**Precedencia de Operadores en Java**

**Introducción**

Los operadores son un tipo de *tokens* que indican una evaluación o computación para ser realizada en objetos o datos, y en definitiva sobre identificadores o constantes.

Además de realizar la operación, un operador devuelve un valor, ya que son parte fundamental de las expresiones.

El valor y tipo que devuelve depende del operador y del tipo de sus operandos. Por ejemplo, los operadores aritméticos devuelven un número como resultado de su operación.

Los operadores realizan alguna función sobre uno, dos o tres operandos.

Los operadores que requieren un operando son llamados *operadores unarios.* Por ejemplo, el operador "*++*" es un operador unario que incrementa el valor de su operando en una unidad.

Los operadores unarios en Java pueden utilizar tanto la notación prefija como la posfija.

La notación prefija indica que el operador aparece antes que su operando.

++contador // Notación prefija, se evalúa a: contador+1

La notación posfija indica que el operador aparece después de su operando:

contador++ // Notación posfija, se evalúa a: contador

Los operadores que requieren dos operandos se llaman *operadores binarios*. Por ejemplo el operador "*=*" es un operador binario que asigna el valor del operando del lado derecho al operando del lado izquierdo.

Todas los operadores binarios en Java utilizan notación infija, lo cual indica que el operador aparece entre sus operandos.

operando1 operador operando2

Por último, los operadores ternarios son aquellos que requieren tres operandos. El lenguaje Java tiene el operador ternario, "*?*":, que es una sentencia similar a la if-else.

Este operador ternario usa notación infija; y cada parte del operador aparece entre operandos:

expresión ? operación1 : operación2

Los operadores de Java se pueden dividir en las siguientes cuatro categorías:

* Aritméticos.
* De comparación y condicionales.
* A nivel de bits y lógicos.
* De asignación.

**Lista de Precedencia de Operadores**

|  |  |
| --- | --- |
| **Descripción** | **Operadores** |
| operadores posfijos | op++ op-- |
| operadores unarios | ++op --op +op -op ~ ! |
| multiplicación y división | \* / % |
| suma y resta | + - |
| desplazamiento | << >> >>> |
| operadores relacionales | < > <= => |
| equivalencia | == != |
| operador AND | & |
| operador XOR | ^ |
| operador OR | | |
| AND booleano | && |
| OR booleano | || |
| condicional | ?: |
| operadores de asignación | = += -= \*= /= %= &= ^= |= <<= >>= >>>= |

En la tabla se muestra el nivel de precedencia de los operadores. Los de mayor nivel se encuentran arriba. Podemos ver que los últimos son los de asignación. Esto es lógico, ya que se debe calcular primeramente la expresión antes de asignar un resultado a una variable.

int j = 1 + 3 \* 4; // resultado j = 13

Desde que aprendimos aritmética básica, conocemos la regla que nos obliga a calcular la multiplicación antes de una suma. Esto también se cumple en Java.

int j = 1 + 3 – 4; resultado j= 0;

Si todos los operadores tienen un nivel idéntico de precedencia se evalua la expresión de izquierda a derecha.

## Utilización de paréntesis

Se utilizan para aislar una porción de la expresión de forma que el cálculo se ejecute de forma independiente. Puede forzar a una expresión a ignorar las reglas de precedencia.

int j = 1 + 3 \* 4; // resultado j = 13

int h = (1 + 3) \* 4 // resultado j = 16

Tomando el primer ejemplo, forzamos al compilador a realizar la suma antes que la multiplicación.

En este ejemplo es imprescindible la utilización de paréntesis :

int k = 1 + (h = 3);

Si quitamos los paréntesis el compilador protestará. Porque al establecer un nivel muy bajo para la asignación, procede primero la suma. Pero estamos sumando con una variable sin valor.

Como en matemáticas, podemos anidar los paréntesis. Se comenzara a evaluar los internos hasta llegar a los externos.

Cabe agregar que los paréntesis no disminuyen el rendimiento de los programas. Por lo tanto, agregar paréntesis no afecta negativamente al programa.

int k = ((12 - 2) \* ( 21 - 11)) / ((1+1)\*(15-10)) + 1 ;

**Bibliografía**

* http://pisuerga.inf.ubu.es/lsi/Invest/Java/Tuto/II\_3.htm
* http://es.wikibooks.org/wiki/Programaci%C3%B3n\_en\_Java\_/\_Precedencia\_de\_operadores