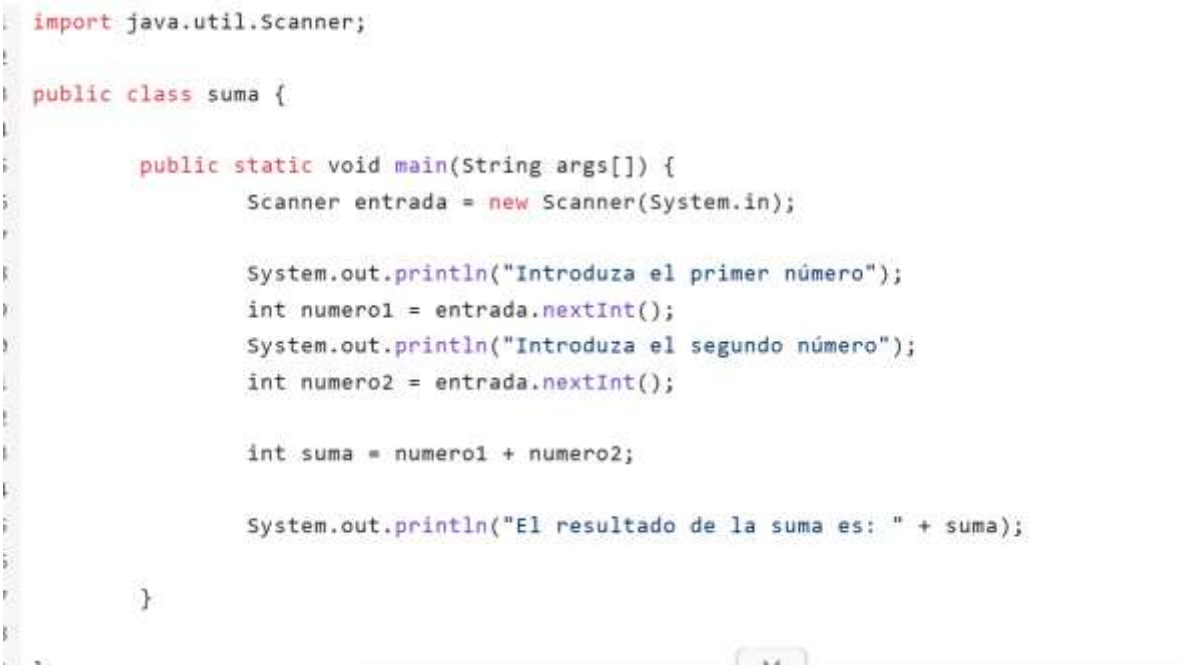
Cuáles son los operadores aritméticos en Java?

En Java existen varios operadores aritméticos que se utilizan para realizar operaciones matemáticas. Los **operadores aritméticos** en Java son los siguientes:

- Suma: (**+**) Este operador se utiliza para sumar dos valores.
- Resta: (**–**) Este operador se utiliza para restar dos valores.
- Multiplicación: (*****) Este operador se utiliza para multiplicar dos valores.
- División: (**/**) Este operador se utiliza para dividir dos valores.
- Módulo: (**%**) Este operador se utiliza para obtener el resto de la división de dos valores.
- Incremento: (**++**) Este operador se utiliza para aumentar en 1 el valor de una variable.
- Decremento: (**—**) Este operador se utiliza para disminuir en 1 el valor de una variable.

Estos operadores se pueden utilizar para realizar cálculos matemáticos simples o complejos en programas de Java.

A continuación te mostrare ejemplos sencillos de como utilizar todos los operadores aritmeticos que nos ofrece Java. Al final del articulo haremos un ejercicio donde utilizaremos funciones para realizar todas la operaciones aritméticas en java y hacer un código mas legible y limpio.



```java
import java.util.Scanner;

public class suma {

	public static void main(String args[]) {
		Scanner entrada = new Scanner(System.in);

		System.out.println("Introduza el primer número");
		int numero1 = entrada.nextInt();
		System.out.println("Introduza el segundo número");
		int numero2 = entrada.nextInt();

		int suma = numero1 + numero2;

		System.out.println("El resultado de la suma es: " + suma);

	}
```

Los **operadores** son símbolos especiales que le indican a Java cómo manipular los datos de nuestros programas. En esta sección exploraremos los operadores más fundamentales: los **aritméticos** para realizar cálculos matemáticos y los de **asignación** para almacenar valores en variables.

Los operadores aritméticos funcionan de manera similar a las **matemáticas básicas** que conoces, pero con algunas particularidades específicas de la programación. Estos operadores nos permiten crear expresiones que el compilador evalúa para producir un resultado.

Operadores aritméticos básicos

Java proporciona cinco **operadores aritméticos fundamentales** que trabajan con tipos numéricos:

- **Suma (+)**: Añade dos valores
- **Resta (-)**: Sustrae el segundo valor del primero
- **Multiplicación (*)**: Multiplica dos valores
- **División (/)**: Divide el primer valor entre el segundo
- **Módulo (%)**: Devuelve el resto de una división entera

```
public class OperadoresAritmeticos {
  public static void main(String[] args) {
```

```
        int a = 15;
        int b = 4;

        // Operaciones básicas
        int suma = a + b;       // 19
        int resta = a - b;      // 11
        int multiplicacion = a * b;  // 60
        int division = a / b;    // 3 (división entera)
        int modulo = a % b;     // 3 (resto de 15/4)

        System.out.println("Suma: " + suma);
        System.out.println("División: " + division);
        System.out.println("Módulo: " + modulo);
    }
}
```

Es importante entender que cuando trabajamos con **tipos enteros** (int, long), la división produce un resultado entero, descartando la parte decimal. Si necesitamos decimales, debemos usar tipos como **double** o **float**:

```
int entero1 = 15;
int entero2 = 4;
int divisionEntera = entero1 / entero2;  // 3

double decimal1 = 15.0;
double decimal2 = 4.0;
double divisionDecimal = decimal1 / decimal2;  // 3.75
```

El **operador módulo** (**%**) es especialmente útil para determinar si un número es par o impar, o para crear ciclos que se repiten cada cierto número de iteraciones:

```
int numero = 17;
int resto = numero % 2;


if (resto == 0) {
   System.out.println("El número es par");
} else {
   System.out.println("El número es impar");
}
```

Operadores de asignación

El **operador de asignación básico** (**=**) almacena un valor en una variable. Sin embargo, Java proporciona operadores de asignación **compuestos** que combinan una operación aritmética con la asignación, haciendo el código más conciso.

- **Asignación simple (=)**: Asigna un valor a una variable
- **Asignación con suma (+=)**: Suma y asigna el resultado
- **Asignación con resta (-=)**: Resta y asigna el resultado
- **Asignación con multiplicación (*=)**: Multiplica y asigna el resultado
- **Asignación con división (/=)**: Divide y asigna el resultado
- **Asignación con módulo (%=)**: Calcula el módulo y asigna el resultado

```
public class OperadoresAsignacion {
   public static void main(String[] args) {
      int contador = 10;


      // Asignación compuesta equivale a: contador = contador + 5
      contador += 5;   // contador ahora vale 15


      contador -= 3;   // contador ahora vale 12
      contador *= 2;   // contador ahora vale 24
```

```
        contador /= 4;   // contador ahora vale 6
        contador %= 5;   // contador ahora vale 1

        System.out.println("Valor final: " + contador);
    }
}
```

Los operadores de asignación compuestos son **más eficientes** y legibles que escribir la operación completa. Compara estos dos enfoques:

```
// Forma larga (menos eficiente)
puntuacion = puntuacion + bonificacion;

// Forma corta (más eficiente y clara)
puntuacion += bonificacion;
```

Operadores de incremento y decremento

Java incluye dos **operadores especiales** para incrementar o decrementar una variable en una unidad:

- **Incremento (++)**: Aumenta el valor en 1
- **Decremento (--)**: Disminuye el valor en 1

Estos operadores tienen dos **variantes** según su posición respecto a la variable:

```
int numero = 5;

// Pre-incremento: incrementa primero, luego usa el valor
int preIncremento = ++numero;   // numero = 6, preIncremento = 6

// Post-incremento: usa el valor actual, luego incrementa
int postIncremento = numero++;  // postIncremento = 6, numero = 7
```

```
System.out.println("Número final: " + numero);  // 7
```

La **diferencia** entre pre y post incremento es crucial en expresiones complejas:

```
int a = 10;

int b = 10;


// Pre-incremento: incrementa a antes de asignar

int resultado1 = ++a * 2;  // a = 11, resultado1 = 22


// Post-incremento: usa b actual, luego incrementa

int resultado2 = b++ * 2;  // resultado2 = 20, b = 11


System.out.println("a: " + a + ", b: " + b);

System.out.println("resultado1: " + resultado1 + ", resultado2: " + resultado2);
```

Precedencia y asociatividad

Los operadores siguen reglas de **precedencia** similares a las matemáticas. Los operadores con mayor precedencia se evalúan primero:

1. **Incremento/Decremento** (**++**, **--**)
2. **Multiplicación, División, Módulo** (*****, **/**, **%**)
3. **Suma, Resta** (**+**, **-**)
4. **Asignación** (**=**, **+=**, **-=**, etc.)

```
int resultado = 2 + 3 * 4;  // 14, no 20 (se evalúa 3*4 primero)


// Para cambiar el orden, usamos paréntesis

int resultadoConParentesis = (2 + 3) * 4;  // 20
```

Es una **buena práctica** usar paréntesis para hacer explícito el orden de evaluación, especialmente en expresiones complejas:

```
double promedio = (nota1 + nota2 + nota3) / 3.0;
```

```
int puntos = (basePoints * multiplier) + bonusPoints;
```

Los operadores aritméticos y de asignación forman la base de la mayoría de cálculos en Java. Dominar su uso y entender cómo se evalúan las expresiones te permitirá escribir código **más eficiente** y fácil de mantener.

Operadores de comparación y lógicos

Los **operadores de comparación** nos permiten evaluar relaciones entre valores y obtener un resultado booleano (**true** o **false**). Estos operadores son fundamentales para crear **condiciones** que determinen el flujo de ejecución de nuestros programas.

A diferencia de los operadores aritméticos que producen valores numéricos, los operadores de comparación siempre devuelven un **valor booleano**. Este resultado se puede almacenar en variables de tipo **boolean** o usar directamente en estructuras de control.

Operadores de comparación básicos

Java proporciona seis **operadores de comparación** que evalúan la relación entre dos valores:

- **Igual a (==)**: Verifica si dos valores son exactamente iguales
- **Distinto de (!=)**: Verifica si dos valores son diferentes
- **Mayor que (>)**: Verifica si el primer valor es mayor que el segundo
- **Menor que (<)**: Verifica si el primer valor es menor que el segundo
- **Mayor o igual que (>=)**: Verifica si el primer valor es mayor o igual que el segundo
- **Menor o igual que (<=)**: Verifica si el primer valor es menor o igual que el segundo

```
public class OperadoresComparacion {

  public static void main(String[] args) {

    int edad = 18;

    int edadMinima = 18;


    boolean esIgual = edad == edadMinima;     // true
```

```
        boolean esDiferente = edad != 21;        // true
        boolean esMayor = edad > 16;             // true
        boolean esMenor = edad < 25;             // true
        boolean esMayorIgual = edad >= 18;       // true
        boolean esMenorIgual = edad <= 18;       // true

        System.out.println("¿Es mayor de edad? " + esMayorIgual);
        System.out.println("¿Es diferente de 21? " + esDiferente);
    }
}
```

Es crucial entender que el operador **== compara valores**, no referencias. Para tipos primitivos funciona correctamente, pero con objetos como **String** requiere consideraciones especiales:

```
// Comparación correcta con tipos primitivos
int numero1 = 5;
int numero2 = 5;
boolean sonIguales = numero1 == numero2;  // true

// Cuidado con Strings - mejor usar equals()
String texto1 = "Hola";
String texto2 = "Hola";
boolean textosIguales = texto1.equals(texto2);  // true (forma correcta)
```

Operadores lógicos

Los **operadores lógicos** combinan múltiples condiciones booleanas para crear expresiones más complejas. Estos operadores siguen las reglas del **álgebra booleana** y son esenciales para construir lógica de decisión sofisticada.

- **AND lógico (&&)**: Devuelve **true** solo si ambas condiciones son verdaderas

- **OR lógico (||)**: Devuelve **true** si al menos una condición es verdadera
- **NOT lógico (!)**: Invierte el valor booleano (true se convierte en false y viceversa)

```
public class OperadoresLogicos {
  public static void main(String[] args) {
    int edad = 25;
    boolean tieneLicencia = true;
    boolean tieneSeguro = false;

    // AND lógico: ambas condiciones deben ser true
    boolean puedeConducir = edad >= 18 && tieneLicencia;
    System.out.println("¿Puede conducir? " + puedeConducir);  // true

    // OR lógico: al menos una condición debe ser true
    boolean necesitaDocumentacion = !tieneLicencia || !tieneSeguro;
    System.out.println("¿Necesita documentación? " + necesitaDocumentacion);  // true

    // NOT lógico: invierte el resultado
    boolean noEsMenorDeEdad = !(edad < 18);
    System.out.println("¿No es menor de edad? " + noEsMenorDeEdad);  // true
  }
}
```

Evaluación de cortocircuito

Los operadores **&&** y **||** implementan **evaluación de cortocircuito**, una característica que optimiza el rendimiento y puede prevenir errores:

- **Con &&**: Si la primera condición es **false**, no evalúa la segunda
- **Con ||**: Si la primera condición es **true**, no evalúa la segunda

```
int dividendo = 10;
int divisor = 0;

// Sin cortocircuito, esto causaría una excepción por división entre cero
if (divisor != 0 && dividendo / divisor > 2) {
   System.out.println("División válida y resultado mayor que 2");
}

// La segunda condición nunca se evalúa porque divisor == 0
```

Esta característica es **especialmente útil** para verificar condiciones que podrían causar errores:

```
String texto = null;

// Evita NullPointerException usando cortocircuito
if (texto != null && texto.length() > 0) {
   System.out.println("El texto no está vacío: " + texto);
}
```

Combinando operadores

En aplicaciones reales, frecuentemente necesitamos **combinar múltiples operadores** para crear condiciones complejas. La precedencia de operadores determina el orden de evaluación:

1. **NOT lógico** (**!**)
2. **Operadores de comparación** (**<**, **>**, **<=**, **>=**)
3. **Igualdad** (**==**, **!=**)
4. **AND lógico** (**&&**)
5. **OR lógico** (**||**)

```
public class LogicaCompleja {
```

```
  public static void main(String[] args) {
    int puntuacion = 85;
    int asistencia = 90;
    boolean entregoProyecto = true;
    boolean esBecario = false;

    // Condición compleja para aprobar el curso
    boolean aprueba = (puntuacion >= 60 && asistencia >= 80) ||
                      (entregoProyecto && puntuacion >= 50) ||
                      (esBecario && puntuacion >= 40);

    System.out.println("¿Aprueba el curso? " + aprueba);
  }
}
```

Para mejorar la **legibilidad** en expresiones complejas, es recomendable usar paréntesis y dividir la lógica en variables intermedias:

```
// Más legible con variables intermedias
boolean cumpleNotaAsistencia = puntuacion >= 60 && asistencia >= 80;
boolean cumpleProyecto = entregoProyecto && puntuacion >= 50;
boolean cumpleBecario = esBecario && puntuacion >= 40;

boolean aprueba = cumpleNotaAsistencia || cumpleProyecto || cumpleBecario;
```

Aplicaciones prácticas

Los operadores de comparación y lógicos son **fundamentales** en validaciones, filtros de datos y lógica de negocio:

```
public class ValidacionUsuario {
  public static void main(String[] args) {
```

```
        String email = "usuario@email.com";
        String password = "password123";
        int intentosLogin = 2;
        boolean cuentaActiva = true;

        // Validación de login compleja
        boolean emailValido = email != null && email.contains("@");
        boolean passwordValido = password != null && password.length() >= 8;
        boolean intentosPermitidos = intentosLogin < 3;

        boolean puedeLoguearse = emailValido && passwordValido &&
                                 intentosPermitidos && cuentaActiva;

        if (puedeLoguearse) {
            System.out.println("Login exitoso");
        } else {
            System.out.println("Login fallido - revisa las credenciales");
        }
    }
}
```

Comparación con cadenas de texto

Cuando trabajamos con **objetos String**, debemos usar métodos específicos en lugar del operador **==**:

```
String nombre1 = "Ana";
String nombre2 = "ana";
```

```
// Comparación exacta
boolean sonExactamenteIguales = nombre1.equals(nombre2);  // false

// Comparación ignorando mayúsculas/minúsculas
boolean sonSimilares = nombre1.equalsIgnoreCase(nombre2);  // true

// Verificar si está vacío o es null
boolean esValido = nombre1 != null && !nombre1.trim().isEmpty();
```