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 `      `.

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.

```
1
2   System.out.println("24 and 45 concatenated: " + 24 + 45);
3
```

Resultado:

```
1
2   24 and 45 concatenated: 2445
3
```

Para sumar valores numéricos:

```
1
2   System.out.println("24 and 45 added: " + (24 + 45));
3
```

Resultado:

```
1
2   24 and 45 added: 69
3
```

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;
```

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;
```

```
3
```

Tipos de datos primitivos

Tipo	Tamaño	Valor mínimo	Valor máximo
byte	8 bits	-128	127
short	16 bits	-32,768	32,767
int	32 bits	-2.1×10^{9}	2.1×10^{9}
long	64 bits	-9×10^{18}	9×10^{18}
float	32 bits	-3.4×10^{38} (7 dec)	$+3.4\times10^{38}$ (7 dec)
double	64 bits	-1.7×10^{308} (15 dec)	$+1.7\times10^{308}$ (15 dec)

Caracteres y Unicode

- El tipo `` representa un solo carácter Unicode.

```
1
2   char letra = 'A';
3
```

- Unicode: Unicode

Booleanos

El tipo `` solo acepta dos valores:

- `true`
- `false`

```
1
```

```
2   boolean activo = true;
3
```

Usado para condiciones y control de flujo.

Operadores aritméticos

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

Precedencia de operadores (PEMDASA)

```
1
2   int result = 14 + 8 / 2; // 18
3
4   int result2 = 3 * ((18 - 4) / 2); // 21
5
```

Operadores de incremento y decremento

```
1
2   count++; // Incrementa en 1 (postfijo)
3
4   ++count; // Incrementa en 1 (prefijo)
5
```

Equivalente a:

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

Asignación compuesta

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;`

Conversiones de tipo (Casting)

Conversión implícita (widening)

```
int num = 10;

double result = num / 3; // int → double
```

Conversión explícita (narrowing)

```
double value = 9.8;

int truncated = (int) value; // 9
```

```
5
```

Entrada de datos con Scanner

```
1
2  import java.util.Scanner;
3
4
5
6  Scanner teclado = new Scanner(System.in);
7
8
9
10 System.out.print("Ingrese su nombre: ");
11
12 String nombre = teclado.nextLine();
13
14
15
16 System.out.println("Hola " + nombre + "!");
17
```

Notas sobre Scanner

- `nextLine()` → lee línea completa
- `nextInt()`, `nextDouble()` → leen números
- Ignora espacios en blanco automáticamente como separadores