



UNIVERSIDAD DE ALMERÍA

# Grado en Ingeniería Informática


## Introducción a la Programación

### 2021-2022



# Tema 1. Fundamentos de Programación

## Tipos de datos primitivos en Java

-  ☐ Números enteros
- ☐ Números en coma flotante
- ☐ Caracteres
- ☐ Cadenas de caracteres
- ☐ Booleanos

# Tema 1. Fundamentos de Programación

## Tipos de datos primitivos en Java

El lenguaje Java define 8 **tipos de datos primitivos**:

*Datos de tipo numérico*

- Números enteros **byte, short, int, long**
- Números en coma flotante **float, double**

*Datos de tipo carácter* **char**

*Datos de tipo booleano* **boolean**

# Tema 1. Fundamentos de Programación

## ➤ Números enteros - Tamaños y Rangos

Tipo de dato	Espacio en memoria	Valor mínimo	Valor Máximo
byte	8 bits	-128	127
short	16 bits	-32768	32767
int	32 bits	-2147483648	2147483647
long	64 bits	-9223372036854775808	9223372036854775807

## Operaciones con números enteros

*Desbordamiento* ⇒ Cuando un dato ocupa más espacio del asignado.

Los rangos en Java funcionan de forma **circular**: 0, 1, 2, ... 127, -128, -127, .

Tipo	Operación	Resultado
int	1000000 * 1000000	-727379968
long	1000000 * 1000000	1000000000000

Tipo	Operación	Resultado
byte	127 + 1	-128
short	32767 + 1	-32768
int	2147483647 + 1	-2147483648

# Tema 1. Fundamentos de Programación

## ➤ Números enteros - Error de división por cero

### *División por cero*

Si dividimos un número entero por cero, se produce un error o **excepción** (Exception) en tiempo de ejecución:

Exception in thread "main"

java.lang.ArithmeticException: / by zero

at ...

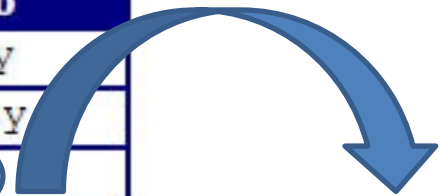
# Tema 1. Fundamentos de Programación

## ➤ Números en coma flotante - Tamaño y Rangos

Tipo de dato	Espacio en memoria	Mínimo (valor absoluto)	Máximo (valor absoluto)	Dígitos significativos
float	32 bits	$1.4 \times 10^{-45}$	$3.4 \times 10^{38}$	6
double	64 bits	$4.9 \times 10^{-324}$	$1.8 \times 10^{308}$	15

## Operaciones con números en coma flotante

Operación	Resultado
1.0 / 0.0	Infinity
-1.0 / 0.0	-Infinity
0.0 / 0.0	NaN



Not a Number

# Tema 1. Fundamentos de Programación

## Operadores aritméticos

Operador	Operación
+	Suma
-	Resta o cambio de signo
*	Multiplicación
/	División
%	Módulo (resto de la división)

- ✓ Si los operandos son enteros, se realizan operaciones enteras.
- ✓ En cuanto uno de los operandos es de tipo **float** o **double**, la operación se realiza en *coma flotante*.
- ✓ No existe un operador de exponenciación: para calcular  $x^a$  hay que utilizar la función **Math.pow(x, a)**

# Tema 1. Fundamentos de Programación

## Operadores aritméticos

### *División (/)*

Operación	Tipo	Resultado
7 / 3	int	2
7 / 3.0f	float	2.3333333333f
5.0 / 2	double	2.5
7.0 / 0.0	double	+Infinity
0.0 / 0.0	double	NaN

### *Módulo (%): Resto de dividir*

Operación	Tipo	Resultado
7 % 3	int	1
4.3 % 2.1	double	~ 0.1



# Tema 1. Fundamentos de Programación

## Expresiones aritméticas

**Expresión:** Construcción que se evalúa para devolver un valor. Se pueden combinar literales y operadores para formar expresiones complejas. Por ejemplo,

$$\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)$$

En Java se escribiría:

$$(3+4*x)/5 - 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)$$

# Tema 1. Fundamentos de Programación

## ➤ Caracteres

Tipo de dato	Espacio en memoria	Codificación
char	16 bits	UNICODE

### Literales de tipo carácter

Valores entre comillas simples “

‘a’ ‘b’ ‘c’ ... ‘1’ ‘2’ ‘3’ ... ‘\*’ ...

Secuencias de escape para representar caracteres especiales

La clase **Character** define funciones (métodos) estáticos para trabajar con caracteres: **isDigit()**, **isLetter()**, **isLowerCase()**, **isUpperCase()**, **toLowerCase()**, **toUpperCase()**

# Tema 1. Fundamentos de Programación



## Caracteres

Secuencia de escape	Descripción
<code>\t</code>	Tabulador (tab)
<code>\n</code>	Avance de línea (new line)
<code>\r</code>	Retorno de carro (carriage return)
<code>\b</code>	Retroceso (backspace)
<code>\'</code>	Comillas simples
<code>\"</code>	Comillas dobles
<code>\\</code>	Barra invertida

# Tema 1. Fundamentos de Programación

## ➤ Cadenas de caracteres

### La clase String

- **String** no es un tipo primitivo, sino una clase predefinida
- Una cadena (String) es una secuencia de caracteres
- Las cadenas de caracteres, en Java, son **inmutables**, es decir, no se pueden modificar los caracteres individuales de la cadena.

### *Literales*

Texto entra comillas dobles “ ”

“Esto es una cadena”

“‘Esto’ también es una cadena”

# Tema 1. Fundamentos de Programación

## ➤ Cadenas de caracteres

### *Concatenación de cadenas de caracteres*

El operador + sirve para concatenar cadenas de caracteres

Operación	Resultado
<code>"Total = " + 3 + 4</code>	<code>Total = 34</code>
<code>"Total = " + (3+4)</code>	<code>Total = 7</code>

# Tema 1. Fundamentos de Programación

## ➤ Booleanos

Representan algo que puede ser verdadero (**true**) o falso (**false**)

Espacio en memoria		Valores
<code>boolean</code>	1 bit	Verdadero o falso

### Expresiones de tipo booleano

- Se construyen a partir de expresiones de tipo numérico con **operadores relacionales** (>, <, ==, !=, <=, >=)
- Se construyen a partir de otras expresiones booleanas con **operadores lógicos o booleanos** (!, &&, ||, ^)

# Tema 1. Fundamentos de Programación

## Operadores relacionales

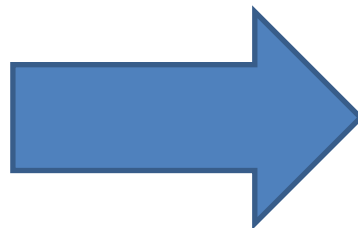
Operador	Significado
==	Igual
!=	Distinto
<	Menor
>	Mayor
<=	Menor o igual
>=	Mayor o igual

## Operadores lógicos/booleanos

Operador	Nombre	Significado
!	NOT	Negación lógica
&&	AND	'y' lógico
	OR	'o' inclusivo
^	XOR	'o' exclusivo

X	!X
true	false
False	true

Tablas de verdad



A	B	A&&B	A  B	A^B
false	false	false	false	false
false	true	false	true	True
true	false	false	true	True
true	true	true	True	False

# Tema 1. Fundamentos de Programación

## ➤ Booleanos

### Ejemplos:

- $3 < 5 \Rightarrow \text{true}$ ;  $3 == 5 \Rightarrow \text{false}$ ;  $3 \leq 5 \Rightarrow \text{true}$ ;  $3 == 3 \Rightarrow \text{true}$ ;  $3 \leq 3 \Rightarrow \text{true}$ ;
- $3 \geq 3 \Rightarrow \text{true}$ ;  $3 \neq 3 \Rightarrow \text{false}$ ;  $3 \neq 4 \Rightarrow \text{true}$ ;  $6 > 8 \Rightarrow \text{false}$ ;  $8 > 6 \Rightarrow \text{true}$
- $3 \leq 5 \ \&\& \ 3 == 3 \Rightarrow \text{true}$
- $3 \leq 5 \ \&\& \ 2 > 10 \Rightarrow \text{false}$
- $1 \neq 2 \ || \ 5 < 3 \Rightarrow \text{true}$
- $1 == 2 \ || \ 5 < 3 \Rightarrow \text{false}$
- $1 \neq 2 \ || \ 3 < 5 \Rightarrow \text{true}$
- $!(1 < 2) \Rightarrow \text{false}$
- $!(1 > 2) \Rightarrow \text{true}$
- $!((3 \leq 5) \ || \ (2 == 3)) \Rightarrow \text{false}$



# Tema 1. Fundamentos de Programación



## Variables

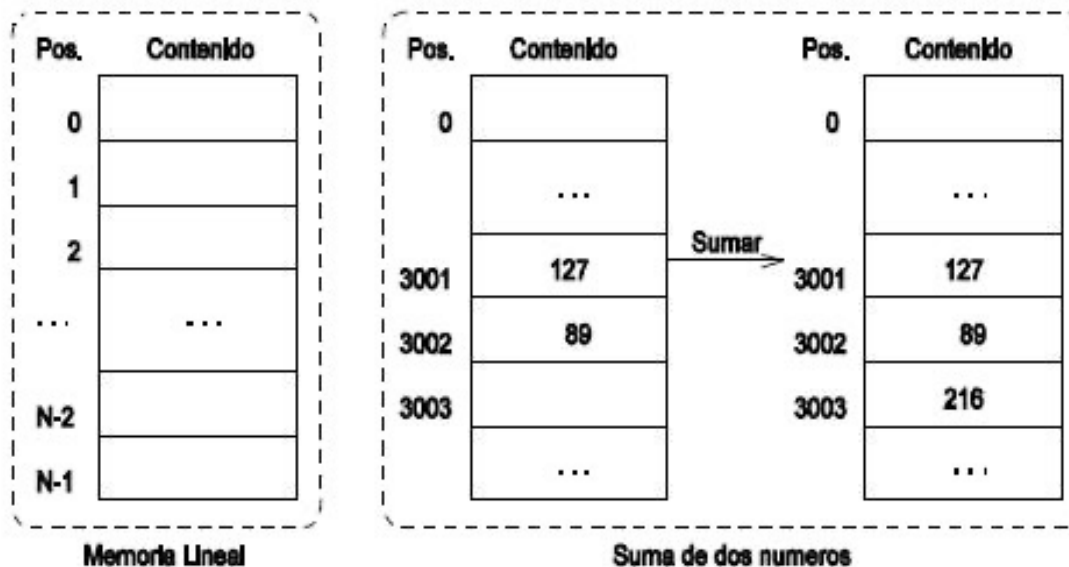


☐ Declaración de variables

☐ Definición de constantes

# Tema 1. Fundamentos de Programación

Una **variable** no es más que un nombre simbólico que identifica una dirección de memoria  $\Rightarrow$  Representación (identificador) de un valor cambiar



“Suma el contenido de la posición 3001 y la 3002 y lo almacenas en la posición 3003”

vs.

total = cantidad1 + cantidad2

“Suma cantidad1 y cantidad2 y lo almacenas en total”

# Tema 1. Fundamentos de Programación

## ➤ Declaración de variables

Para usar una variable debe de estar previamente declarada ⇒ **Identificador**

```
<tipo> identificador;  
<tipo> lista de identificadores;
```

### *Ejemplos*

// Declaración una variable entera x de tipo **int**

```
int x;
```

// Declaración de una variable real r de tipo **double**

```
double r;
```

// Declaración de una variable c de tipo **char**

```
char c;
```

// Múltiples declaraciones en una sola línea

```
int i, j, k;
```

# Tema 1. Fundamentos de Programación

## Identificadores en Java

➤ El primer símbolo del identificador será un carácter alfabético (a, ..., z, A, ..., Z, '\_', '\$') pero no un dígito. Después de ese primer carácter, podremos poner caracteres alfanuméricos (a, ..., z) y (0, 1, ..., 9), signos de dólar '\$' o guiones de subrayado '\_'.

➤ Los identificadores no pueden coincidir con las **palabras reservadas**, que ya tienen significado en Java

➤ Las mayúsculas y las minúsculas se consideran diferentes.

abstract	continue	for	new	switch
boolean	default	goto	null	synchronized
break	do	if	package	this
byte	double	implements	private	threadsafe
byvalue	else	import	protected	throw[s]
case	extends	instanceof	public	transient
catch	false	int	return	true
char	final	interface	short	try
class	finally	long	static	void
const	float	native	super	while
cast	future	generic	inner	
operator	outer	rest	var	

# Tema 1. Fundamentos de Programación

## Identificadores en Java

### Convenciones para la nomenclatura de variables en Java

- Los identificadores deben ser descriptivos: deben hacer referencia al significado de aquello a lo que se refieren.

```
int n1, n2;           // MAL
```

```
int anchura, altura; // BIEN
```

- Los identificadores asociados a las variables se suelen poner en minúsculas.

```
int CoNTaDoR; // MAL
```

```
int contador; // BIEN
```

- Cuando el identificador está formado por varias palabras, la primera palabra va en minúsculas y el resto de las palabras se inician con una letra mayúscula.

# Tema 1. Fundamentos de Programación

## Identificadores en Java

### Convenciones para la nomenclatura de variables en Java

```
int mayorvalor;    // MAL  
int mayor_valor;  // ACEPTABLE  
int mayorValor;   // MEJOR
```

### Inicialización de variables

```
int i = 0;  
float pi = 3.1415927f;  
double x = 1.0, y = 1.0;
```

# Tema 1. Fundamentos de Programación

## ➤ Definición de constantes

```
final <tipo> identificador = valor;
```

- Las constantes se definen igual que cuando se declara una variable y se inicializa su valor, permaneciendo dicho valor **inmutable** el resto del programa.
- Con la palabra reservada **final** se impide la modificación del valor almacenado.
- Si intentamos cambiar el valor a una constante, se produce un error en tiempo de compilación.



```
final double CARGA_ELECTRON = 1.6E-19;  
CARGA_ELECTRON = 1.7E-19;
```



The final local variable CARGA\_ELECTRON cannot be assigned.

## Convenciones

Los identificadores asociados a las constantes se suelen poner en mayúsculas.

```
final double PI = 3.141592;
```

Si el identificador está formado por varias palabras, las distintas palabras se separan con un guión de subrayado '\_'. Por ejemplo, MAXIMA\_CANTIDAD

# Tema 1. Fundamentos de Programación



## Expresiones y sentencias



☐ Construcción de expresiones

☐ Sentencia de asignación



# Tema 1. Fundamentos de Programación

## Expresiones y sentencias

### *Expresión*

Construcción (combinación de tokens) que se evalúa para devolver un valor.

### *Sentencia*

Representación de una acción o una secuencia de acciones.  
En Java, todas las sentencias terminan con un punto y coma [;].

# Expresiones y sentencias

## ➤ Construcción de expresiones

- **Literales y variables** son expresiones primarias:

1.7      // Literal real de tipo `double`. Un literal es la  
          // especificación de un valor concreto de un tipo dado  
`double sum` // Variable

- Los literales se *evalúan* a sí mismos.
- Las variables se *evalúan* a su valor.

- Los **operadores** nos permiten combinar **expresiones** primarias y otras expresiones formadas con operadores:

$1 + 2 + 3 * 1.2 + (4 + 8) / 3.0$

# Expresiones y sentencias

## ➤ Construcción de expresiones

### *Operadores y tipo del resultado*

Operadores	Descripción	Resultado
+ - * / %	Operadores aritméticos	Número*
== != < > <= >=	Operadores relacionales	Booleano
! &&    ^	Operadores booleanos	
~ &   ^ << >> >>>	Operadores a nivel de bits	Entero
+	Concatenación de cadenas	Cadena

# Expresiones y sentencias

## ➤ Sentencia de asignación - Operador de asignación

### Sintaxis:

<variable> = <expresión>;

1. Se evalúa la **expresión** (una serie de operaciones) que aparece a la derecha del operador de asignación (=).
2. El **valor** que se obtiene como **resultado de evaluar la expresión** se almacena en la variable que aparece a la izquierda del operador de asignación (=).

### Restricción:

El tipo del valor que se obtiene como resultado de evaluar la expresión ha de ser *compatible* con el tipo de la variable.

# Expresiones y sentencias

## ➤ Sentencia de asignación

### *Ejemplos*

```
x = x + 1;
```

```
int miVariable = 20;
```

```
otraVariable = miVariable;
```

```
// Declaración con inicialización
```

```
// Sentencia de asignación
```

*Tipo de resultado de la expresión debe ser compatible con el tipo de la variable a la que se asigna*

# Expresiones y sentencias

## Conversión de tipos

En determinadas ocasiones, nos interesa convertir el tipo de un dato en otro tipo para poder operar con él.

- La conversión de un tipo **con menos bits a un tipo con más bits** es *automática* (por ejemplo, de **int** a **long**, de **float** a **double**), ya que el tipo mayor puede almacenar cualquier valor representable con el tipo menor (además de valores que “no caben” en el tipo menor).
- La conversión de un tipo **con más bits a un tipo con menos bits** hay que realizarla de forma explícita con “**castings**”. Como se pueden perder datos en la conversión, el compilador nos obliga a ser conscientes de que se está realizando una conversión en la que se va a **perder información** (truncado)

# Expresiones y sentencias

## Conversión de tipos

### Ejemplos

```
int i;
```

```
byte b;
```

```
i = 13;
```

```
// No se realiza conversión alguna
```

```
b = 13;
```

```
// Se permite porque 13 está dentro
```

```
// del rango permitido de valores del tipo byte
```

```
b = i;
```

```
// No permitido (incluso aunque
```

```
// 13 podría almacenarse en un byte)
```

```
b = (byte)i;
```

```
// Fuerza la conversión
```

```
i = (int) 14.456;
```

```
// Almacena 14 en la variable i
```

```
i = (int) 14.656;
```

```
// Sigue almacenando 14
```

El compilador de Java comprueba siempre los tipos de las expresiones y nos avisa de posibles errores:  
*“Incompatible types” (N)* y *“Possible loss of precision” (C)*

# Expresiones y sentencias

## Evaluación de expresiones

- La **precedencia de los operadores determina el orden de** evaluación de una expresión (el orden en que se realizan las operaciones que forman la expresión):

$3*4+2$  es equivalente a  $(3*4)+2$

porque el operador  $*$  es de mayor precedencia que el operador  $+$

Prioridad de operaciones o precedencia	
( )	(paréntesis)
++, --, !	(más prioridad <i>postfijos</i> (a++, a--) que <i>unarios prefijos</i> (++a, --a, +a, -a, !a))
*, /, %	(aritméticos)
+, -	(aritméticos)
<, <=, >, >=	(relacionales)
==, !=	(comparación)
&&	(AND lógico)
	(OR lógico)
? :	(operador condicional o ternario)
=, +=, -=, *=, /=, %=, ^=	(asignación y operadores opera_y_asigna)



## Operadores incremento y decremento. Operadores unarios

Java ofrece una notación abreviada para una operación de programación muy común, la de incrementar o decrementar el valor de una variable en 1.

Los **operadores unarios** de incremento y decremento son:

Operador	Significado
<code>x++</code>	Primero evalúa <code>x</code> , y después incrementa <code>x</code> ( <b>post-incremento</b> )
<code>++x</code>	Primero incrementa <code>x</code> , y luego evalúa ( <b>pre-incremento</b> )
<code>x--</code>	Primero evalúa <code>x</code> , y después decrementa <code>x</code> ( <b>post-decremento</b> )
<code>--x</code>	Primero decrementa <code>x</code> , y después evalúa ( <b>pre-decremento</b> )

### Ejemplos

Suponiendo `i = 3` y `c = 10`, los resultados de las siguientes operaciones son:

- A. `x = i++ = 3++`  $\Rightarrow$  incrementa después de la asignación, así **`x = 3`**
- B. `x = ++i = ++3`  $\Rightarrow$  incrementa antes de la asignación, así **`x = 4`**
- C. `x = c++ = 10++`  $\Rightarrow$  incrementa después de la asignación, así **`x = 10`**
- D. `x = --c + 2 = --10 + 2 = 9 + 2`, así **`x = 11`**
- E. `x = i-- + ++c = 3-- + ++10 = 3 + 11`, así **`x = 14`**

## Operadores combinados de asignación y operación

➤ A estos operadores se les denomina **opera y asigna**, que realizan la operación indicada, tomando como operandos el valor de la variable a la izquierda y el valor a la derecha del = (igual). El resultado se asigna a la misma variable utilizada como primer operando.

Operador	Nombre	Ejemplo	Equivale
+=	Asignación adición	<code>i += 8</code>	<code>i = i + 8</code>
-=	Asignación sustracción	<code>i -= 8</code>	<code>i = i - 8</code>
*=	Asignación multiplicación	<code>i *= 8</code>	<code>i = i * 8</code>
/=	Asignación división	<code>i /= 8</code>	<code>i = i / 8</code>
%=	Asignación resto	<code>i %= 8</code>	<code>i = i % 8</code>

# Tema 1. Fundamentos de Programación



## Programas



- ☐ Estructura de un programa simple
- ☐ Estilo y documentación del código
- ☐ Errores de programación

# Programas



## Estructura de un programa simple

- Entrada de datos
- Procesamiento de los datos
- Salida de resultados

El punto de entrada de un programa en Java es la función **main**:

```
public static void main (String[] args)
```

```
{
```

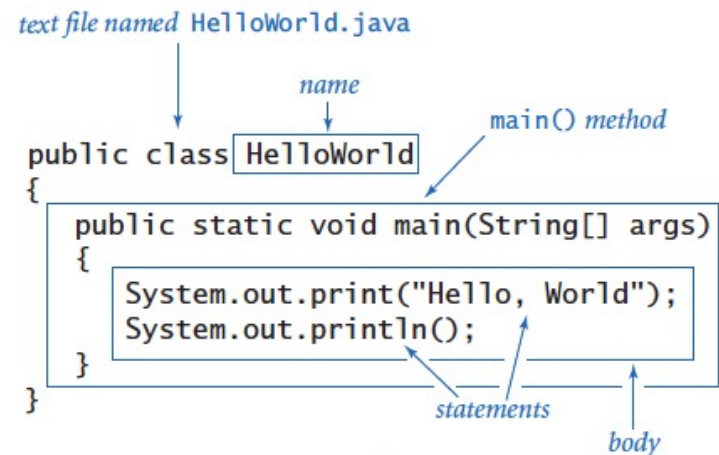
*Declaraciones y sentencias escritas en Java*

```
}
```

# Programas

Java es un lenguaje de programación orientada a objetos y todo debe estar dentro de una **clase**, incluida la función **main**, tal como se muestra a continuación en nuestro primer programa

```
package org.ip.sesion01;
public class HolaMundo {
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("HOLA, MUNDO");
    }
}
```



Se guardará en un archivo de texto con el mismo nombre que la clase y con la extensión `.java`. Es decir: **HolaMundo.java**

# Programas

- ❑ La máquina virtual Java (JVM, Java Virtual Machine) ejecuta el programa invocando a la función **main**.
- ❑ Las llaves **{}** delimitan bloques en Java (conjuntos de elementos de un programa, delimitadores de ámbito).

## Comentarios

Los comentarios sirven para incluir aclaraciones en el código.

Java permite tres tipos de comentarios:

`//` Comentarios de una línea

`/*` Comentarios de varias líneas `*/`

`/**` Comentarios de documentación `*/`

Son comentarios al estilo javadoc

# Programas

## *Errores comunes de programación*

### **Errores sintácticos**

Errores detectados por el compilador en tiempo de compilación.

### **Errores semánticos**

Sólo se detectan en tiempo de ejecución: Causan que el programa finalice inesperadamente su ejecución (por ejemplo, división por cero) o que el programa proporcione resultados incorrectos.

### **Errores lógicos**

Los causados por un mal diseño del algoritmo.

# Programas

Nuestros programas se construirán dentro de un paquete o **package**.

Un paquete es un “contenedor” de un grupo de clases relacionadas entre sí.

```
package org.ip.sesionNN;
```

```
public class IdentificadorClase {
```

```
    /**
```

```
    * @param args
```

```
    */
```

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

```
        declaración de variables y/o constantes
```

```
        sentencias (asignación, salida)
```

```
    }
```

```
}
```

Fichero: Armonico.java

```
package org.ip.sesion03;
```

```
import java.util.Scanner;
```

```
public class Armonico {
```

Si desde una claseA necesitamos acceder a otra claseB que se encuentra en otro paquete tenemos que importar (**import**) el paquete que contiene la claseB.

Por ejemplo, la clase de Java **Scanner** que está dentro del paquete **java.util**



# Programas

## Ejemplo 1

```
package org.ip.sesion01;
```

```
/*  
 * Muestra lo que ocurre cuando divides por cero con enteros y reales  
 *  
 * 17.0 / 0.0 = Infinity  
 * 17.0 % 0.0 = NaN  
 * Exception in thread "main" java.lang.ArithmeticException: / by zero  
 */  
*****/
```

```
public class DivisionPorCero {  
    public static void main(String[] args) {  
        System.out.println("17.0 / 0.0 = " + (17.0 / 0.0));    // Infinity  
        System.out.println("17.0 % 0.0 = " + (17.0 % 0.0));    // NaN => not a number  
        System.out.println("17 / 0 = " + (17 / 0));            // ERROR  
        System.out.println("17 % 0 = " + (17 % 0));            // ERROR  
    }  
}
```

# Programas

## Ejemplo 2

```
package org.ip.sesion01;
```

```
/* **** */
```

- \* Muestra un entero pseudo-aleatorio entre 0 y N - 1.
- \* Ilustra una conversión explícita de tipos (cast) de double a int.
- \*

```
**** */
```

```
public class EnteroAleatorio {
```

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

```
        int N = 10;
```

```
        // genera un real pseudo-aleatorio entre 0.0 y 1.0
```

```
        double r = Math.random();
```

```
        // lo convertiremos a un entero pseudo-aleatorio entre 0 y N-1
```

```
        int n = (int) (r * N);
```

```
        System.out.println("Su entero aleatorio es: " + n);
```

```
    }
```

```
}
```

## Salida por consola

Java utiliza **System.out** para referirse al dispositivo *estándar de salida* y **System.in** para referirse al dispositivo *estándar de entrada*. Por defecto el dispositivo de salida estándar es el *monitor* y el de entrada es el *teclado*.

La clase **System** la podemos usar sin importarla porque está en el paquete **java.lang** que es el único que se importa automáticamente y por tanto no es necesario indicarlo.

Para realizar una salida por consola, utilizaremos algunos de los siguientes métodos:

- **print**
- **println**
- **printf**

El método **print** es idéntico a **println** excepto que este último mueve el cursor a la siguiente línea después de mostrar el String y **print** no avanza el cursor a la línea siguiente.

Para combinar mensajes de texto y el valor de variables utilizamos: +

## Salida por consola formateada

En ocasiones estamos interesados en mostrar un número real con solo dos decimales o un número entero justificado a la derecha o a la izquierda etc. Para ello utilizamos la función **printf** cuya sintaxis es:

```
System.out.printf(especificadores de formato, item1, item2, ...);
```

- Los *especificadores de formato* se dan entre comillas dobles (") e indican cómo se quiere mostrar la salida (formato de salida).
- Consisten en un signo de porcentaje (%) seguido de un carácter cuyo significado se muestra en la siguiente tabla.

## Tabla de especificadores más frecuentes

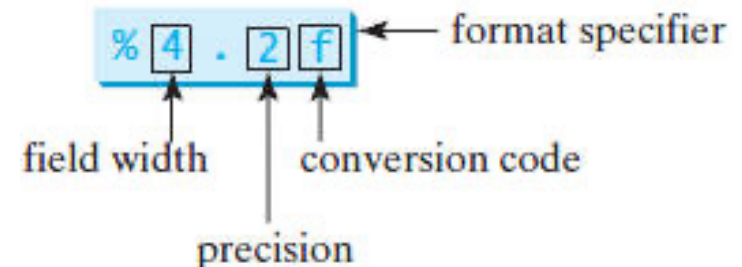
Especificador	Salida	Ejemplo
%b	Un valor booleano	true o false
%c	Un carácter	'a'
%d	Un entero	200
%f	Un real	45.460000
%s	Una cadena	"Esto es Java"

Los *items* de la sintaxis de **printf** pueden ser un valor numérico, un carácter, un booleano o una cadena de caracteres.

Ejemplo: `double x = 2.0 / 3;`

`System.out.printf("x es %4.2f ", x);`

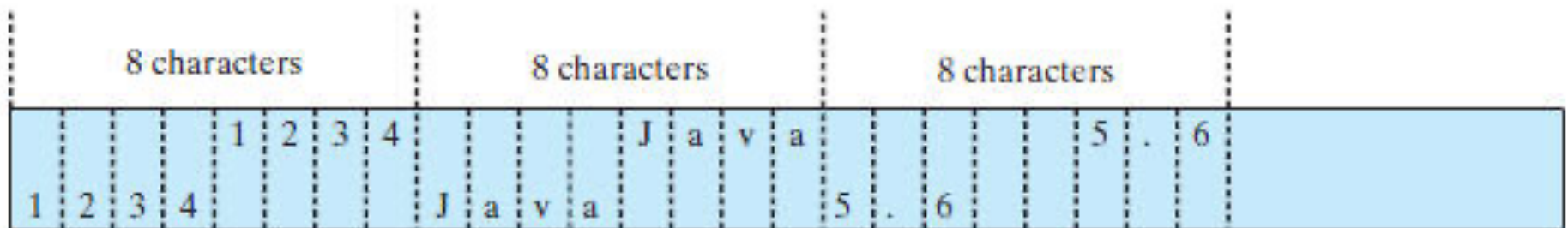
Mostraría: **x es 0.67**



Por defecto, la salida está justificada a la *derecha*. Podemos poner el signo (-) para especificar que un *item* se quiere justificar a la *izquierda*. Por ejemplo, las siguientes sentencias:

```
System.out.printf("%8d%8s%8.1f\n", 1234, "Java", 5.6);
System.out.printf("%-8d%-8s%-8.1f \n", 1234, "Java", 5.6);
```

Mostrarían



# Tema 1. Fundamentos de Programación



## Estructuras de control



- ☐ Programación estructurada
- ☐ Estructuras selectivas/condicionales
- ☐ Estructuras repetitivas/iterativas

# Tema 1. Fundamentos de Programación

## ➤ Programación estructurada

### ☐ Crisis del software (Dijkstra)

☐ Las estructuras de control controlan la ejecución de las instrucciones de un programa (especifican el orden en el que se realizan las acciones)

### ☐ IDEA PRINCIPAL:

Las estructuras de control de un programa sólo deben tener **un punto de entrada y un punto de salida.**



### Teorema de Böhm y Jacopini (1966):

Cualquier programa de ordenador puede diseñarse e implementarse utilizando únicamente las tres construcciones estructuradas (secuencia, selección e iteración; esto es, sin sentencias goto).

En programación estructurada sólo se emplean tres construcciones:

#### ☐ **Secuencia**

Conjunto de sentencias que se ejecutan en orden

Ejemplos: Sentencias de asignación (=).

#### ☐ **Selección**

Elige qué sentencias se ejecutan en función de una condición.

Ejemplos: Estructuras de control condicional **if ... else** y **switch**

#### ☐ **Iteración**

Las estructuras de control repetitivas repiten conjuntos de instrucciones.

Ejemplos: Bucles **while**, **do...while** y **for**.

**¡MUCHAS GRACIAS!**



UNIVERSIDAD DE ALMERÍA

