capítulo 5

Estructuras de selección

objetivos

En este capítulo aprenderá a:

- Distinguir entre una sentencia simple y una compuesta.
- Entender el concepto de selección.
- Construir sentencias de selección simple.
- Diseñar sentencias de selección múltiple.
- Crear un menú con múltiples alternativas.

introducción

Un programa escrito de modo secuencial ejecuta una sentencia después de otra; comienza con la primera y prosigue hasta la última, cada una se ejecuta una sola vez; el modo secuencial es adecuado para resolver problemas sencillos. Sin embargo, para solucionar problemas de tipo general, se necesita la capacidad de controlar cuáles son las sentencias a ejecutar en cada momento. Las estructuras o construcciones de control determinan la secuencia o flujo de ejecución de las sentencias; se dividen en tres grandes categorías en función del flujo de ejecución: secuencia, selección y repetición.

Este capítulo considera las sentencias `if` y `switch`, estructuras selectivas o condicionales que controlan si una sentencia o lista de sentencias se ejecutan en función del cumplimiento o no de una condición; para soportar estas construcciones, Java tiene el tipo lógico `boolean`.

5.1 Estructuras de control

Las estructuras de control determinan el comportamiento de un programa; permiten combinar instrucciones o sentencias individuales en una simple unidad lógica con un punto de entrada y otro de salida, se organizan en tres tipos que sirven para controlar el flujo de la ejecución: secuencia, selección o decisión y repetición. Además, ejecutan una sentencia simple o compuesta; esta última es un conjunto de sentencias encerradas entre llaves ({}) que se utiliza para especificar un flujo secuencial; así se representa:

```
{
  sentencia 1;
  sentencia 2;
  .
  .
  .
  sentencia n;
}
```

El control fluye de la sentencia 1 a la 2 y así sucesivamente; sin embargo, existen problemas que requieren etapas con dos o más opciones o alternativas a elegir en función del valor de una condición o expresión.

5.2 Sentencia `if`

En Java, la estructura de control de selección principal es una sentencia `if`; la cual, tiene dos alternativas o formatos posibles, el más sencillo tiene la sintaxis siguiente:

```
if (expresión) Acción
```

La sentencia `if` funciona de la siguiente manera: cuando se alcanza, se evalúa la siguiente expresión entre paréntesis; si `expresión` es verdadera se ejecuta `Acción`, en caso contrario no se efectúa y sigue la ejecución en la siguiente sentencia. `Acción` es una sentencia simple o compuesta; la figura 5.1 muestra un diagrama de flujo que indica el flujo de ejecución del programa.

EJEMPLO 5.1

Prueba de divisibilidad. Éste es un programa en el que se introducen dos números enteros y mediante una sentencia de selección se determina si son divisibles.

```
import java.util.Scanner;
class Divisible
{
  public static void main(String[] a)
```

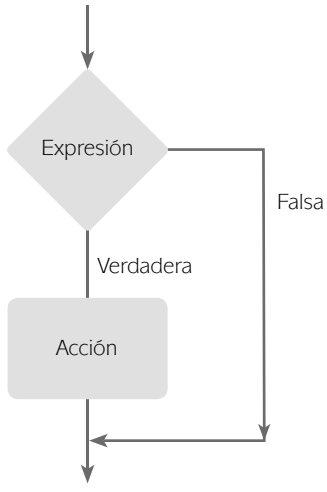

Figura 5.1 Diagrama de flujo de una sentencia básica `if`.

```
  {
    int n, d;
    Scanner entrada = new Scanner(System.in);
    System.out.println("Introduzca dos enteros:");
    n = entrada.nextInt();
    d = entrada.nextInt();
    if (n%d == 0)
     System.out.println(n +" es divisible por "+d);
  }
}
```

Ejecución

```
Introduzca dos enteros:
36
4
36 es divisible por 4
```

Este programa lee dos números enteros y comprueba cuál es el valor del resto de la división `n` entre `d` (`n%d`); si es cero, `n` es divisible entre `d`; en este caso `36` es divisible entre `4` y el resto es `0`.

EJEMPLO 5.2

Representar la superación de un examen considerando `>=5`, aprobado.

```
import java.util.Scanner;
class NotaAprobado
{
  public static void main(String[] a)
  {
    int nota;
    Scanner entrada = new Scanner(System.in);
    System.out.println("Introduzca nota a analizar:");
    nota = entrada.nextInt();
    if (nota > 5)
      System.out.println("Prueba superada ");
  }
}
```

EJEMPLO 5.3

El programa selecciona el signo que tiene un número real.

```
import java.util.Scanner;
class Positivo
{
  public static void main(String[] a)
  {
    float numero;
    Scanner entrada = new Scanner(System.in);
    System.out.println("Introduzca un número real");
    numero = entrada.nextFloat();
    // comparar número con cero
    if (numero > 0)
      System.out.printl(numero + " es mayor que cero");
  }
}
```

Ejecución

```
Introduzca un número real
5.4
5.4 es mayor que cero
```

¿Qué sucede si se introduce un número negativo en lugar de uno positivo? Nada; el programa es tan simple que sólo comprueba si el número es mayor que cero. La versión del programa que se presenta a continuación añade un par de sentencias `if`: una comprueba si el número que se introduce es menor que cero, mientras que la otra comprueba si el número es igual a cero.

```
import java.util.Scanner;

class SignoNumero
{
  public static void main(String[] a)
  {
    float numero;
    Scanner entrada = new Scanner(System.in);
    System.out.println("Introduzca un número real");
    numero = entrada.nextFloat();
    // comparar número con cero
    if (numero > 0)
      System.out.printl(numero + " es mayor que cero");
    if (numero < 0)
      System.out.println(numero + " es menor que cero");
    if (numero == 0)
      System.out.println(numero + " es igual que cero");
  }
}
```

5.3 Sentencia `if` de dos alternativas: `if-else`

Un segundo formato de `if` es `if-else`, cuyo formato tiene la siguiente sintaxis:

```
if (expresión)
  acción 1
else
  acción 2
```

En este formato `acción 1` y `acción 2` son, de forma individual, una única sentencia que termina en un punto y coma, o un grupo de sentencias entre llaves; `expresión` se evalúa cuando se ejecuta la sentencia: si es verdadera, se efectúa `acción 1`; en caso contrario se ejecuta `acción 2`, la figura 5.2 muestra su semántica.

Por ejemplo:

1.
```
if (salario > 100000)
  salario_neto = salario - impuestos;
else
  salario_neto = salario;
```

Si `salario` es mayor que 100 000, se calcula el salario neto restándole los impuestos; en caso contrario (`else`), el salario neto es igual al salario bruto.

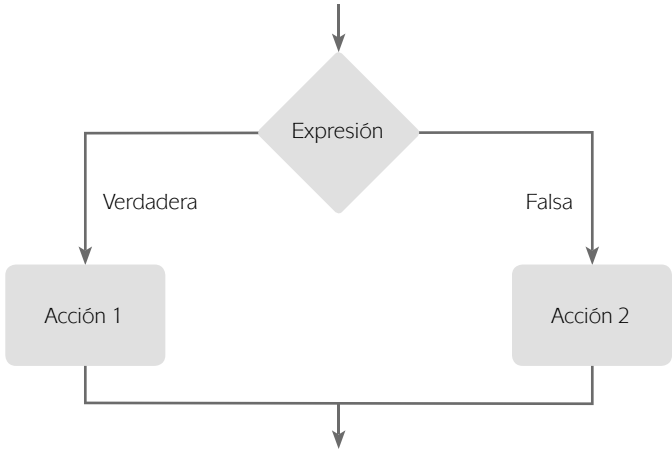

Figura 5.2 Diagrama de flujo de la representación de una sentencia `if-else`.

2.
```
if (Nota >= 5)
  System.out.println("Aprobado");
else
  System.out.println("Suspenso");
```

Formatos

1.
```
if (expresión_lógica)
  sentencia
```
2.
```
if (expresión lógica)
  sentencia1
else
  sentencia 2
```
3.
```
if (expresión lógica) sentencia1 else sentencia2
```

Si `expresión lógica` es verdadera, se ejecuta `sentencia` o `sentencia1`; si es falsa, se lleva a cabo `sentencia2`.

Por ejemplo:

1.
```
if (x > 0.0) producto * = X; else producto = 0.0;
  producto = producto * x;
```
2.
```
if (x != 0.0)
  producto = producto * x;
// se ejecuta la sentencia de asignación cuando x no es igual a 0.
// en este caso producto se multiplica por x y el nuevo valor se
// guarda en producto reemplazando el valor antiguo.
// si x es igual a 0, la multiplicación no se ejecuta.
```

EJEMPLO 5.4

Prueba de divisibilidad; es el programa 5.1 al que se añadió la cláusula `else`; se leen dos números enteros y con el operador módulo (`%`) se comprueba si son divisibles o no.

```
import java.util.Scanner;
class Divisible
{
  public static void main(String[] a)
  {
    int n, d;
    Scanner entrada = new Scanner(System.in);
    System.out.print("Introduzca primer valor ");
    n = entrada.nextInt();
    System.out.print("Introduzca segundo valor ");
    d = entrada.nextInt();
    if (n%d == 0)
      System.out.println(n +" es divisible entre "+d);
    else
      System.out.println(n +" no es divisible entre "+d);
  }
}
```

Ejecución

```
Introduzca primer valor 36
Introduzca segundo valor 5
36 no es divisible entre 5
```

Cabe mencionar que 36 no es divisible entre 5 pues produce un residuo de 1 (`n%d ==0`, es falsa), y se ejecuta `else`.

EJEMPLO 5.5

Este programa determina el mayor de dos números ingresados y lo visualiza en pantalla; la entrada de los datos enteros se realiza de la misma forma que en el ejemplo anterior; por último, la selección del mayor se realiza con el operador > y la sentencia `if`.

```
import java.util.Scanner;
class MayorNumero
{
  public static void main(String[] a)
  {
     int n1, n2;
     Scanner entrada = new Scanner(System.in);
     System.out.print("Introduzca primer entero: ");
     n1 = entrada.nextInt();
     System.out.print("Introduzca segundo entero: ");
     n2 = entrada.nextInt();
     if (n1 > n2)
       System.out.println(" El mayor es "+n1);
     else
       System.out.println(" El mayor es "+n2);
  }
}
```

NOTA

La condición es (`n1 > n2`); si `n1` es mayor que `n2` la condición es verdadera; en caso de que `n1` sea menor o igual que `n2`, la condición es falsa; así se imprime `n1` cuando es mayor que `n2`, como en el ejemplo de la ejecución.

Ejecución

```
Introduzca primer entero: 172
Introduzca segundo entero: 54
El mayor es 172
```

5.4 Sentencias `if-else` anidadas

Hasta este momento, las sentencias `if` implementan decisiones que implican una o dos opciones; pero en esta sección, se mostrará que una sentencia `if` es anidada cuando alguna de las ramas, sin importar si es verdadera o falsa, también es `if`; entonces se puede utilizar para tomar decisiones con varias opciones o multiopciones.


sintaxis

```
if (condición 1)
  sentencia 1
else if (condición 2)
  sentencia 2
...
else if (condición n)
  sentencia n
else
  sentencia e
```


EJEMPLO 5.6

Se incrementan contadores de números positivos, números negativos o ceros.

```
if (x > 0)
  num_pos = num_pos + 1;
else
  if (x < 0)
    num_neg = num_neg + 1;
  else
    num_ceros = num_ceros + 1;
```

La sentencia `if` que está anidada tiene tres variables (`num_pos`, `num_neg` y `num_ceros`), incrementa una de ellas en `1`, dependiendo de si `x` es mayor que cero, menor que cero o igual a cero, respectivamente; su ejecución se realiza de la siguiente forma: se comprueba la primera condición `(x > 0)` y, si es verdadera, `num_pos` se incrementa en `1` y se omite el resto de la sentencia; si es falsa, se comprueba la segunda condición `(x < 0)` y, si ésta es verdadera, `num_neg` se incrementa en uno; en caso contrario `num_ceros` se aumenta en uno. Observe que la segunda condición sólo se comprueba si la primera condición es falsa.

5.4.1 Sangría en las sentencias `if` anidadas

El formato multibifurcación se compone de una serie de sentencias `if` anidadas que se pueden escribir en cada línea; la sintaxis de multibifurcación anidada es:

- Formato 1:


sintaxis

```
if (expresión_lógica1)
  sentencia 1
```

```
  else
    if (expresión_lógica2)
      sentencia2
    else
      if (expresión_lógica3)
        sentencia 3
      else
        if (expresión_lógica4)
          sentencia 4
        else
          sentencia 5
```

- Formato 2:

sintaxis

```
if (expresión_lógica1)
  sentencia 1
else if (expresión_lógica2)
  sentencia2
else if (expresión_lógica3)
  sentencia3
else if (expresión_lógica4)
  sentencia 4
else
  sentencia5
```

Por ejemplo:

```
if (x > 0)
  if (y > 0)
    z = Math.sqrt(x) + Math.sqrt(y);
  else
    System.out.println("*** Imposible calcular z");
```

EJEMPLO 5.7

Comparación de un valor entero leído desde el teclado; muestra las sentencias compuestas `if-else`.

```
import java.util.Scanner;
class pruebaCompuesta
{
  public static void main(String[] a)
  {
    int numero;
    Scanner entrada = new Scanner(System.in);
    System.out.print("Introduzca un valor entero: ");
    numero = entrada.nextInt();
    // comparar número a cero
    if (numero > 0)
    {
```

```
        System.out.println(numero + "es mayor que cero");
        System.out.println
          ("Pruebe de nuevo introduciendo un número negativo");
      }
      else if (numero < 0)
      {
        System.out.println(numero + "es menor que cero");
        System.out.println
          ("Pruebe de nuevo introduciendo un número positivo");
      }
      else
      {
        System.out.println(numero + "es igual a cero");
        System.out.println
          ("¿Por qué no introduce un número negativo?");
      }
    }
  }
```

5.4.2 Comparación de sentencias `if` anidadas y secuencias de sentencias `if`

Los programadores tienen dos opciones: 1) usar una secuencia de sentencias `if` o 2) emplear una única sentencia `if` anidada; es decir, la sentencia `if` del ejemplo 5.6 se puede reescribir como una secuencia de sentencias `if`:

```
if (x > 0)
  num_pos = num_pos + 1;
if (x < 0)
  num_neg = num_neg + 1;
if (x == 0)
  num_ceros = num_ceros + 1;
```

Aunque la secuencia anterior es lógicamente equivalente a la original, no es tan legible ni eficiente; contrario a la sentencia `if` anidada, ésta no muestra claramente cuál es la sentencia a ejecutar para un valor determinado de `x`. En cuanto a eficiencia, la sentencia `if` anidada se ejecuta más rápidamente cuando `x` es positivo, ya que la primera condición `(x >0)` es verdadera, lo que significa omitir la sentencia después del primer `else`; además, se comprueban siempre las dos condiciones en la secuencia; si `x` es negativa, se comprueban dos condiciones en las sentencias anidadas frente a las tres de las secuencias completas.

Una estructura típica `if-else` anidada permitida es:

```
if (numero > 0)
{
  // ...
}
else
{
  if (// ...)
  {
    // ...
  }
  else
  {
```

```
    if (// ...)
    {
      // ...
    }
  }
  // ...
}
```

EJEMPLO 5.8

Éstas son tres formas de escribir sentencias `if` anidadas:

1. La siguiente manera es muy engorrosa; su sintaxis es correcta pero su uso constituye una mala práctica en la programación.

```
if (a > 0) if (b > 0) ++a; else if (c > 0)
if (a < 5) ++b; else if (b < 5) ++c; else --a;
else if (c < 5) --b; else --c; else a = 0
```

2. Ésta forma es adecuada: una simple lectura del código muestra la sentencia seleccionada si se cumple la condición, o bien, aparece la estipulada en caso contrario; también se pueden observar los `if` anidados.

```
if (a > 0)
  if (b > 0) ++a;
  else
    if (c > 0)
      if (a < 5) ++b;
      else
       if (b < 5) ++c;
       else --a;
     else
      if (c < 5) --b;
       else --c;
else
  a = 0;
```

3. Ésta también es adecuada; quizá es más legible que la anterior porque la sentencia seleccionada se encuentra en la siguiente línea con una sangría apropiada.

```
if (a > 0)
  if (b > 0)
    ++a;
  else if (c > o)
    if (a < 5)
      ++b;
    else if (b < 5)
      ++c;
    else
      --a;
  else if (c < 5)
    --b;
  else
    --c;
else
  a = 0;
```

EJEMPLO 5.9

Este programa calcula el mayor de tres números reales al tratarlos como si fueran de doble precisión. La entrada se realiza desde el teclado y, para realizar la selección, se comparan los pares de valores entre sí.

```
import java.util.Scanner;
class Mayorde3
{
  public static void main(String[] a)
  {
    double x,y,z;
    Scanner entrada = new Scanner(System.in);
    System.out.print("Introduzca primer número real");
    x = entrada.nextDoble();
    System.out.print("Introduzca segundo número real");
    y = entrada.nextDouble ();
    System.out.print("Introduzca el tercer número real");
    z = entrada.nextDouble();
    double mayor;
    if (x > y)
      if (x > z)
        mayor = x;
      else
        mayor = z;
    else
      if (y > z)
        mayor = y;
      else
        mayor = z;
    System.out.println("El mayor es "+mayor);
  }
}
```

Ejecución

```
Introduzca primer número real
77
Introduzca segundo número real
33
Introduzca el tercer número real
85
El mayor es 85
```

Análisis

Al ejecutar el primer `if`, la condición `(x > y)` es verdadera, entonces se efectúa el segundo `if`; en este último, la condición `(x > z)` es falsa, en consecuencia se ejecuta el primer `else: mayor = z`; se termina la sentencia `if` y se efectúa la línea que visualiza `El mayor es 85`.

5.5 Sentencia de control `switch`

En Java, `switch` es una sentencia que se utiliza para elegir una de entre múltiples opciones; es especialmente útil cuando la selección se basa en el valor de una variable simple o de una expresión simple denominada *expresión de control* o *selector*; el valor de dicha expresión puede ser de tipo `int` o `char` pero no de tipo `double`.


```
switch (selector)
{
  case etiqueta1 : sentencias1 ;
    break;
  case etiqueta2 : sentencias2 ;
    break;
  .
  .
  .
  case etiquetan : sentenciasn ;
    break;
  default: sentenciasd ; // opcional

}
```

La expresión de control o selector se evalúa y compara con cada una de las etiquetas de `case`; además, debe ser un tipo ordinal, por ejemplo, `int`, `char`, `bool` pero no `float` o `string`; cada etiqueta es un valor único, constante y debe tener un valor diferente de los otros. Si el valor de la expresión es igual a una de las etiquetas `case`, por ejemplo `etiqueta1`, entonces la ejecución comenzará con la primera sentencia de *`sentencias1`* y continuará hasta encontrar `break` o el final de `switch`.

El tipo de cada etiqueta debe ser el mismo que la expresión de selector; las expresiones están permitidas como etiquetas pero sólo si cada operando de la expresión es por sí misma una constante; por ejemplo, `4`, `+8` o `m*15`, siempre que `m` hubiera sido definido anteriormente como constante nombrada.

Si el valor del selector no está listado en ninguna etiqueta `case` no se ejecutará ninguna de las opciones a menos que se especifique una acción predeterminada. La omisión de una etiqueta `default` puede crear un error lógico difícil de prever; aunque ésta es opcional, se recomienda su uso, a menos de estar absolutamente seguro de que todos los valores de *selector* están incluidos en las etiquetas `case`.

5.5.1 Sentencia `break`

Para alterar el flujo de control de una sentencia de selección múltiple o de los bucles, existe la sentencia `break` la cual termina el bucle.


```
break;
break etiqueta;
```

Una sentencia `break` consta de la misma palabra reservada seguida por un punto y coma; cuando la computadora ejecuta las sentencias siguientes a una etiqueta `case`, continúa hasta que se alcanza una sentencia `break`; al hacerlo, la computadora termina la sentencia `switch`. Si se omiten las sentencias `break` después de ejecutar el código `case`, la computadora ejecutará el código siguiente hasta el próximo `case`.

- Ejemplo 1

```
switch (opcion)
{
  case 0:
    System.out.println("Cero!");
    break;
  case 1:
    System.out.println("Uno!");
    break;
  case 2:
    System.out.println("Dos!");
    break;
  default:
    System.out.println("Fuera de rango!");
}
```

- Ejemplo 2

```
switch (opcion)
{
  case 0:
  case 1:
  case 2:
    System.out.println("Menor que 3!");
    break;
  case 3:
    System.out.println("Igual a 3!");
    break;
  default:
    System.out.println("Mayor que 3!");
}
```

Java ofrece una sentencia `break` etiquetada que permite romper el flujo de control determinado por una sentencia compuesta; es útil para salir del control de bucles anidados, como se verá en el capítulo 6. La siguiente es una sentencia compuesta de la cual se sale si la condición `x < 0` se cumple:

```
bloque1:
{
  ...
  if (x < 0) break bloque1;  // salida del bloque
  ...
}
```

EJEMPLO 5.10

Comparación de las sentencias `if-else-if` y `switch`; se quiere determinar si un carácter `car` es vocal y escribir el resultado.

Solución con `if-else-if`.

```
if ((car == 'a') || (car == 'A'))
  System.out.println(car + " es una vocal");
else if ((car == 'e') || (car == 'E'))
  System.out.println(car + " es una vocal");
```

```
else if ((car == 'i') || (car == 'I'))
  System.out.println(car + " es una vocal");
else if ((car == 'o') || (car == 'O'))
  System.out.println(car + " es una vocal");
else if ((car == 'u') || (car == 'U'))
  System.out.println(car + " es una vocal");
else
  System.out.println(car + " no es una vocal");
```

Solución con `switch`.

```
switch (car)
{
  case 'a': case 'A':
  case 'e': case 'E':
  case 'i': case 'I':
  case 'o': case 'O':
  case 'u': case 'U':
    System.out.println(car + " es una vocal");
    break;
  default:
    System.out.println(car + " no es una vocal");
  }
```

EJEMPLO 5.11

Considerando un rango entre 1 y 10 para asignar la nota de un curso, el programa ilustra la selección múltiple con la sentencia `switch`.

```
import java.util.Scanner;
class Pruebacompuesta
{
  public static void main(String[] a)
  {
     int nota;
     Scanner entrada = new Scanner(System.in);
     System.out.print
       ("Introduzca calificación (1 - 10), pulse Intro:");
     nota = entrada.nextInt();
     switch (nota)
     {
       case 10:
       case 9 : System.out.println("Excelente.");
                break;
       case 8 :
       case 7 : System.out.println("Notable.");
                break;
       case 6 :
       case 5 : System.out.println("Aprobado.");
                break;
       case 4 :
       case 3 :
       case 2 :
       case 1 :
       case 0 : System.out.println("Suspendido.");
                break;
```

```
        default:
                System.out.println("no es posible esta nota.");
    }
    System.out.println("Final de programa.");
 }
```

Cuando se ejecuta la sentencia `switch`, se evalúa `nota` si el valor de la expresión es igual al valor de una etiqueta; entonces se transfiere el flujo de control a las sentencias asociadas con la etiqueta correspondiente. Si ninguna etiqueta coincide con el valor de `nota`, se ejecuta la sentencia `default` y las siguientes; por lo general, la última sentencia que sigue a `case` es `break`; esta última hace que el flujo de control del programa salte a la última sentencia de `switch`. Si no existiera `break` también se ejecutarían las sentencias restantes de `switch`.

- Ejecución de prueba 1

```
Introduzca calificación (1- 10), pulse Intro: 9
Excelente.
Final de programa.
```

- Ejecución de prueba 2

```
Introduzca calificación (1- 10), pulse Intro: 8
Notable.
Final de programa.
```

- Ejecución de prueba 3

```
Introduzca calificación (1- 10), pulse Intro: 12
No es posible esta nota.
Final de programa.
```

PRECAUCIÓN

Si se olvida `break` en una sentencia `switch`, el compilador no emitirá mensaje de error pues se habrá escrito una sentencia `switch` correcta en cuanto a sintaxis, pero no realizará las tareas asignadas.

EJEMPLO 5.12

Este ejemplo selecciona tipo de vehículo y, en concordancia, asigna peaje y salta la ejecución a la sentencia que sigue `switch`.

```
int tipo_vehiculo;
System.out.println("Introduzca tipo de vehículo:");
tipo_vehiculo = entrada.nextInt();

switch(tipo_vehículo)
{
  case 1:
    System.out.println("turismo");
    peaje = 500;
    break;
```

Si se omite este `break`, el primer vehículo será turismo y luego, autobús.

```
case 2:
  System.out.println("autobus");
  peaje = 3000;
  break;
```

```
    case 3:
      System.out.println("motocicleta");
      peaje = 300;
      break;
    default:
      System.out.println("vehículo no autorizado");
  }
```

Cuando la computadora comienza a ejecutar `case` no se detiene hasta que encuentra una sentencia `break` o bien termina `switch` y sigue en secuencia.

5.5.2 Caso particular de `case`

Está permitido tener varias expresiones `case` en una alternativa dada en `switch`; por ejemplo, se puede escribir:

```
switch(c)
{
  case '0':case '1': case '2': case '3': case '4':
  case '5':case '6': case '7': case '8': case '9':
    num_digitos++; // se incrementa en 1 el valor de num_digitos
    break;
  case '': case '\t': case '\n':
    num_blancos++; // se incrementa en 1 el valor de num_blancos
    break;
  default:
    num_distintos++;
}
```

5.5.3 Uso de `switch` en menús

`if-else` es más versátil que `switch` y se pueden utilizar `if-else` anidadas en cualquier parte de una sentencia `case`; sin embargo, normalmente `switch` es más clara; por ejemplo, es idónea para implementar menús como el de un restaurante, el cual presenta una lista para que el cliente elija entre diferentes opciones. Un menú en un programa de computadora hace lo mismo: presenta una lista de alternativas en pantalla para que el usuario elija. En los capítulos siguientes veremos ejemplos prácticos de ellos.

5.6 Expresiones condicionales, operador `?:`

Java mantiene, a semejanza de C, un tercer mecanismo de selección, una expresión que produce uno de dos valores como resultado de una expresión lógica o booleana, también llamada *condición*; este mecanismo es realmente una operación ternaria denominada *expresión condicional* y tiene el formato `C ? A : B`, en el que `C`, `A` y `B` son tres operandos y `?` es el operador.


```
condición ? expresión1 : expresión2

condición es una expresión lógica
expresión1 /expresión2 son expresiones compatibles de tipos
```

Se evalúa *condición*; si su valor es verdadero, se devuelve *expresión1*; si es falso, resulta en *expresión2*.

Uno de los usos más sencillos del operador condicional es utilizar `?:` y llamar a una de dos funciones; el siguiente ejemplo lo utiliza para asignar el menor de dos valores de entrada a `menor`.

```
int entrada1;
int entrada2;
int menor;
entrada1 = entrada.nextInt();
entrada2 = entrada.nextInt();
menor = (entrada1 <= entrada2) ? entrada1 : entrada2
```

EJEMPLO 5.13

Este ejemplo determina y escribe el mayor y el menor de dos números reales; se realiza de dos formas, con la sentencia `if-else`, y con el operador `?:`.

```
import java.util.Scanner;
class MayorMenor
{
  public static void main(String[] a)
  {
    float n1,n2;
    Scanner entrada = new Scanner(System.in);
    System.out.println("Introduzca dos números reales");
    n1 = entrada.nextFloat();
    n2 = entrada.nextFloat();
    // Selección con if-else
    if (n1 > n2)
      System.out.println(n1 + " > " + n2);
    else
      System.out.println(n1 + " < " + n2);
    // operador condicional
    n1 > n2 ? System.out.println(n1 + " > " + n2);
            : System.out.println(n1 + " < " + n2);
  }
}
```

5.7 Evaluación en cortocircuito de expresiones lógicas

Cuando se valoran expresiones lógicas en Java, se puede emplear una técnica denominada *evaluación en cortocircuito*; la cual implica, como ya se dijo en el capítulo anterior, que se puede detener la evaluación de una expresión lógica tan pronto se determine su valor con absoluta certeza; por ejemplo: si el valor de `soltero == 's'` es falso, la expresión lógica `soltero == 's'&& sexo = 'h'&& (edad > 18 && edad <= 45)` también lo será independientemente del valor de las demás condiciones; esto es porque una expresión lógica de tipo `falso && (...)` siempre debe ser falsa cuando uno de los operandos de la operación `and` lo es. En consecuencia, no hay necesidad de continuar la evaluación del resto de las condiciones cuando `soltero == 's'` se evalúa como falso.

En el compilador de Java, la evaluación de una expresión lógica de la forma `a1 && a2` se detiene si la subexpresión `a1` de la izquierda se evalúa como falsa.

Java realiza evaluación en cortocircuito con los operadores `&&` y `||`, de modo que primero evalúa la expresión que se encuentra más hacia la izquierda de las que están unidas por `&&` o por `||`; si esta evaluación muestra información suficiente para determinar el valor final de la expresión, sin importar el valor de la segunda expresión, el compilador de Java no evalúa esta última.

EJEMPLO 5.14

Si `x` es negativo, la expresión

```
(x >= 0) && (y > 1)
```

se evalúa en cortocircuito ya que `x >= 0` es falso y, por tanto, el valor final de la expresión también lo será.

En el caso del operador `||`, se produce una situación similar: si la primera de las dos expresiones que une es verdadera, entonces la expresión completa también lo será, sin importar si el valor de la segunda expresión es verdadero o falso; esto es porque `or (||)` produce un resultado verdadero si el primer operando lo es.

Lenguajes distintos de Java utilizan una evaluación completa, esto implica que cuando dos expresiones se unen por un símbolo `&&` o `||`, se evalúan siempre ambas expresiones y a continuación se utilizan sus tablas de verdad para obtener el valor de la expresión final; por ejemplo, si `x` es cero, la condición

```
if ((x != 0.0) && (y/x > 7.5))
```

es falsa ya que `x != 0.0` también lo es; por consiguiente no hay necesidad de evaluar la expresión `y/x > 7.5` cuando `x` sea cero; sin embargo, si altera el orden de las expresiones, al evaluar la sentencia

```
if ((y/x > 7.5) && (x != 0.0))
```

el compilador produciría un error en tiempo de ejecución por la división entre cero, reflejando que el orden de las expresiones con operadores `&&` y `||` puede ser crítico en determinadas situaciones.

5.8 Puesta a punto de programas

Estilo y diseño

1. El estilo de escritura de una sentencia `if` o `if-else` es el sangrado de las diferentes líneas en el formato siguiente:

```
if(expresión_lógica)
  sentencia1;
else
  sentencia2;

if(expresión_lógica)
{
  sentencia 1;
  sentencia 2;
  ...
```

```
  sentencia k
}
else
{
  sentencia k+1;
  sentencia k+2;
  ...
  sentencia k+n;
}
```

En el caso de las sentencias `if-else-if` que se utilizan para implementar una estructura de selección entre varias alternativas, se escribe de la siguiente forma:

```
if (expresión_lógica 1 )
 sentencia 1;
else if (expresión_lógica 2)
 sentencia 2;
.
.
.
else if (expresión_lógica n)
 sentencia n
else
 sentencia n+1
```

2. Una construcción de selección múltiple se puede implementar de forma más eficiente con una estructura `if-else-if` que con una secuencia de sentencias independientes `if`; por ejemplo:

```
System.out.print("Introduzca nota");
nota = entrada.nextInt();
if (nota < 0 || nota > 100)
{
  System.out.println(nota+" no es una nota válida.");
  return'?';
}
if ((nota >= 90) && (nota <= 100))
  return 'A';
if ((nota >= 80) && (nota < 90))
  return 'B';
if ((nota >=70) && (nota < 80))
  return 'C';
if ((nota >= 60) && (nota < 70))
  return 'D';
if (nota < 60)
  return 'F';
```

Se ejecutan todas las sentencias `if` sin que el valor de `nota` afecte; cinco de las expresiones lógicas son compuestas, de modo que se ejecutan 16 operaciones. En contraste, las sentencias `if` anidadas reducen considerablemente las operaciones a realizar (entre 3 y 7); todas las expresiones son simples y no siempre se evalúan.

```
System.out.print("Introduzca nota");
nota = entrada.nextInt();
if (nota < 0 || nota > 100)
{
```

```
    System.out.println(nota+" no es una nota válida.");
    return'?';
  }
  else if (nota >= 90)
    return 'A';
  else if (nota >= 80)
    return 'B';
  else if (nota >=70)
    return 'C';
  else if (nota >= 60)
    return 'D';
  else
    return 'F';
```

5.9 Errores frecuentes de programación

1. Uno de los errores más comunes en una sentencia `if` es utilizar el operador de asignación = en lugar del operador relacional de igualdad ==.
2. En una sentencia `if` anidada, cada cláusula `else` corresponde con la `if` precedente más cercana; por ejemplo, en el segmento de programa siguiente:

```
if (a > 0)
if (b > 0)
c = a + b;
else
c = a + abs(b);
d = a * b * c;
```

   ¿cuál es la sentencia `if` asociada a `else`? El sistema más fácil para evitar errores es el sangrado o indentación; lo que permite apreciar que la cláusula `else` corresponde a la sentencia que contiene la condición `b > 0`.

```
if (a > 0)
  if (b > 0)
    c = a + b;
  else
    c = a + abs(b);
d = a * b * c;
```

3. Las comparaciones con operadores == de cantidades algebraicamente iguales pueden producir una expresión lógica falsa debido a que la mayoría de los números reales no se almacenan exactamente; por ejemplo, aunque las expresiones reales siguientes son equivalentes:

```
a * (1/a)
1.0
```

   la expresión

```
a * (1/a) == 1.0
```

   puede ser falsa debido a que `a` es un número real.
4. Cuando en una sentencia `switch` o en un bloque de sentencias falta una llave, ({) o (}), aparece un mensaje de error similar a éste:

```
Error ...: Cumpound statement missing }in ...
```

Si no se tiene cuidado con la presentación de la escritura del código, puede ser muy difícil localizar la llave que falta.

5. El selector de una sentencia `switch` debe ser de tipo entero o compatible; así, las constantes reales no pueden utilizarse en el selector; por ejemplo:

```
2.4, -4.5, 3.1416
```

6. Cuando utilice una sentencia `switch` asegúrese de que el selector de `switch` y las etiquetas `case` sean del mismo tipo `int`, `char` o `bool` pero no `float`. Si el selector evalúa un valor no listado en ninguna de las etiquetas `case`, la sentencia `switch` no gestionará ninguna acción; para resolver este problema, se coloca una etiqueta `default`.

Sentencia `if`

Una alternativa:

```
if (a != 0)
  resultado = a/b;
```

Dos opciones:

```
if (a >= 0)
  System.out.println(a+" es positivo");
else
  System.out.println(a+" es negativo");
```

Múltiples opciones:

```
if (x < 0)
{
  System.out.println("Negativo");
  abs_x = -x;
}
else if (x == 0)
{
  System.out.println("Cero");
  abs_x = 0;
}
else
{
  System.out.println("Positivo");
  abs_x = x;
}
```

Sentencia `switch`

```
switch (sig_car)
{
  case 'A': case'a':
    System.out.println("Sobresaliente");
```

```
      break;
    case 'B': case 'b':
      System.out.println("Notable");
      break;
   case 'C': case 'c':
     System.out.println("Aprobado");
     break;
   case 'D': case 'd':
     System.out.println("Suspenso");
     break;
   default:
     System.out.println("nota no válida");
  } // fin de switch
```

conceptos clave

- Estructura de control.
- Estructura de control selectiva.
- Sentencia `break`.
- Sentencia compuesta.
- Sentencia `if`.
- Sentencia `switch`.
- Tipo de dato `boolean`.

ejercicios

5.1 ¿Cuáles errores de sintaxis tiene la siguiente sentencia?

```
if x > 25.0
  y = x
else
  y = z;
```

5.2 ¿Qué valor se asigna a consumo en la sentencia `if` siguiente si `velocidad` es 120?

```
if (velocidad > 80)
  consumo = 10.00;
else if (velocidad > 100)
  consumo = 12.00;
else if (velocidad > 120)
  consumo = 15.00;
```

5.3 Explicar las diferencias entre las sentencias de la columna izquierda y las de la derecha; para ambas deducir el valor final de `x` si su valor inicial es `0`.

```
if (x >= 0)                    if (x >= 0)
  x = x+1;                       x = x+1;
else if (x >= 1);              if (x >= 1)
  x = x+2;                       x = x+2;
```

5.4 ¿Qué salida produce el código siguiente cuando se empotra en un programa completo?

```
int primera_opcion = 1;
switch (primera_opcion + 1);
{
  case 1:
    System.out.println("Cordero asado");
    break;
  case 2:
    System.out.println("Chuleta lechal");
```

```
    break;
  case 3:
    System.out.println("Chuletón");
  case 4:
    System.out.println("Postre de pastel");
    break;
  default:
    System.out.println("Buen apetito");
}
```

5.5 ¿Qué salida produce el siguiente código cuando se empotra en un programa completo?

```
int x = 2;
System.out.println("Arranque");
if (x <= 3)
  if (x != 0)
    System.out.println("Hola desde el segundo if.");
  else
    System.out.println("Hola desde el else.");
System.out.println("Fin");
System.out.println("Arranque de nuevo");
if (x > 3)
 if (x != 0)
   System.out.println("Hola desde el segundo if.");
 else
   System.out.println("Hola desde el else.");
 System.out.println("De nuevo fin");
```

5.6 Escribir una sentencia `if-else` que visualice la palabra `Alta` si el valor de la variable `nota` es mayor que 100 y `Baja` si su valor es menor que 100.

5.7 Identificar el error en el siguiente código:

```
if (x = 0) System.out.println(x + " = 0");
else System.out.println(x + " != 0");
```

5.8 Localizar la errata que hay en el código siguiente:

```
if (x < y < z) System.out.println(x + "<" + y "<" + z);
```

5.9 Ubicar la falla en el siguiente código:

```
System.out.println("Introduzca n:");
n = entrada.nextInt();
if (n < 0)
  System.out.println("Este número es negativo. Pruebe de nuevo.");
else
  System.out.println("Conforme. n = " + n);
```

5.10 Escribir un programa que lea tres enteros y que emita un mensaje que indique si están o no en orden numérico.

5.11 Crear una sentencia `if-then-else` que clasifique un entero `x` en una de las siguientes categorías y que escriba un mensaje adecuado:

```
x < 0, o 0<x<100, o x > 100
```

5.12 Redactar un programa que introduzca número del mes (1-12) y que visualice su número de días.

5.13 Escribir un programa que clasifique enteros leídos en el teclado considerando las siguientes condiciones: si 30 es mayor o negativo, visualizar un mensaje en ese sentido; si es primo, potencia de 2 o número compuesto, visualizar el mensaje correspondiente; si es cero o 1,visualizar "cero" o "unidad".

5.14 Crear un programa que determine cuál es el mayor de tres números.

5.15 El Domingo de Pascua es el primer domingo después de la primera luna llena posterior al equinoccio de primavera y se determina mediante el siguiente cálculo:

```
A =año resto 19
B =año resto 4
C =año resto 7
D =(19 *A +24)resto 30
E =(2 *B +4*C +6*D +5)resto 7
N =(22 +D +E )
```

donde `N` indica el número de día del mes de marzo, si es igual o menor que 3; o abril, si es mayor que 31. Construir un programa que determine las fechas de los domingos de Pascua.

5.16 Codificar un programa que escriba la calificación correspondiente a una nota de acuerdo con el siguiente criterio:

```
0 a <5.0 Suspenso
5 a <6.5 Aprobado
6.5 a <8.5 Notable
8.5 a <10 Sobresaliente
10 Matrícula de honor.
```

5.17 Determinar si el carácter asociado a un código introducido por el teclado es alfabético, dígito, de puntuación, especial o no imprimible.

problemas

5.1 Cuatro enteros entre 0 y 100 representan las puntuaciones de un estudiante de un curso de informática. Escribir un programa que encuentre la media de estas puntuaciones y que visualice una tabla de notas de acuerdo con el siguiente cuadro:

Media puntuación

90-100 A 80-89 B 70-79 C 60-69 D 0-59 E

5.2 Escribir un programa que lea la hora en notación de 24 horas y que imprima en notación de 12; por ejemplo, si la entrada es 13:45, la salida será 1:45 pm. El programa debe solicitar al usuario que introduzca exactamente cinco caracteres para especificar una hora; por ejemplo, las 9 en punto se debe introducir así: 09:00.

5.3 Crear un programa que acepte fechas escritas de modo usual y que las visualice en tres números; por ejemplo: la entrada 15, febrero, 1989 debe producir la salida: 15 2 1989.

5.4 Dadas dos fechas en el formato día (1 a 31), mes (1 a 12) y año (entero de cuatro dígitos), correspondientes a la fecha de nacimiento y fecha actual, respectivamente. Redactar un programa que deduzca y visualice la edad del individuo; si es la fecha de un bebé de menos de un año de edad, la edad se debe dar en meses y días; en caso contrario, en años.

5.5 Codificar un programa que determine si un año es bisiesto; esto se presenta cuando es múltiplo de 4, por ejemplo, 1984; sin embargo, los años que son múltiplos de 100 sólo son bisiestos cuando también son múltiplos de 400; por ejemplo, 1800 no es bisiesto, mientras que 2000, sí lo es.

5.6 Escribir un programa que calcule el número de días de un mes, dados los valores numéricos del mes y el año.

5.7 Crear un programa que valore el salario neto semanal de los trabajadores de una empresa de acuerdo a las siguientes normas:

- Horas semanales trabajadas <38 a una tasa.
- Horas extras (38 o más) a una tasa 50% superior a la ordinaria.
- Impuestos de 0%, si el salario bruto es menor o igual a 750 euros; 10%, si el salario bruto es mayor que 750 euros.

5.8 Redactar y ejecutar un programa que simule una calculadora simple y que lea 2 enteros y un carácter. Si el carácter es un signo +, debe imprimir la suma; si es un signo –, la diferencia; si es *, el producto; si es /, el cociente; y si es %, el resto. Utilizar la sentencia `switch`.

5.9 Escribir un programa que resuelva la ecuación cuadrática $(ax^2 + bx + c = 0)$ y comprobar que así sea.