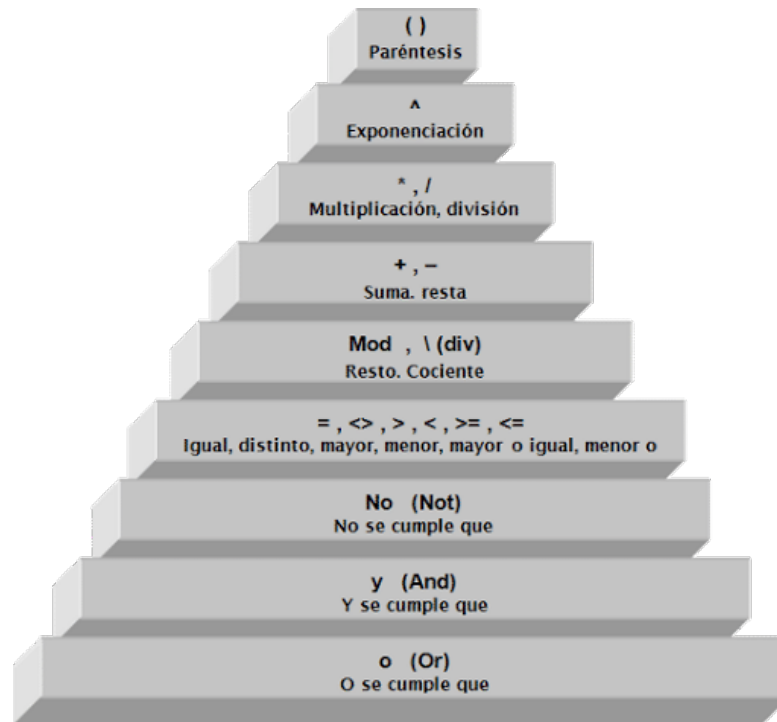


OPERADORES

PROGRAMACIÓN



Prioridades:



Operadores Aritméticos:

Los habituales

- Suma + .
- Resta - .
- Multiplicación * .
- División / .
- Resto de la División % .

Ejemplo:

int a = 10, b = 3;

double v1 = 12.5, v2 = 2.0;

char c1='P', c2='T';

| Operación | Valor | Operación | Valor | Operación | Valor |
|-----------|-------|-----------|-------|---------------|-------|
| a+b | 13 | v1+v2 | 14.5 | c1 | 80 |
| a-b | 7 | v1-v2 | 10.5 | c1 + c2 | 164 |
| a*b | 30 | v1*v2 | 25.0 | c1 + c2 + 5 | 169 |
| a/b | 3 | v1/v2 | 6.25 | c1 + c2 + '5' | 217 |
| a%b | 1 | v1%v2 | 0.5 | | |

En aquellas operaciones en las que aparecen operandos de distinto tipo, java convierte los valores al tipo de dato de mayor precisión de todos los datos que intervienen. Esta conversión es de forma temporal, solamente para realizar la operación. Los tipos de datos originales permanecen igual después de la operación.

Los tipos short y byte se convierten automáticamente a int.

Ejemplo:

int i = 7;

double f = 5.5;

char c = 'w';

| Operación | Valor | Tipo |
|-----------------------|-------|--------|
| i + f | 12.5 | double |
| i + c | 126 | int |
| i + c - '0' | 78 | int |
| (i + c) - (2 * f / 5) | 123.8 | double |

Operadores de Asignación:

El principal es '=' pero hay más operadores de asignación con distintas funciones que explicamos brevemente ahora.

- '+=' : op1 += op2 op1 = op1 + op2
- '-=' : op1 -= op2 op1 = op1 - op2
- '*=' : op1 *= op2 à op1 = op1 * op2
- '/=' : op1 /= op2 à op1 = op1 / op2
- '%=' : op1 %= op2 à op1 = op1 % op2

Operadores Unarios: El mas (+) y el menos (-). Para cambiar el signo del operando.

Operador instanceof: Nos permite saber si un objeto pertenece a una clase o no.

- NombreObjeto instanceof NombreClase

Operadores Incrementales: Son los operadores que nos permiten incrementar las variables en una unidad. Se pueden usar delante y detrás de la variable dependiendo de lo que queramos, es decir, si queremos que incremente o viceversa antes de utilizar o lo contrario.

'++'

'--'

Ejemplo de operador decremento:

```
int i = 1;
i--; // decrementa en 1 la variable i.
// Es lo mismo que hacer i = i - 1; i toma el valor 0
```

Los operadores incremento y decremento pueden utilizarse como prefijo o sufijo, es decir, pueden aparecer antes o después de la variable.

Por ejemplo:

```
i = 5;
i++; // i vale ahora 6
++i; // i vale ahora 7
```

Cuando estos operadores intervienen en una expresión, si preceden al operando (++i), el valor se modificará antes de que se evalúe la expresión a la que pertenece.

En cambio, si el operador sigue al operando (i++), entonces el valor del operando se modificará después de evaluar la expresión a la que pertenece.

Por ejemplo:

```
int x, i = 3;
x = i++;
```

En esta asignación a x se le asigna el valor 3 y a continuación i se incrementa, por lo tanto, después de ejecutarla:

x contiene 3, i contiene 4.

Si las instrucciones son:

```
int x, i = 3;
x = ++i;
```

En esta instrucción primero se incrementa i y el resultado se asigna a x. Por lo tanto, después de ejecutarla:

x contiene 4, i contiene 4.

Otro ejemplo:

```
int i = 1;
System.out.println(i);
System.out.println(++i);
System.out.println(i);
```

Estas instrucciones mostrarían por pantalla:

```
1
2
2
```

En cambio, si se cambia la posición del operador:

```
int i = 1;
System.out.println(i);
System.out.println (i++);
System.out.println (i);
```

Estas instrucciones mostrarían por pantalla:

```
1
1
2
```

El operador complemento a 1 `~` cambia de valor todos los bits del operando (cambia unos por ceros y ceros por unos). Solo puede usarse con datos de tipo entero. El carácter `~` es el ASCII 126.

Por ejemplo:

```
int a = 1, b = 0, c = 0;
c = ~a;
```

Operadores Relacionales: Permiten comparar variables según relación de igualdad/desigualdad o relación mayor/menor. Devuelven siempre un valor boolean.

- `'>'`: Mayor que
- `'<'`: Menor que
- `'=='`: Iguales
- `'!='`: Distintos
- `'>='`: Mayor o igual que
- `'<='`: Menor o igual que

Ejemplo:

int a = 7, b = 9, c = 7;

| Operación | Resultado |
|-----------|-----------|
| a==b | false |
| a >=c | true |
| b < c | false |
| a != c | false |

Operadores Lógicos: Nos permiten construir expresiones lógicas.

- '&&' : devuelve true si ambos operandos son true.
- '||' : devuelve true si alguno de los operandos son true.
- '!' : Niega el operando que se le pasa.
- '&' : devuelve true si ambos operandos son true, evaluándolos ambos.
- '|' : devuelve true uno de los operandos es true, evaluándolos ambos.

Las definiciones de las operaciones OR, AND y NOT se recogen en unas tablas conocidas como **tablas de verdad**.

| A | B | A OR B |
|---|---|--------|
| F | F | F |
| F | V | V |
| V | F | V |
| V | V | V |

| A | B | A AND B |
|---|---|---------|
| F | F | F |
| F | V | F |
| V | F | F |
| V | V | V |

| A | NOT A |
|---|-------|
| F | V |
| V | F |

F:

Falso

V: Verdadero

Como ejemplo, en la siguiente tabla vemos una serie de expresiones lógicas y su valor:

int i = 7;

float f = 5.5F;

char c = 'w';

| Expresión | Resultado |
|-----------|-----------|
|-----------|-----------|

| | |
|--|-------|
| <code>(i >= 6) && (c == 'w')</code> | true |
| <code>(i >= 6) (c == 119)</code> | true |
| <code>(f < 11) && (i > 100)</code> | false |
| <code>(c != 'p') ((i + f) <= 10)</code> | true |
| <code>i + f <= 10</code> | false |
| <code>i >= 6 && c == 'w'</code> | true |
| <code>c != 'p' i + f <= 10</code> | true |

Las expresiones lógicas en java se evalúan sólo hasta que se ha establecido el valor cierto o falso del conjunto. Cuando, por ejemplo, una expresión va a ser seguro falsa por el valor que ha tomado uno de sus operandos, no se evalúa el resto de expresión.

Operador de concatenación con cadena de caracteres '+':

- Por Ejemplo: `System.out.println("El total es"+ result +"unidades");`

Operadores que actúan a nivel de bits: Son mucho menos utilizados por eso los explicamos mas por encima.

- '>>': desplazamiento a la derecha de los bits del operando
- '<<': desplazamiento a la izquierda de los bits de operando
- '&': operador and a nivel de bit.
- '|': operador or a nivel de bit