

CURSO DE TÉCNICO EN DESARROLLO NATIVO SOBRE PLATAFORMAS ANDROID

EOI – Escuela de Organización Industrial

TEMA 2. JAVA

LENGUAJES EN ANDROID

- JAVA
 - Lenguaje de programación principal
- XML
 - Metalenguaje para definir interfaz de usuario
- Kotlin
 - Nuevo lenguaje (2011) basado en Java

JAVA

- JAVA es un lenguaje de programación:
 - Simple
 - Orientado a objetos
 - Independiente de la plataforma sobre la que se ejecuta
 - Recolecta de forma automática la *basura* (memoria ocupada que ya no se utilizará más)

JAVA

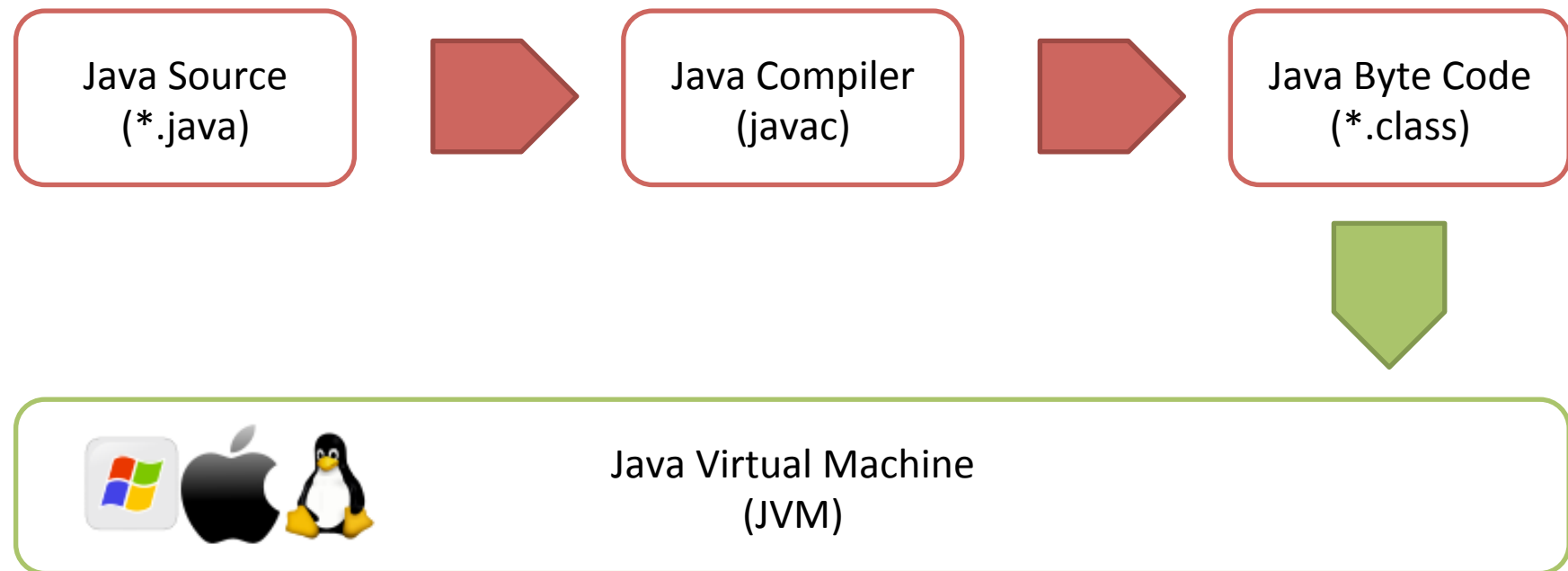
- Simple
 - Lenguaje de alto nivel nacido para responder la necesidad de crear software complejo a medida que el hardware se desarrollaba
 - Más calidad y mejor organización en proyectos de gran tamaño

JAVA

- Programación Orientada a Objetos
 - Diseño de software centrado en “cosas” (objetos) y cómo interaccionan entre ellos
 - Abstracción y encapsulación
 - Visibilidad

JAVA

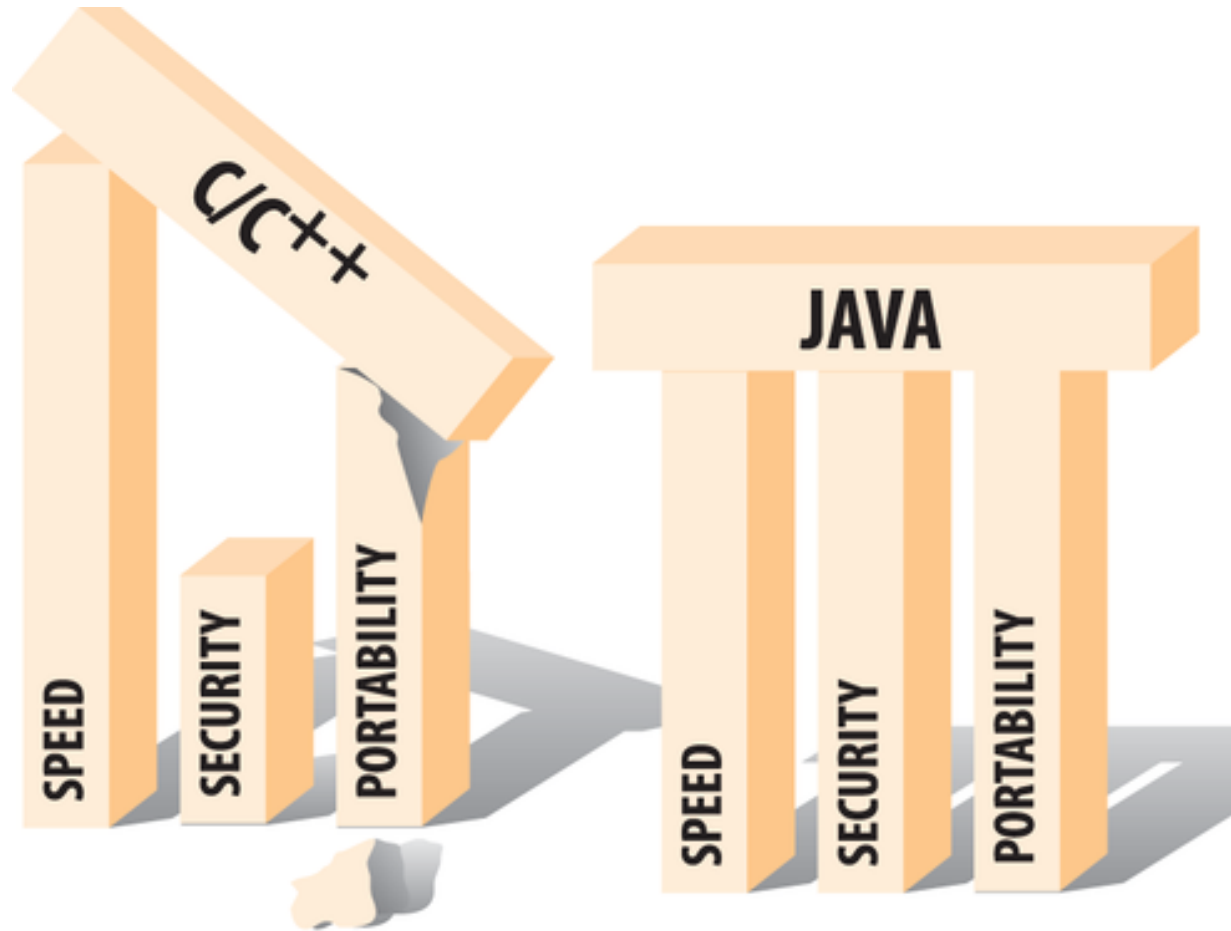
- Independencia de plataforma
 - Java no es un lenguaje compilado, es **interpretado**



JAVA

- Recolección automática de *basura*
 - En lenguajes anteriores, el programador debía **reservar** y **liberar** memoria. Si la memoria no se liberaba, se producía un *memory leak*.
 - Java se encarga de liberar dicha memoria cuando ya no se utiliza más.

JAVA



SINTAXIS

- Todas las instrucciones terminan con ;
- Las **clases** deben llamarse **exactamente igual que el fichero que la contiene**

```
package com.teaching.android;

/**
 * Empty Java App for teaching purposes.
 */
public class JavaApp {

    public static void main(String[] args) {
        // Display the string.
        System.out.println("Hello World!");
    }
}
```

Fichero *JavaApp.java*

SINTAXIS

- **Identificadores**

- Definen nombres de variables, métodos, clases e interfaces
- Secuencia ilimitada de caracteres alfabéticos o dígitos que comienzan con un carácter alfabético
- El carácter “_” es aceptado

numero

1valor

nombre de persona

valor1

Nombre_persona

nombrePersona

COMENTARIOS

```
// Esto es un comentario
```

```
/* Esto también es un comentario */
```

```
/*  
    Esto es otro comentario,  
    pero de varias líneas  
*/
```

KEYWORDS

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

VARIABLES

- Java es un lenguaje **fuertemente tipado**
 - Todas las variables tienen un tipo de dato en tiempo de compilación
 - Java nos avisará de una asignación errónea de tipos

```
int a = "esto es un entero";
```



VARIABLES

- Deben tener identificadores válidos y únicos
- Existen cuatro tipos de variables
 - Variables de instancia
 - Variables de clase
 - Parámetros o argumentos
 - Variables locales

VARIABLES

- Tipos de variables - ejemplo

```
public class JavaApp {  
    public int id = 0;  
    public static boolean isClassUsed;  
  
    public void processData(String parameter) {  
        Object currentValue = null;  
    }  
}
```

TIPOS PRIMITIVOS

Category	Types	Size (bits)	Minimum Value	Maximum Value	Precision	Example
Integer	<code>byte</code>	8	-128	127	From +127 to -128	<code>byte b = 65;</code>
	<code>char</code>	16	0	$2^{16}-1$	All Unicode characters	<code>char c = 'A';</code> <code>char c = 65;</code>
	<code>short</code>	16	-2^{15}	$2^{15}-1$	From +32,767 to -32,768	<code>short s = 65;</code>
	<code>int</code>	32	-2^{31}	$2^{31}-1$	From +2,147,483,647 to -2,147,483,648	<code>int i = 65;</code>
	<code>long</code>	64	-2^{63}	$2^{63}-1$	From +9,223,372,036,854,775,807 to -9,223,372,036,854,775,808	<code>long l = 65L;</code>
Floating-point	<code>float</code>	32	2^{-149}	$(2^{-2^{23}}) \cdot 2^{127}$	From 3.402,823,5 E+38 to 1.4 E-45	<code>float f = 65f;</code>
	<code>double</code>	64	2^{-1074}	$(2^{-2^{52}}) \cdot 2^{1023}$	From 1.797,693,134,862,315,7 E+308 to 4.9 E-324	<code>double d = 65.55;</code>
Other	<code>boolean</code>	1	--	--	false, true	<code>boolean b = true;</code>
	<code>void</code>	--	--	--	--	--

TIPOS PRIMITIVOS

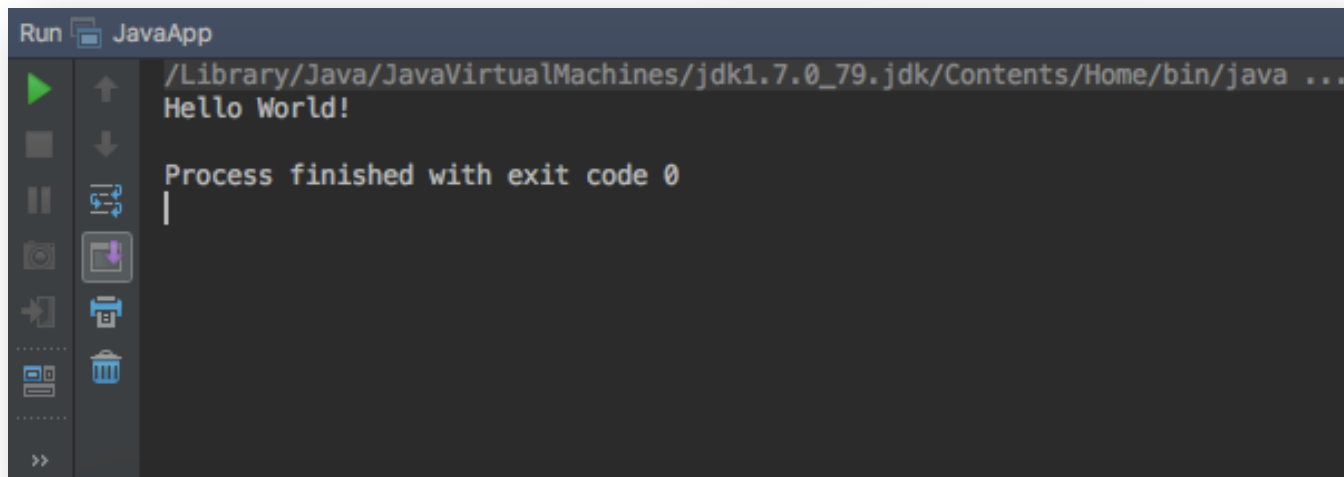
- Conversión entre tipos
 - Pérdida de precisión

	from <code>byte</code>	from <code>char</code>	from <code>short</code>	from <code>int</code>	from <code>long</code>	from <code>float</code>	from <code>double</code>	from <code>boolean</code>
to <code>byte</code>	-	(byte)	(byte)	(byte)	(byte)	(byte)	(byte)	N/A
to <code>char</code>		-	(char)	(char)	(char)	(char)	(char)	N/A
to <code>short</code>		(short)	-	(short)	(short)	(short)	(short)	N/A
to <code>int</code>				-	(int)	(int)	(int)	N/A
to <code>long</code>					-	(long)	(long)	N/A
to <code>float</code>						-	(float)	N/A
to <code>double</code>							-	N/A
to <code>boolean</code>	N/A	N/A	N/A	N/A	N/A	N/A	N/A	-

HELLO WORLD

1. **Fork** repositorio 'empty-java-project'
 - <https://github.com/AndroidTeaching/empty-java-project>
2. Abrir en IntelliJ IDEA
3. Ejecutar aplicación

✖ No se puede mostrar la imagen. Puede que su equipo no tenga suficiente memoria para abrir la imagen o que ésta esté dañada. Reinicie el equipo y, a continuación, abra el archivo de nuevo. Si sigue apareciendo la x roja, puede que tenga que borrar la imagen e insertarla de nuevo.



The screenshot shows the 'Run' console in IntelliJ IDEA for a project named 'JavaApp'. The console output displays the command to run Java, the output 'Hello World!', and the message 'Process finished with exit code 0'. The left sidebar contains various icons for running and debugging the application.

```
Run JavaApp  
/Library/Java/JavaVirtualMachines/jdk1.7.0_79.jdk/Contents/Home/bin/java ...  
Hello World!  
  
Process finished with exit code 0
```

PRÁCTICA – TIPOS PRIMITIVOS

1. Crear 8 tipos de variables primitivas en **main**
2. Imprimir por pantalla cada variable

PRÁCTICA – TIPOS PRIMITIVOS

```
public static void main(String[] args){  
    byte b = 65;    // 8 bits  
    char c = 'A';   // 16 bits  
    short s = 65;   // 16 bits  
    int i = 65;     // 32 bits  
    long l = 65L;   // 64 bits  
    float f = 65f;  // 32 bits  
    double d = 65.55; // 64 bits  
    boolean bol = true; // 1 bit  
  
    System.out.println(b);  
    System.out.println(c);  
    System.out.println(s);  
    System.out.println(i);  
    System.out.println(l);  
    System.out.println(f);  
    System.out.println(d);  
    System.out.println(bol);  
}
```

PRÁCTICA – TIPOS PRIMITIVOS

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java ...  
65  
A  
65  
65  
65  
65.0  
65.55  
true  
  
Process finished with exit code 0
```

EXPRESIONES ARITMÉTICAS

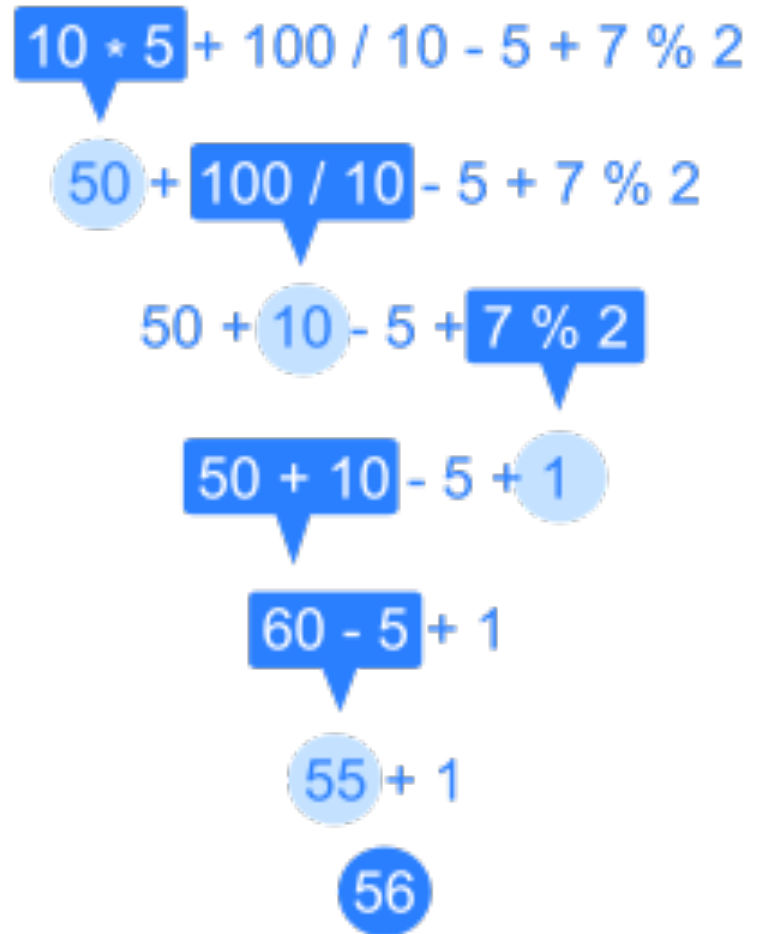
+	suma
-	resta
*	multiplicación
/	división entera
%	módulo
++	incremento
--	decremento

```
int a = 10, b = 5;  
System.out.println(a + b);  
System.out.println(a - b);  
System.out.println(a * b);  
System.out.println(a / b);  
System.out.println(a % b);  
System.out.println(++a);  
System.out.println(--a);  
System.out.println(a++);  
System.out.println(a--);  
System.out.println(a);
```

EXPRESIONES ARITMÉTICAS

- Orden de ejecución implícito

$$10 * 5 + 100 / 10 - 5 + 7 \% 2$$



EXPRESIONES ARITMÉTICAS

- Se puede alterar con paréntesis

$$10 * (5 + 100) / 10 - 5 + 7 \% 2$$

1. $5 + 100 = 105$
2. $10 * 105 = 1050$
3. $1050 / 10 = 105$
4. $7 \% 2 = 1$
5. $105 - 5 + 1 = 101$

PRÁCTICA - EXPRESIONES ARITMÉTICAS

- Calcular a mano el resultado y después comprobar en Java de las siguientes expresiones:

1. $2 + 5 - 3 * 2$

2. $2 * 4 * 2 / 2$

3. $1 / 2$

4. $5 * (1 / 2)$

5. $5f * 1 / 2$

EXPRESIONES BOOLEANAS

- Sirven para evaluar a **verdadero** o **falso** cualquier expresión

>	mayor
<	menor
>=	mayor o igual
<=	menor o igual
==	igual
!=	distinto

```
int a = 5, b = 3;  
System.out.println(a > b);
```

OPERADORES BOOLEANOS

- Sirven para evaluar a **verdadero** o **falso** según el álgebra booleana

! NOT

&& AND

|| OR (booleano inclusivo)

^ XOR (booleano exclusivo)

OPERADORES BOOLEANOS

a	b	!a	a && b	a b	a ^ b
true	true	false	true	true	false
true	false	false	false	true	true
false	true	true	false	true	true
false	false	true	false	false	false

CONDICIONALES

- if/else

```
int a = 7;
if (a > 0) {
    System.out.println("a is positive");
} else if (a == 0) {
    System.out.println("a is zero");
} else {
    System.out.println("a is negative");
}
```

CONDICIONALES

- if/else condensado

```
int a = 7;  
String result = a > 0 ? "a is positive"  
                    : "a is zero or negative";  
System.out.println(result);
```

CONDICIONALES

- switch – cuando hay **muchos** if/else anidados

```
int a = 2;
switch (a) {
    case 1:
        System.out.println("a is 1");
        break;
    case 2:
        System.out.println("a is 2");
        break;
    case 3:
        System.out.println("a is 3");
        break;
    default:
        // When no 'case' matches the value
        System.out.println("a is unknown");
        break;
}
```

CONDICIONALES

- switch – ‘case’ sin ‘break’

```
int a = 2;
switch (a) {
    case 1:
        System.out.println("a is 1");
        break;
    case 2:
    case 3:
        System.out.println("a is bigger than 1");
        break;
    default:
        // When no 'case' matches the value
        System.out.println("a is unknown");
        break;
}
```


PRÁCTICA – CONDICIONALES

- Crear un programa que determine si el número introducido es par o impar
 - PISTA: Un número es par si **al dividirlo por 2 el resto es 0.**
 - PISTA: Leer un número introducido por el teclado

```
// Read the input value
Scanner keyboard = new Scanner(System.in);
int valor = keyboard.nextInt();
```

PRÁCTICA – CONDICIONALES 2

- Crear un programa para que el usuario adivine un número al azar entre 1 y 10 en 3 intentos.
 - Si el número a adivinar es mayor que el introducido, el programa debe imprimir “**Mayor!**”, si es menor, debe imprimir “**Menor!**”, y si es el correcto, debe imprimir “**Correcto!**”
 - PISTA: Crear un número aleatorio entre 1 y 10

```
// Create a random number between 1 and 10
Random generator = new Random();
int random = generator.nextInt(10) + 1;
```

BUCLES

- Sirven para ejecutar **tareas repetitivas** con el mínimo esfuerzo. Por ejemplo:
 - Contar hasta 1.000
 - Recorrer una colección de objetos
 - Pedir al usuario un número hasta que acierte
- Aportan
 - Flexibilidad, escalabilidad y mantenibilidad

BUCLES

- **while**
 - El cuerpo del bucle se repite mientras la condición se evalúe como **true**

```
while (<condicion>) {  
    System.out.println("Dentro del bucle");  
}
```

BUCLES

- while

```
int valor = 0;

while (valor * valor < 200) {
    valor = valor + 1;
}
```

BUCLES

- do...while
 - Igual que *while*, solo que la condición se ejecuta **después de la primera ejecución del cuerpo del bucle**

```
do {  
    System.out.println("Dentro del bucle");  
} while (<condicion>;
```

BUCLES

- do...while

```
int valor = 0;  
  
do {  
    valor = valor + 1;  
} while (valor * valor < 200);
```

BUCLES

- **for**
 - *while* especializado para iterar fácilmente sobre una secuencia de números

```
for (<variable>; <condition>; <iteration>) {  
    System.out.println("Dentro del bucle");  
}
```


BUCLES

- for

```
for (int i = 0; i < 5; i++) {  
    System.out.println("Número " + i);  
}
```

```
for (int i = 5; i > 0; i--) {  
    System.out.println("Número " + i);  
}
```

```
for (int i = 1, j = 10; i <= 10; i++, j--) {  
    System.out.print(i);  
    System.out.println(j);  
}
```

BUCLES

- for - anidados

```
int[][] matriz = {  
    { 1, 2 },  
    { 3, 4 },  
    { 5, 6 }  
};  
for (int i = 0; i < matriz.length; i++) {  
    for (int j = 0; j < matriz[i].length; j++) {  
        System.out.println("Item: " + matriz[i][j]);  
    }  
}
```

BUCLES

- for-each
 - *for* mejorado que recorre una colección de elementos y los asigna a una variable en cada iteración

```
for (Object item : items_list ) {  
    System.out.println(item);  
}
```

BUCLES

- for-each

```
int[] primos = { 2, 3, 5, 7, 11 };  
for (int primo : primos) {  
    System.out.println(primo);  
}
```

PRÁCTICA – BUCLES

- Modificar el juego **Guess The Number** añadiendo un bucle
 - Eliminar el código *duplicado* cada vez que se comprueba si el número es el correcto
 - Ofrecer al usuario 3 intentos. Al tercer error, salir del bucle.

STRINGS

- Es una clase predefinida en Java
 - Representa una cadena de caracteres
 - Es **immutable**

```
String str = "This is a string";
```

Docs: <https://docs.oracle.com/javase/7/docs/api/java/lang/String.html>

STRINGS

- Operaciones

trim()

substring()

replace()

contains()

...

equalsIgnoreCase()

isEmpty()

startsWith()

STRINGS

- Concatenación
 - Se pueden concatenar con el operador +

```
String str1 = "Hola, me llamo ";  
String str2 = "Messi";  
System.out.println(str1 + str2);
```


STRINGS

- Concatenación
 - Concatenar otros tipos de datos

```
String str1 = "Somos ";  
int alumnos = 18;  
String str2 = " alumnos en clase";  
System.out.println(str1 + alumnos +  
str2);
```

STRINGS

- Comparación
 - Nunca comparar strings con ==
 - Utilizar siempre `.equals()`

```
String greeting = "Hello World!";  
if (greeting == "Hello World!") {  
    System.out.println("Match found.");  
}
```



STRINGS

- Comparación
 - Nunca comparar strings con ==
 - Utilizar siempre `.equals()`

```
String greeting = "Hello World!";  
if (greeting.equals("Hello World!")) {  
    System.out.println("Match found.");  
}
```

CLASES

- Una clase es una **plantilla** que se utiliza para crear **objetos**.
- Una clase está compuesto de:
 - **Atributos**: describen el estado del objeto
 - **Métodos**: describen las acciones asociadas al objeto

CLASES

```
public class Persona {  
    String nombre;  
    int edad;  
  
    public Persona(String nombre, int edad) {  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
  
    public int getEdad() {  
        return edad;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
}
```

Fichero ***Persona.java***

CLASES

- Constructor
 - Se llama automáticamente una vez el objeto se ha instanciado en memoria
 - Inicializa los atributos del objeto
 - Si no existe constructor, el compilador genera uno vacío por defecto

```
public Persona(String nombre, int edad)
{
    this.nombre = nombre;
    this.edad = edad;
}
```

OBJETOS

- Es una **instancia** concreta de una **clase**

```
Persona p = new Persona("Messi", 29);
```

OBJETOS

- Diferencias con tipos primitivos
 - Los tipos primitivos no se instancian
 - Un tipo primitivo no tiene métodos
 - Un tipo primitivo no se puede heredar
 - En memoria, los primitivos solo guardan su valor, mientras que los objetos guardan también una **referencia** a una instancia

METODOS

- Los métodos son **cómo nos comunicamos con los objetos**
 - Cuando llamamos a un método sobre una instancia, estamos pidiendo a dicho objeto que ejecute una acción
 - Contienen:
 - Nombre
 - Parámetros de entrada
 - Parámetro de salida

METODOS

```
public class JavaApp {  
    public ReturnType methodName(  
        ParameterType parameter1,  
        ParameterType parameter2) {  
        // Do something  
        ReturnType returnType = new ReturnType();  
        return returnType;  
    }  
}
```

METODOS

- Si el método no **devuelve** ningún valor, el tipo de retorno deberá ser **void**.

```
public void methodName(  
    ParameterType parameter1,  
    ParameterType parameter2) {  
    // Do something  
}
```

METODOS

- Paro de parámetros: primitivos

```
public int methodName(  
    int parameter1,  
    int parameter2) {  
    return parameter1 * parameter2;  
}
```

METODOS

- Paso de parámetros: objetos complejos

```
public void changeName(Person p) {  
    p.name = "Paco";  
}
```

METODOS

- Sólo pueden devolver **un valor**.
 - No pueden devolver c y d
 - Se debe construir un objeto más complejo que contenga dentro los valores

```
public int methodName(int a, int b) {  
    int c = a+1;  
    int d = b+1;  
    return c,d;  
}
```



METODOS

- Método especial: **el constructor**
 - Método especial que se llama automáticamente cuando se crea un objeto de este tipo con la *keyword* **new**

```
public class MyClass {  
    /**  
     * MyClass Empty Constructor  
     */  
    public MyClass() {  
    }  
}
```

```
MyClass myClass = new MyClass();
```

METODOS

- Métodos estáticos
 - Método que se puede llamar **sin ninguna instancia del objeto**

```
Integer a = Integer.parseInt("12");
```


ALCANCE (SCOPE)

- El alcance de una clase, variable o método define su **visibilidad** y su **accesibilidad**.
 - Ejemplos.
 - Alcance de un parámetro de un método: accesible únicamente dentro de dicho método.
 - Alcance de una variable local: accesible desde su declaración hasta el final del bloque donde ha sido declarada

ALCANCE (SCOPE)

- Modificadores de visibilidad
 - public
 - private
 - protected
 - package (o vacío)
- Se aplican sobre clases, métodos o atributos de clase

ALCANCE (SCOPE)

	Class	Method, or Member variable
public	visible from anywhere	same as its class
protected	N/A	its class and its subclass, and from its package
package	only from its package	only from its package
private	N/A	only from its class

ALCANCE (SCOPE)

- Getters & Setters
 - Los atributos **private** pueden ser accedidos por un *getter*, y modificados por un *setter*.

```
public class Persona {  
    private String nombre;  
    private int edad;  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
}
```

CLASES – PRÁCTICA

- Crear la clase **Persona** con los siguientes atributos:
 - Nombre (tipo String)
 - Dinero (tipo float)
- Y los siguientes métodos:
 - Constructor que reciba un nombre y dinero inicial
 - Método **pagar(Persona quien, float cantidad)**, que añada el valor de *cantidad* al dinero de la persona que recibe el pago y lo reste de la persona que paga.
- Instanciar 2 personas (Paco y Pepe), y hacer que Paco pague a Pepe 200€.

CLASES – PRÁCTICA

- Crear la clase **Punto** con los siguientes atributos:
 - x: int
 - y: int
- Y los siguientes métodos:
 - Constructor sin parámetros
 - Constructor que reciba dos parámetros: **x, y**
 - Métodos **set** y **get** para cada atributo
 - Método **distancia(int x, int y)** que calcule la distancia al punto representado por los parámetros
 - Método **distancia(Punto p)** que calcule la distancia al punto representado por el parámetro
 - Método **toString()**, que imprima por pantalla una descripción útil del objeto. Ej: ***"Punto (2, 3)"***

CLASES – PRÁCTICA

- Crear la clase **Circulo** con los siguientes atributos:
 - centro: Punto
 - radio: int
- Y los siguientes métodos:
 - Constructor sin parámetros
 - Constructor que reciba tres parámetros: **x, y, radio**
 - Constructor que reciba dos parámetros: **Punto, radio**
 - Métodos **set** y **get** para cada atributo
 - Método **getArea()** que calcule el área del circulo
 - Método **getCircumference()** que calcule la circunferencia del círculo
 - Método **toString()**, que imprima por pantalla una descripción útil del objeto. Ej: ***“Círculo con centro en (2, 3) y radio 5”***

CLASES – PRÁCTICA

- Crear la clase **Triangulo** con los siguientes atributos:
 - v1: Punto
 - v2: Punto
 - v3: Punto
- Y los siguientes métodos:
 - Constructor sin parámetros
 - Constructor que reciba tres parámetros: **v1, v2, v3**
 - Métodos **set** y **get** para cada atributo
 - Método **getArea()** que calcule el área del circulo
 - Método **getPerimeter()** que calcule el perímetro del triangulo
 - El perímetro es la suma de sus tres lados, o la suma de las distancias entre todos los puntos
 - Método **toString()**, que imprima por pantalla una descripción útil del objeto. Ej: ***“Triangulo[(2,3),(5,8),(3,7)]”***