APÉNDICE

C

Estructuras de control en Java

C.1 Sentencias de selección

If-else

La sentencia *if-else* tiene dos formas:

```
if (expresión) {
     sentencias
}
```

```
if (expresión) {
     sentencias
}
else {
     sentencias
}
```

Ejemplos:

```
if (campo.size() == 0) {
   System.out.println("El campo está vacío");
}

if (numero < 0) {
   informarError();
}
else {
   procesarNumero(numero);
}

if (numero < 0) {
   procesarNegativo();
}
else if (numero == 0) {
   procesarCero();
}
else {
   procesarPositivo();
}
```

switch

La sentencia *switch* selecciona un único valor de un número arbitrario de casos. Existen dos esquemas posibles:

```
switch (expresión) {
    case valor: sentencias;
                break;
    case valor: sentencias;
                break;
    (se omiten los restantes
casos)
    default: sentencias;
                break;
}
```

```
switch (expresión) {
    case valor1:
    case valor2:
    case valor3:
        sentencias;
        break;
    case valor4:
    case valor5:
        sentencias;
        break;
    (se omiten los restantes
casos)
    default:
        sentencias;
        break;
}
```

Notas:

- Una sentencia *switch* puede tener cualquier número de etiquetas `case`.
- La instrucción `break` después de cada `case` es necesaria; en caso contrario la ejecución continúa pasando a través de las sentencias de la etiqueta siguiente. La segunda forma descrita anteriormente usa este esquema. En este caso, los tres primeros valores ejecutarán la primera sección de sentencias mientras que los valores cuatro y cinco ejecutarán la segunda sección de sentencias.
- El caso `default` es opcional. Si no se da ningún valor por defecto puede ocurrir que este caso no se ejecute nunca.
- No es necesaria la instrucción `break` al final del caso por `default` (o del último `case`, si es que no hay sección `default`) pero se considera de buen estilo incluirla.

Ejemplos:

```
switch (dia) {
  case 1:   stringDia = "Lunes";
            break;
  case 2:   stringDia = "Martes";
            break;
  case 3:   stringDia = "Miércoles";
            break;
  case 4:   stringDia = "Jueves";
            break;
  case 5:   stringDia = "Viernes";
            break;
  case 6:   stringDia = "Sábado";
            break;
  case 7:   stringDia = "Domingo";
            break;
  default: stringDia = "Día no válido";
            break;
}

switch (mes) {
  case 1:
  case 3:
  case 5:
  case 7:
  case 8:
```

```
    case 10:
    case 12:
      numeroDeDias = 31;
      break;
    case 4:
    case 6:
    case 9:
    case 11:
      numeroDeDias = 30;
      break;
    case 2:
      if (esAnioBisiesto())
        numeroDeDias = 29;
      else
        numeroDeDias = 28;
      break;
}
```

C.2 Ciclos

Java tiene tres tipos de ciclos: *while*, *do-while* y *for*.

while

El *ciclo while* ejecuta un bloque de sentencias tantas veces como la evaluación de la expresión resulte verdadera. La expresión se evalúa antes de la ejecución del cuerpo del ciclo, por lo tanto, el cuerpo del ciclo podría ejecutarse cero veces (es decir, no ejecutarse).

```
while (expresión) {
  sentencias
}
```

```
Ejemplos:
  int i = 0;
  while (i < texto.size()) {
    System.out.println(texto.get(i));
    i++;
  }

  while (iter.hasNext()) {
    procesarObjeto(iter.next());
  }
```

do-while

El *ciclo do-while* ejecuta un bloque de sentencias tantas veces como la expresión resulte verdadera. La expresión es evaluada después de la ejecución del cuerpo del ciclo, por lo que el cuerpo de este ciclo se ejecuta siempre por lo menos una vez.

```
do {
      sentencias
} while (expresión);
```

```
Ejemplo:
```

```
do {
   entrada = leerEntrada();
   if (entrada == null) {
      System.out.println("Pruebe nuevamente");
   }
} while (entrada == null);
```

for

El *ciclo for* tiene dos formas diferentes. La primera se conoce también como *ciclo for-each* y se usa exclusivamente para recorrer los elementos de una colección. A la variable del ciclo se le asigna el valor de los sucesivos elementos de la colección en cada iteración del ciclo.

```
for (declaración-de-variable : colección) {
     sentencias
}
Ejemplo:
for (String nota : lista) {
      System.out.println(nota);
}
```

La segunda forma del *ciclo for* ejecuta un bloque de sentencias tantas veces como la condición se evalúe verdadera. Antes de iniciar el ciclo, se ejecuta exactamente una vez, una sentencia de *inicialización*. La condición es evaluada antes de cada ejecución del cuerpo del ciclo (por lo que el cuerpo del ciclo podría no ejecutarse). Se ejecuta una sentencia de *incremento* al finalizar cada ejecución del cuerpo del ciclo.

```
for (inicialización; condición; incremento) {
      sentencias
}
```

Ejemplo:

```
for(int i = 0; i < texto.size(); i++) {
     System.out.println(texto.get(i));
   }
```

C.3 Excepciones

El lanzamiento y la captura de excepciones proporciona otro par de construcciones que alteran el flujo del control.

```
try {
   sentencias
}
catch (tipo-de-excepción nombre) {
   sentencias
}
finally {
   sentencias
}
```

Ejemplo:

```
try {
```

```
    FileWriter writer = new FilwWriter("foo.txt");
    writer.write(texto);
    writer.close();
}
catch (IOException e) {
    Debug.reportError("Falló la grabación del texto");
    Debug.reportError("La excepción es: " + e );
}
```

Una sentencia de excepción puede tener cualquier número de cláusulas *catch* que son evaluadas en el orden en que aparecen y se ejecuta sólo la primera cláusula que coincida. (Una cláusula coincide si el tipo dinámico del objeto excepción que ha sido lanzado es compatible en la asignación con el tipo de excepción declarado en la cláusula *catch*.) La cláusula *finally* es opcional.

C.4 Aserciones

Hay dos formas de sentencias de aserción:

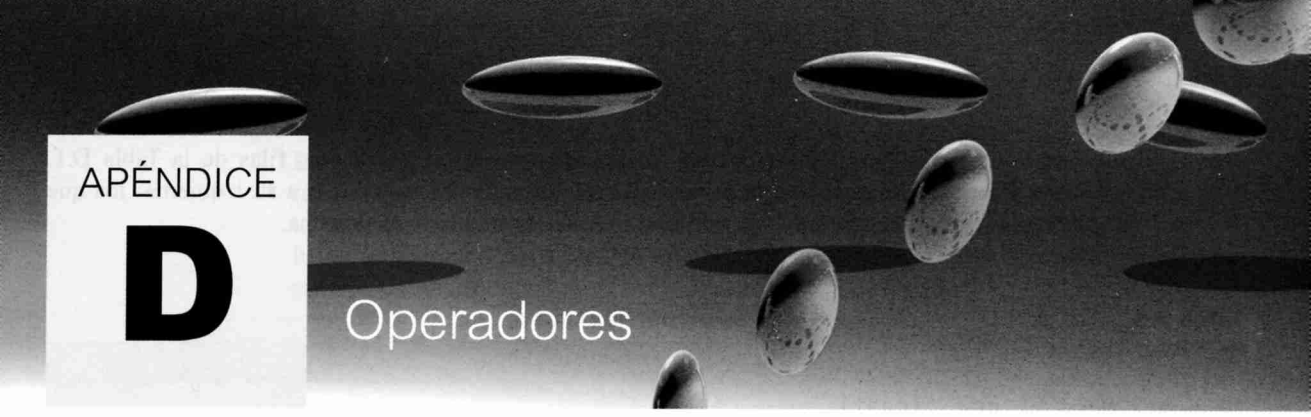
```
assert expresión-booleana;
assert expresión-booleana : expresión;
```

Ejemplos:

```
assert getDatos(clave) != null;

assert esperado = actual :
    " El valor actual: " + actual +
          " no coincide con el valor esperado: " + esperado;
```

Si la expresión de la aserción se evalúa falsa, se disparará un `AssertionError`.



APÉNDICE

D Operadores

D.1 Expresiones aritméticas

Java dispone de una cantidad considerable de operadores para expresiones aritméticas y lógicas. La tabla D.1 muestra todo aquello que se clasifica como un operador, incluyendo la conversión de tipos (*casting*) y el pasaje de parámetros. Los principales operadores aritméticos son:

+	*suma*
-	*resta*
*	*multiplicación*
/	*división*
%	*módulo o resto de una división entera*

Tanto en la división como en el módulo, los resultados de las operaciones dependen de si sus operandos son enteros o si son valores de punto flotante. Entre dos valores enteros, la división retiene el resultado entero y descarta cualquier resto; pero entre dos valores de punto flotante, el resultado es un valor de punto flotante:

`5 / 3` da por resultado `1`
`5.0 / 3` da por resultado `1.6666666666666667`

(Observe que es necesario que uno sólo de los operandos sea de punto flotante para que se produzca un resultado de punto flotante.)

Cuando en una operación aparecen más operadores, se deben usar las *reglas de precedencia* para indicar el orden de su aplicación. En la Tabla D.1, los operadores se presentan por nivel de precedencia, de mayor a menor (en la primera fila aparecen los operadores de nivel de precedencia más alto). Por ejemplo, podemos ver que la multiplicación, la división y el módulo preceden a la suma y a la resta y esta es la razón por la que los dos ejemplos siguientes dan por resultado 100:

```
51 * 3 - 53
154 - 2 * 27
```

Los operadores que tienen el mismo nivel de precedencia se evalúan de izquierda a derecha.

Se pueden usar paréntesis cuando se necesite alterar el orden de evaluación. Es por este motivo que los dos ejemplos siguientes dan por resultado 100:

```
(205 - 5) / 2
2 * (47 + 3)
```

Observe que algunos operadores aparecen en las dos primeras filas de la Tabla D.1. Los que aparecen en la primera fila admiten un solo operando a su izquierda; los que están en la segunda fila admiten un solo operando a su derecha.

Tabla D.1 Operadores Java por nivel de precedencia (de mayor a menor)

```
[]    .     ++    - -  (parámetros)
++    - -   +     -     !     "
new   (cast)
*     /     %
+     -
<<    >>    >>>
<     >     >=    <=    instanceof
==    !=
&
^
|
&&
||
?:
=     +=    - =   *=    /=    %=    >>=   >>>=  &=          |=     ^=
```

D.2. Expresiones lógicas

En las expresiones lógicas, se usan los operadores para combinar operandos y producir un único valor lógico, ya sea verdadero o falso (*true* o *false*). Las expresiones lógicas generalmente se encuentran en las condiciones de las sentencias *if-else* y en las de los ciclos.

Los operadores relacionales o de comparación combinan generalmente un par de operandos aritméticos, aunque también se utilizan para evaluar la igualdad y la desigualdad de referencias a objetos. Los operadores relacionales de Java son:

==	*igual*	!=	*distinto*
<	*menor*	<=	*menor o igual*
>	*mayor*	>=	*mayor o igual*

Los operadores lógicos binarios combinan dos expresiones lógicas para producir otro valor lógico. Los operadores son:

&& *y (and)*
|| *o (or)*
^ *o excluyente*

Y además,

! *no (not)*

que toma una expresión lógica y cambia su valor de verdadero a falso y viceversa.

La manera en que se aplican los operadores && y || es un poco extraña. Si el operando izquierdo es falso entonces resulta irrelevante el valor del operando derecho y no será evaluado; de igual manera, si el operando izquierdo es verdadero, no será evaluado el operando derecho. Por este motivo, se conoce a estos operadores como operadores en «cortocircuito».