = "Parr";
nombreCompleto = primerNombre + apellidoPaterno;
saludo = "Saludos de " + nombreCompleto; // se convierte en "Saludos de Mike
Parr"
```

En el ejemplo anterior declaramos algunas variables `String` y les asignamos valores iniciales mediante el uso de comillas dobles. Observe la `S` mayúscula en `String`.

Después utilizamos 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 signo `=`, de forma similar a la asignación numérica.

Las siguientes líneas ilustran el uso del operador `+`, que (al igual 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 `Mike Parr`.

Además hay un amplio rango de métodos de cadenas que proporcionan operaciones tales como búsqueda y modificación de cadenas. Hablaremos sobre estos métodos en el capítulo 15.

Anteriormente mencionamos que el operador `/` considera a los elementos que divide como números `double` si uno de ellos es `double`. El operador `+` trabaja en forma similar con las cadenas. He aquí un ejemplo:

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

En este ejemplo el operador `+` detecta que `nombre` es una variable `String` y convierte el valor de `i` en una cadena antes de unir ambas variables. Éste es un método abreviado conveniente para evitar la conversión explícita que veremos a continuación. Pero puede ser 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, el primer caso es el que ocurre. Java 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 colocamos:

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

primero se calcula la operación `2 + 3` y se obtiene un `5`. Por último se lleva a cabo la unión de las cadenas.

Sin embargo, la mayoría de los casos son simples, como en nuestro programa del área:

```
JOptionPane.showMessageDialog(null, "El área es: " + área);
```

El cuadro de mensaje (que presentamos en el capítulo 2) puede mostrar una cadena. En el ejemplo anterior, el cuadro de mensaje muestra una cadena entre comillas unida con la cadena que representa a un entero. La alternativa es que podemos hacer la conversión explícita utilizando el método `toString` como se muestra a continuación.

PRÁCTICA DE AUTOEVALUACIÓN

- 4.7** Los cuadros de mensaje pueden mostrar una cadena de texto. ¿Qué muestran en pantalla los siguientes cuadros de mensaje?

```
JOptionPane.showMessageDialog(null,
    "5" + "5" + 5 + 5);
JOptionPane.showMessageDialog(null,
    "5" + "5" + (5 + 5));
```

● Conversiones entre cadenas y números

Uno de los usos más importantes del tipo de dato `String` es en las operaciones de entrada y salida, en donde procesamos los datos que introduce el usuario y mostramos los resultados en la pantalla. Muchas de las clases de GUI de Java trabajan con cadenas de caracteres en vez de números, por lo cual necesitamos saber cómo realizar conversiones entre números y cadenas.

Para convertir un cálculo o una variable numérica (una expresión en general) en una cadena, podemos utilizar los métodos `toString` de las clases `Integer` y `Double`. He aquí algunos ejemplos:

```

String s1, s2;
int num = 44;
double d = 1.234;
s1 = Integer.toString(num);    // s1 es "44"
s2 = Double.toString(d);       // s2 es "1.234"

```

Por lo general, el nombre de un método (como `toString`) va precedido de un objeto con el que debe trabajar, pero aquí suministramos el nombre de una clase (`Integer` o `Double`). Los métodos que funcionan de esta forma se denominan estáticos (`static`); cada vez que los utilicemos debemos identificar la clase a la que pertenecen. Las clases `Double` e `Integer` contienen herramientas adicionales para los tipos `double` e `int`. Tome en cuenta que (como siempre) los nombres de las clases empiezan con mayúscula. En el capítulo 9 veremos los métodos estáticos.

Para convertir cadenas en números utilizamos los métodos “parse” de las clases `Double` e `Integer`. El término *parse* se utiliza en el sentido de explorar un bloque de texto para examinarlo. A diferencia de los métodos que utilizamos en el capítulo 2 (como `drawLine`), estos métodos devuelven un valor cuando se les invoca y podemos guardar el valor devuelto en una variable, o podemos usarlo de alguna otra forma. He aquí algunos ejemplos que muestran cómo se puede convertir una cadena de texto en un valor `int` o `double`:

```

int i;
double d;
String s1 = "1234";
String s2 = "1.23";
i = Integer.parseInt(s1);
d = Double.parseDouble(s2);

```

Los métodos `parseInt` y `parseDouble` requieren un parámetro, el cual debe ser de tipo `String`. Podríamos proveer una cadena de texto entre comillas, una variable o una expresión de cadena de texto.

Como veremos más adelante, el usuario puede haber escrito la cadena de texto y, por ende, podría contener caracteres que no se permitan en los números. Los métodos `parseInt` y `parseDouble` detectarán esos errores y el programa terminará. En el capítulo 16 veremos cómo puede un programa detectar esos errores (“excepciones”) y pedir al usuario que vuelva a escribir un número. Por ahora vamos a suponer que el usuario siempre introducirá datos correctos.

PRÁCTICA DE AUTOEVALUACIÓN

- 4.8** En el siguiente código, ¿cuáles son los valores finales de `m`, `n` y `s`?

```

int m, n;
String s;
String v = "3";
m = Integer.parseInt(v + v + "4");
n = Integer.parseInt(v + v) + 4;
s = Integer.toString(Integer.parseInt(v)
+ Integer.parseInt(v)) + "4";

```

Cuadros de mensaje y de entrada

Aquí vamos a analizar los cuadros de diálogo con más detalle. En nuestro programa para calcular el área de un rectángulo, utilizamos un cuadro de mensaje para mostrar el valor del área. Este valor debe estar en forma de una cadena de texto. En vez de sólo mostrar el número, lo unimos en un mensaje:

```
JOptionPane.showMessageDialog(null, "El área es: " + área);
```

Recuerde nuestra descripción anterior sobre el operador `+` al aplicarlo a una cadena y un número: el número se convierte de manera automática en una cadena, por lo que no necesitamos usar `toString`. Sin embargo, podríamos haberlo convertido en forma explícita, como en el siguiente ejemplo:

```
int n = 33;
JOptionPane.showMessageDialog(null,
    "n es: " + Integer.toString(n));
```

Algunas veces tal vez tengamos que mostrar el número sin texto que lo acompañe. Tenga en cuenta que el siguiente código no se compilará debido a que el método `showMessageDialog` espera un valor `String` como parámetro:

```
JOptionPane.showMessageDialog(null, área); // NO - ¡no se compilará!
```

En vez de eso, debe usar:

```
JOptionPane.showMessageDialog(null, Integer.toString(área));
```

Por cierto, el primer parámetro para un cuadro de mensaje siempre será `null` en nuestros ejemplos. Esta palabra clave de Java hace que el cuadro de mensaje se posicione en el centro de la pantalla. También se puede posicionar sobre una ventana específica, pero no mostraremos esa posibilidad aquí.

La clase `JOptionPane` también provee un cuadro de entrada, el cual permite al usuario escribir una cadena. He aquí algunos ejemplos:

```
String primerNombre, apellidoPaterno;
primerNombre = JOptionPane.showInputDialog(
    "Escriba su primer nombre");
apellidoPaterno = JOptionPane.showInputDialog(
    "Escriba su apellido paterno");
```

El único parámetro del método `showInputDialog` es una indicación, que se utiliza para informar al usuario sobre los datos requeridos. Al hacer clic en “Aceptar”, la cadena se devuelve al programa para poder asignarla a una variable.

Ahora regresemos a nuestro programa del área. En realidad, es poco probable que conozcamos las medidas del rectángulo al momento de escribir el programa. Vamos a enmendar nuestro programa para que solicite las medidas al momento de ejecutarse. He aquí el programa modificado (en la figura 4.3 se muestra la pantalla del primer cuadro de entrada):

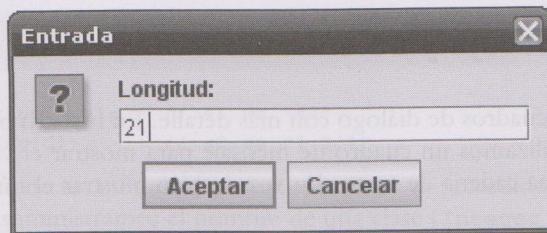


Figura 4.3 Pantalla de un cuadro de entrada del programa CuadrosDiálogoÁrea.

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class CuadrosDiálogoÁrea extends JFrame
    implements ActionListener {
    private JButton botón;
    public static void main(String[] args) {
        CuadrosDiálogoÁrea marco = new CuadrosDiálogoÁrea();
        marco.setSize(400, 300);
        marco.crearGUI();
        marco.setVisible(true);
    }
    private void crearGUI() {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container ventana = getContentPane();
        ventana.setLayout(new FlowLayout());
        botón = new JButton("Haga clic");
        ventana.add(botón);
        botón.addActionListener(this);
    }
    public void actionPerformed(ActionEvent event) {
        int área;
        int longitud;
        int anchura;
        String longitudString;
        String anchuraString;
        longitudString = JOptionPane.showInputDialog("Longitud:");
        longitud = Integer.parseInt(longitudString);
        anchuraString = JOptionPane.showInputDialog("Anchura:");
        anchura = Integer.parseInt(anchuraString);
        área = longitud * anchura;
        JOptionPane.showMessageDialog(null, "El área es: " + área);
    }
}

```

Recuerde que los cuadros de entrada proveen cadenas de texto al programa, las cuales se deben convertir en enteros mediante el método `parseInt`.

● Aplicación de formato al texto en cuadros de diálogo mediante \n

Al mostrar texto, a menudo es conveniente mostrarlo como varias líneas cortas en vez de una sola línea extensa. Para ello utilizamos un par especial de caracteres: el carácter “diagonal inversa” seguido de una `n`. Esta combinación, a la que comúnmente se le conoce como nueva línea, se encarga de separar la línea. La combinación `\n` se debe usar como parte del mensaje, entre comillas. He aquí un ejemplo. El programa se llama `CuadrosDiálogoDólares` y muestra los resultados de dividir una cantidad de centavos en dólares enteros y centavos restantes. Sólo mostramos el método `actionPerformed`. Observe la salida en la figura 4.4 y compárela con el uso de `\n` en el cuadro de mensaje.

```
public void actionPerformed(ActionEvent event) {  
    int totalCentavos;  
    int dólares;  
    int centavosRestantes;  
    String totalCentavosString;  
  
    totalCentavosString = JOptionPane.showInputDialog(  
        "Escriba su monto, en centavos");  
    totalCentavos = Integer.parseInt(totalCentavosString);  
    dólares = totalCentavos / 100;  
    centavosRestantes = totalCentavos % 100;  
    JOptionPane.showMessageDialog(null,  
        totalCentavosString + " centavos se dividen en:\n" +  
        dólares + " dólares\n" +  
        centavosRestantes + " centavos.");  
}
```

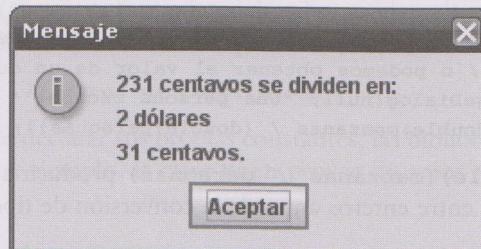


Figura 4.4 Salida del programa `CuadrosDiálogoDólares`.

● Conversiones entre números

Algunas veces necesitamos convertir valores numéricos de un tipo a otro. Los casos más comunes son la conversión de un `int` a un `double` y de un `double` a un `int`.

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

Antes de proporcionarle la respuesta, veamos 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 importantes son:

- Para asignar un `int` a un `double` no se requiere de programación adicional. Es un proceso seguro ya que no se puede perder información; no hay números decimales por los cuales precuparse.
- Al asignar un `double` a un `int` se pueden perder los dígitos después del punto decimal, ya que no caben en el entero. Debido a esta pérdida potencial de información, Java requiere que especifiquemos esta conversión de manera explícita mediante la conversión de tipos o *casting* (utilizaremos 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)`. El valor se trunca, eliminando los dígitos después del 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;    // o podemos obtener el valor de un cuadro de texto
int personas = 4;    // o podemos obtener el valor de un cuadro de texto
 JOptionPane.showMessageDialog(null, "Una persona recibe: " +
 Double.toString((double)manzanas / (double)personas));
```

Tome en cuenta que `(double)(manzanas / personas)` produciría la respuesta incorrecta, ya que se realizaría una división entre enteros antes de la conversión de tipos.

PRÁCTICA DE AUTOEVALUACIÓN

4.9 ¿Cuáles son los valores de `a`, `b`, `c`, `i`, `j` y `k` después de ejecutar el siguiente código?

```
int i, j, k;
double a, b, c;
int n = 3;
double y = 2.7;
i = (int)y;
j = (int)(y + 0.6);
k = (int)((double)n + 0.2);
a = n;
b = (int)n;
c = (int)y;
```

Constantes: uso de final

Hasta ahora, hemos usado variables; es decir, elementos cuyos valores cambian a medida que se ejecuta el programa. Pero algunas veces tenemos valores que nunca cambian. He aquí un ejemplo:

```
double millas, km = 4.7;
millas = km * 0.6214;
```

El significado de esta línea no está claro. ¿Qué es lo que representa `0.6214`? De hecho, es el número de millas en un kilómetro, y el valor nunca cambia.

He aquí cómo podemos reformular el programa mediante la palabra clave `final` de Java:

```
final double millasPorKm = 0.6214;
double millas, km = 4.7;
millas = km * millasPorKm;
```

La palabra `final` indica que ahora la variable especificada tiene su valor final fijo, el cual no puede cambiar cuando el programa se ejecuta. En otras palabras, `millasPorKm` es una constante. Las constantes pueden ser `double`, `int`, `String` o `boolean`.

Cabe mencionar que algunos programadores usan mayúsculas para las constantes (como `MILLASPORKM`), para poder distinguirlas claramente de las variables.

Si usted escribe código (ya sea en forma intencional o accidental) para modificar la constante, se produce un error de compilación, como en el siguiente ejemplo:

```
millasPorKm = 2.1; // error de compilación
```

Además de que usted puede declarar sus propias constantes, las bibliotecas de Java también proveen constantes matemáticas. Por ejemplo, la clase `Math` provee un valor para `pi`, que se puede usar de esta forma:

```
double radio = 1.4, circunferencia;
circunferencia = 2 * Math.PI * radio;
```

Los beneficios de utilizar constantes son:

- Mejora la legibilidad del programa al utilizar nombres en vez de números.
- El uso de constantes minimiza los errores de escritura, ya que al usar repetidas veces un número largo en un programa se pueden cometer errores. Por ejemplo, podríamos cometer un error al escribir `0.6124` en vez de `0.6214`. Dichos errores son difíciles de detectar. En cambio, si escribimos `millasPorKm` como `millasBorKm` se producirá un error de compilación y podremos corregir la ortografía.

PRÁCTICA DE AUTOEVALUACIÓN

- 4.10** Hay 2.54 cm en una pulgada. Declare una constante llamada `cmPorPulg` con el valor correcto. Muestre cómo se podría usar en un cálculo para convertir pulgadas en centímetros.

● El papel que desempeñan las expresiones

Aunque hemos recalado que las expresiones (cálculos) pueden formar parte del lado derecho de las instrucciones de asignación, también pueden estar en otros lugares. De hecho, podemos colocar una expresión `int` en cualquier parte en donde podamos colocar un valor `int` individual. Considere el método `drawLine` que vimos en ejemplos anteriores, el cual requiere cuatro enteros para especificar el inicio y el final de la línea a dibujar. Podríamos (si fuera conveniente) sustituir los números con variables o con expresiones:

```
int x = 100;
int y = 200;
papel.drawLine(100, 100, 110, 110);
papel.drawLine(x, y, x + 50, y + 50);
papel.drawLine(x * 2, y - 8, x * 30 - 1, y * 3 + 6);
```

Las expresiones se calculan y los valores resultantes se pasan a `drawLine` para que los utilice. He aquí otro ejemplo. El método `parseInt` requiere un parámetro de cadena de texto y el cuadro de entrada devuelve una cadena de texto. Podemos combinar éstos en una instrucción, como en el siguiente ejemplo:

```
int edad;
edad = Integer.parseInt(JOptionPane.showInputDialog(null,
    "Escriba la edad"));
```

En el ejemplo anterior no tuvimos que inventar una variable de cadena temporal para transmitir un valor de cadena de texto del cuadro de entrada a `parseInt`.

Fundamentos de programación

- Una variable tiene un nombre que el programador puede elegir.
- Una variable tiene un tipo que el programador puede elegir.
- Una variable contiene un valor.
- El valor de una variable se puede modificar mediante una instrucción de asignación.
- Las constantes proveen valores que no cambian. Se les puede asignar un nombre significativo.

Errores comunes de programación

- Tenga cuidado con la ortografía en 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 “I” (uno) en vez de una “L” minúscula. El compilador de Java se quejará de 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 programación al principio. Aunque el compilador de Java nos da una indicación de la posición en la que cree que se encuentra el error, en realidad éste podría estar en una línea anterior.
- Los paréntesis deben estar balanceados; debe haber el mismo número de “(“ que de “)”.
- Al utilizar números a través de cuadros de mensaje, recuerde usar las herramientas de conversión de cadenas o unir el número a una cadena.
- Al multiplicar elementos debe colocar un * entre ellos, mientras que en matemáticas se omite este signo. Al dividir elementos, recuerde 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**.

Secretos de codificación

- Para declarar variables indicamos su tipo y su nombre, como en:

```
int miVariable;  
String tuVariable = "¡Saludos desde acá!";
```

- Los tipos más útiles son `int`, `double` y `String`.
- Los principales operadores aritméticos son `*`, `/`, `%`, `+` y `-`.
- El operador `+` se utiliza para unir cadenas de texto.
- Los operadores `++` y `--` se pueden utilizar para incrementar y decrementar.
- Podemos convertir números a cadenas con los métodos `Integer.toString` y `Double.toString`.
- Podemos convertir cadenas a números con los métodos `Integer.parseInt` y `Double.parseDouble`.
- 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` y `string`.
- Los operadores `+`, `-`, `*`, `/`, `%`.
- `++` y `--` para incremento y decremento.
- `=` para asignación.
- `final` para las constantes.
- Conversión de tipos: las clases `Integer` y `Double`, los operadores de conversión (`double` e `int`).
- La clase `JOptionPane` y sus métodos `showMessageDialog` y `showInputDialog`.
- El uso de `\n` para representar una nueva línea en una cadena.

Resumen

- Las variables se utilizan para contener (guardar) valores. Mantienen su valor hasta que éste se modifica en forma explícita (por ejemplo, mediante otra instrucción de asignación).
- Los operadores operan sobre valores.
- Una expresión es un cálculo que produce un valor. Se puede utilizar en diversas situaciones, incluyendo el lado derecho de una asignación o como parámetro para la llamada a un método.

Ejercicios

En estos ejercicios, seleccione un nuevo nombre de clase para cada programa, pero base todos los ejercicios en el programa **CalculoÁrea**. Use cuadros de mensaje para las operaciones de salida y cuadros de entrada para las operaciones de entrada.

4.1 Escriba un programa para calcular el volumen de una caja, dadas sus tres dimensiones.

4.2 (a) Dado el siguiente valor:

```
double radio = 7.5;
```

utilice instrucciones de asignación para calcular la circunferencia de un círculo, el área de un círculo y el volumen de una esfera, con base en el mismo radio. Muestre los resultados con cuadros de mensaje. El mensaje debe indicar qué es el resultado, en vez de sólo mostrar un número. Use la constante **Math.PI**, como en:

```
volumen = (4 * Math.PI / 3) * radio * radio * radio;
```

4.3 Escriba un programa que reciba como entrada tres calificaciones de exámenes representadas por enteros y muestre la calificación promedio como un valor **double**. Verifique su respuesta con una calculadora.

4.4 Escriba un programa que reciba como entrada tres calificaciones de exámenes representadas por números **double** y muestre la calificación promedio como un valor **double**. Verifique su respuesta con una calculadora.

4.5 Suponga que un grupo de personas tienen que pagar impuestos del 20% de sus ingresos. Obtenga el valor del ingreso por medio de un cuadro de entrada; después calcule y muestre la cantidad inicial, la cantidad después de las deducciones y la cantidad que se dedujo. Asegúrese de que los usuarios puedan entender fácilmente la salida de su programa. Modifique su programa para que utilice una constante **final** para la tasa de impuestos.

4.6 Use tipos **int** para escribir un programa que convierta una temperatura en Fahrenheit a su equivalente en Celsius (centígrados). La fórmula es:

```
c = (f - 32) * 5 / 9
```

- 4.7** Escriba un programa que reciba como entrada un número entero de segundos; después debe convertir este valor en horas, minutos y segundos. Por ejemplo, 3669 segundos deberían mostrarse en el cuadro de mensaje como:

H:1 M:1 S:9

- 4.8** Este problema está relacionado con las resistencias eléctricas, las cuales se "resisten" al flujo de la corriente eléctrica que pasa a través de ellas. Las mangueras son una analogía: una manguera delgada tiene una alta resistencia al agua y una gruesa, una baja resistencia. Imagine que tenemos dos mangueras. Si las conectamos en serie se obtendría una mayor resistencia, y si las conectamos en paralelo se reduciría la resistencia (ya que obtendríamos el equivalente a una manguera más gruesa). Calcule y muestre la resistencia en serie con base en:

$$\text{serie} = r_1 + r_2$$

y la resistencia en paralelo, con base en:

$$\text{paralelo} = \frac{r_1 \times r_2}{r_1 + r_2}$$

Reciba como entrada los valores para r_1 y r_2 .

- 4.9** Suponga que necesitamos instalar cierto software en una máquina europea dispensadora de bebidas. He aquí los detalles: todos los productos cuestan menos de 1 euro (100 centavos de euro) y una moneda de 1 euro es la denominación más alta que podemos insertar. Dado el monto insertado y el costo del producto, su programa debe regresar cambio utilizando el menor número de monedas. Por ejemplo, si un producto cuesta 45 centavos y pagamos con 100 centavos, el resultado debería ser una serie de cuadros de mensaje (uno para cada moneda) de la siguiente forma:

```
La cantidad de monedas de 50 centavos es 1
La cantidad de monedas de 20 centavos es 0
La cantidad de monedas de 10 centavos es 0
La cantidad de monedas de 5 centavos es 1
La cantidad de monedas de 2 centavos es 0
La cantidad de monedas de 1 centavo es 0
```

Sugerencia: trabaje con centavos y utilice el operador % todas las veces que pueda. Las monedas de euro son:

100, 50, 20, 10, 5, 2, 1

Respuestas a las prácticas de autoevaluación

- 4.1 **volumen** – permitido, estilo correcto.
ÁREA – permitido, pero es preferible usar **área**.
Longitud – permitido, pero es preferible usar la 'l' minúscula.
3lados - no está permitido ya que empieza con un dígito.
lado1 – permitido, estilo correcto.
lonitud – permitido, aun cuando está mal escrita la palabra "longitud".
Misalario – permitido, pero es preferible usar **miSalario**.
su salario – no está permitido (no se permiten espacios en medio de un nombre).
tamañoPantalla - permitido, estilo correcto.
- 4.2 En la línea 2, **b** no está inicializada. Se producirá un error de compilación debido a que estamos tratando de almacenar una variable no asignada en **a**.
- 4.3 Los valores finales de **a**, **b**, **c** y **d** son 5, 7, 12 y 8.
- 4.4 Por desgracia usted no recibe nada, ya que primero se calcula **(1 / 2)** y se obtiene un 0. Es mejor que multiplique por 0.5.
- 4.5 Los valores finales de **a**, **b**, **c** y **d** son 2, 8, 1 y 0.
- 4.6
- ```
int totalSegundos = 307;
int segundos, minutos;
minutes = totalSegundos / 60;
segundos = totalSegundos % 60;
```
- 4.7 Los cuadros de mensaje muestran las cadenas **5555** y **5510**, respectivamente.  
En el primer caso procedemos de izquierda a derecha, uniendo cadenas. En el segundo caso se llevan a cabo las operaciones dentro de los paréntesis y se obtiene el entero **10**. Despues se lleva a cabo la unión de cadenas.
- 4.8 Los valores finales de **m**, **n** y **s** son 334, 37 y 64.
- 4.9 Los valores de las variables **int i, j y k** son 2, 3 y 3, y los valores de las variables **double a, b y c** son 3.0, 3.0 y 2.0.
- 4.10
- ```
final double cmPorInch = 2.54;
double cm, pulgadas = 23.6;
cm = pulgadas * cmPorPulg;
```