1.6 Sentencias y Expresiones

1. Sentencias

Una sentencia es la unidad mínima de ejecución de un programa. Un programa se compone de un conjunto de sentencias que acaban resolviendo un problema. Al final de cada una de las sentencias encontraremos un punto y coma (`;`).

Veamos algunos ejemplos de sentencias en java:

- Sentencias de declaración: `int x;`
- Invocaciones o llamadas a métodos de tipo void: `System.out.println("Bienvenidos a Programación");`
- Sentencias de control de flujo: alteran el fujo de ejecución para tomar decisiones o repetir sentencias.

2. Expresiones

Una expresión es una combinación de operadores y operandos que se evalúa generándose un único resultado de un tipo determinado.

La diferencia entre las sentencias y los operadores es que las expresiones devuelven un valor y las sentencias no devuelven nada.

3. Prioridad entre operadores

A veces hay expresiones con operadores que resultan confusas. Por ejemplo:

```
resultado=8+4/2;
```

Es difícil saber el resultado. ¿Cuál es? ¿seis o diez? La respuesta es 10 y la razón es que el operador de división siempre precede en el orden de ejecución al de la suma. Es decir, siempre se ejecuta antes la división que la suma. Siempre se pueden usar paréntesis para forzar el orden deseado:

```
resultado=(8+4)/2;
```

Ahora no hay duda, el resultado es seis.

¿Cómo podemos saber en qué orden se van a ejecutar los operadores en una expresión en Java? Pues se ejecutan en función de una prioridad, es decir, primero se ejecuta el que tenga más prioridad. La siguiente tabla muestra todos los operadores Java ordenados de mayor a menor prioridad. La primera línea de la tabla contiene los operadores de mayor prioridad y la última los de menor prioridad. Los operadores que aparecen en la misma línea tienen la misma prioridad.

Cuando una expresión tenga dos operadores con la misma prioridad, la expresión se evalúa según su asociatividad.

Nivel	Operador	Descripcion	Asociatividad
1	[ ] . ()	acceso elementos array acceso miembros objetos paréntesis	de izquierda a derecha
2	++ - -	unario post-incremento unario post-decremento	no asociativos
3	++ - - + - ! ~	unario pre-incremento unario pre-decremento unario más unario menos unario lógico NOT unario NOT a nivel de bits	de derecha a izquierda
4	() new	cast creación objetos	de derecha a izquierda
5	* / %	multiplicación división módulo	de izquierda a derecha
6	+ - +	suma resta concatenación cadenas	de izquierda a derecha
7	<< >> >>>	desplazamientos a nivel de bits	de izquierda a derecha
8	< <= > >= instanceof	relacionales	no asociativos
9	= = ! =	igual distinto	de izquierda a derecha
10	&	AND a nivel de bits	de izquierda a derecha

Nivel	Operador	Descripcion	Asociatividad
11	^	XOR a nivel de bits	de izquierda a derecha
12	|	OR a nivel de bits	de izquierda a derecha
13	&&	AND	de izquierda a derecha
14	||	OR	de izquierda a derecha
15	?:	ternario condicional	de derecha a izquierda
16	= += -= *= /= %= &= ^= |= <<= >>= >>>=	asignaciones	de derecha a izquierda

Por ejemplo: `resultado = 9 / 3 * 3 ;` En este caso, la multiplicación y la división tienen la misma prioridad y su asociatividad es de izquierda a derecha por lo que se realiza primero la operación que esté más a la izquierda, que en este caso es la división. El resultado por lo tanto es nueve. Si se desea que se haga primero la multiplicación, habría que utilizar un paréntesis: `resultado = 9 / (3 * 3);` En este caso, el resultado sería 1.

Otro ejemplo: `x = y = z = 17;` Como la asociatividad de la asignación es de derecha a izquierda, primero se asigna el valor 17 a `z`, luego a `y` y por último a `x`. Esto se puede realizar porque el operador de asignación devuelve el valor asignado.

Algunos operadores son no asociativos, por ejemplo, la expresión `x <= y <= z` es inválida ya que el valor devuelto por estos operadores es de un tipo diferente (booleano) al de los operandos que necesita (numérico o carácter).

4. Evaluación de cortocircuito

La evaluación de cortocircuito denota la semántica de algunos operadores booleanos en algunos lenguajes de programación en los cuales si con la evaluación de la primera expresión ya se conoce el resultado, ya no se evalúan el resto de expresiones. En Java, se utiliza la evaluación de cortocircuito.

Por ejemplo, veamos la siguiente expresión que utiliza operadores `AND`:

```
12 < 9 && 5 > 1 && 8 <= 13
```

Se evalúa la primera expresión `12 < 9` dando *false*. Como el resultado va a ser *false* independientemente del resultado de la segunda y tercera expresión, entonces no se evalúan ni la segunda `5 > 1` ni la tercera expresión `8<= 13`, sino que solamente se evalúa la primera, dando como resultado *false*.

Lo mismo ocurre con el operador `OR`:

```
8 <= 13 || 12 < 9 || 5 > 1
```

Se evalúa la primera expresión `8 <= 13` dando *true*. Como el resultado va a ser *true* independientemente del resultado de la segunda y tercera expresión, entonces no se evalúan ni la segunda `12 < 9` ni la tercera expresión `5 > 1`, sino que solamente se evalúa la primera, dando como resultado *true*.