# Datos y Expresiones

Programación Orientada a Objetos (CC2008) - 2026

# Datos y Expresiones

Semestre 02, 2025

## Cadenas (Strings)

Una cadena puede contener cualquier carácter válido, incluyendo números, signos de puntuación y otros símbolos.
Se escribe entre **comillas dobles**.

Ejemplos:

```
"Hola, mundo!"
"10 de octubre de 2012"
""
```

Una cadena vacía es válida: `""`

En Java, las cadenas pertenecen a la clase `String`.

### Concatenación de cadenas

```
System.out.print("Hola");
System.out.println(" Mundo!");
```

- `System.out.print()` no avanza de línea.
- `System.out.println()` imprime y luego hace salto de línea.

### El operador +

El operador `+` se usa para concatenar cadenas.

```
System.out.println("24 and 45 concatenated: " + 24 + 45);
```

Resultado:

```
24 and 45 concatenated: 2445
```

Para sumar valores numéricos:

```
System.out.println("24 and 45 added: " + (24 + 45));
```

Resultado:

```
24 and 45 added: 69
```

## Secuencias de escape

Java tiene **secuencias de escape** para representar caracteres especiales dentro de cadenas.

| Secuencia | Significado | |-----------|-----------------| | `\b` | backspace | | `\t` | tabulación | | `\n` | nueva línea | | `\r` | retorno de carro | | `\"` | comilla doble | | `\'` | comilla simple | | `\\` | barra invertida |

Ejemplo:

```
System.out.println("Hola\nMundo");
```

# Variables y tipos de datos

Una **variable** es un nombre para una ubicación en memoria que almacena datos.

Declaración:

```
int total;
double precio;
char letra;
```

Asignación:

```
total = 10;
int numero = 6;
```

Se pueden declarar múltiples variables del mismo tipo:

```
int x, y, z;
```

## Constantes

- Se declaran con la palabra clave `final`.
- Por convención, se escriben en **mayúsculas**.

```
final int MAX_USERS = 100;
```

# Tipos de datos primitivos

Son los tipos de datos más básicos en Java.

Sirven para representar valores simples como números, caracteres y valores lógicos.

No son objetos (aunque Java tiene clases wrapper como Integer, Double, etc.).

Java tiene 8 tipos de datos primitivos, agrupados en 4 categorías principales.

| Categoría | Tipos incluidos | | ---------- | ---------- | | Enteros | `byte`, `short`, `int`, `long` | | Punto flotante | `float`, `double` | | Caracteres | `char` | | Booleanos | `boolean` |

## Enteros

| Tipo | Tamaño | Rango | | ------- | ------- | ---------- | | `byte` | 8 bits | -128 a 127 | | `short` | 16 bits | -32,768 a 32,767 | | `int` | 32 bits | -2,147,483,648 a 2,147,483,647 | | `long` | 64 bits | $-9\times10^{18}$ a $9\times10^{18}$ (aprox.) |

## Decimales

| Tipo | Tamaño | Precisión | | ------- | ------- | ---------- | | `float` | 32 bits | ~7 dígitos decimales | | `double` | 64 bits | ~15 dígitos decimales |

## Caracteres

El tipo `char` representa un solo carácter Unicode (16 bits).
Puede almacenar cualquier símbolo Unicode.

**Ejemplo:**

```
char letra = 'A';
char simbolo = '$';
char unicode = '\u0041'; // A
```

## Booleanos

El tipo `boolean` solo puede tener dos valores:

- `true`
- `false`

**Ejemplo:**

```
boolean activo = true;
boolean terminado = false;
```

### Características

**Eficientes:** Se almacenan directamente en memoria sin la sobrecarga de los objetos.

**Inmutables:** El valor almacenado no puede cambiar de tipo.

**Fuertemente tipados:** No puedes asignar un `double` a un `int` sin casting explícito.

## Operadores

### Aritméticos

| Operador | Descripción | |----------|--------------------| | `+` | Suma | | `-` | Resta | | `*` | Multiplicación | | `/` | División | | `%` | Residuo (módulo) |

### Precedencia de operadores (PEMDASA)

```
int result = 14 + 8 / 2;         // 18
int result2 = 3 * ((18 - 4) / 2); // 21
```

### Incremento y decremento

En Java, los operadores de incremento (`++`) y decremento (`--`) permiten aumentar o disminuir el valor de una variable en una unidad de forma más concisa.

Postfijo vs. Prefijo:

```
count++;   // Postfijo: después de usar el valor actual
++count;   // Prefijo: antes de usar el valor actual
```

Equivalentes a:

```
count = count + 1;
```

**Ejemplo:**

```
int a = 5;
int b = a++;  // b = 5, luego a = 6
int c = ++a;  // a = 7, luego c = 7
```

## Relacionales

Los operadores relacionales comparan dos valores y devuelven un resultado booleano (`true` o `false`).

| Operador | Significado | Ejemplo | |----------|--------------------|--------------------| | `==` | Igual a | `a == b` | | `!=` | Distinto de | `a != b` | | `>` | Mayor que | `a > b` | | `<` | Menor que | `a < b` | | `>=` | Mayor o igual que | `a >= b` | | `<=` | Menor o igual que | `a <= b` |

**Ejemplo:**

```
int x = 10, y = 20;
boolean resultado = x < y; // true
```

### Lógicos

Los operadores lógicos permiten combinar expresiones booleanas.

| Operador | Nombre lógico | Descripción | |----------|---------------|-------------------------------------| | `&&` | AND | Verdadero si ambos operandos son `true` | | `||` | OR | Verdadero si al menos un operando es `true` | | `!` | NOT | Invierte el valor lógico |

**Ejemplo:**

```
boolean a = true;
boolean b = false;

System.out.println(a && b); // false
System.out.println(a || b); // true
System.out.println(!a);     // false
```

## Asignación compuesta

La asignación compuesta permite abreviar operaciones aritméticas combinadas con asignación.

| Operador | Ejemplo | Equivalente | |----------|-------------|----------------------| | `+=` | `x += y;` | `x = x + y;` | | `-=` | `x -= y;` | `x = x - y;` | | `*=` | `x *= y;` | `x = x * y;` | | `/=` | `x /= y;` | `x = x / y;` | | `%=` | `x %= y;` | `x = x % y;` |

**Ventajas:**

- Hace el código más limpio y conciso.
- Reduce errores en cálculos repetitivos.

**Ejemplo en Java:**

```
int total = 10;
total += 5;  // total = 15
total *= 2;  // total = 30
total %= 7;  // total = 2
```

## Conversión de tipos

En Java, las conversiones de tipo permiten transformar un valor de un tipo de dato primitivo a otro.

### Widening (ampliación)

Es una conversión **automática**.

Ocurre cuando un tipo de dato "más pequeño" se asigna a uno "más grande" sin pérdida de datos.

También llamada **type promotion**.

**Ejemplo:**

```
int num = 42;
double resultado = num; // int -> double
```

Aquí el valor `num` se convierte automáticamente a `double`.

### Narrowing (reducción)

Es una conversión **explícita**.

Requiere un **cast** porque puede haber pérdida de datos.

También llamada **type casting**.

**Ejemplo:**

```
double pi = 3.14;
int entero = (int) pi; // 3 (pierde la parte decimal)
```

⚠

¡Cuidado! La conversión puede truncar valores si el rango del tipo destino es menor.

### Resumen

| Conversión | Automática | Necesita cast | Posible pérdida | |------------|------------|---------------|-----------------| | Widening | □ | □ | □ | | Narrowing | □ | □ | □ |

## Scanner

La clase `Scanner` en Java permite leer datos desde distintas fuentes, como la entrada estándar del teclado. Pertenece al paquete `java.util` y debe ser importada.

**Ejemplo:**

```
import java.util.Scanner;

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

        System.out.print("Ingrese su nombre: ");
        String nombre = teclado.nextLine();
        System.out.print("Ingrese su edad: ");
        int edad = teclado.nextInt();
```

```
        System.out.println(nombre + ", " + edad + " años");
    }
}
```

## Notas importantes

`nextLine()` → Lee una línea completa (incluye espacios).

`nextInt()`, `nextDouble()` → Lee valores numéricos (ignora saltos de línea).

`next()` → Lee una sola palabra (hasta un espacio).

Scanner utiliza espacios en blanco como delimitadores predeterminados.

Después de usar métodos como `nextInt()`, puede ser necesario limpiar el buffer con `nextLine()` para evitar errores de lectura.

**Ejemplo:**

```
System.out.print("Ingrese un número entero: ");
int numero = teclado.nextInt();

teclado.nextLine(); // Limpia el salto de línea pendiente

System.out.print("Ingrese su dirección: ");
String direccion = teclado.nextLine();
```