UD2. Elementos del lenguaje Java

Módulo: Programación

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

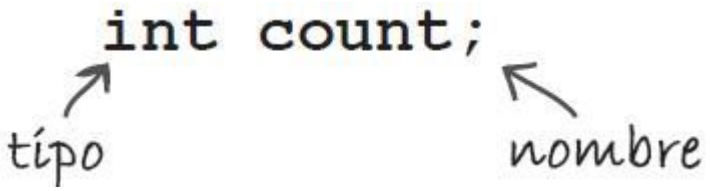
Germán Gascón Grau
g.gascongrau@edu.gva.es

Contenidos

- Variables
- Tipo de datos
- Constantes
- Literales
- Operadores

Variables

- Permiten almacenar datos, resultados y valores intermedios de un programa
- Tienen asociado un **tipo de datos**, que determina:
  - Conjunto de valores que la variable puede guardar
  - Operaciones que se pueden realizar con esa variable
- **Declaración** de variables en Java:



Identificadores

- Los **identificadores** son los nombres que se le dan a las variables, clases, interfaces, atributos y métodos de un programa.
- ¿Qué nombres pueden tener las variables?
  - Tiene que empezar con una letra, subrayado (_) o el carácter $. No puede empezar con un número.
  - Después del primer carácter, sí que se pueden utilizar números. No se pueden utilizar espacios en blanco ni los símbolos de los operadores.
  - No pueden coincidir con ninguna palabra reservada.
  - El % no está permitido, sí el $ y la ç. También acentos y ñ, pero no son recomendables.
  - Recuerda que Java distingue mayúsculas y minúsculas (case sensitive).

Listado de palabras reservadas en Java

boolean	byte	char	double	float	int	long	short	public	private
protected	abstract	final	native	static	strictfp	synchronized	transient	volatile	if
else	do	while	switch	case	default	for	break	continue	assert
class	extends	implements	import	instanceof	interface	new	package	super	this
catch	finally	try	throw	throws	return	void	const	goto	enum

Y los valores true y false;

Convenciones para los identificadores

- Los nombres de las **variables** y los **métodos** deben **empezar por minúscula** y los de las **clases** por **mayúscula**.
- Si el identificador está formado por varias palabras, la primera se escribe en minúsculas (excepto para las clases) y el resto de palabras empiezan por mayúscula (CamelCase)
- Ejemplos:
  - variable: anyoDeCreacion
  - clase: HolaMundo

Tipo de datos

- Tipo de datos **primitivos**:

  boolean, byte, char, double, float, int, long, short

  El nombre de los tipos de datos primitivos empieza por minúscula.

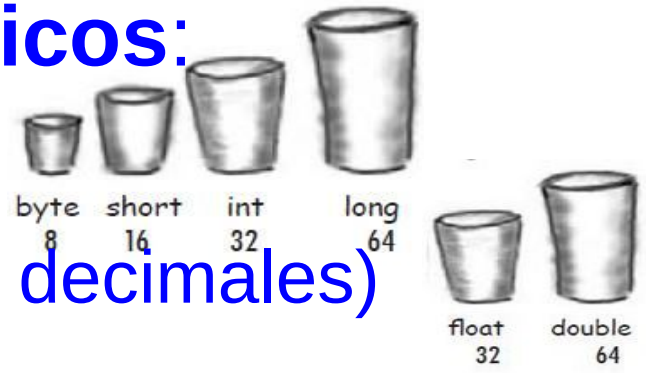
- Tipo de datos **referencia**:

  String, Class (objetos), Interface

  El nombre de los tipos referencia empieza por mayúscula.

Tipo de datos primitivos

- Tipos **numéricos**:
  - Enteros
  - Reales (con decimales)



NOMBRE	TAMAÑO EN BITS	VALOR MÁXIMO
byte	8	127
short	16	32767
int	32	2147483647
long	64	9223372036854775807
float	32	3.4E+38
double	64	1.7E+308

Tipo de datos primitivos

- Tipo **carácter**

NOMBRE	TAMAÑO EN BITS	REPRESENTA
char	16	Representa carácteres (letras, números y símbolos especiales) Java utiliza la codificación UNICODE de 2 bytes. UTF-16

- Tipo **booleano**

NOMBRE	TAMAÑO EN BITS	REPRESENTA
boolean	8	Puede tomar los valores *true* o false.

Ejemplos de tipos primitivos

- Declaración:
-
```
byte b;
char caracter;
int x;
double d;
float f;
boolean activo;
```
- Asignación
-
```
b = 89;
caracter = 'a';
x = 14500;
d = 4.55;
f = 2.31f;
activo = true;
```

Es necesario añadir una **f** al final para indicar que se trata de un float, ya que por defecto Java interpreta los números con decimales como si fueran doubles.

¿Qué pasa si el valor es mayor?

- Si el compilador es capaz de detectarlo en tiempo de compilación, **lanzará un error** avisando del problema. Por ejemplo:

```
int x = 724;
byte b = x; // error de compilación
```

- Si el compilador no es capaz de detectar esta situación, por ejemplo se trata de un dato introducido por el usuario, **se desbordará**. Es decir, se perderán bits y por tanto el valor de la variable carecerá de sentido.

```
byte b = 127;
b++;
```

- El valor de b carecerá de sentido.

Asignación de variables

- Asignando un valor literal después del símbolo =

```
int x = 12;
boolean isGood = true;
```

- Asignando el valor de una variable a otra

```
x = y;
```

- Utilizando una expresión combinación de las dos anteriores

```
x = y + 43;
```

- Asignando el valor en la misma sentencia de declaración

```
char caracter1 = 'A', caracter2 = '\u0041';
float precioPatata = 1.2f, precioChoco = 2.3f;
```

Ejemplos de asignación de variables

`int size = 32;`	declara un entero llamado ***size*** asignándole el valor 32
`char initial = 'j';`	declara un carácter llamado ***initial*** asignándole el valor 'j'
`double d = 456.709;`	declara un doble llamado ***d*** asignándole el valor 456.709
`boolean isCrazy;`	declara un boolean llamado ***isCrazy*** sin asignarle valor
`isCrazy = true;`	asigna el valor true a la variable ***isCrazy***, antes definida
`int y = x + 456;`	declara un entero llamado ***y*** y le asigna el valor de la suma del valor de x más el valor 456

Tipos de variables

- En Java hay **tres tipos** de variables:
  - Variables **locales**: son las que se utilizan en los métodos.
  - Variables **de instancia**: son aquellas que tendrán todos los objetos que se crearán de una clase. Tendrán, en términos generales, valores diferentes para cada objeto.
  - Variables **de clase**: serán iguales a todos los objetos de la clase.

Ámbito de las variables

- El **ámbito** de una variable es la parte del programa en la cual es conocida y se puede utilizar.
- Una **variable local** se declara dentro del cuerpo de un método de una clase y es visible únicamente dentro de ese método.
- Las variables se pueden declarar en cualquier lugar del cuerpo del método, incluso después de instrucciones ejecutables, aunque es una **buena costumbre declararlas al principio**.
- También pueden declararse variables dentro de un bloque entre {…}
  - Únicamente serán “visibles” dentro de ese bloque.
  - Las variables definidas en un bloque tienen que tener nombres diferentes.

Constantes

- El valor de una **constante no puede ser modificado** a lo largo de un programa, a diferencia de las variables.

- Para declarar una constante utilizamos el modificador **final**.

```
final double PI = 3.1415926536;
```

- Tenemos que darle un valor cuando lo declaramos.

Literales enteros

- Un literal entero es la expresión que indica el valor del número.
- Un literal entero puede expresarse:
  - en decimal (base 10) → Ejemplo: 21
  - en octal (base 8) → Ejemplo: 025
  - en hexadecimal (base 16) → Ejemplo: x03A
  - puede añadirse al final la letra L o l para indicar que el entero es considerado como un long → Ejemplo: 234L
  - Pueden añadirse _ (guión bajo) para que actúe como separador de miles. Es muy útil para interpretar más fácilmente las cantidades → Ejemplo: 4_500_000 representa 4500000

Literales reales o decimales

- Un literal real o decimal puede expresarse:
  - parte entera, (.) y parte fraccionaría
    - Ejemplos:
      - 345.678
      - 0.00056
  - notación exponencial o científica
    - Ejemplo: 3.45678e2
- Un literal real **por defecto es de tipo double**. Si queremos que se interprete como float tendremos que añadir el sufijo F o f.

Literales carácter

- Literal carácter: se representan entre comillas simples (‘). Puede ser:
  - Un símbolo (letra)
    - Ejemplos: ‘a’ , ‘B’ , ‘{‘ , ‘ñ’ , ‘á’
  - El código Unicode del carácter en octal o hexadecimal.
    - ’\141’ → código Unicode en octal para ’a’
    - ’\u0061’ → código Unicode en hexadecimal para ’a’

- Secuencias de escape para caracteres especiales

Secuencia	Significado
'\"'	Comilla simple
'\”'	Comilla doble
'\\'	Contrabarra
'\b'	Backspace
'\n’	Salto de línea
'\r'	Retorno de carro
'\t’	Tabulador

Literales String o cadena de caracteres

- No forman parte de los tipos de datos elementales (primitivos) de Java.
- Se representan mediante la clase (tipo referencia) String.
- El literal se expresa entre **comillas dobles** (“). Ejemplos:

```
System.out.println("Primera línea\nSegunda línea del string\n");

System.out.println("Hol\u0061");

String texto = "Hola mundo";
```

Literales booleanos

- Se expresan mediante las palabras reservadas **true** y **false**.
- Ejemplo

```
boolean permiso = false;
```

Conversión de tipo de datos

- Cuando se realiza una asignación:

  identificador = expresión;

- Tanto la variable como la expresión tienen que ser del **mismo tipo o de tipos compatibles**.
- Una expresión puede asignarse a una variable siempre que sea de un tipo de menor tamaño que el tipo de la variable. Por lo tanto, podemos asignar en el siguiente orden:
  - byte → short → int → long → float → double
- Este tipo de conversión se llama **conversión implícita** ya que es realizada automáticamente por el compilador.

Conversión de tipo de datos

- Otras formas de conversión de tipo se pueden realizar explícitamente a través del **casting**.
  - (tipo) expresión
    - Ejemplo:

      ```
      int num = (int) 34.56;
      ```

      El valor de la variable num será 34, ya que el casting trunca el valor ignorando los decimales.
- Este tipo de conversión se llama **conversión explícita** ya que debe ser indicada por el programador explícitamente.
- También utilizando funciones adecuadas de algunos paquetes de Java.

Operadores unarios

- **Signo**
  - Poner un signo + o un signo – antes de una expresión.
  - Ejemplos: +45, -32
- **Incremento (++) y decremento (--)**
  - Aumentar y disminuir en 1 el valor de la variable
  - Pueden ir antes (pre) o detrás (post) de la variable. Ejemplos:

```
int valor, i = 5;
i++;     // ahora i vale 6; es equivalente a i = i+1
i--;     //ahora i vale 5; es equivalente a i = i-1
```

- Veamos la diferencia entre pre y post:

```
valor = i++;  // ahora valor vale 5, i vale 6
                 // equivale a { valor = i; i = i+1; }
valor = ++i; // ahora valor vale 7, i vale 7
                // equivale a {i = i+1; valor = i; }
```

Operadores aritméticos

- Permiten realizar operaciones aritméticas que implican el cálculo de valores numéricos representados por literales, variables, otras expresiones, invocaciones a funciones y propiedades y constantes.
- Sintaxis:

  `expresión1 operador_aritmético expresión2`

- Operadores aritméticos
  - `+` suma
  - `-` resta
  - `*` multiplicación
  - `/` división
  - `%` resto de la división entera
- Ejemplos

```
int x;
x = 52 * 17;
x = 12 / 5;
x = 67 + 34;
x = 32 - 12;
x = 7 % 2;
```

OJO

División entera.
En Java el tipo del operador viene determinado por el tipo del mayor de los operandos. En este caso son los 2 enteros.

Operadores de comparación

- Símbolos que **evalúan expresiones condicionales** y devuelven un boolean.
- Sintaxis:

```
expresión1 operador_de_comparación expresión2
```

- Operadores de comparación

Operador	*true* si...	*false* si...
< (menor que)	expr1 < expr2	expr1 >= expr2
<= (menor o igual que)	expr1 <= expr2	expr1 > expr2
> (mayor que)	expr1 > expr2	expr1 <= expr2
>= (mayor o igual que)	expr1 >= expr2	expr1 < expr2
= = (igual)	expr1 == expr2	expr1 != expr2
!= (distinto)	expr1 != expr2	expr1 == expr2

Operadores de comparación

- Ejemplo:

```
int cantidad = 500;
boolean pedidoGrande;
pedidoGrande = cantidad > 1000;
```

- Ejemplo:

```
boolean testResult;
testResult = ( 45 < 35 );
testResult = ( 45 == 45 );
testResult = (4 != 3 );
testResult = ('a' > 'b');
```

Método equals()

- Para comparar valores de tipos referencia se debe utilizar el método equals().
- Uno de los usos más comunes es para comparar dos cadenas String (tipo referencia)
- Ejemplo:

```
String s = new String("Hola");
if(s.equals("Hola")) {
    System.out.println("Son iguales");
}
```

Método equals() vs operador ==

- El operador == se utiliza para comparar valores de tipos primitivos de datos, es decir, devolverá true cuando los dos valores sean iguales.
- Si aplicamos el operador == a objetos (variables de referencia a objetos), lo que comparará es si las referencias apuntan al mismo objeto, es a decir, apuntan a la misma dirección de memoria.
- Por lo tanto, no tenemos que utilizar el operador == para comparar variables de tipo referencia, ya que estaremos comparando si son el mismo objeto.

Método equals() vs operador ==

```
String cadena = new String("Hola");
String mensaje = new String("Hola");

// Con == serán diferentes porque son objetos distintos
if(cadena == mensaje) {
    System.out.println("Son iguales");
} else {
    System.out.println("Son diferentes");
}
// Con equals serán iguales porque se compara su valor
if(cadena.equals(mensaje)) {
    System.out.println("Son iguales");
} else {
    System.out.println("Son diferentes");
}
```

Operadores lógicos

- Realizan una **evaluación lógica de expresiones** y devuelven un valor boolean.
- Sintaxis:

```
expresión1 operador_lógico expresión2
```

- Operadores:

Operador	Función
&&	Conocido como AND. Combina dos expresiones donde cada una debe ser *true* para que toda la expresión sea *true*.
||	Conocido como OR. Combina dos expresiones donde si una expresión es *true,* toda la expresión es *true*.
!	Conocido como NOT. Devuelve el negativo lógico de la entrada.

Operadores lógicos

- Ejemplo:

```
boolean resultadoAND, resultadoOR;
int edad = 21;
char genero = 'M';
resultadoAND = edad > 18 && genero == 'H';
resultadoOR = edad > 18 || genero == 'H';
```

Operadores lógicos. Tablas de verdad

- Operador **AND**

expr1	expr2	resultado
false	false	false
false	true	false
true	false	false
true	true	true

- Operador **NOT**

expr1	resultado
false	true
true	false

- Operador **OR**

expr1	expr2	resultado
false	false	false
false	true	true
true	false	true
true	true	true

Operadores de asignación

Operador	Ejemplo
=	edad = 34;

+=	edad += 1; // es lo mismo que:
	edad = edad + 1;

-=	edad -= 3; // es lo mismo que:
	edad = edad - 3;

*=	edad * =2; // es lo mismo que:
	edad = edad * 2;

/=	edad /= 2; // es lo mismo que:
	edad = edad / 2;

%=	edad %= 2; // es lo mismo que:
	edad = edad % 2;

Operadores de concatenación

- Permite generar una cadena de caracteres a partir de otras dos
- Sintaxis:

  `expresión1 + expresión2`

- Ejemplo

```
String nombre = "Juan";
String s = "Hola " + nombre;
```

Precedencia de los operadores

1. Paréntesis, de dentro hacia fuera.
2. Unarios
3. Aritméticos
   - A. Multiplicativos: * / %
   - B. Sumativos: + -
4. Comparativos o relacionales
5. Lógicos o booleanos
   - A. NOT
   - B. AND
   - C. OR
6. Asignación

Cuando aparecen operadores de la misma prioridad juntos en una expresión, el compilador evalúa cada operación de izquierda a derecha.

Precedencia de los operadores. Ejemplos

- Ejemplo:

  A = -3 + 5 + 2 * 4 – 4 / 2 * 3 – 5 % 2;

  ¿Cuál es el valor final de esta expresión?

  ¿Y de la siguiente expresión?

  B = -3 + 5 + 2 * 4 – 6 / 4 * 3 – 5 % 2;

Visualización por pantalla. Salida estándar

- La clase **System** tiene un objeto llamado **out** (System.out) que representa la salida estándar, que en la mayoría de SO está asociada por defecto a la pantalla, pero se puede cambiar.
- Mediante los métodos `print()` y `println()` podemos mostrar texto en la salida estándar.
- La diferencia entre `print()` y `println()` es que println() incluye un salto de línea al final de la salida. Ejemplos:

```
System.out.print("Este mensaje se muestra sin salto de línea");
System.out.println("Este mensaje se muestra con un salto de línea");
```

Visualización por pantalla. Salida de error

- Otro objeto de la clase **System** es **err** (System.err) que representa la salida de error, que en la mayoría de SO también está asociada por defecto a la pantalla, pero se puede cambiar.
- Existen los mismos métodos `print()` y `println()` pero en este caso el mensaje se envía a la salida de error..

```
System.err.print("Este mensaje se muestra sin salto de línea");
System.err.println("Este mensaje se muestra con un salto de línea");
```

Lectura de datos. Entrada estándar

- La clase **System** tiene un objeto llamado **in** (System.in) que nos permite leer caracteres desde la entrada estándar, habitualmente asociada al teclado, mediante el método `read()`
- El método `read()` lee un único carácter

```
char c = (char) System.in.read();
```

- Esta forma de leer la información del teclado es poco práctica (imagina que tuviéramos que leer cada línea de texto carácter a carácter...)
- Solución: declarar un objeto de la clase **Scanner** y emplear sus métodos.

Lectura de datos. Entrada estándar. Scanner

- La clase **Scanner** dispone de varios métodos que permiten leer diferentes tipos de datos desde la entrada estándar.

Método	Tipo de datos leído
lector.nextByte()	byte
lector.nextShort()	short
lector.nextInt()	int
lector.nextLong()	long
lector.nextFloat()	float
lector.nextDouble()	double
lector.nextBoolean()	boolean
lector.next()	String
lector.nextLine()	String

Ejemplo de lectura de datos con Scanner

```
// declaramos el objeto lector de la clase Scanner
// conectándolo a la entrada estándar (System.in)
Scanner lector = new Scanner(System.in);
int num;
System.out.println("Escribe un número y pulsa intro");
// leemos el número con el método nextInt()
num = lector.nextInt();
// consumimos el salto de línea que queda en el buffer
lector.nextLine();
System.out.println("Has introducido el número " + num);
// al terminar con la lectura de datos debemos cerrar el
Scanner
lector.close();
```

Tipos enumerados

- Los tipos enumerados se utilizan para representar **datos acotados a un conjunto fijo de valores**.
- Por ejemplo: para representar los días de la semana {Lunes, Martes, Miércoles, Jueves, Viernes}, estaciones del año {Primavera, Verano, Otoño, Invierno}, género {Hombre, Mujer}, etc.
- La principal **ventaja** que nos proporcionan los tipos enumerados, es que **solo admiten valores que estén dentro del posible conjunto de valores**.

Ejemplo tipos enumerados

```
public enum DiaSemana {
    LUNES, MARTES, MIERCOLES, JUEVES, VIERNES,
SABADO, DOMINGO
}

DiaSemana hoy = DiaSemana.LUNES;
System.out.println("Hoy es " + hoy);

// Si intentamos asignar a la variable hoy un valor fuera
// del rango de valores, el compilador avisará del error
hoy = DiaSemana.LUNIS;  // ERROR
```