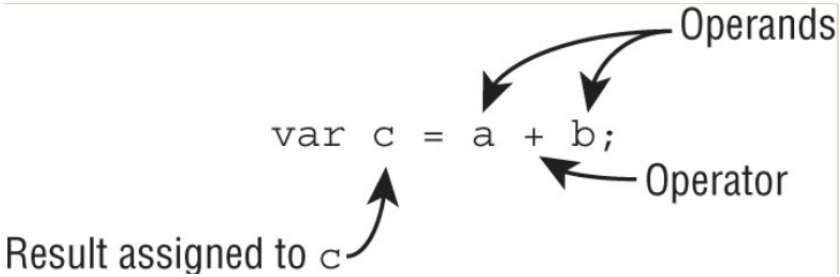
Operadores

Un operador de Java es un símbolo especial que se aplica a variables, valores, o literales el cual retorna un resultado.



Java tiene 3 tipos de operadores: unario, binario y ternario

Operator	Symbols and examples	Evaluation
Post-unary operators	*expression*++, *expression*--	Left-to-right
Pre-unary operators	++*expression*, --*expression*	Left-to-right
Other unary operators	-, !, ~, +, (*type*)	Right-to-left
Cast	(Type)reference	Right-to-left
Multiplication/division/modulus	*, /, %	Left-to-right
Addition/subtraction	+, -	Left-to-right
Shift operators	<<, >>, >>>	Left-to-right
Relational operators	<, >, <=, >=, instanceof	Left-to-right
Equal to/not equal to	==, !=	Left-to-right
Logical AND	&	Left-to-right
Logical exclusive OR	^	Left-to-right
Logical inclusive OR	|	Left-to-right
Conditional AND	&&	Left-to-right
Conditional OR	||	Left-to-right
Ternary operators	*boolean expression* ? *expression1* : *expression2*	Right-to-left
Assignment operators	=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=	Right-to-left
Arrow operator	->	Right-to-left

Operator	Examples	Description
Logical complement	!a	Inverts a boolean's logical value
Bitwise complement	~b	Inverts all 0s and 1s in a number
Plus	+c	Indicates a number is positive, although numbers are assumed to be positive in Java unless accompanied by a negative unary operator
Negation or minus	-d	Indicates a literal number is negative or negates an expression
Increment	++e f++	Increments a value by 1
Decrement	--f h--	Decrements a value by 1
Cast	(String)i	Casts a value to a specific type

Figura 1: operadores unarios

Para el examen, también necesitas saber sobre el operador de complemento a uno (~), el cual invierte todos los 0s y 1s de un número. Solo puede aplicarse a tipos numéricos enteros como ***byte***, ***short***, ***char***, ***int*** y ***long***. Intentemos un ejemplo. Por simplicidad, solo mostraremos los últimos cuatro bits (en lugar de todos los 32 bits).

```
1 int value = 3 // es 0011
2 int comp = ~value; // es 1100
3 System.out.println(value); // 3
4 System.out.println(comp); // -4
```

No necesitas saber cómo hacer aritmética binaria complicada en el examen, siempre y cuando recuerdes esta regla: para encontrar el complemento a uno de un número, multiplícalo por -1 y luego resta uno.

```
1 int pelican = !5; // no compila porque no se puede hacer negación de un número
2 boolean pen = -true; // no compila porque no se puede hacer negativo un booleano
3 boolean pek = !0; // no compila porque no se puede negar un número
```

los operadores de incremento y decremento se aplican a valores numéricos y tienen mas precedencia que los operadores binarios o sea se aplican primero que ellos. para este tipo de operadores debemos tener en cuenta que importa la forma en que se aplica al numero, como se detalla en la siguiente imagen:

Operator	Example	Description
Pre-increment	++w	Increases the value by 1 and returns the *new* value
Pre-decrement	--x	Decreases the value by 1 and returns the *new* value
Post-increment	y++	Increases the value by 1 and returns the *original* value
Post-decrement	z--	Decreases the value by 1 and returns the *original* value

```
1 int park = 0;
2 System.out.println(park); // 0
3 System.out.println(++park); // 1
4 System.out.println(park); // 1
5 System.out.println(park--); // 1
6 System.out.println(park); // 0
```

los operadores aritméticos binarios van a ser los más comunes, las operaciones de multiplicación, división y residuo tienen mas precedencia que la suma y resta.

Operator	Example	Description
Addition	a + b	Adds two numeric values
Subtraction	c - d	Subtracts two numeric values
Multiplication	e * f	Multiplies two numeric values
Division	g / h	Divides one numeric value by another
Modulus	i % j	Returns the remainder after division of one numeric value by another

int price = 2 * 5 + 3 * 4 - 8 (primero se evalúa 2 * 5 y 3 * 4) quedando:

int price = 10 + 12 - 8 (después se evalúa de izquierda a derecha obteniendo 14)

los paréntesis modifica el orden de ejecución por ejemplo:

```
1 int price = 2 * ((5 + 3) * 4 - 8);
2 int price = 2 * (8 * 4 - 8);
3 int price = 2 * (32 - 8);
4 int price = 2 * 24;
```

hay que tener cuidado ya que siempre deben haber paréntesis completos para que el código compile. También tener en cuenta que solo los paréntesis se permiten en operaciones, las llaves o corchetes no son aceptados por java.

el operador modulo (%) permite obtener el residuo de una división.

```
1 System.out.println(10/3); // 3
2 System.out.println(10%3); // 1
```

para números decimales el valor del piso (floor) quiere decir que en 4.03, 4.3, 4.9999 va ser 4.

La promoción numérica es un proceso automático en Java donde un tipo de dato numérico de menor rango se convierte temporalmente a un tipo de dato de mayor rango durante una operación aritmética. Esto se hace para evitar la pérdida de precisión y garantizar que el resultado de la operación sea correcto.

Reglas:

- Si dos valores tienen tipos de datos diferentes, Java promoverá automáticamente uno de los valores al tipo de dato más grande de los dos.
- Si uno de los valores es un número entero y el otro es de punto flotante, Java promoverá automáticamente el valor entero al tipo de dato de punto flotante.
- Los tipos de datos más pequeños, como ***byte***, ***short*** y ***char***, se promocionan primero a ***int*** cada vez que se usan con un operador aritmético binario de Java con una variable (en oposición a un valor), incluso si ninguno de los operando es ***int***.

Para la tercera regla, ten en cuenta que los operadores unarios están excluidos de esta regla. Por ejemplo, aplicar ++ a un valor ***short*** resulta en un valor ***short***.

```
int x = 1;
log y = 33;
var z = x * y; // z va ser long

double x = 33.21;
float y = 2.1; // esto arroja error de compilación ya que debe acabar en "f"
var z = x + y; // si se corrige la linea anterior z seria double

short x = 10;
short y = 3;
var z = x * y; // el valor de z sera int por la tercera regla

short w = 14; // se promueve a int y después a float
float x = 13;
double y = 30;
var z = w * x / y; // W * X se promueve a double y se opera con Y
```

el símbolo "=" sirve como operador de asignación y lo hace de derecha a izquierda. arrojara un error si detecta que se desea asignar a la izquierda un valor mas grande desde la derecha, sin haber echo casting.

casteo de valores: La conversión de tipos (casting) es una operación unaria en la que un tipo de dato se interpreta explícitamente como otro tipo de dato. La conversión es obligatoria al convertir a un tipo de dato más pequeño (narrowing). Sin la conversión, el compilador generará un error al intentar colocar un tipo de dato más grande dentro de uno más pequeño. Aquí hay algunos ejemplos de conversión de tipos:

```
int fur = (int) 5; // compila
short tail = (short) (4 + 10); // compila, se pone la suma en paréntesis para que se aplique al resultado.
long x1 = 10(long); // no compila
```

hay que tener en cuenta que el casteo en números cambia el tipo entre estos mientras que el casteo de objetos solo cambia la referencia a un objeto, no el objeto en si mismo.

```
float egg = 2.0 / 9; // no compila porque el resultado double no se puede asignar a un float directamente.

int tadpole = (int)5 * 2L; // no compila porque el casting se aplica al 5 y el resultado de la operación va ser un double.

short frog = 3 - 2.0; // no compila porque el resultado es un double

short a = 10;
short b = 3;
short c = a * b; // no compila porque se vuelve los short se vuelven int por defecto al operarse y después no se puede guardar en un short si no se castea.
```

Operator	Example	Description
Addition assignment	a += 5	Adds the value on the right to the variable on the left and assigns the sum to the variable
Subtraction assignment	b -= 0.2	Subtracts the value on the right from the variable on the left and assigns the difference to the variable
Multiplication assignment	c *= 100	Multiplies the value on the right with the variable on the left
		and assigns the product to the variable
Division assignment	d /= 4	Divides the variable on the left by the value on the right and assigns the quotient to the variable

estos operadores también sirven para hacer un casteo interno por ejemplo:

```
long a1 = 10;
int a2 = 5;
a2 *= a1; // a2 primero se va castear a long y después de multiplicarse con a1 va castearse nuevamente a int.
```

retorno y asignación de valores: el siguiente ejemplo es totalmente correcto:

```
1 long a1 = 5;
2 long a2 = (a1 = 3);
3 System.out.println(a1); // 3
4 System.out.println(a2); // 3
```

variables de comparación: Cuando comparas dos valores en Java, es importante entender si estás comparando si son exactamente el mismo objeto en memoria (usando ==) o si tienen el mismo valor (usando == para tipos primitivos o equals() para objetos)

Operator	Example	Apply to primitives	Apply to objects
Equality	a == 10	Returns true if the two values represent the same value	Returns true if the two values reference the same object
Inequality	b != 3.14	Returns true if the two values represent different values	Returns true if the two values do not reference the same object

```
1 var a1 = new File("diario.txt");
2 var a2 = new File("diario.txt");
3 var a3 = a2;
4 System.out.println(a1 == a2); // FALSE porque cada uno esta referenciado a diferentes objetos
5 System.out.println(a2 == a3); // TRUE porque cada uno esta referenciado al mismo objeto
6 System.out.println(null == null); // TRUE porque hace referencial mismo objeto, no cae en error
```

Operadores de relación: permiten comparar 2 valores y retornar un booleano. las 4 primeras opciones son para datos numéricos y si ambos números no son del mismo tipo el numero menor es promovido al tipo del mayor.

Operator	Example	Description
Less than	a < 5	Returns true if the value on the left is strictly *less than* the value on the right
Less than or equal to	b <= 6	Returns true if the value on the left is *less than or equal* to the value on the right
Greater than	c > 9	Returns true if the value on the left is strictly *greater than* the value on the right
Greater than or equal to	3 >= d	Returns true if the value on the left is *greater than or equal* to the value on the right
Type comparison	e instanceof String	Returns true if the reference on the left side is an instance of the type on the right side (class, interface, record, enum, annotation)

el operador ***instanceof***, es útil para determinar, en tiempo de ejecución, si un objeto arbitrario es miembro de una clase o interfaz en particular.

```
1 Integer zooTime = Integer.valueOf(9);
2 Number num = zooTime;
3 Object obj = zooTime; //  obj es instancia de Number e Integer
```

si Java detecta que los tipos de datos con ***instanceof*** no puede ser casteada para evaluarse arrojara error.

Respecto a ***null*** debemos saber que ***instanceof*** aplicado a un ***null*** literal o referenciado siempre arrojara ***false***.

```java
Obj a1 = null;
System.out.println(a1 instanceof String); // false
System.out.println(null instanceof null); // esta nunca compilara
```

operadores lógicos: Son ***&, |,*** ^ que pueden ser aplicados a ***boolean*** o numéricos, tener en cuenta que para estos operadores. ***AND*** solo es si ambos son verdaderos. ***OR*** solo da falso si ambos son falsos y ***XOR*** da verdadero si ambos son diferentes.

Operator	Example	Description
Logical AND	a & b	Value is true only if both values are true.
Logical inclusive OR	c | d	Value is true if at least one of the values is true.
Logical exclusive OR	e ^ f	Value is true only if one value is true and the other is false.

AND (x & y)

	y = true	y = false
x = true	true	false
x = false	false	false

INCLUSIVE OR (x | y)

	y = true	y = false
x = true	true	true
x = false	true	false

EXCLUSIVE OR (x ^ y)

	y = true	y = false
x = true	false	true
x = false	true	false

operadores condicionales:

Operator	Example	Description
Conditional AND	a && b	Value is true only if both values are true. If the left side is false, then the right side will not be evaluated.
Conditional OR	c || d	Value is true if at least one of the values is true. If the left side is true, then the right side will not be evaluated.

```java
int hour = 10;
boolean z1 = true || (hour < 4);
System.out.println(z1); // siempre arrojara verdad ya que para el tipo "O" si el primero es verdad ya no se evalúa el segundo.
```

uso de operadores condicionales para evitar los ***NullPointerException***: Un ejemplo muy común donde se utilizan los operadores condicionales es para verificar si un objeto es nulo antes de realizar una operación. En el siguiente ejemplo, si **duck** es nulo, el programa lanzará una excepción ***NullPointerException*** en tiempo de ejecución:

```java
if (duck != null & duck.getAge() < 5) { // Podría lanzar una NullPointerException
    // Hacer algo
}
```

El problema es que el operador lógico ***AND (&)*** evalúa ambos lados de la expresión. Podríamos agregar una segunda condición ***if***, pero esto se volvería engorroso si tenemos muchas variables que verificar. Una solución fácil de leer es usar el operador ***AND*** condicional (***&&***):

```java
if (duck != null && duck.getAge() < 5) {
    // Hacer algo
}
```

En este ejemplo, si ***duck*** es nulo, la condición impide que se lance una ***NullPointerException***, ya que la evaluación de ***duck.getAge() < 5*** nunca se alcanza.

operador ternario: sirve para evaluaciones de **if** sencillas y tiene la forma.

booleanExpression ? ResultexpressionTrue : ResultexpressionFalse

```java
int a1 = 5;
int b1 = a1 < 2 ? 3 : 4;
System.out.println(b1); // 4
```