# ADSI

## Operadores en Java - POO


Instructor: Gustavo Adolfo Rodríguez Q.

garodriguez335@misena.edu.co

ADSI

# OPERADORES EN JAVA

En esta lección vamos a aprender sobre los distintos operadores que nos brinda el lenguaje Java para la utilización dentro de nuestros programas. Los operadores del lenguaje Java son muy similares a los operadores del lenguaje C y C++. Estos operadores pueden ser categorizados así:

- Operadores aritméticos.
- Operadores de asignación.
- Operadores unarios.
- Operadores incrementales
- Operadores relacionales
- Operadores lógicos
- Operadores a nivel de bit
- Operador ***instanceof***

Los operadores, al igual que los métodos, se pueden sobrecargar, es decir se puede redefinir su funcionalidad dependiendo de los tipos de datos de los operandos que reciba. Así, podemos indicar que el operador (+) realice una suma aritmética si los operandos que recibe son dos enteros y una concatenación si recibe una cadena y otro objeto. Los detalles los veremos a continuación.

## 1. OPERADORES ARITMÉTICOS

Este tipo de operadores son los que nos permiten realizar operaciones aritméticas básicas tales como: suma, resta, multiplicación, división y módulo. Estos operadores se aplican a datos de tipo numérico. La siguiente tabla muestra los operadores aritméticos.

| OPERADOR | DESCRIPCIÓN |
|---|---|
| **+** | Realiza la suma de los operandos |
| **-** | Realiza la resta de los operandos |
| ***** | Realiza la multiplicación de los operandos |
| **/** | Realiza la división de los operandos |
| **%** | Calcula el módulo o residuo en una división de los operandos. |

## 2. OPERADORES DE ASIGNACIÓN

Este tipo de operadores son los que permiten asignar un valor a un dato primitivo o asignar una referencia hacia un objeto. El principal de los operadores de asignación es el =, sin embargo existen otros más que se explicarán a continuación.

| OPERADOR | DESCRIPCIÓN |
|---|---|
| **+=** | Realiza la suma de los operandos y el resultado lo almacena en el operando de la izquierda. |
| **-=** | Realiza la resta de los operandos y el resultado lo almacena en el operando de la izquierda. |
| ***=** | Realiza la multiplicación de los operandos y el resultado lo |

| | |
|---|---|
| | almacena en el operando de la izquierda. |
| **/=** | Realiza la división de los operandos y el resultado lo almacena en el operando de la izquierda. |
| **%=** | Calcula el módulo o residuo en una división de los operandos y el resultado lo almacena en el operando de la izquierda. |

Teniendo en cuenta la tabla anterior, en las siguientes secciones de código, las asignaciones del lado derecho son equivalentes a las asignaciones del lado izquierdo.

```
int a = 1, b = 1;
a = a + b;          a += b;
a = a - b;          a -= b;
a = a * b;          a *= b;
a = a / b;          a /= b;
a = a % b;          a %= b;
```

## 3. OPERADORES UNARIOS

Este tipo de operadores son los que se aplican a un único operando como se muestra en la siguiente tabla.

| OPERADOR | DESCRIPCIÓN |
|---|---|
| **+** | Especifica el signo positivo en un literal numérico. |
| **-** | Especifica el signo negativo en un literal numérico. |

## 4. OPERADORES INCREMENTALES/DECRAMENTALES

Este tipo de operadores son los que permiten realizar el incremento o decremento unitario de una variable como se muestra en la siguiente tabla.

| OPERADOR | DESCRIPCIÓN |
|---|---|
| **++** | Realiza el incremento en una unidad del operando. Puede escribirse antes o después del operando |
| **--** | Realiza el decremento en una unidad del operando. Puede escribirse antes o después del operando |

Aunque los operadores ++, -- realizan el incremento y decremento respectivamente, se puede obtener diferentes resultados dependiendo de si el operador aparece antes o después del operador. El siguiente fragmento de código aclara lo expuesto.

```
int a = 0;
int b = 0;

b = a++;
```

En este caso, la variable *a* tendrá el valor 1, mientras que la variable *b* tendrá el valor 0. Es ocurre debido a que primero se realiza la asignación del valor de la variable *a* hacia la variable *b* y posteriormente se realiza el incremento en la variable *a*

```
int a = 0;
int b = 0;

b = ++a;
```

En este caso, la variable *a* tendrá el valor 1 y la variable *b* tendrá el valor 1. Es ocurre debido a que primero se realiza el incremento en la variable *a* y posteriormente se realiza la asignación del valor de la variable *a* hacia la variable *b*

## 5. OPERADORES RELACIONALES

Los operadores relacionales son aquellos que permiten comparar variables para determinar su relación de igualdad/desigualdad o relación mayor/menor. Este tipo de comparaciones siempre retornan un valor booleano. Los operadores relacionales se muestran en la siguiente tabla.

| OPERADOR | DESCRIPCIÓN |
|---|---|
| **>** | Relación *mayor que*. |
| **<** | Relación *menor que*. |
| **==** | Relación *igual a*. |
| **!=** | Relación *diferente de*. |
| **>=** | Relación *mayor o igual que* |
| **<=** | Relación *menor o igual que* |

## 6. OPERADORES LÓGICOS

Este tipo de operadores nos permiten determinar el valor de verdad de una expresión. Los operadores lógicos se muestran en la siguiente tabla.

| OPERADOR | DESCRIPCIÓN |
|---|---|
| **&&** | Realiza la operación AND lógica entre dos operandos. Es decir, retorna true sólo si ambos operadores son true, en caso contrario retorna false. |
| \|\| | Realiza la operación OR lógica entre dos operandos. Es decir, sólo retorna false si ambos operadores son false, en caso contrario retorna true. |
| **!** | Realiza la operación NOT lógica al operando. Es decir, cambia el valor de verdad del operando |

El siguiente fragmento de código muestra la tabla de verdad para un operador AND lógico.

```
boolean t = true;
boolean f = false;
boolean result;

result = t && t;  //verdadero
result = t && f;  //falso
result = f && t;  //falso
result = f && f;  //falso
```

El siguiente fragmento de código muestra la tabla de verdad para un operador OR lógico.

```
boolean t = true;
boolean f = false;
boolean result;

result = t || t;  //verdadero
result = t || f;  //verdadero
result = f || t;  //verdadero
result = f || f;  //falso
```

El siguiente fragmento de código muestra la tabla de verdad para una operación NOT lógica.

```
boolean t = true;
boolean f = false;
boolean result;

result = !t;  // false
result = !f;  // true
```

## 7. OPERADORES DE BITS

En los sistemas computacionales, los microprocesadores sólo entienden la información que se encuentra representada en un conjunto de bits. Java nos permite alterar los conjuntos de bits mediante los operadores listados en la siguiente tabla.

| OPERADOR | USO | DESCRIPCIÓN |
|---|---|---|
| << | X << Y | Desplaza a la izquierda Y veces los bits de X. Se rellena con bits iguales al signo |
| >> | X >> Y | Desplaza a la derecha Y veces los bits de X. Se rellena con bits iguales al signo |
| >>> | X >>> Y | Desplaza a la derecha Y veces los bits de X. Se rellena con 0. |
| & | X & Y | Operación AND a nivel de bits. |
| \| | X \| Y | Operación OR a nivel de bits. |
| ^ | X ^Y | Operación XOR a nivel de bits. |
| ~ | ~X | Operación NOT a nivel de bits. |

A continuación se ilustra la utilización del operador de desplazamiento a la izquierda.

```
int x = 33;          //00000000000000000000000000100001
int k = x << 2;      //00000000000000000000000010000100
```

Después de la operación la variable *k* tendrá el valor 132

El siguiente fragmento de código muestra la utilización del operador de desplazamiento a la derecha.

```
int x = 33;          //00000000000000000000000000100001
int k = x >> 2;      //00000000000000000000000000001000
```

Después de la operación la variable *k* tendrá el valor 8

El siguiente fragmento de código muestra la utilización del operador de desplazamiento a la derecha sin signo.

```
int x = -1;          //11111111111111111111111111111111
int k = x >>> 10;    //00000000001111111111111111111111
```

Después de la operación la variable *k* tendrá el valor 4194303

El siguiente fragmento de código muestra la utilización del operador AND.

```
int k = 132;    //00000000000000000000000010000100
int l = 144;    //00000000000000000000000010010000
int m = k & l; //00000000000000000000000010000000
```

Después de la operación la variable *m* tendrá el valor 128.

El siguiente fragmento de código muestra la utilización del operador OR

```
int k = 132;    //00000000000000000000000010000100
int l = 144;    //00000000000000000000000010010000
int m = k | l; //00000000000000000000000010010100
```

Después de la operación la variable *m* tendrá el valor 148.

El siguiente fragmento de código ilustra la utilización del operador XOR

```
int k = 132;    //00000000000000000000000010000100
int l = 144;    //00000000000000000000000010010000
int m = k ^ l; //00000000000000000000000000010100
```

Después de la operación la variable *m* tendrá el valor 20

El siguiente fragmento de código ilustra la utilización del operador NOT.

```
int k = 359;    //00000000000000000000000101100111
int m = ~k;     //11111111111111111111111010011000
```

Después de la operación la variable *m* tendrá el valor -360

## 8. OPERADOR instanceof

El operador **instanceof** se utiliza para consultar si un objeto es una instancia de una clase determinada, o de su padre. Por ejemplo, si tenemos la clase Punto:

```
public class Punto {
      protected int x, y;

      public Punto ( int x, int y ) {
          this.x = x;
          this.y = y;
      }

      // Devuelve la distancia al eje de coordenadas
      public float distancia() {
          return (float) Math.sqrt(Math.pow(x,2.0)+Math.pow(y,2.0));
      }
}
```

Y tenemos la clase Punto3D:

```
public class Punto3D extends Punto {
      private int z;

      public Punto3D ( int x, int y, int z ) {
          super(x,y);
          this.z = z;
      }

      // Devuelve la distancia al eje de coordenadas
      public float distancia() {
          return (float) Math.sqrt(Math.pow(x,2.0) +
                  Math.pow(y,2.0)+Math.pow(z,2.0));
      }
}
```

Se puede observar que hemos sobre escrito el método distancia en la clase punto 3D para que devuelva la distancia hasta el eje de coordenadas desde una posición en el espacio tridimensional.

Si tenemos que escribir algún método general, que pueda tratar tanto con Puntos como con Puntos3D, posiblemente deberemos hacer algún *casting* de objetos, y para saber con cuál

clase instanciada estamos tratando, deberemos usar el operador **instanceof** como se muestra a continuación:

```
public static void main(String[] args){
    Punto p1 = new Punto(5, 5);
    Punto3D p2 = new Punto3D(5, 5, 5);
    Punto p3 = new Punto3D(10, 10, 10);

    if(p1 instanceof Punto){
        System.out.println("p1 instancia de Punto");
    }
    if(p1 instanceof Punto3D){
        System.out.println("p1 instancia de Punto3D");
    }
    if(p2 instanceof Punto){
        System.out.println("p2 instancia de Punto");
    }
    if(p2 instanceof Punto3D){
        System.out.println("p2 instancia de Punto3D");
    }
    if(p3 instanceof Punto){
        System.out.println("p3 instancia de Punto");
    }
    if(p3 instanceof Punto3D){
        System.out.println("p3 instancia de Punto3D");
    }
}
```

El resultado obtenido es el siguiente:

```
p1 instancia de Punto
p2 instancia de Punto
p2 instancia de Punto3D
p3 instancia de Punto
p3 instancia de Punto3D
```