OPERADORES

- Existen diferentes tipos de operadores en Java:
  - Operadores **Aritméticos**.
  - Operadores de **Asignación**.
  - Operadores **Incrementales**.
  - Operadores **Relacionales**.
  - Operadores **Lógicos**.
  - Operadores **Unarios**.
  - Operador **Condicional**.
  - Operador de **Concatenación** de Cadenas de caracteres.
  - Operador **instanceof**.

TIPOS DE DATOS PRIMITIVOS Y OPERADORES

OPERADORES ARITMÉTICOS

- Son operadores binarios (requieren siempre de dos operandos) que realizan las operaciones aritméticas habituales: **suma** (+), **resta** (-), **multiplicación** (*), **división** (/) y **resto de la división** (%).

Operación	Operador	Expresión
Suma	+	`f + 7`
Resta	-	`p - c`
Multiplicación	*	`b * m`
División	/	`x / y`
Módulo	%	`r % s`

- Ejemplos:

```
int result, a = 8, b = 2;
result = a % b;          /* result = 0 */
result = a - b;          /* result = 6 */
b = 3;
result = a % b;          /* 8 % 3 --> result = 2 */
result = a / 5;          /* 8 / 5 --> result = 1 */
```

OPERADORES ARITMÉTICOS

- Diferencias de la división según el tipo de los operandos:

  - Si uno o más de los operandos es una variable de tipo punto flotante (**double, float**), entonces la división es real.

  - Si los 2 operandos son variables enteras (**int, long, short, byte**), entonces la división es entera.

Tipo de División	Resultado	Ejemplo
int / int	int	3 / 2 = 1
int / double	double	3.0 / 2.0 = 1.5
double / int	double	1.0 / 2 = 0.5
double / double	double	6.0 / 5.0 = 1.2

OPERADORES ARITMÉTICOS

× *Reglas de Precedencia de Operadores Aritméticos.*

+ Las expresiones contenidas entre paréntesis se evalúan primero, aplicando primero los operadores del par de paréntesis más interno.

+ A continuación se aplican las operaciones de multiplicación, división y residuo de izquierda a derecha.

+ A continuación se aplican las operaciones de suma y resta de izquierda a derecha.

× Ejemplo:

```
                        z = p * r % q + w / x - y
orden de evaluación:        1   2   4   3   5

                        y = a * x * x + b * x + c
orden de evaluación:        1   2   4   3   5

equivalente a:          y = (a * x * x) + (b * x) + c
```

OPERADORES DE ASIGNACIÓN

- Los operadores de Asignación permiten asignar un valor a una variable.
- El operador de asignación por excelencia en el operador igual (=).
- La forma general de las sentencias de asignación con este operador es:
  - Variable = expresion;
- Java dispone de versiones abreviadas del operador (=) que realizan operaciones "acumulativas" sobre una variable:

Operador	Utilización	Expresión equivalente
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2

- Ejemplo:
  - int a = 2, b = 3; /* Se declaran dos variables enteras a = 2, b = 3 */
  - a += b; /* a = a + b = 2 + 3 = 5 */

OPERADORES INCREMENTALES

+ Java dispone del operador **incremento** (++) y **decremento** (--).
  - × El operador (**++**) incrementa en una unidad la variable a la que se aplica.
  - × El operador (**--**) la reduce en una unidad.

+ Estos operadores se pueden utilizar de dos formas:
  - × **Precediendo** a la variable (++**i**): en este caso primero se incrementa la variable y luego se utiliza (ya incrementada) en la expresión en la que aparece.
  - × **Post** a la variable (**i**++): en este caso primero se utiliza la variable en la expresión (con el valor anterior) y luego se incrementa.

+ Ejemplos:
  - × int a = 7, b; /* a = 7 y b = 0 */
  - × b = a++; /* primero b = 7 y luego a = a + 1 → 8 */
    - * La variable **b** toma el valor de **a** (o sea b = 7). Luego de la asignación, la variable **a** se incrementa, tomando un valor final de 8.
  - × a = 8;
  - × b = ++a;
    - * En este caso , lo primero que se hace es incrementar la variable **a** , para luego asignar ese valor a la variable **b** (o sea b = 9).

OPERADORES INCREMENTALES

- Ejemplos:
  - int a = 2, b = 3; /* Se declaran dos variables enteras a = 2, b = 3 */
  - a += b; /* a = a + b = 2 + 3 = 5 */
  - b *= 5; /* b = b * 5 = 3 * 5 = 15 */
  - a = ++b; /* b = b + 1 = 15 + 1 = 16 ; a = b = 16 */
    - Como existe una variable que se está pre-incrementando, lo primero que se hace es modificar el valor de esa variable. Después de haber hecho esto, se evalúan los valores.
  - b += --a + 5; /* a = a - 1 = 16 - 1 = 15 ; b = b + a + 5 = 16 + 15 + 5 = 36 */
    - Después de incrementar la variable **a**, se escribe y evalúa la expresión equivalente.
  - a += b--; /* a = a + b = 15 + 36 = 51 ; b = b - 1 = 36 - 1 = 35 */
    - Como ahora b, se está post-incrementando, lo primero que se hace es evaluar la expresión equivalente, y después finalmente incrementar la variable b.Valores finales : a = 51 , b = 35.

OPERADORES RELACIONALES

- Los operadores relacionales sirven para realizar comparaciones de **igualdad, desigualdad** y relación de **mayor** o **menor**.

- El resultado de estos operadores es siempre un valor boolean (**true** o **false**) según se cumpla o no la relación considerada. Son fundamentales para implementar las condiciones de las sentencias de control.

Operador	Descripción	Ejemplo
<	menor	a < b
<=	menor o igual	a++ <= 122
>	mayor	x > 5 * b++
>=	mayor o igual	p >= 0.01
==	igual	a%2 == 0
!=	distinto	t != true

- Ejemplo :

```
+   5 > 1          /* true */
+   1 >= 5         /* false */
+   2 == 2         /* true */
+    2 != 2        /* false */
```

OPERADORES LÓGICOS

- Java proporciona operadores lógicos que se usan para simular los conceptos **Y**, **O** y **NO** de algebra de Boole. Estos se conocen con el nombre de conjunción, disyunción y negación respectivamente, y los operadores correspondientes son **&&**, **||** y **!**.

- Cabe mencionar que en ciertos casos el segundo operando no se evalúa porque ya no es necesario (si ambos tienen que ser *true* y el primero es *false*, ya se sabe que la condición de que ambos sean *true* no se va a cumplir).

Operador	Descripcion	Ejemplo
&&	AND (y)	f > 0.0 && f <= 1.0
||	OR (o)	e == 1 || d == 2
!	NOT (no)	!encendido

OPERADORES LÓGICOS

- En la siguiente tabla se muestran los resultados de aplicar los operadores lógicos para todos los posibles valores de entrada.

X	Y	X && Y	X || Y	!X
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

- Ejemplos:
  - true || false && true → true
  - (false || false) && (!true) → false
  - !true && !false → false
  - !(5 > 7 || 8 > 0 && 3 == 2) → true

```
sexo == mujer &&  edad >= 65
```

Si `sexo` no es igual a `mujer` no se evalúa el segundo operando `edad >= 65`.

```
notaLab == noApto || notaTeoría < 5
```

Si `notaLab` es `noApto` no se evalúa el segundo operando.

OPERADORES UNARIOS

- Los operadores **más** (+) y **menos** (-) unarios sirven para mantener o cambiar el signo de una variable, constante o expresión numérica.
  - Ejemplo:
    - x = +a;
    - y = -8;

OPERADOR CONDICIONAL

- Este operador (**?**) permite realizar expresiones condicionales sencillas. Su forma general es la siguiente:

  + booleanExpression **?** res1 **:** res2;
    - Donde se evalúa booleanExpression y se devuelve *res1* si el resultado es **true** y *res2* si el resultado es **false**. Es el único operador ternario (3 argumentos) de Java.

  + Ejemplo:
    - x = 1; y = 10; z = (x < y) ? x + 3 : y + 8; → z = 4

TIPOS DE DATOS PRIMITIVOS Y OPERADORES

OPERADOR DE CONCATENACIÓN

- En Java el operador (+) se utiliza también para concatenar cadenas de caracteres.

- Ejemplo:
  - System.out.println("El total a cobrar es: " + cantidad + "pesos ");

    - Donde el operador de concatenación se utiliza 2 veces para construir la cadena de caracteres que se desea imprimir por medio del método *println()*.

OPERADOR instanceof

- En Java este operador nos permite saber si un objeto pertenece o no a una determinada clase. Es un operador binario cuya forma general es:

  - objectName **instanceof** ClassName

    - Devuelve *true* o *false* según el objeto pertenezca o no a la clase.

OPERADORES INCREMENTALES

- Ejemplos:
  - int a = 2, b = 3; /* Se declaran dos variables enteras a = 2, b = 3 */
  - a += b; /* a = a + b = 2 + 3 = 5 */
  - b *= 5; /* b = b * 5 = 3 * 5 = 15 */
  - a = ++b; /* b = b + 1 = 15 + 1 = 16 ; a = b = 16 */
    - Como existe una variable que se está pre-incrementando, lo primero que se hace es modificar el valor de esa variable. Después de haber hecho esto, se evalúan los valores.
  - b += --a + 5; /* a = a - 1 = 16 - 1 = 15 ; b = b + a + 5 = 16 + 15 + 5 = 36 */
    - Después de incrementar la variable **a**, se escribe y evalúa la expresión equivalente.
  - a += b--; /* a = a + b = 15 + 36 = 51 ; b = b - 1 = 36 - 1 = 35 */
    - Como ahora b, se está post-incrementando, lo primero que se hace es evaluar la expresión equivalente, y después finalmente incrementar la variable b. Valores finales : a = 51 , b = 35.

RESUMEN DE OPERADORES, ASOCIATIVIDAD Y PRECEDENCIA

Operadores	Asociatividad	Tipo
( )	Izda a dcha	Paréntesis
++ -- !	Dcha a izda	Unarios
* / %	Izda a dcha	Multiplicativos
+ -	Izda a dcha	Aditivos
< <= > >=	Izda a dcha	Relacionales
== !=	Izda a dcha	De igualdad
&&	Izda a dcha	AND lógico
||	Izda a dcha	OR lógico
?:	Dcha a izda	Condicional
= += -= *= /= %=	Dcha a izda	Asignación

TIPOS DE DATOS PRIMITIVOS

- Los rangos y la memoria que ocupa cada uno de estos tipos de dato primitivo son los siguientes:

Tipo	Lo que almacena	Rango
byte	entero de 1 byte (8 bits)	de -128 a 127
short	entero de 2 byte (16 bits)	de -32768 a 32767
int	entero de 4 byte (32 bits)	de -2147483648 a 2147483647
long	entero de 8 byte (64 bits)	de -2^{-63} a $2^{63} - 1$
float	entero de 4 byte (32 bits)	6 dígitos significativos (10^{-46} , 10^{38})
double	entero de 8 byte (64 bits)	15 dígitos significativos (10^{-324} , 10^{308})
char	caracter UNICODE 2 bytes (16 bits)	Comprende el código ASCII
boolean	variable booleana de 1 byte (8 bits)	false y true

TIPOS DE DATOS PRIMITIVOS

- Se llaman ***tipos primitivos*** de variables de Java a aquellas variables sencillas que contienen los tipos de información más habituales: valores boolean, caracteres y valores numéricos enteros o de punto flotante.

- Java cuenta con 8 tipos de datos primitivos para almacenar distintos rangos de valores:

  + 1 tipo para almacenar valores *true* y *false* (**boolean**);

  + 1 tipo para almacenar caracteres (**char**) y;

  + 6 tipos para guardar valores numéricos, de los cuales 4 son para enteros (**byte**, **short**, **int** y **long**) y 2 para valores reales de punto flotante (**float** y **double**).

TIPOS DE DATOS PRIMITIVOS

- ¿Cómo se definen e Inicializan la variables?

  + Cualquier variable, se declara proporcionando su **tipo**, su **nombre**, y opcionalmente su **valor inicial**.

  + Las variables pueden ser tanto de tipo **primitivo** como **referencias** a objetos de alguna clase perteneciente a Java.

  + Si no se especifica un valor en su declaración, las variables primitivas se inicializan a **cero** (salvo boolean que se inicializa a ***false*** y char a **'\0'**).

  + Las variables de tipo referencia son inicializadas a ***null***.

  + Ejemplos:

    - **int a;** /* declara una variable entera de 32 bits */
    - **double b;** /* declara una variable de punto flotante de 64 bits */

TIPOS DE DATOS PRIMITIVOS

- Ahora bien, si se quiere declarar una lista de variables, se debe usar el separador coma.
  - **int x, y, z;** /* declara tres variables enteras */
  - **boolean est, wq;** /* declara dos variables lógicas */

- Para dar un valor inicial a la variable, se debe usar el operador asignación =.

```
int contador = 0,sum = 0;
double tiempo, radio = 1.0;
boolean encendido = false;
char entrada = 's', salida = 'w';
int s, t = 100;          /* s tiene un valor inicial cero y t un valor cien */
 double k = 3.6, u;      /* k vale 3.6 y u vale 0.0 */
boolean par;             /* par vale false */
char c, f = 'x';          /* c vale '\0' y f vale 'x' */
```

TIPOS DE DATOS PRIMITIVOS

- Ejemplos de Variables Primitivas y Variables Referencia.

```
int x;                  // Declaración de la variable primitiva x. Se inicializa a 0
int y = 5;              // Declaración de la variable primitiva y. Se inicializa a 5

MyClass unaRef;                     // Declaración de una referencia a un objeto MyClass.
                                    // Se inicializa a null
unaRef = new MyClass();             // La referencia "apunta" al nuevo objeto creado
                                    // Se ha utilizado el constructor por defecto
MyClass segundaRef = unaRef;        // Declaración de una referencia a un objeto MyClass.
                                    // Se inicializa al mismo valor que unaRef
int [] vector;          // Declaración de un array. Se inicializa a null
vector = new int[10]; // Vector de 10 enteros, inicializados a 0
double [] v = {1.0, 2.65, 3.1};// Declaración e inicialización de un vector de 3
                                    // elementos con los valores entre llaves
MyClass [] lista=new MyClass[5];// Se crea un vector de 5 referencias a objetos
                                    // Las 5 referencias son inicializadas a null
lista[0] = unaRef;                  // Se asigna a lista[0] el mismo valor que unaRef
lista[1] = new MyClass();           // Se asigna a lista[1] la referencia al nuevo objeto
                                    // El resto (lista[2]…lista[4] siguen con valor null
```