💾 Operadores

Los operadores son símbolos especiales en Java que realizan operaciones entre uno o varios operandos y devuelve un resultado. Uno de los más usados es el operador suma (+) como hemos visto en clases anteriores.

Operando

Es cualquier término, que puede ser una variable o valor y que es manipulado por un operador.

```
1     int valor = 8;
2     int numero = valor + 12;
```

En el ejemplo anterior, *+* es el operador y *valor* y *12* son los operandos. *valor + 12* es una expresión que devuelve el resultado de 20.

Expresiones

Una expresión es una combinación de literales, operadores, nombres de variables y paréntesis que se utilizan para calcular un valor.

```
1    int miPrimerEntero = 7 + 5;
2    int resultado = 0;
3
4    resultado = (miPrimerEntero * 10) / (32 + 12);
```

Java examina la expresión de la derecha del signo igual y realiza el cálculo de una expresión matemática. Después asigna ese valor a la variable resultado. Podríamos complicar más la expresión utilizando **operadores** como paréntesis, multiplicaciones, divisiones, etc.

Las partes de una expresión deben estar ordenadas correctamente. Las reglas para las expresiones Java correctas son casi las mismas que las del álgebra:

1. Cada operador debe tener el número correcto de operandos.
    - Multiplicación *, División /, Suma +, Resta: debe tener dos operandos, uno en cada lado.
    - La negación - y unario más + deben ir seguidos de un operando.
2. Los paréntesis () pueden rodear una expresión legal para convertirla en operando.

Expression	Correct or Not Correct?	Expression	Correct or Not Correct?
25	CORRECT	25 - value	CORRECT
2(a - b)	NOT correct	(a-b) * (c-d)	CORRECT
A - b/c + D	CORRECT	-sum + partial	CORRECT
((x+y) / z) / (a - b)	CORRECT	((m-n) + (w-x-z) / (p % q)	NOT correct

Expresiones mixtas con int y double

Si ambos operandos de un operador aritmético son de tipo int, entonces la operación es una operación entera. Si algún operando es de punto flotante, entonces la operación es de punto flotante.

Tipos de operadores en Java

Java proporciona muchos tipos de operadores que se pueden usar según la necesidad. Se clasifican según la funcionalidad que brindan. Algunos de los tipos son los siguientes:

Operadores aritméticos, unarios, de asignación, relacionales, lógicos, etc.

Operador de asignación (=)

Es uno de los operadores más usados. Se usa para asignar un valor a cualquier variable. Tiene una asociación de derecha a izquierda, es decir, el valor dado en el lado derecho del operador se asigna a la variable de la izquierda y, por lo tanto, el valor del lado derecho debe declararse antes de usarlo o debe ser una constante.

Operadores aritméticos

Se utilizan para realizar operaciones aritméticas simples.

Símbolo	Operación	Descripción
+	Suma	Realiza la suma de los operandos.
-	Resta	Realiza la resta de los operandos.
*	Producto	Multiplica los operandos.
/	División	Realiza la división.

Símbolo	Operación	Descripción
%	Módulo	Calcula el resto.

```
int a = 10, b = 5;
int suma = a + b;         // 15
int division = a / b;     // 2
int modulo = a % b;       // 0
```

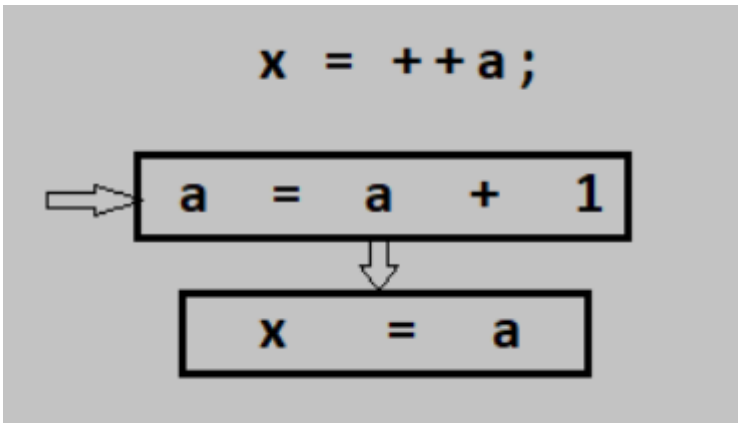
Operadores unarios

Los operadores unarios solo necesitan un operando. Se usan para incrementar, disminuir o negar un valor.

Símbolo	Operación	Descripción
++	Incremento	Incrementa el valor en 1 unidad.
--	Decremento	El valor disminuye en 1 unidad.
!	NOT lógico	Invierte un valor booleano.

Existen **dos versiones** de estos operadores:

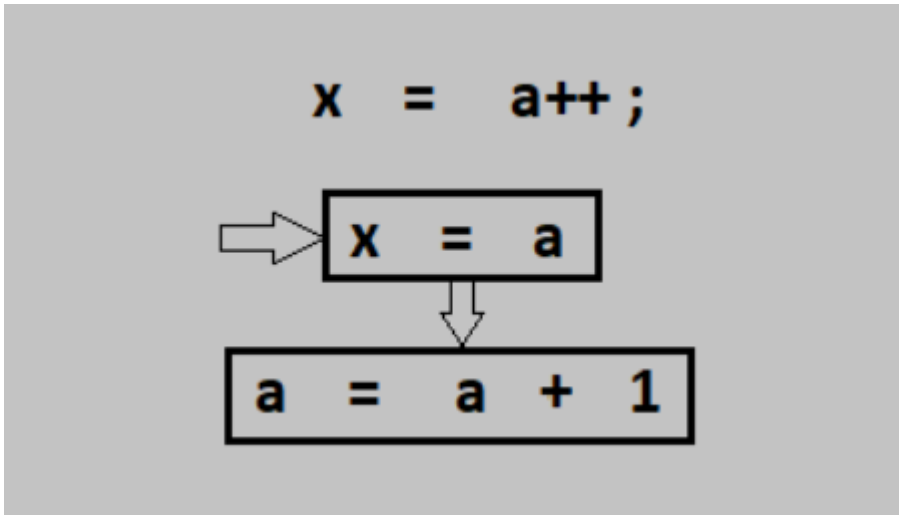
- Pre-incremento y pre-decremento. El valor se aumenta/disminuye primero y luego se calcula el resultado.




```
int a = 8, b = 1;
b = ++a; //b=9, a=9
```

- Post-Incremento y post-decremento: el valor se usa por primera vez para calcular el resultado y luego se incrementa/decrementa.

```
int a = 8, b = 1;
b = a++;//b = 8, a = 9
```



Operadores relacionales

Estos operadores se utilizan para verificar relaciones como igualdad, mayor que, menor que. Devuelven el resultado booleano(true o false) después de la comparación.

Operador	Descripción	Ejemplo
`==`	Igual a	`a == b`
`!=`	No igual a	`a != b`
`>`	Mayor que	`a > b`
`<`	Menor que	`a < b`
`>=`	Mayor o igual que	`a >= b`
`<=`	Menor o igual que	`a <= b`

```
int a = 20, b = 10;


boolean esIgual = a == b;      // false
```

```
5   boolean esMayor = a > b;       // true
6   boolean esMenorOIgual = a <= b; // false
```

Operadores lógicos

Estos operadores se utilizan para realizar operaciones lógicas **AND** y **OR**. Se usa ampliamente en sentencias if-then o bucles para verificar condiciones, establecer un punto de salida de un bucle o la toma de decisiones. Los operadores condicionales son:

Símbolo	Operación	Descripción
&&	AND lógico	Devuelve verdadero cuando ambas condiciones son ciertas.
||	OR lógico	Devuelve verdadero si al menos una condición es cierta.
!	NOT lógico	Invierte el valor lógico de la expresión lógica

INPUT		OUTPUT
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

INPUT		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

En Java

a	b	a && b	a || b
falso	falso	falso	falso
cierto	falso	falso	cierto
falso	cierto	falso	cierto
cierto	cierto	cierto	cierto

a	!a
falso	cierto
cierto	falso

```
1       int a = 20, b = 10, c= 10;
2       System.out.println((b == c && a == c)); //False
3       System.out.println((a == c && b == c)); //False
4       System.out.println(!(a == c && b == c)); //True
```

```
5       System.out.println((a == b || b == c)); //True
6       System.out.println(!(a == b || b == c)); //False
```

Operadores de bits

Símbolo	Operación	Descripción
&	AND	Si ambos bits de entrada son 1, establece como resultado 1. De lo contrario 0.
|	OR	Si por lo menos uno de los bits de entrada es 1, establece como resultado 1. De lo contrario 0.
^	XOR	Si uno de los bits de entrada es 1 y el otro 0, establece como resultado 1. Si los bits son iguales establece 0.
~	NOT	Invierte todos los bits y devuelve el resultado en complemento a 2.

Abreviaciones

En muchos casos, el operador de asignación se puede combinar con otros operadores para construir una versión más corta de la declaración llamada Declaración Compuesta (Compound Statement).

- += , para sumar el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.

```
1     int a = 5;
2     a += 5; // a = a + 5;
```

- – = , para restar el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.

```
1     int a = 5;
2     a -= 5; // a = a - 5;
```

- *= , para multiplicar el operando izquierdo con el operando derecho y luego asignándolo a la variable de la izquierda.

```
int a = 5;
a *= 5; // a = a * 5;
```

- / = , para dividir el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.

```
int a = 5;
a /= 5; // a = a / 5;
```

- % = , para asignar el módulo del operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.

```
int a = 5;
a %= 5; // a = a % 5;
```

Operador ternario

El operador ternario es un operador condicional que evalúa una expresión y devuelve un valor basado en si la condición es verdadera o falsa.

Operador	Descripción	Ejemplo
`?:`	Operador ternario (condicional)	`condición ? valor1 : valor2`

```
int a = 10, b = 5;
int max = (a > b) ? a : b;
// max=10 ya que a>b=true

int num = 15;
String resultado = (num % 2 == 0) ? "Par" : "Impar";  // Si el número es divisible por 2, es par
//resultado="Par"

int edad=8;
String mensaje = (edad>=18) ? "Es mayor de edad" : "Es menor de edad";
//mensaje="Es menor de edad"
```

Precedencia de operadores

La prioridad de operadores en Java define el orden en que las operaciones son evaluadas cuando hay múltiples operadores en una expresión. Algunos operadores tienen mayor prioridad que otros, lo que significa que se evaluarán antes a menos que el orden sea alterado explícitamente mediante el uso de paréntesis.

Reglas generales:

- Los operadores con mayor prioridad se evalúan antes que los de menor prioridad.
- Los operadores con la misma prioridad se evalúan de acuerdo a su asociatividad, que puede ser de izquierda a derecha o de derecha a izquierda.

Las prioridades en Java son:

Tabla de prioridad de operadores en Java (de mayor a menor prioridad)

Nivel	Operadores	Asociatividad
1	`()` (paréntesis), `[]` (acceso a arreglo), `.` (acceso a miembros de clase)	Izquierda a derecha
2	`++` (post-incremento), `--` (post-decremento)	Izquierda a derecha
3	`++` (pre-incremento), `--` (pre-decremento), `+` (unario), `-` (unario), `~` (complemento), `!` (negación lógica)	Derecha a izquierda
4	`*`, `/`, `%`	Izquierda a derecha
5	`+`, `-`	Izquierda a derecha
6	`<<`, `>>`, `>>>` (operadores de desplazamiento de bits)	Izquierda a derecha
7	`<`, `<=`, `>`, `>=`, `instanceof`	Izquierda a derecha
8	`==`, `!=`	Izquierda a derecha
9	`&` (AND bit a bit)	Izquierda a derecha
10	`^` (XOR bit a bit)	Izquierda a derecha
11	`	` (OR bit a bit)
12	`&&` (AND lógico)	Izquierda a derecha
13	`	
14	`?:` (operador ternario)	Derecha a izquierda
15	`=` (asignación), `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `	`=`, `^=`, `<<=`, `>>=`, `>>>=` `
16	`,` (separador de expresiones)	Izquierda a derecha

```java
int resultado = 10 + 3 * 2;
//16
```

Explicación:

1. El operador * (multiplicación) tiene mayor prioridad que + (suma).
2. Por lo tanto, la multiplicación 3 * 2 se evalúa primero, dando 6.
3. Luego se evalúa la suma: 10 + 6, que da 16.

Uso de paréntesis para alterar la prioridad:

Si deseas cambiar el orden de evaluación, puedes utilizar paréntesis:

```
1   int resultado = (10 + 3) * 2;
2   //26
```

Entender la prioridad de operadores es crucial para evitar errores sutiles y mejorar la legibilidad del código. Aunque Java sigue un conjunto de reglas predefinidas, es recomendable usar paréntesis para aclarar la intención en expresiones complejas. Esto facilita tanto el entendimiento del código como su mantenimiento.

Conversiones de tipo

Conversiones por defecto

Las reglas de Java para saber el tipo de datos resultante de una expresión se siguen las siguientes reglas:

Si algún operando es	El otro operando se transforma a
double	double
float	float
long	long
byte or short	int

Si se aplican dos regla, se elige la que aparece primero en la tabla.

```
1       //imprime 5, es decir un int
2       System.out.println((34 + 1) / 7);
3       //imprime 5.0, es decir un float, ya que 1.0 es un float
4       System.out.println((34 + 1.0) / 7);
```

Conversiones forzosas (casting entre tipos nativos)

Es una forma de convertir un número de un tipo a otro tipo de dato. Para hacerlo ponemos en paréntesis el tipo de dato al que queremos convertirlo.

```
1       byte miByte = (byte) (14 / 2); //conviero la operación división que
devuelve un int a byte
```

```
2
3       //Otra forma
4       float a = 8.0f;
5       int b = 10;
6       b = (int) a;//convierto el tipo float a int
```

Otros operadores

Existen más operadores que no se han mencionado en el curso. Si se desea consultar todos los operadores de Java se pueden ver en su documentación oficial. More information

API de Java

Una de las ventajas de los lenguajes modernos es que se dispone de herramientas que resuelve muchos de los algoritmos necesarios para solucionar un problema. En Java, a estas herramientas disponibles para el programador se les llama API de Java y se encuentran organizadas en paquetes, donde cada paquete contiene un conjunto de clases relacionadas.

La API de Java es extensa y diversa, lo que permite a los desarrolladores aprovechar una amplia gama de funcionalidades sin tener que implementar todo desde cero. Los programadores pueden importar los paquetes y clases necesarios en sus programas y utilizar los métodos y atributos proporcionados por la API para realizar tareas específicas.

Por ejmplo, ya hemos utilizado la clase System para la salida por teclado.

```
1       System.out.println("Hola mundo");
```

Iremos aprendiendo la API de Java poco a poco. En la siguiente sección podemos ver la clase **Math** del paquete java.lang

Sintaxis de las expresiones matemáticas

Método	Returns	Ejemplo
Math.abs	valor absoluto	Math.abs(-308) returns 308
Math.ceil	redondeo hacia arriba	Math.ceil(2.13) returns 3.0
Math.floor	redondeo hacia abajo	Math.floor(2.93) returns 2.0
Math.max	valor máx. de dos valores	Math.max(45, 207) returns 207

Método	Returns	Ejemplo
Math.min	valor min. de dos valores	Math.min(3.8, 2.75) returns 2.75
Math.pow	potencia	Math.pow(3, 4) returns 81.0
Math.round	redondear al entero más cercano	Math.round(2.718) returns 3
Math.sqrt	raíz cuadrada	Math.sqrt(81) returns 9.0

Algunos ejemplos

```java
public class EjemploMath {
    public static void main(String[] args) {
        // Calcular el valor absoluto de un número
        double numeroNegativo = -10.5;
        double valorAbsoluto = Math.abs(numeroNegativo);
        System.out.println("Valor absoluto de " + numeroNegativo + " es " + valorAbsoluto);

        // Calcular la raíz cuadrada de un número
        double numero = 25.0;
        double raizCuadrada = Math.sqrt(numero);
        System.out.println("La raíz cuadrada de " + numero + " es " + raizCuadrada);

        // Calcular el valor máximo entre dos números
        int num1 = 30;
        int num2 = 45;
        int maximo = Math.max(num1, num2);
        System.out.println("El valor máximo entre " + num1 + " y " + num2 + " es " + maximo);

        // Calcular el valor mínimo entre dos números
        int minimo = Math.min(num1, num2);
        System.out.println("El valor mínimo entre " + num1 + " y " + num2 + " es " + minimo);

        // Calcular un número elevado a una potencia
        double base = 2.0;
        double exponente = 3.0;
        double resultadoPotencia = Math.pow(base, exponente);
        System.out.println(base + " elevado a la " + exponente + " es igual a " + resultadoPotencia);

        // Generar un número aleatorio entre 0.0 (inclusive) y 1.0 (exclusivo)
        double numeroAleatorio = Math.random();
        System.out.println("Número aleatorio: " + numeroAleatorio);
    }
}
```

La clase **Math** es parte del paquete **java.lang**, que se importa automáticamente en todos los programas de Java sin necesidad de una declaración de importación explícita