



UNIVERSIDAD DE CÓRDOBA

PROGRAMACIÓN WEB – JAVA

---

# Introducción a JAVA

Rafael Barbudo Lunar  
rbarbudo@uco.es



# Contenidos

1. Introducción
2. Configuración del entorno
3. Aspectos básicos del lenguaje

1.

# Introducción

# Características de Java

- **Orientación a objetos “pura”**. Cualquier función debe declararse dentro de una clase que heredará de `java.lang.Object`. No existen las estructuras (`struct`)
- **Simplicidad**. Elimina los punteros (todos los objetos por referencia), posee un recolector de basura (`gc`), hay tipo *boolean*, etc.
- **Interpretado**. Los programas en Java (“`.java`”) se traducen a *bytecode* (“`.class`”) y la máquina virtual de Java (JVM) los interpreta pasándolos a código máquina propio de cada plataforma.
- **Portable**. Se ejecuta en cualquier plataforma que disponga de una JVM que interprete los archivos *bytecode* generados a código máquina

2.

## Configuración del entorno

# Instalación de Java

- Se recomienda **utilizar –al menos– la versión Java 8**

- ☐ En los servidores de la UCO está instalado Java 7
- ☐ Novedad: [expresiones lambda](#)

- Enlace de descarga del *Java Development Kit* (JDK)

<https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

- ☐ Instrucciones para [Linux](#)

- Validar la instalación:

- ☐ `java -versión`
- ☐ `javac -versión`



Puede ser necesario  
actualizar el `path`

```
Microsoft Windows [Versión 10.0.19041.1083]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Rafael Barbudo Lunar>java -version
java version "1.8.0_301"
Java(TM) SE Runtime Environment (build 1.8.0_301-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.301-b09, mixed mode)

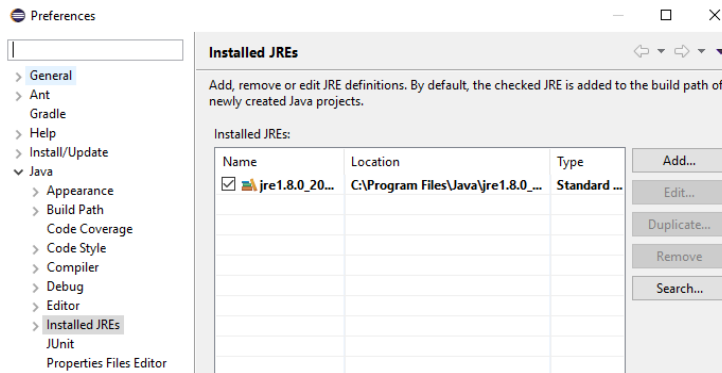
C:\Users\Rafael Barbudo Lunar>
```

# Instalación de Eclipse

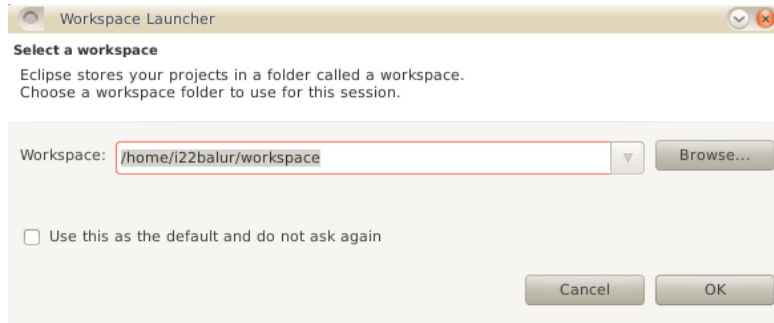
- Enlace de descarga de **Eclipse** (no estrictamente necesario)

<https://www.eclipse.org/downloads/packages/>

- ☐ En la UCO, se utilizará Eclipse Luna para las prácticas
- Eclipse debería reconocer automáticamente la instalación de Java
  - ☐ *Window > Preferences > Java > Installed JREs*



# Ejemplo\_ “Hola Mundo”



➤ *Aplicaciones > Programación > Eclipse*

➤ *Workspace Launcher*

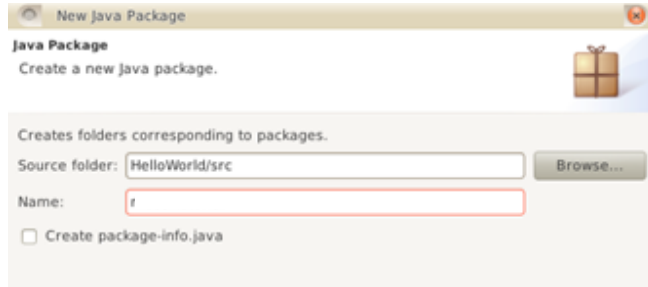
❑ **Workspace.** Espacio de trabajo común donde se albergarán los proyectos creados

➤ *File > New > Java Project*





# Ejemplo\_ “Hola Mundo”



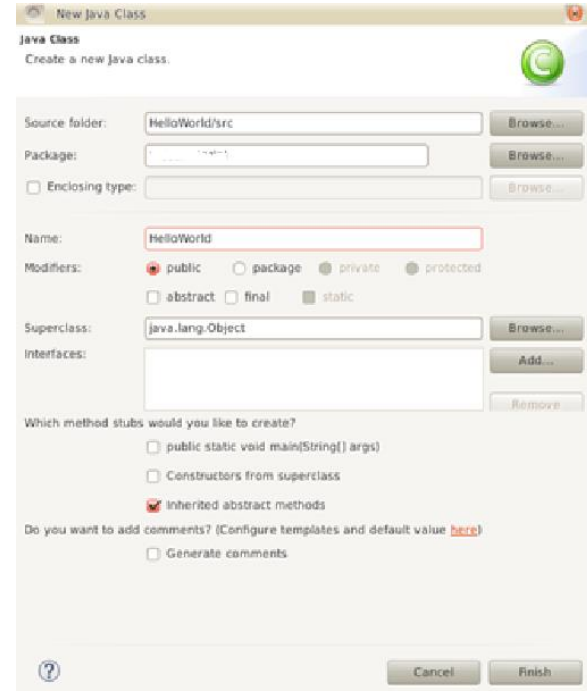
➤ *File > New > Package*



❑ **Paquete.** Mecanismo que permite organizar las clases/interfaces de Java en **espacios de nombres**

➤ *File > New > Class*

❑ ***es.uco.pw.HelloWorld***

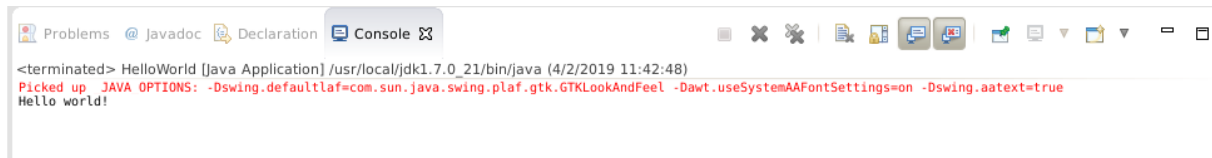
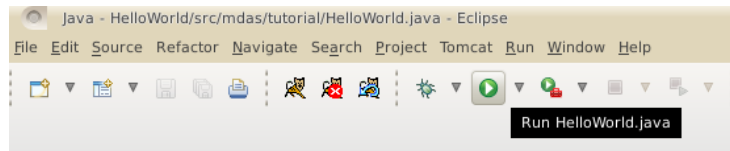


# Ejemplo\_ "Hola Mundo"

- Cuando una clase sea un programa ejecutable, debe definir el método **main()**
  - ❑ **public static void** *main*(String [] args)
- Como buena práctica, **main()** siempre **debe ir aparte del código de las clases del dominio**, generalmente en una clase llamada como el programa que deseamos ejecutar



```
1 package mdas.tutorial;
2
3 public class HelloWorld {
4
5     public static void main(String[] args) {
6         System.out.println("Hello world!");
7     }
8
9 }
10
```



```
<terminated> HelloWorld [Java Application] /usr/local/jdk1.7.0_21/bin/java (4/2/2019 11:42:48)
Picked up JAVA_OPTIONS: -Dswing.defaultlaf=com.sun.java.swing.plaf.gtk.GTKLookAndFeel -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Hello world!
```

3.

## Aspectos básicos del lenguaje

# Tipos en Java

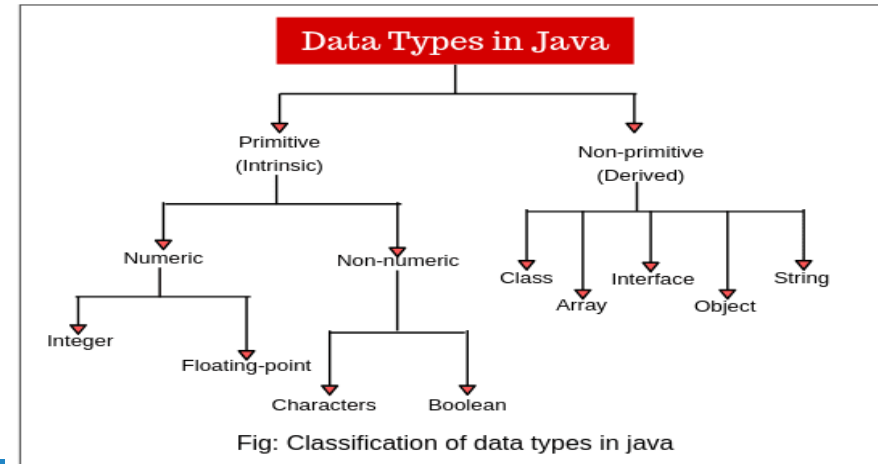
## ❑ Tipos primitivos con sus clases equivalentes (*Wrappers*)

### ❑ Numéricos:

- byte, Byte
- short, Short
- int, Integer
- long, Long
- float, Float
- double, Double

### ❑ No numéricos:

- boolean, Boolean
- char, Character



# Tipos en Java

## ❑ Tipos no primitivos (referenciados)

- Clase. Plantilla o abstracción que engloba un conjunto de instancias con propiedades similares
- Interfaz. Entidad que define una colección de métodos que se deben implementar en una clase
- Array. Colección de objetos del mismo tipo cuyo tamaño es fijo
- Cadenas. En realidad, objetos de la clase `String`



Aunque se pueden crear *Arrays* de primitivas, no se pueden crear colecciones (p.ej. listas) de primitivas: ~~`List<int> myList`~~

# Clases



Por convención, las clases se nombran con la primera letra en mayúscula, mientras que las interfaces comienzan por “I”

```
[<Modificadores>] class <NombreClase> [extends  
<ClasePadre>] [implements <Interfaz>]
```

## ❑ [<Modificadores>]

- **public**: la clase es accesible desde otras. Para acceder desde otros paquetes primero debe ser importada
- **abstract**: tiene al menos un método abstracto y no puede ser instanciado.
- **final**: termina una cadena de herencia

## ❑ **extends**

- Indica de qué clase hereda. Si no se especifica por defecto heredará de `Object`

## ❑ **Implements**

- Indica que la clase implementa uno o varios interfaces
- Las interfaces fuerzan a que un conjunto de métodos (cuya signatura es declarada en la interfaz) deban ser implementados

# Clases

## ➤ Cuerpo de clases

- ❑ **Constructores.** Si no se indica se crea uno por defecto
- ❑ **Variables.** Accesibles dentro de toda la clase. Fuera depende de su visibilidad: `public`, `package`, `protected`, `private`. Generalmente privadas.
- ❑ **Métodos.** Accesibles dentro de toda la clase. Fuera depende de su visibilidad: `public`, `package`, `protected`, `private`. Generalmente públicos.

## ➤ Ámbito

- ❑ Clase (`static`) o instancia



Los métodos estáticos pueden simular las funciones de C/C++ no declaradas dentro de clases

```
Person.java
1 package mdas;
2
3 public class Person {
4
5     private int dni;
6
7     static private String species;
8
9     public Person () {
10
11     }
12
13     public Person (int dni) {
14         this.dni = dni;
15         Person.species = "Homo sapiens";
16     }
17
18     public int getDni() {
19         return dni;
20     }
21
22     public void setDni(int dni) {
23         this.dni = dni;
24     }
25
26     public static String getSpecies() {
27         return species;
28     }
29
30     public static void setSpecies(String species) {
31         Person.species = species;
32     }
33
34     public static void main(String[] args) {
35         Person p1 = new Person(1234);
36         System.out.println(p1.getDni()); // output: 1234
37         System.out.println(Person.getSpecies()); // output: Homo sapiens
38         Person.setSpecies("Computer technician");
39         System.out.println(Person.getSpecies()); // output: Computer technician
40     }
41 }
```

# Clases

## ➤ Cuerpo de clases

- ☐ **Constructores**. Si no se indica se crea uno por defecto
- ☐ **Variables**. Accesibles dentro de toda la clase. Fuera depende de su visibilidad: `public`, `package`, `protected`, `private`. Generalmente privadas.
- ☐ **Métodos**. Accesibles dentro de toda la clase. Fuera depende de su visibilidad: `public`, `package`, `protected`, `private`. Generalmente públicos.

## ➤ Ámbito

- ☐ Clase (`static`) o instancia

Modificador (métodos y variables)	Clase	Package	Subclase	Todos
public	Sí	Sí	Sí	Sí
protected	Sí	Sí	Sí	No
No especificado	Sí	Sí	No	No
private	Sí	No	No	No





# Interfaces

## ➤ Declaración de interfaces:

[<Modificadores>] **interface** <NombreInterfaz> [**extends** <InterfazPadre>]

### ❑ [<Modificadores>]

- **public**: las interfaces solo pueden ser públicas. Para acceder desde otros paquetes primero debe ser importada.
- **abstract**: aunque se pueda indicar, las interfaces son abstractas por definición.

### ❑ **extends**

- indica que la interfaz hereda o deriva de otra(s). Permiten la herencia múltiple.

## ➤ Cuerpo de interfaces

- ❑ **Variables**. Aunque no se especifique, son **public**, **final** y **static**. Uso desaconsejado.
- ❑ **Métodos**. Por lo general, solo se indica su declaración (deben ser **public**).
- ❑ Es posible implementar el cuerpo si se usan los modificadores **static** o **default**.

```
1 package mdas;
2
3 public interface ExampleInterface {
4
5     public void printSomething();
6
7     // discouraged
8
9     public int num = 3;
10
11     public default void doSomething() {
12         System.out.println("Doing something...");
13     }
14
15     public static int getNum() {
16         return num;
17     }
18 }
19
```

# Arrays en Java

- Se diferencia entre **arrays como variable** (que almacena lista fija de valores de un mismo tipo) y **arrays como listas dinámicas**:

```
// Arrays como variable

double[] data;
//double data[];    -- Sintaxis alternativa

data = new float[10]; // Array fijo de 10 flotantes

int[] pares = {2,4,6,8,10}; // Los arrays son zero-index
String semana[] = {"Lunes", "Martes", "Miércoles",
"Jueves", "Viernes", "Sábado", "Domingo"};

out.println("Número de días:" + semana.length);

int[][] matriz = new int[3][4]; // [filas][cols]
```

# Arrays en Java

- Se diferencia entre **arrays como variable** (que almacena lista fija de valores de un mismo tipo) y **arrays como listas dinámicas**:
  - Los arrays dinámicos son objetos de la clase `java.util.ArrayList`
  - En la declaración se indica el tipo de los elementos: `ArrayList<tipo>`, donde **tipo** debe ser una clase (no datos primitivos)

```
ArrayList<String> nombres = new ArrayList<String>();
```

- Los `ArrayList` proporcionan métodos para añadir, obtener, eliminar o cambiar elementos, así como para obtener su longitud: `add`, `get`, `remove`, `set`, `size`

# Estructuras de control

## ➤ Condicionales:

```
if (<expresión-booleana>) {  
    sentencias;  
}  
[else if (<expresión-booleana>)  
{  
    sentencias;  
}]  
[else {  
    sentencias;  
}]
```

```
switch( <expresión> ) {  
    case <valor1>:  
        sentencias;  
        break;  
    case <valor2>:  
        sentencias;  
        break;  
    [default:  
        sentencias;]  
}
```

¡Se desaconseja el uso de SWITCH!

# Estructuras de control

## ➤ Bucles:

```
for ( <inicialización>; <terminación>; <iteración> )  
{  
    sentencias;  
}
```

```
while ( <terminación-expresión-booleana> )  
{  
    sentencias;  
}
```

```
do {  
    sentencias;  
} while ( <terminación-expresión-booleana> )
```

# Excepciones

- Una **excepción** es una condición anormal que surge en una secuencia de código durante su ejecución.
- Son el mecanismo de Java para el **tratamiento de errores**, que permite a los métodos finalizar abruptamente ante una situación anómala.

```
try {  
    // Código que puede generar una excepción;  
}  
catch( TipoExcepción e ) {  
    // Código para tratar la excepción;  
}  
  
[catch(...) ]  
  
[finally {  
    // Código para finalizar el bloque;  
}]
```

# Excepciones

```
1 package ;
2
3 public class HolaIterativo
4 {
5     public static void main( String args[] )
6     {
7         String Saludos[] = {"Hola Mundo!", "HOLA Mundo!"};
8         try {
9             for(int i=0; i<5; i++)
10                 System.out.println( Saludos[i] );
11         }
12         catch (ArrayIndexOutOfBoundsException e) {
13             System.out.println("Se ha desbordado el array 'Saludos'");
14         }
15         catch (Exception e) {
16             System.out.println(e);
17         }
18         finally {
19             System.out.println("Esto se imprime siempre");
20         }
21     }
22 }
```

Nombre	Significado
Excepcion	Es la excepción más genérica.
ArithmeticException	Suele ser el resultado de división por 0.
NullPointerException	Intento de acceder a una variable no definida.
ClassCastException	Intento de convertir a una clase inválida.
NoClassDefFoundException	Clase no encontrada.
ArrayIndexOutOfBoundsException	Acceso fuera de los límites de un array.

# Ficheros

- Las clases e interfaces necesarias en Java para la **manipulación de la E/S** se alojan en el paquete `java.io`
- Como en otros lenguajes OO, Java está basado en la manipulación de flujos de datos (***data streams***), es decir, control de una secuencia continua de datos desde algún dispositivo hasta el programa
- Los **flujos de datos** se pueden clasificar **según la dirección de la secuencia**:
  - ❑ ***Flujos de entrada***. Los datos fluyen desde algún dispositivo hacia el programa
  - ❑ ***Flujos de salida***. Los datos fluyen desde el programa hacia un dispositivo
- También se pueden clasificar **según la naturaleza de los datos**:
  - ❑ ***Flujos binarios***. Ocupan un byte
  - ❑ ***Flujos de caracteres***. Cada carácter ocupa 2 bytes, utilizando Unicode



# Ficheros

- Ejemplo de manejo simultáneo de un flujo de entrada y de salida
- Clases importantes:

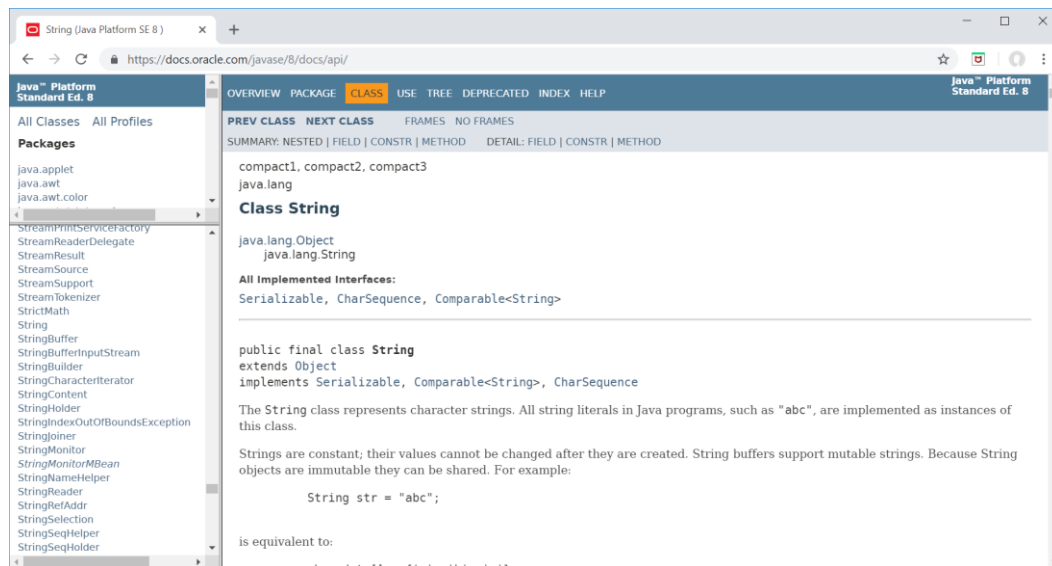
- ❑ `java.io.File`
- ❑ `java.io.BufferedReader`
- ❑ `java.io.BufferedWriter`

```
11 public class RemoveOddLines
12 {
13     public static void main(String[] args) {
14
15         String inputFilename = args[0];
16         String outputFilename = args[1];
17
18         try {
19             BufferedReader br = new BufferedReader(new FileReader(new File(inputFilename)));
20             BufferedWriter bw = new BufferedWriter(new FileWriter(new File(outputFilename)));
21
22             int counter = 0;
23             String line;
24
25             while((line = br.readLine()) != null) {
26                 if(counter%2 == 0)
27                     bw.write(line + "\n");
28                 counter++;
29             }
30
31             br.close();
32             bw.close();
33
34         } catch (FileNotFoundException e) {
35             e.printStackTrace();
36         } catch (IOException e) {
37             e.printStackTrace();
38         }
39     }
40 }
41 }
```

# Documentación de Java

- Es **IMPRESINDIBLE** dominar **Javadoc**, como referencia de información:

<https://docs.oracle.com/javase/8/docs/api/>



# Programación Web



Introducción a Java - Curso 2021/22