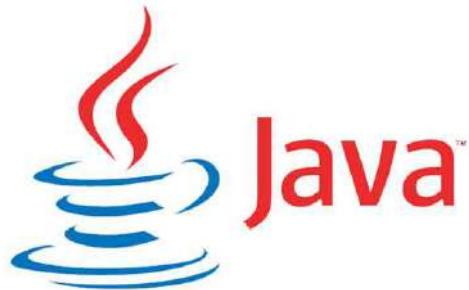


Bloque de Java



David Bernal González



EL CÓDIGO
APRENDIÓ A
PROGRAMAR
EN 1 MES



PROGRAMMING IN

JAVA



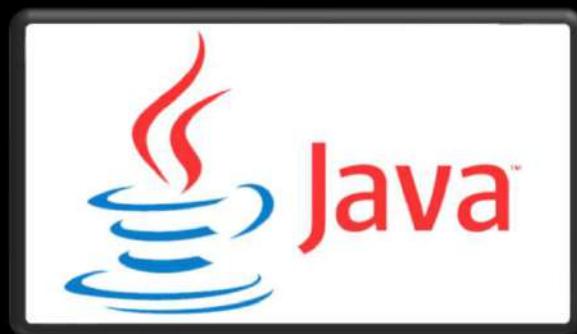


PRESENTANDO

A JAVA



¿QUÉ ES JAVA?



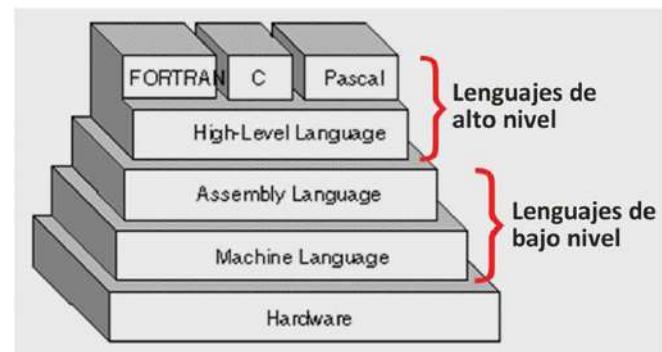
¿Qué es Java?

- Lenguaje de programación de propósito general (diseñados para resolver todo tipo de problemas).
- Catalogado como un lenguaje de alto nivel.

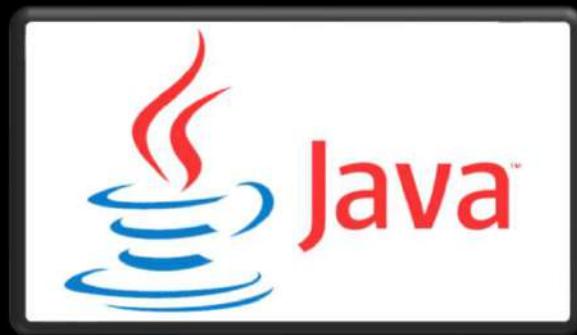


Tipos de lenguajes de programación

- Clasificación en función de la distancia con la que interactúe dicho lenguaje sobre el hardware del dispositivo:
 - **Lenguajes de bajo nivel:** utilizan términos muy cercanos a la máquina. Las instrucciones están escritas en código binario (01001...) y/o código ensamblador.
 - **Lenguajes de alto nivel:** utilizan términos más cercanos a las personas, normalmente en inglés como son: IF (si), ELSE (sino), WHILE (mientras), etc.



¿PORQUÉ JAVA?





Características de Java

Simple

Java, nace de una evolución de C y C++. Pero, reduciendo algunos aspectos que podían incrementar su complejidad.
¡Por tanto Java es “fácil” de usar!

Portable (arquitectura neutral)

Permite la ejecución de un mismo programa (con tan solo realizar una vez el proceso de compilación) en múltiples sistemas operativos gracias a la JVM.

Orientado a Objetos (POO)

Utiliza el paradigma de la programación orientado a objetos.
¡Por lo que para Java prácticamente todo (a excepción de los tipos de datos primitivos) es un objeto!

Robusto

Verifica el código del programa durante el ciclo de desarrollo y nuevamente posteriormente durante su compilación.
Además, es un lenguaje altamente tipado lo que significa que está basado en unas buenas prácticas de programación.

Distribuido

Proporciona una serie de librerías que nos otorgan la capacidad de realizar comunicaciones en red por ejemplo con el protocolo TCP/IP permitiéndonos acceder a ficheros o datos situados en otros equipos fácilmente.

Compilado y posteriormente Interpretado

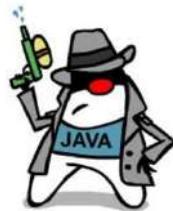
Java se compila preparando los ficheros para posteriormente ser ejecutado en la JVM. Y finalmente, una vez compilado se interpreta por la JVM.

Por ello, podemos afirmar que ¡Java es muy rápido!

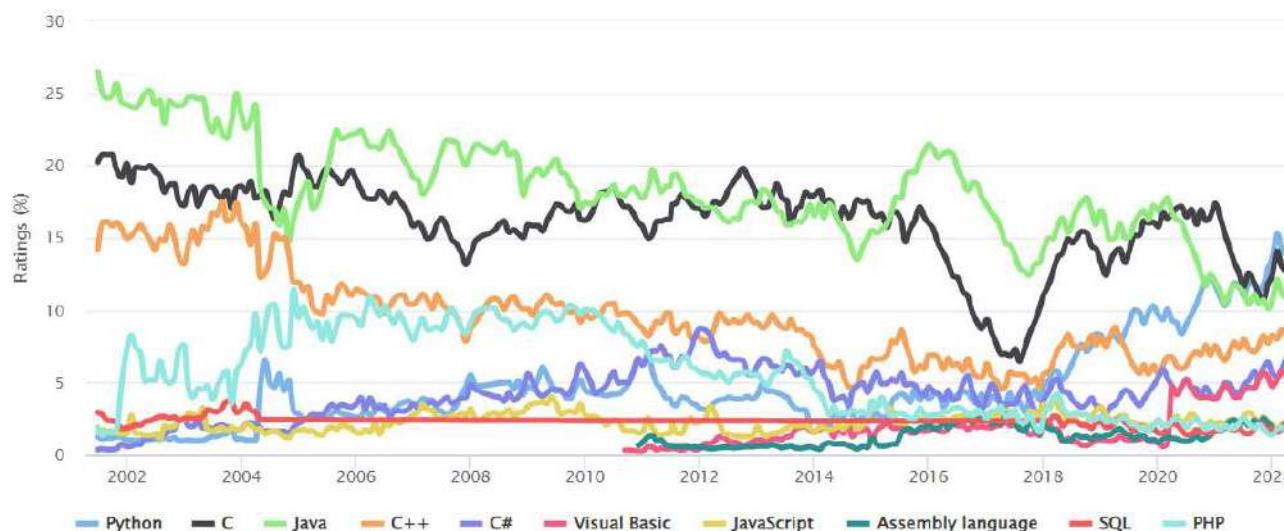
Multitarea (Multihilo o Multithread)

Permite la ejecución concurrente (simultáneos /a la vez) de varios procesos de nuestros programas en un sistema.

¿Porque aprender Java?



- Java es uno de los lenguajes más usados del mundo.
- Tiene una gran comunidad
- Junto a C y C++ ha dominado/reinado la programación comercial más de 25 años. Lo que se traduce en que hay muchos desarrollos realizados en Java que mantener. Y, a dario, se continua realizando nuevos desarrollos en Java.



INSTALACIÓN Y CONFIGURACIÓN DE JAVA



INSTALACIÓN DE JDK



Descargando JDK

- Para instalar JDK vamos a la web de Oracle. En este caso, vamos a descargar la versión 11 que es una LTS (Long Time Support):

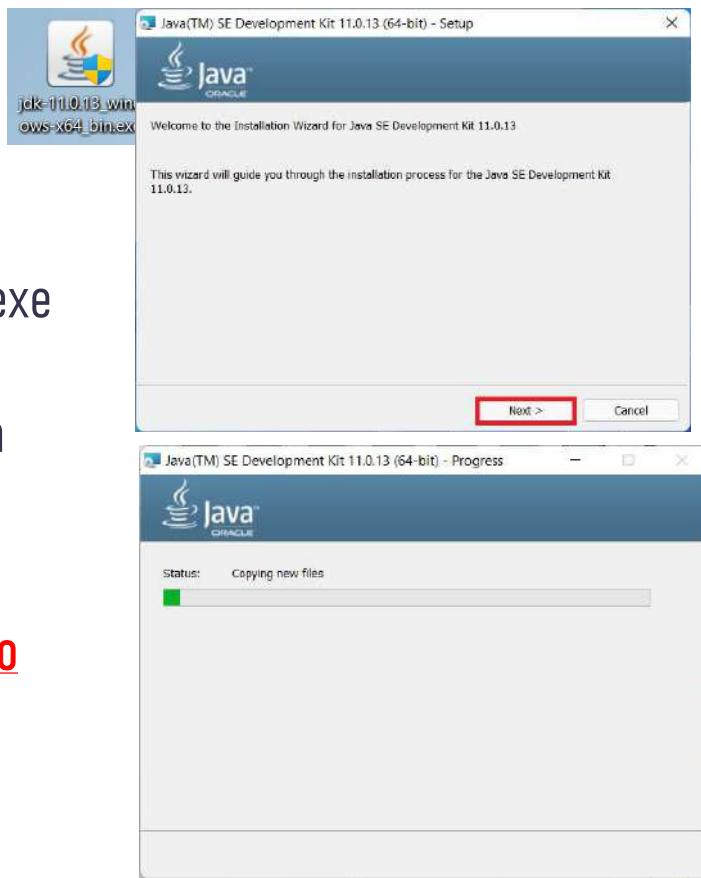
A Google search results page for "oracle jdk 11". The search bar has "oracle jdk 11" highlighted with a red box. Below the search bar, there are tabs for Todo, Noticias, Vídeos, Imágenes, and Shopping. The Todo tab is selected. The search results show approximately 15,500,000 results in 0.51 seconds. The top result is a link to "Java Archive Downloads - Java SE 11 | Oracle España", which is highlighted with a red box. The snippet below the link reads: "Java SE 11 Archive Downloads. Go to the Oracle Java Archive page. The JDK is a development environment for building applications using the Java programming ...".

Java SE Development Kit 11.0.13		
This software is licensed under the Oracle Technology Network License Agreement for Oracle Java SE		
JDK 11.0.13 checksum		
Product / File Description	File Size	Download
Windows x64 Installer	139.83 MB	 jdk-11.0.13_windows-x64_bin.exe
Windows x64 Compressed Archive	157.28 MB	 jdk-11.0.13_windows-x64_bin.zip



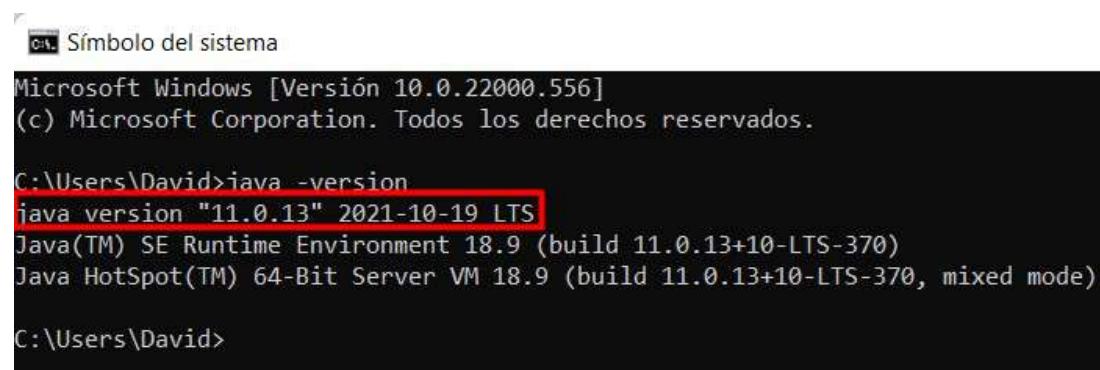
Instalación de JDK

- Una vez descargada, ejecutamos el exe y realizamos la instalación, con siguiente, siguiente.
- **¡IMPORTANTE, no tocar las rutas por defecto!**



Instalando JDK

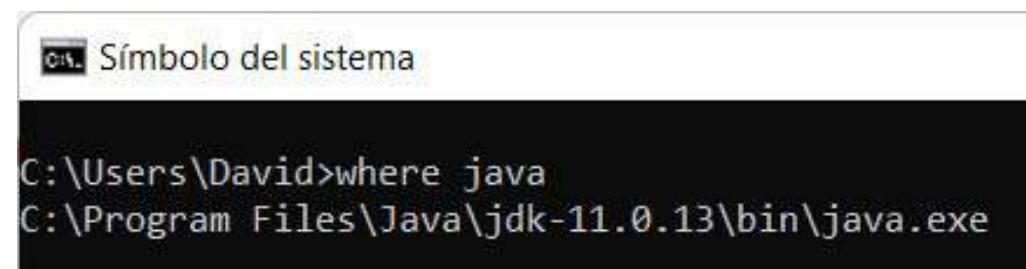
- Una vez instalado java vamos a realizar un java -versión para confirmar que lo hemos instalado correctamente:
- Además, con esto, confirmaremos que estamos trabajando con la versión deseada de Java. Ya que, en este caso queremos trabajar con la versión 11 de Java



```
Microsoft Windows [Versión 10.0.22000.556]
(c) Microsoft Corporation. Todos los derechos reservados.

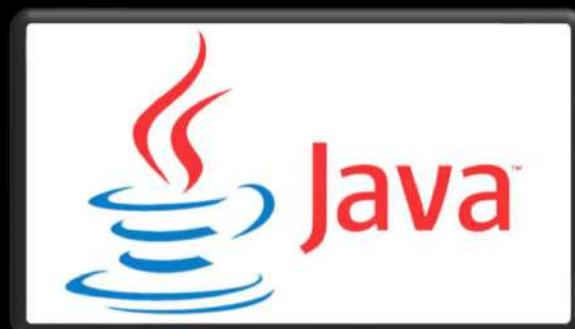
C:\Users\David>java -version
java version "11.0.13" 2021-10-19 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.13+10-LTS-370)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.13+10-LTS-370, mixed mode)

C:\Users\David>
```



```
C:\Users\David>where java
C:\Program Files\Java\jdk-11.0.13\bin\java.exe
```

CONFIGURACIÓN DE VARIABLES DE ENTORNO



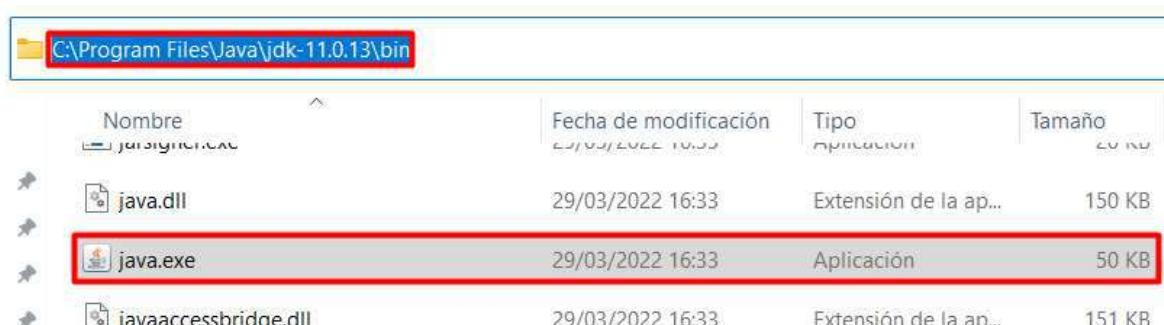


ENVIRONMENTO VARIABO

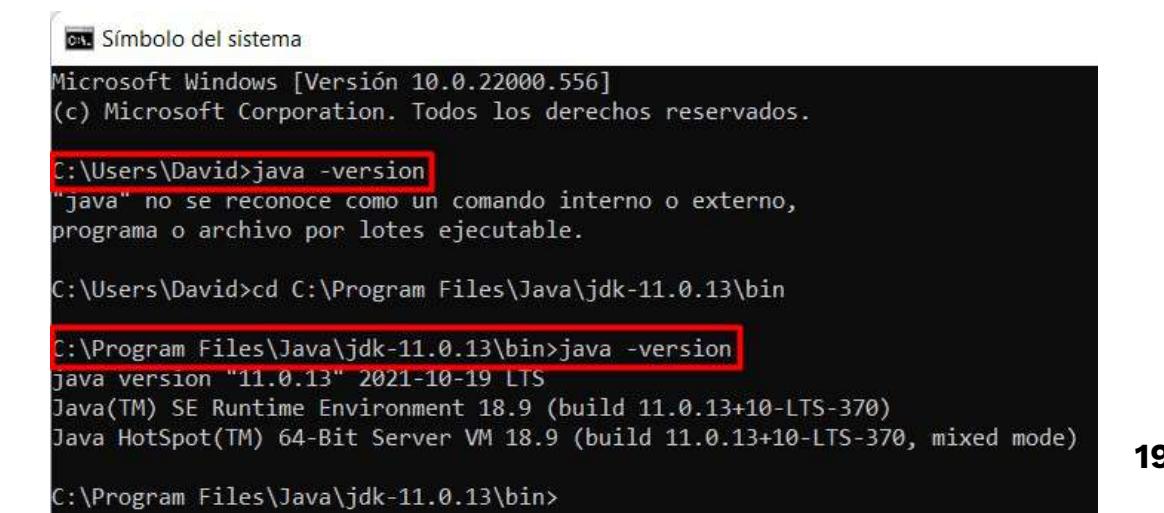
18

Variables de entorno

- Si cuando hemos instalado JDK, no se han configurado correctamente las variables de entorno en el sistema. Solamente podremos trabajar con java siempre y que nos situemos en el directorio \bin de la carpeta sobre la que hemos realizado la instalación de Java



Nombre	Fecha de modificación	Tipo	Tamaño
jarsigner.exe	29/03/2022 16:33	Aplicación	150 KB
java.exe	29/03/2022 16:33	Aplicación	50 KB
javaaccessbridge.dll	29/03/2022 16:33	Extensión de la ap...	151 KB



```
C:\Users\David>java -version
"java" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

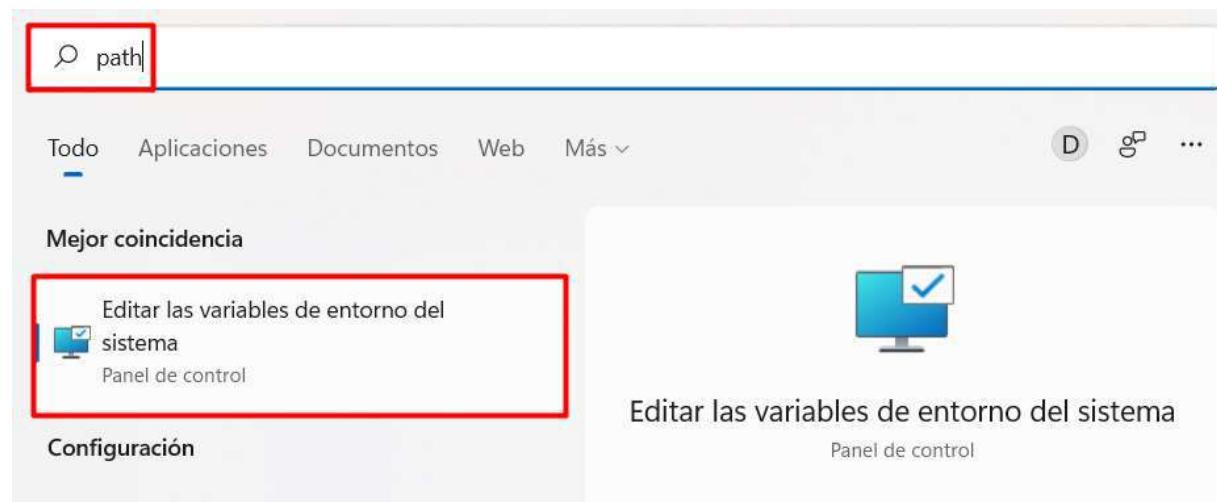
C:\Users\David>cd C:\Program Files\Java\jdk-11.0.13\bin

C:\Program Files\Java\jdk-11.0.13\bin>java -version
java version "11.0.13" 2021-10-19 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.13+10-LTS-370)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.13+10-LTS-370, mixed mode)

C:\Program Files\Java\jdk-11.0.13\bin>
```

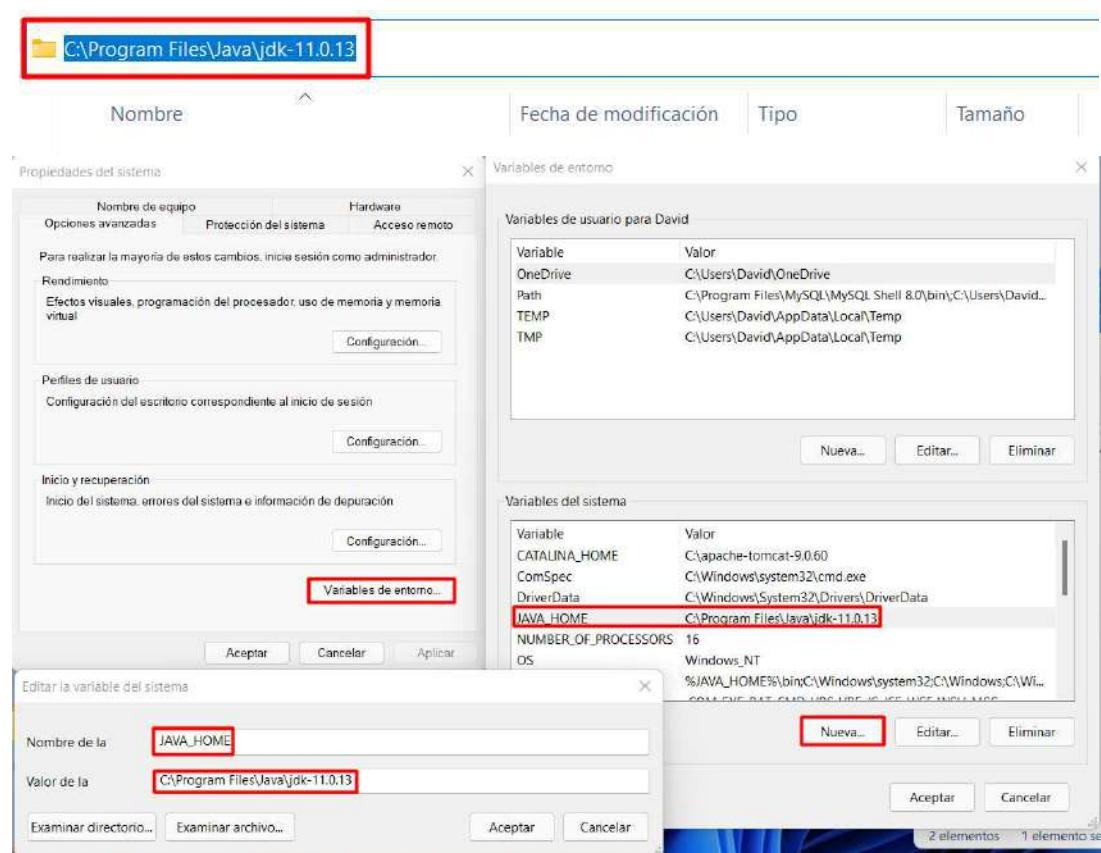
Variables de entorno

- Para poder trabajar desde cualquier ruta, debemos configurar las variables de entorno.
- Para ello, siempre y que estemos trabajando el sistema operativo Windows, tenemos que ir al editor de variables de entorno:



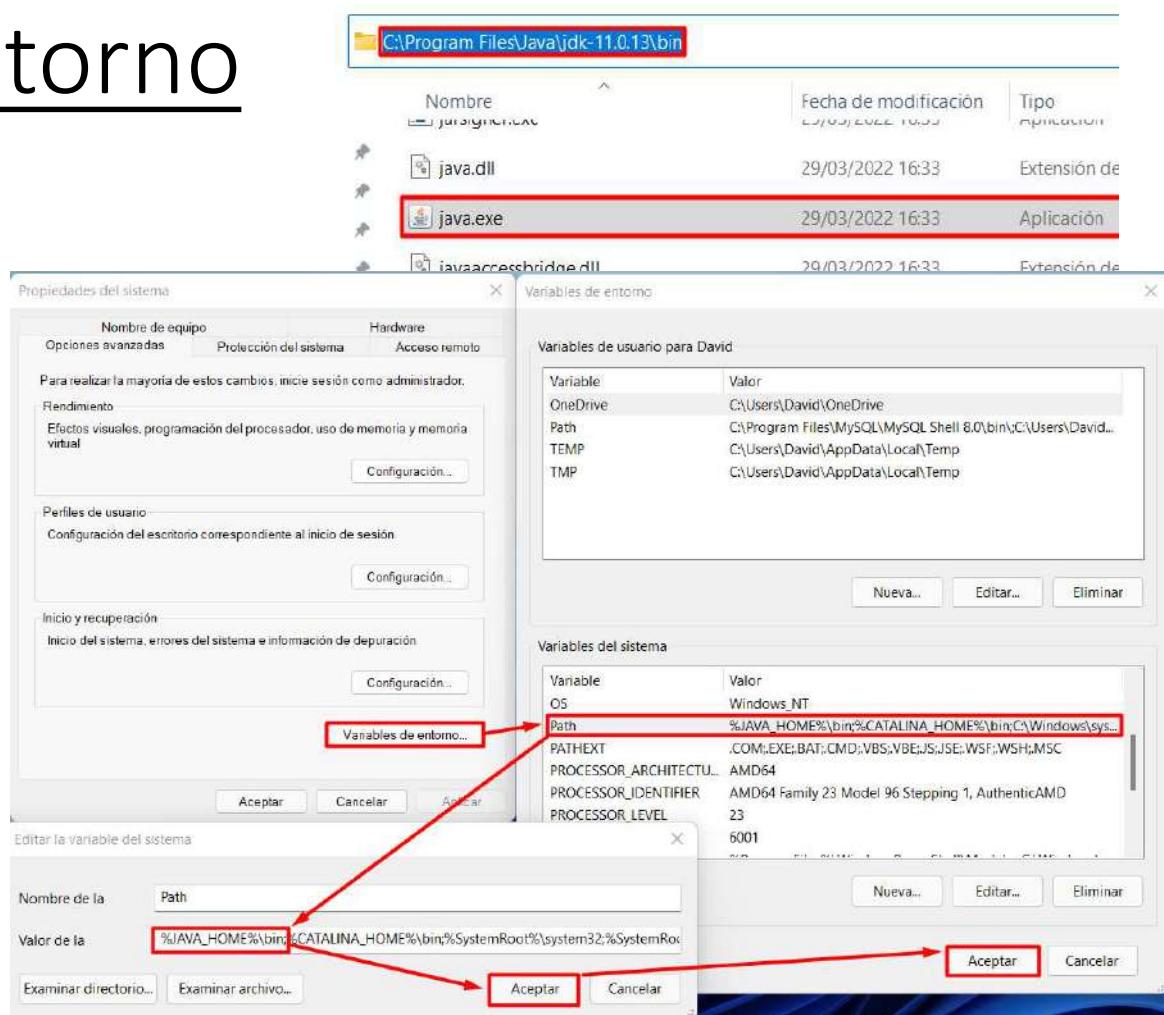
Variables de entorno

- Primeramente, definimos el variable de entorno JAVA_HOME, mediante el cual le indicaremos al sistema cual es la ruta base en la que tenemos ubicada nuestra versión de JDK.
- Esto nos permitirá indicarle al sistema operativo la ruta raíz de nuestra versión de Java (JDK)



Variables de entorno

Finalmente, en la variable de entorno path, debemos definir la ruta a la carpeta bin. Aunque podríamos poner la ruta C:\Program Files\Java\jdk-11.0.13\bin, es mejor utilizar %JAVA_HOME%\bin;. Ya que, de esta manera si queremos cambiar el JDK de la variable de entorno JAVA_HOME, solamente deberemos modificar la variable de entorno JAVA_HOME



Variables de entorno

- Configurar las variables de entorno nos permitirán ejecutar comandos utilizados para trabajar con Java, como por ejemplo:
 - Java c: para realizar compilaciones:
 - Java: para ejecutar las clases ya compiladas
- Sin tener que ir a la ruta en la que se encuentra el fichero java.exe para poder ejecutar Java.
- Que estas variables estén bien configuradas es totalmente necesario para poder trabajar con un IDE (Entorno de trabajo)

```
Símbolo del sistema
C:\Users\David>where java
C:\Program Files\Java\jdk-11.0.13\bin\java.exe
```

```
Símbolo del sistema
Microsoft Windows [Versión 10.0.22000.556]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\David>java -version
java version "11.0.13" 2021-10-19 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.13+10-LTS-370)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.13+10-LTS-370, mixed mode)

C:\Users\David>
```

Ejercicios

1. Instala la última versión de Java, manteniendo la versión 11 LTS que acabamos de instalar, de la 11 y realiza la configuración desde las variables de entorno para que te funcione.

Realiza un java -versión y comprueba que la versión a cambiado

2. Vuelve a apuntar hacia la versión 11 LTS

INSTALACIÓN Y CONFIGURACIÓN DE ECLIPSE



Intalando Eclipse JEE Edition (IDE)

- Para descargar Eclipse, vamos a la web <https://www.eclipse.org/downloads/> y durante su instalación marca la opción de Eclipse Enterprise Java and Web Developers:



HISTORIA DE JAVA



Historia de Java

- El siguiente artículo [Artículo de introducción a Java](#) nos va a permitir entender la filosofía que hay detrás del lenguaje, en él, haremos una vista de pájaro, en el que hablaremos sobre anécdotas, historia, etc.

 **Comenzando con Java: Presentación, orígenes e historia del lenguaje de programación, Hola Mundo...**

♪ [English Version] ♪

En este mega post, vamos a hacer un especie de viaje en el tiempo que nos va a permitir profundizar en aterrizar en la tecnología, entendiendo cuál es la filosofía que esconder este el lenguaje antes de ponernos a picar código, haciendo una pequeña vista de pájaro, cogiendo un poco de “culturilla” sobre varios aspectos, hablando sobre algunas anécdotas de este lenguaje, conocido su historia, etc.



La finalidad de todo esto, será la de obtener un gran conocimiento, una gran visión de cómo funciona internamente Java y entender el porqué realizamos según qué determinados procedimientos.



Cuando Hitler se entera de la compra de SUN por parte de Oracle



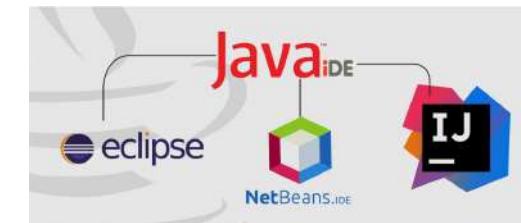
BENEFICIOS DE UN ENTORNO DE TRABAJO (IDE)





Entorno de trabajo

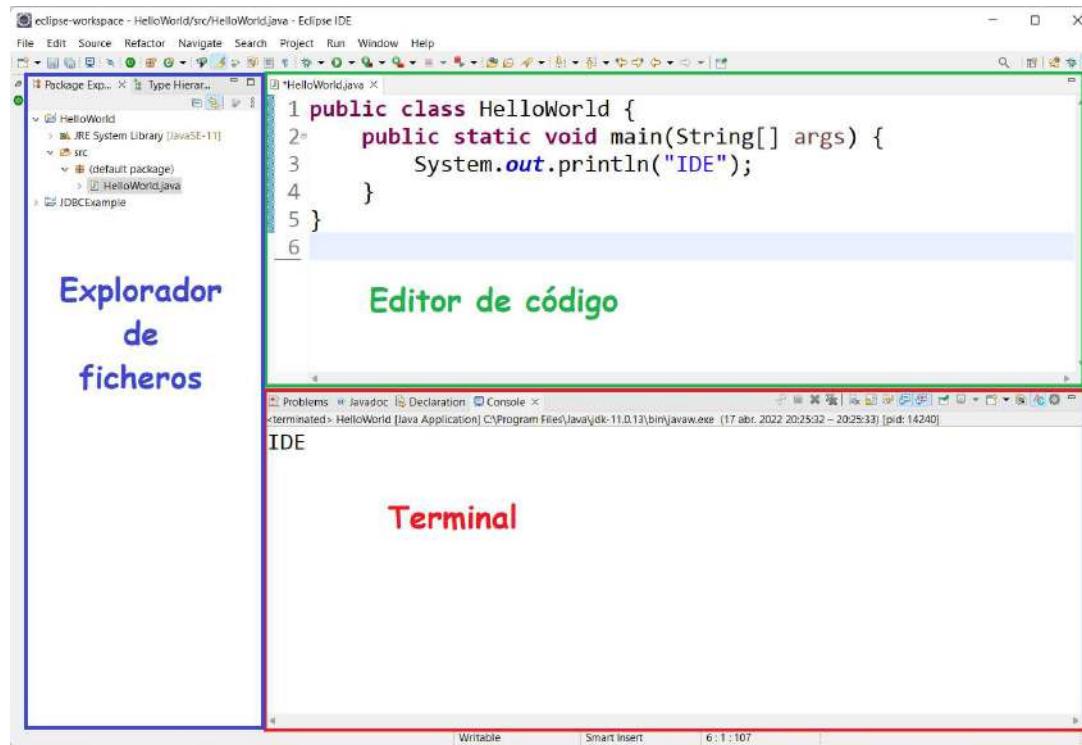
- IDE = Integrated Development Environment (o Entorno de trabajo).
- Si no trabajamos con un IDE, necesitamos tener varias ventanas abiertas: explorador de archivos, editor de texto y una terminal para realizar la compilación/ejecución del programa.
- Un IDE entre otros beneficios nos permite agrupar todas estas herramientas que acabamos de comentar en una sola ventana, haciendo que trabajar con Java sea mucho más sencillo y evitando así el tener que tener tantas ventanas abiertas.
- Los principales IDEs para trabajar con Java son:
 - Eclipse
 - Netbeans
 - IntelliJ
 - Visual Studio Code: también nos permite trabajar con Java. Aunque, está más enfocado hacia la web.





Partes de un IDE

- En mi caso trabajaré con Eclipse, y podemos ver que tenemos todas las partes que hemos comentado anteriormente agrupadas en una misma ventana:





Intellisense

- Además de otras tantas cosas, Eclipse, uno de los mejores beneficios que nos proporciona Eclipse es el conocido como intellisense (auto completado).
- Si por lo que sea se nos cierra el intellisense, lo podemos volver a abrir simplemente pulsando CONTROL + ESPACIO.

The screenshot shows the Eclipse IDE interface with the following details:

- File Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbars:** Standard toolbar and Java-specific toolbar.
- Views:** Package Explorer (HelloWorld project selected), Type Hierarchy, and *HelloWorld.java editor.
- Code Editor:** The code for `HelloWorld.java` is displayed:

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3 }
```

A cursor is positioned at the start of the third line.
- Intellisense Pop-up:** A tooltip appears below the third line, providing information about the `System` class:

The `System` class contains several useful class fields and methods. It cannot be instantiated. Among the facilities provided by the `System` class are standard input, standard output, and error output streams; access to externally defined properties and environment variables; a means of loading files and libraries; and a utility method for quickly copying a portion of an array.

Since: 1.0
- Completion List:** A list of suggestions is shown on the right side of the editor, starting with `System - java.lang`.



Otros beneficios de Eclipse

- El IDE nos ofrece:

- Intellisense: herramienta para ayudarnos a escribir
 - Compiler: herramienta para realizar las compilaciones sin necesidad de tener que ejecutar los comandos javac y java
 - Syntax Highlighter: herramienta para subrayarnos mejores prácticas, recomendaciones, posibles errores, etc.

The screenshot shows the Eclipse IDE interface with the 'HelloWorld.java' file open. A tooltip is displayed over the word 'System' in the code, providing information about the class: 'The system class contains several useful class fields and methods. It cannot be instantiated. Among the facilities provided by the system class are standard input, standard output, and error output streams; access to externally defined properties and environment variables; a means of loading files and libraries; and a utility method for quickly copying a portion of an array.' Below the tooltip, a list of static methods from the 'java.lang.System' class is shown, with 'out.println()' highlighted.

The screenshot shows the Eclipse IDE interface with the 'Main.java' file open. The code prints 'Hola mundo'. In the 'Console' tab, the output 'Hola mundo' is visible. The status bar at the bottom indicates the command used: '<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe {18 abr. 2022 11:38:04 - 11:38:04}'.

System.out.println("Hola mundo");

3



When I am writing the code in IDE..



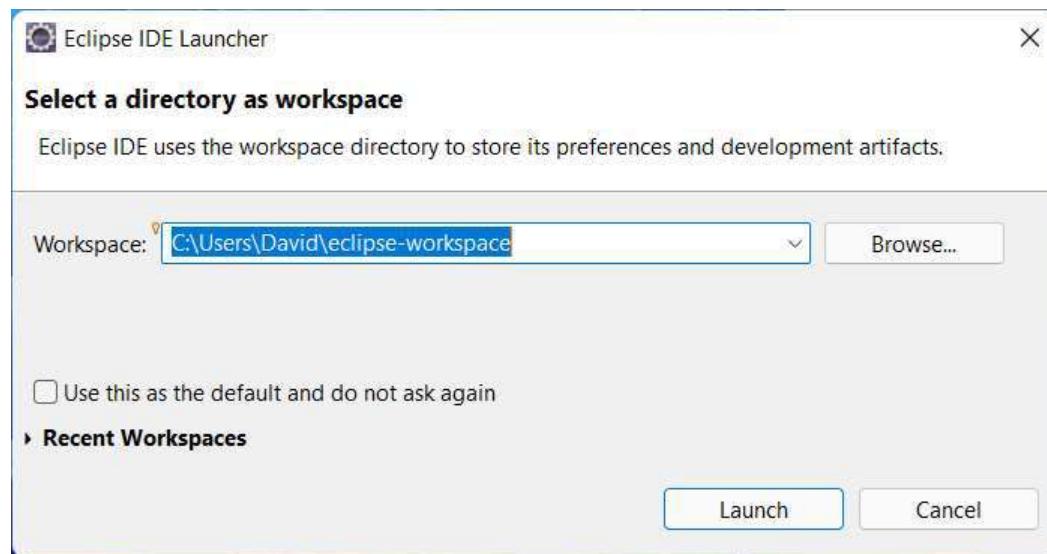
¿QUÉ ES EL WORKSPACE?





¿Qué es el workspace?

- El workspace es el directorio que será nuestro espacio de trabajo, es decir, donde Eclipse alojará nuestros proyectos de Java. Aunque se puede modificar la ruta, la idea del workspace es la de tener todo el trabajo en un sitio predefinido y poder encontrarlo fácilmente.



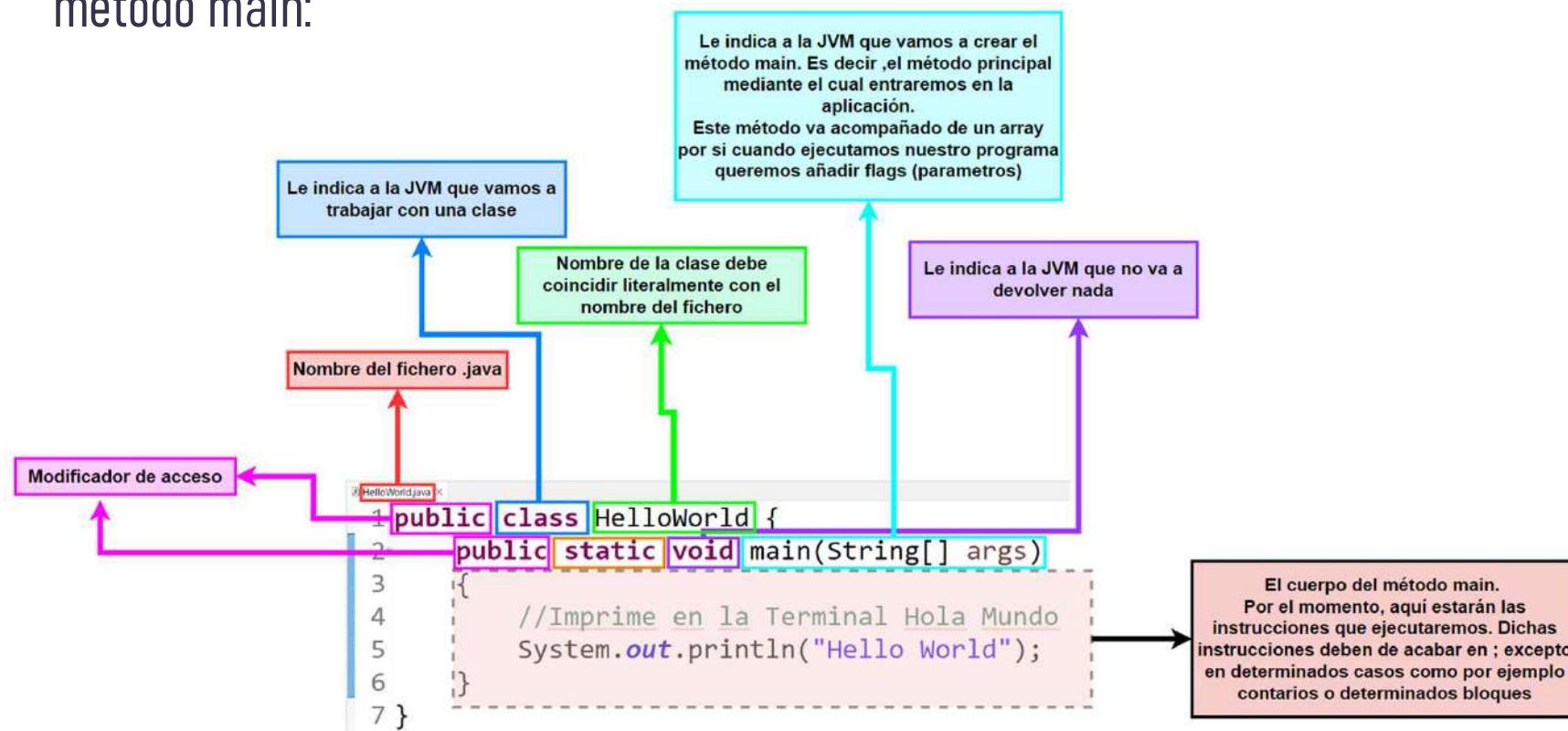
HELLO WORLD





Entendiendo el Hello World

- Esquema que desengrana la estructura de una clase que en su interior contiene el método main:





¿Qué pasa si un programa no tiene un método main?

- El método main, será “la boca de la botella” mediante a la cual “entraremos” en nuestra aplicación. Por tanto, el método main, es el encargado de ejecutar dicha aplicación. Si no existe al menos un método main dentro de nuestra aplicación, la aplicación no podrá arrancar. Cuando intentemos arrancarla, nos aparecerá un error como el siguiente: **X "Main method not found" X**
- Vamos a verlo:

```
<terminated> HelloWord [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (16 abr. 2022 14:02:31 – 14:02:31)
Error: Could not find or load main class HelloWord
Caused by: java.lang.ClassNotFoundException: HelloWord
```

Ejercicios

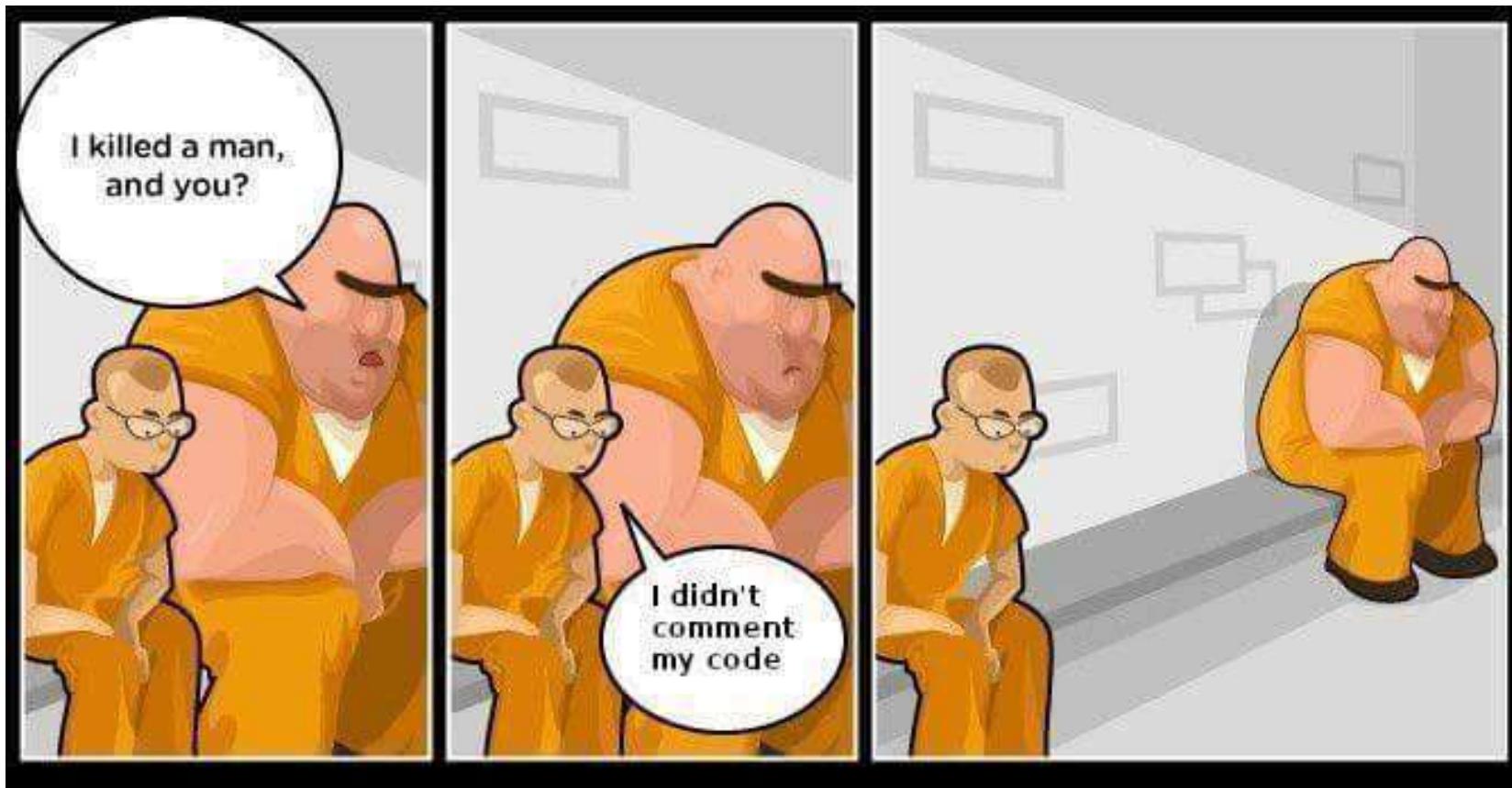
Una vez leído el artículo y configurado Java, realiza los siguientes ejercicios:

1. Realiza la compilación de un programa en Java de forma manual (desde una Terminal) en el cual deberá de mostrar el siguiente mensaje por consola:
 - Hola desde mi primer programa Java (CMD).
2. Realiza la descompilación (en medida de lo posible) de un programa en Java de forma manual (desde una Terminal)
3. Realiza la compilación de un programa en Java desde un IDE (Eclipse) en el cual se deberá de mostrar el siguiente mensaje por terminal:
 - I ❤ Java
4. Realiza la descompilación (en medida de lo posible) de un programa en Java con un software especializado para ello.



COMENTARIOS







¿Qué son los comentarios?

- Los comentarios nos permiten comentar/documentar nuestro código de Java evitando así dichas instrucciones sean ejecutadas por el compilador.
- Otro uso habitual, puede ser el de comentar instrucciones o bloques de código evitando así que se ejecute dicha parte del código.

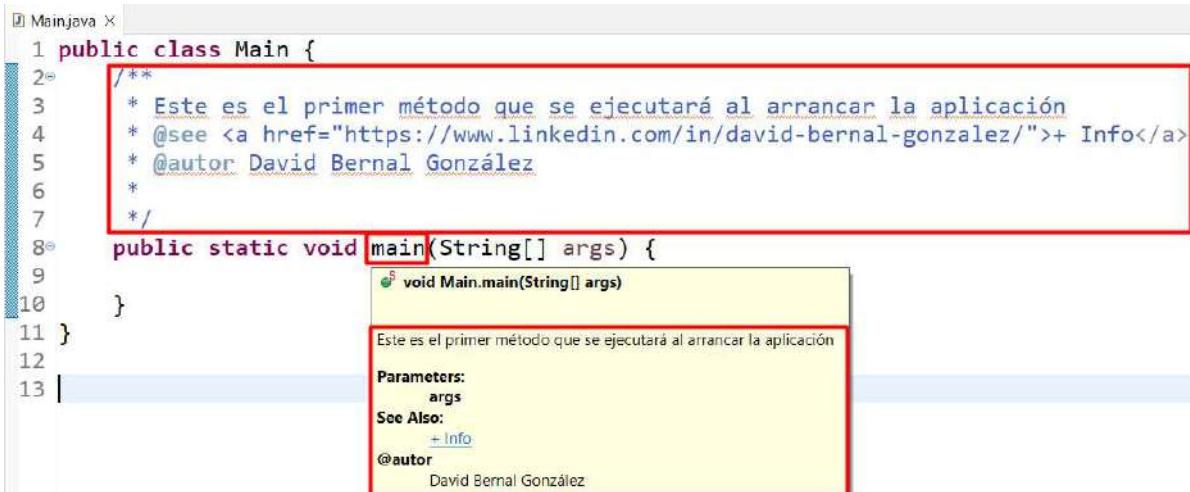




Tipos de comentarios

- En línea:

```
System.out.println("Hola mundo"); // En linea
```
- En bloque:

```
/* Comentario  
en  
bloque */  
System.out.println("Hola mundo");
```
- Comentarios de documentación (Java Docs): nos permite realizar documentación sobre clases, métodos, etc.


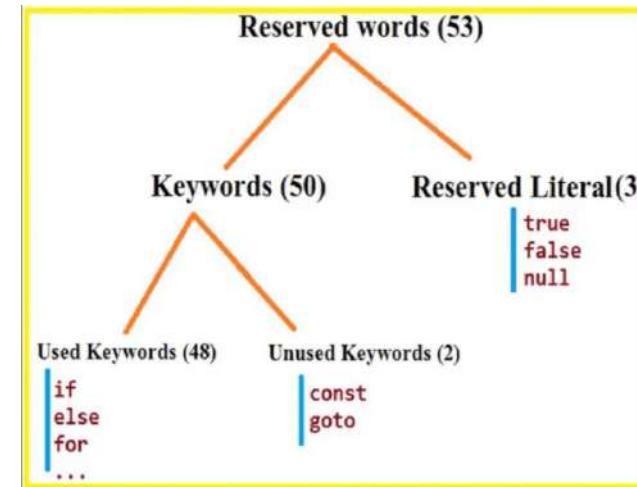
PALABRAS RESERVADAS





¿Qué son las palabras reservadas?

- Las palabras reservadas o reserved words, son las palabras mediante a las cual escribimos en el lenguaje de programación Java.
- Java, tiene en total 53 palabras en total.
- Todas operativas a excepción de const y goto, que pese a que era utilizada inicialmente cuando James Gosling y su equipo cuando crearon la Java Virtual Machine (JVM). Más tarde, ambas, dejaron de ser implementadas (utilizada) dentro del lenguaje. Es decir, fueron desactivadas. Pero pese a ello, continúan existiendo. Por lo que, por ejemplo, no podrían usarse por ejemplo para definir una variable. **48**





¿Qué son las palabras reservadas?

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

Aportaciones de:

lenguaje c, lenguaje c++, lenguaje java

Nota: goto y const se mantienen aunque no se utilizan.

INPUT Y OUTPUT

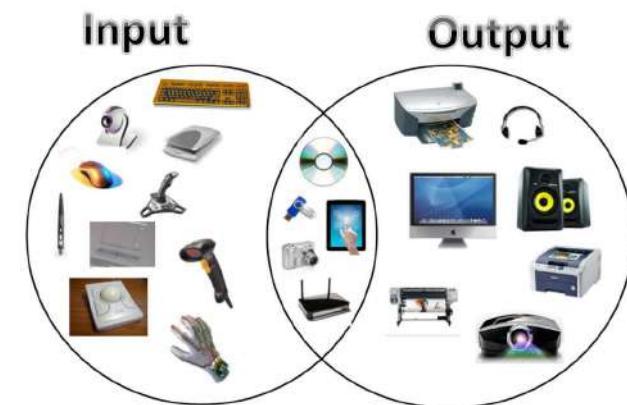




Input y output

En función de la relación que tenga un componente con el dispositivo, nos permitirá:

- Input (o entrada): nos permite introducir datos en el dispositivo.
 - Algunos ejemplos son: un teclado, un scanner, un ratón...
- Output (o salida): nos permiten sacar datos desde el dispositivo.
 - Algunos ejemplos son: una pantalla, un proyector, unos altavoces...
- Existen algunos dispositivos como las pantallas tactiles, los routers, los USB, etc. Que se pueden relacionar de ambas maneras con el computador.



MENSAJES DE SALIDA (OUTPUT) BÁSICOS POR TERMINAL/CONSOLA





Mensajes por consola: print

- La clase System.out nos permite hacer un output por consola. Para ello, podemos utilizar varios métodos:
 - Print: no realiza ningún salto de línea. Por lo que si utilizamos varias veces dicha instrucción, los mensajes se mostrarán en la misma línea

The screenshot shows an IDE interface with two tabs: 'HelloWorld.java' and 'Console'. The code in 'HelloWorld.java' is:

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         System.out.print("Hola");  
4         System.out.print("Mundo");  
5     }  
6 }
```

The output in the 'Console' tab is 'HolaMundo', with both words highlighted in green, indicating they were printed on the same line.

Si queremos que ambas palabras se separen deberíamos dejar un espacio al final de Hola o al principio de Mundo.



Mensajes por consola: println

- La clase System.out nos permite mostrar mensajes por consola. Para ello, podemos utilizar varios métodos:
 - Print: no realiza ningún salto de línea. Por lo que si utilizamos varias veces dicha instrucción, los mensajes se mostrarán en la misma línea

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         System.out.println("Hola");  
4         System.out.println("Mundo");  
5     }  
6 }
```

The screenshot shows an IDE interface with two tabs: 'HelloWorld.java' and 'Console'. The code in 'HelloWorld.java' is as follows:

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         System.out.println("Hola");  
4         System.out.println("Mundo");  
5     }  
6 }
```

The 'System.out.println("Hola");' line is highlighted with a red box and has a red arrow pointing to the word 'Hola' in the 'Console' tab. The 'System.out.println("Mundo");' line is highlighted with a green box and has a green arrow pointing to the word 'Mundo' in the 'Console' tab. The 'Console' tab shows the output: 'Hola' and 'Mundo' on separate lines.



Mensajes por consola: System.err

- La clase `System` nos permite mostrar mensajes por consola. Para ello, podemos utilizar varios métodos:
 - `System.err` también nos permite trabajar con los métodos `print`, `println` y `printf` sin problemas.
 - Aunque, de este último método (`printf`), hablaremos más detenidamente más adelante.

The screenshot shows an IDE interface with two windows. The top window is titled "HelloWorld.java" and contains the following Java code:

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         System.err.println("Este tipo de mensajes aparecen en la consola en rojos y nos sirven para mostrar errores");  
4     }  
5 }
```

The bottom window is titled "Console" and shows the output of the program. The text "Este tipo de mensajes aparecen en la consola en rojos y nos sirven para mostrar errores" is displayed in red, indicating it was printed via `System.err.println`.



Mensajes por consola: trace

- La clase System.out nos permite mostrar mensajes por consola. Para ello, podemos utilizar varios métodos:
□ System.out también nos permite trabajar con los métodos trace que simplemente es un System.out.println que imprime la clase y el método que estamos utilizando. Como una especie de log para identificar que hemos pasado por dicho método.

The screenshot shows an IDE interface with a code editor and a terminal window. The code editor displays a file named Main.java with the following content:

```
Main.java
1 public class Main {
2     public static void main(String[] args) {
3         System.out.println("Main.main()");
4     }
5 }
```

The line `System.out.println("Main.main()");` is highlighted in blue. Below the code editor is a terminal window titled "Console". The terminal shows the output of the program's execution:

```
<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (18 abr.)
Main.main()
```



Mensajes por consola: tip/truquito

- Cuando trabajamos con Eclipse, y queremos mostrar la clase System.out, System.err, tenemos una forma reducida de llamar a dicha clase. Para ello, escribimos sysout, syserr o systrace. O inclusive, bastaría con escribir sys y pulsas control espacio:

The screenshot shows the Eclipse IDE interface. In the top-left corner, there is a tab labeled "Main.java ×". Below it, the code editor displays the following Java code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3     }  
4 }  
5  
6
```

At the bottom of the screen, there is a toolbar with several icons: "Problems", "Javadoc", "Declaration", and "Console ×". The "Console" icon is highlighted with a blue border. To the right of the toolbar, the text "No consoles to display at this time." is visible. On the far right edge of the window, the number "7" is displayed.

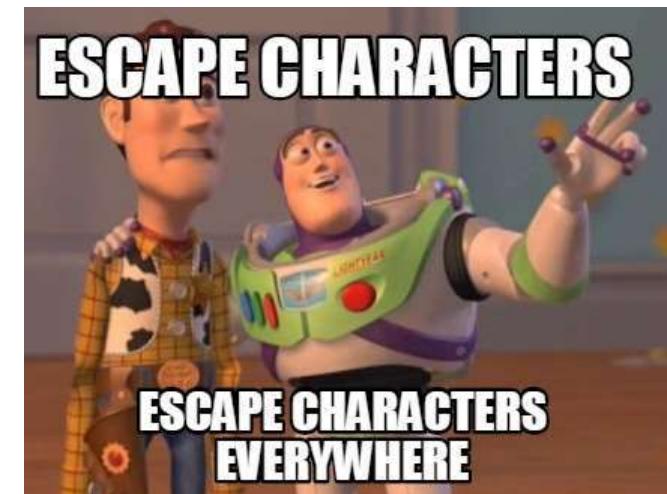
SECUENCIAS DE ESCAPE





Secuencias de escape

- Las secuencias de escape son caracteres especiales que se pueden añadir/introducir dentro de una cadena de texto con la finalidad de representar a una determinada acción sobre dicha cadena de texto que de otra forma sería muy difícil de representar.
 - Algunos ejemplos son: tabulaciones, retornos de carro, saltos de línea, etc.
- Para utilizar las secuencias de escape utilizamos la barra invertida (\) la cual irá acompañada del carácter de escape que queremos utilizar.





Secuencias de escape

- Dentro de las secuencias de escape de Java tenemos las siguientes:

Secuencia de Escape	Descripción
\n	Salto de línea
\t	Tabulador
\\\	Diagonal Inversa \
\"	Comillas Dobles
'	Comilla Simple
\r	Retorno de Carro (Solo en modo Administrador)
\b	Borrado a la Izquierda (Solo en modo Administrador)

- Por ejemplo, para realizar un salto de línea en una cadena de texto, realizaríamos lo siguiente:

The screenshot shows an IDE interface with two windows. On the left, the code editor window titled "HelloWorld.java" contains the following Java code:

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         // TODO Auto-generated method stub  
4         System.out.print("Hello\nWorld!");  
5     }  
6 }
```

A red arrow points from the "\n" character in the code to the resulting output in the console window on the right. The console window is titled "Console" and shows the text "Hello" on one line and "World!" on the next line.



Ejercicio de secuencias de escape

- Basándote en el ejemplo que hemos realizado y apoyándote en la siguiente tabla, realiza un ejemplo en el que puedas demostrar un uso de cada una de las distintas secuencias de escape.

The screenshot shows an IDE interface with two tabs: 'HelloWorld.java' and 'Console'. The code in 'HelloWorld.java' is:

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         // TODO Auto-generated method stub  
4         System.out.print("Hello\nWorld!");  
5     }  
6 }
```

The output in the 'Console' tab is 'Hello World!', with a red arrow pointing from the '\n' character in the code to the new line in the output.

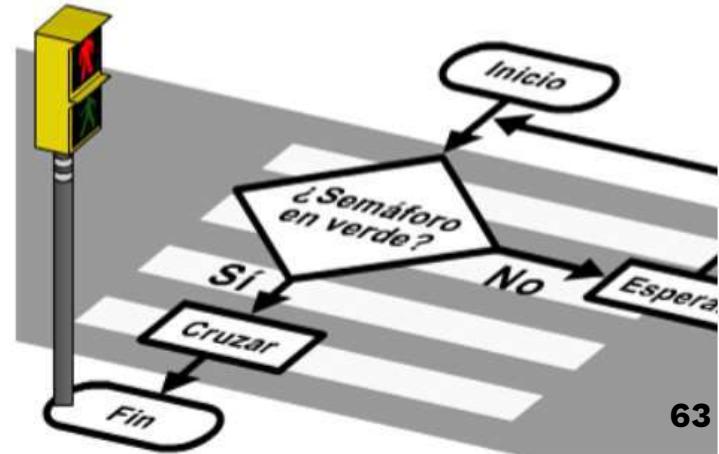
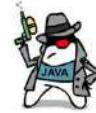
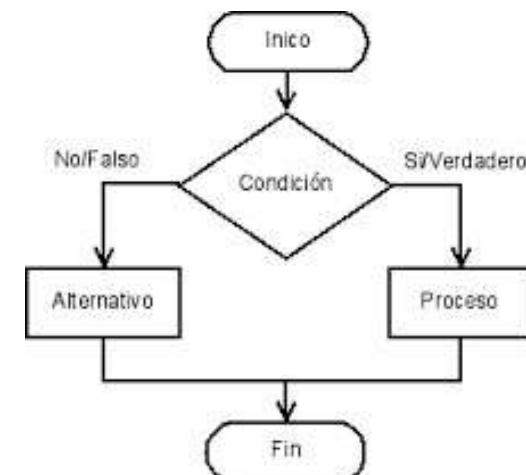
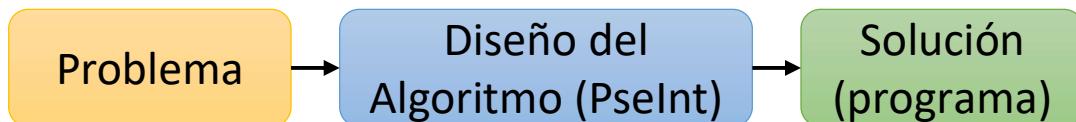
Secuencia de Escape	Descripción
\n	Salto de línea
\t	Tabulador
\\\	Diagonal Inversa \
\"	Comillas Dobles
'	Comilla Simple
\r	Retorno de Carro (Solo en modo Administrador)
\b	Borrado a la Izquierda (Solo en modo Administrador)

¿QUÉ ES UN ALGORITMO?



¿Qué es un algoritmo?

- Una algoritmo es una sucesión de secuencias/serie de pasos que nos permitirán realizar resolver una cierta tarea.
- Si bajamos el concepto al mundo real, por ejemplo, al mundo de la cocina, el algoritmo es la sucesión de pasos (receta) mediante a los cuales realizamos la tarea/elaboración (el bizcocho).



63

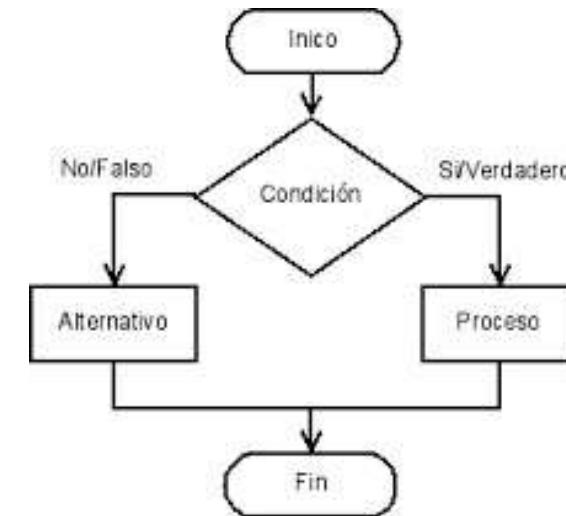
Algoritmo que seguimos para cruzar un paso de cebra



Características de un algoritmo

Algunas características de los algoritmos son:

- Preciso: tiene que resolver problemas sin errores.
- Definido: si ejecutamos el algoritmo varias veces, con los mismos datos, los datos de salida serán iguales en cada repetición.
- Finito: Debe tener inicio y fin.
- Legible: Cualquier persona que vea el algoritmo debe ser capaz de comprenderlo.



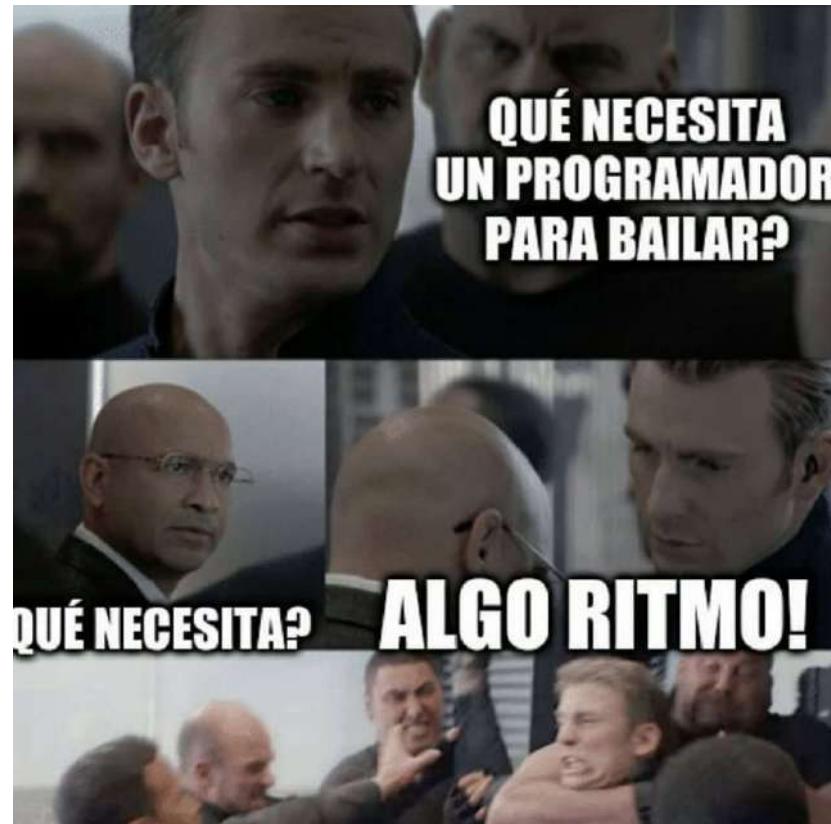


Estructura de un algoritmo

La estructura de un algoritmo, está formada por:

- - Entrada: datos que se le asignan al algoritmo.
 - Por ejemplo: int num1=1; int num2 = 2;
- - Proceso: Operaciones que se hacen con dichos datos.
 - Por ejemplo: int suma = num1+num2;
- - Salida: resultado que se obtiene al finalizar las operaciones.
 - Por ejemplo: mostrar por pantalla el resultado con
System.out.println(suma);

Estructura de un algoritmo





Ejercicio

- Mediante a comentarios al final de las instrucciones, define las distintas partes del siguiente algoritmo:

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         int num1 = 10;  
4         int num2 = 20;  
5         int suma = num1 + num2;  
6         System.out.println(suma);  
7     }  
8 }
```

Console > terminated > 30

Graphical Representation of Three Variables with values in memory-RAM

The memory diagram shows three variables: num1, num2, and suma. Each variable is represented by a vertical stack of memory cells. num1 contains the value 10, num2 contains 20, and suma contains 30. The memory cells are shown as vertical bars of different heights, with the total height representing the value of the variable.

VARIABLES

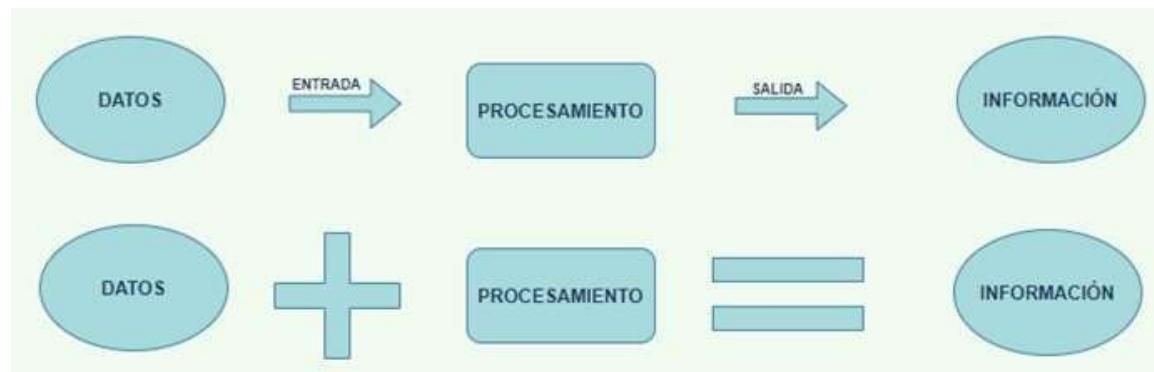


DIFERENCIAS ENTRE DATOS, INFORMACIÓN Y CONOCIMIENTO



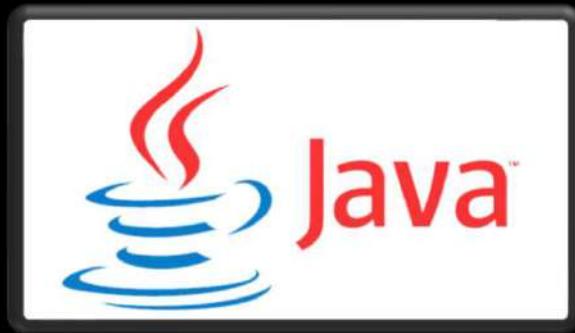
Datos, información y conocimiento

- Aquí os dejo la pirámide del conocimiento y la diferencia entre dato e información.



- Finalmente, la sabiduría es cuando nuestro conocimiento se aplica en el día a día.

¿QUÉ ES UNA VARIABLE?



¿Qué es una variable?

- Cualquier programa, desde los más simples, hasta los más complejos, necesitan almacenar y manipular datos. Para ello, usualmente se utilizan las variables, las cuales “escriben” sobre un espacio que reservamos en una memoria (usualmente la RAM) para almacenar dichos datos.
- Algunos ejemplos de ello son:
 - El resultado de una llamada a una API
 - El resultado de consulta a una BBDD
 - Un input (entrada) que ha realizado el usuario en nuestro programa, etc.
 - El guardar el nombre de un usuario en un juego
- Las variables, por tanto, son “una especie de cajas”, que nos permiten de forma temporal almacenar datos en su interior.

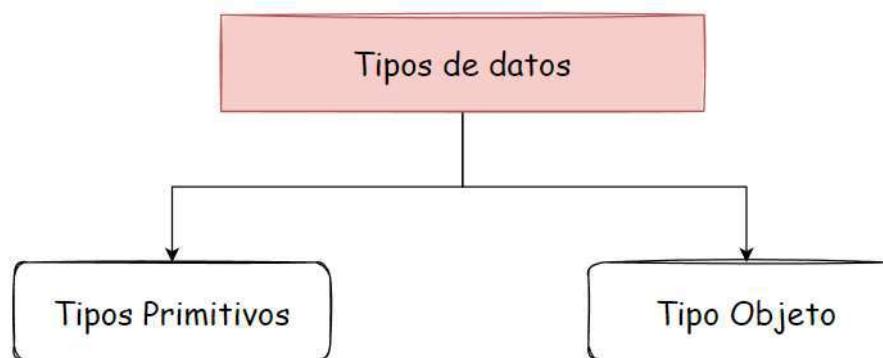


CLASIFICACIÓN DE TIPOS

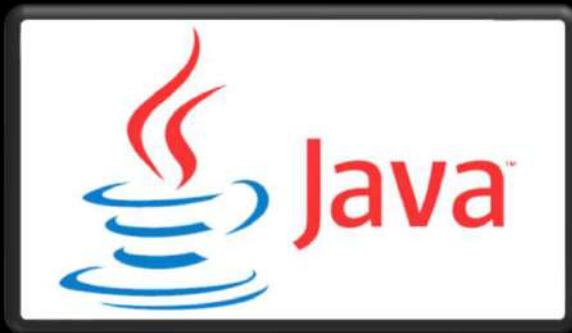


Clasificación de tipos

- Los datos se subdividen principalmente en dos grandes grupos:
 - Tipos Primitivos: contiene tipos de datos básicos como un solo carácter, números enteros, números decimales, condicionales...
 - Tipo objeto: todo lo que no es considerado primitivo es un objeto.
- Pese a ello, para java todo es un objeto, pero nosotros hacemos esta distinción/clasificación en la que diferenciamos a los tipos primitivos de los objetos.

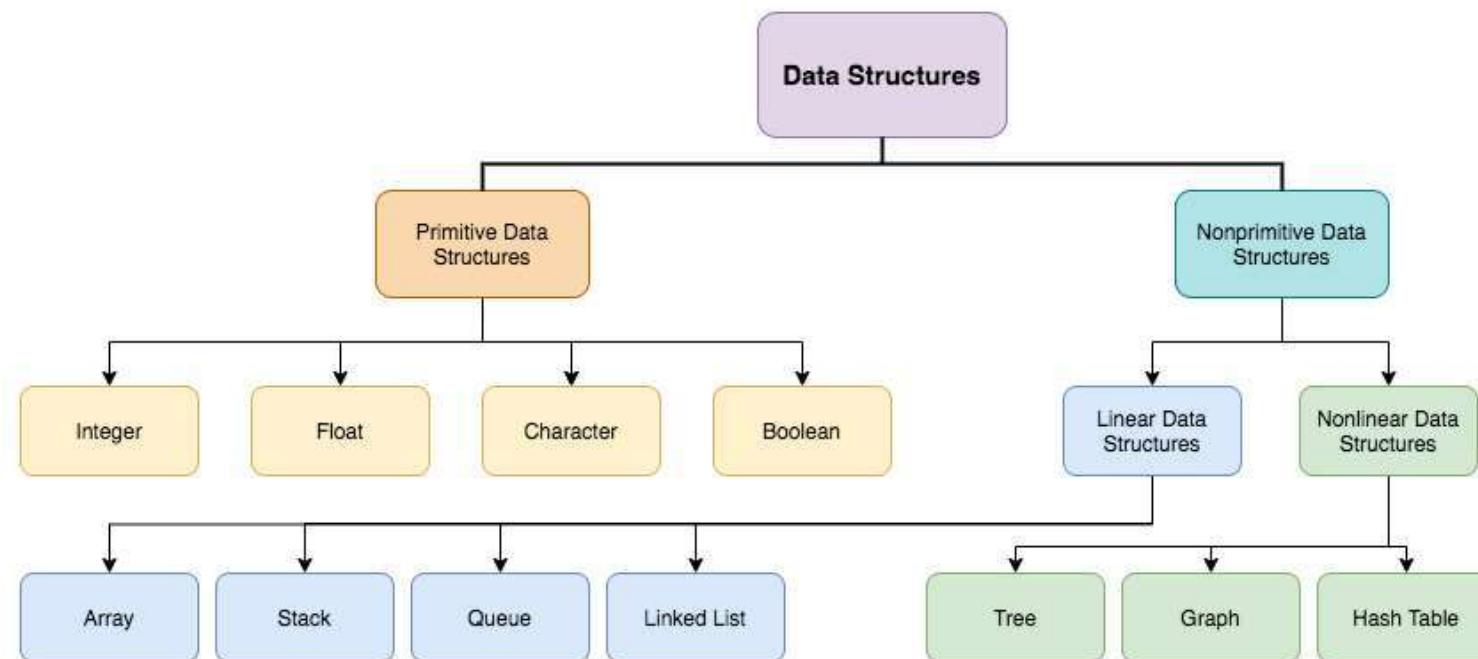


¿QUÉ SON LAS ESTRUCTURAS DE DATOS?



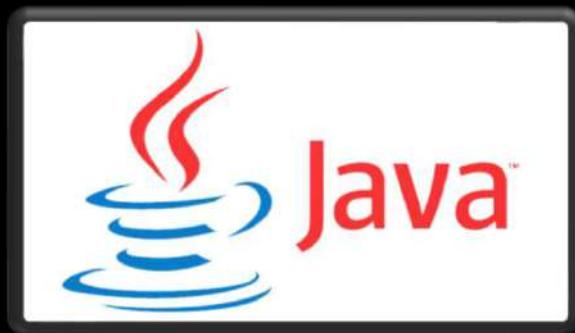
Clasificación de las estructuras de datos

- Las estructuras de datos, son las distintas formas mediante a las cuales Java nos permite almacenar datos que usaremos durante la ejecución de nuestros programas. Vamos a ver un esquema donde se detallan los distintos tipos:



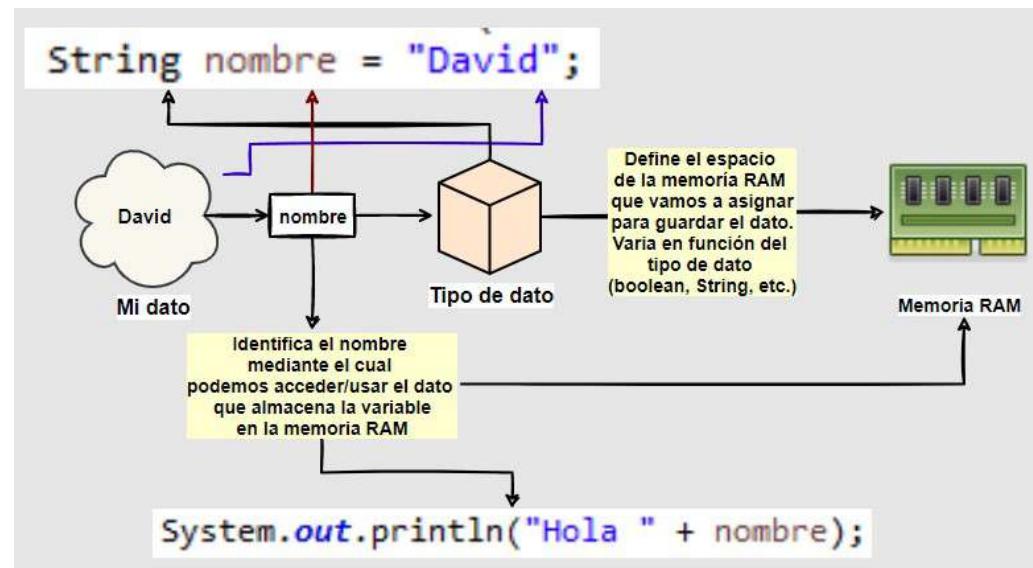


EJEMPLO DE USO DE UNA VARIABLE



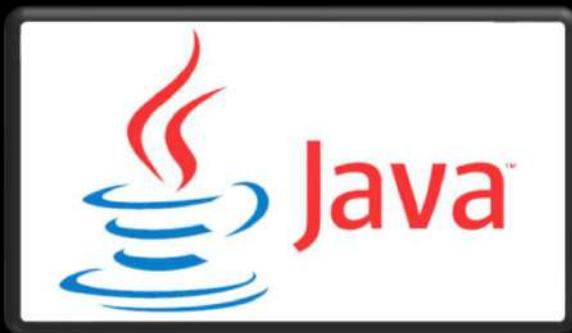
Ejemplo de uso de una variable

- Un ejemplo de uso de una variable podría ser el siguiente:
- En este caso, estamos trabajando sobre la clase String, para formar una cadena de texto (tipo objeto) que internamente trabajará con varios datos primitivos de tipo char para formar dicha cadena de texto.



A screenshot of a Java IDE showing the `HelloWorld.java` file and its execution output. The code is:1 public class HelloWorld {
2 public static void main(String[] args) {
3 String nombre = "David";
4 System.out.println("Hola " + nombre);
5 }
6 }The output in the 'Console' window is: `Hola David`. The line `String nombre = "David";` and the println statement are highlighted with red boxes and green arrows pointing to the 'Hola David' output.

DESENGRANANDO LA DECLARACIÓN DE VARIABLES



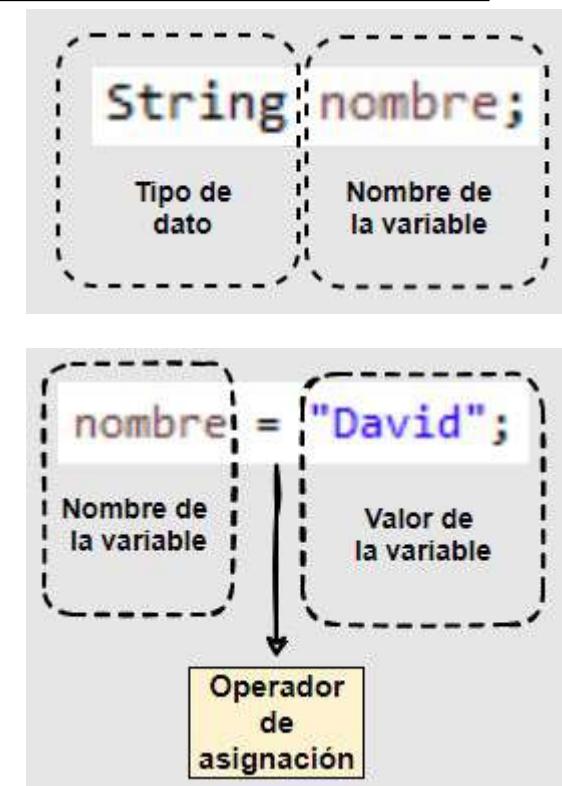
Declaración y asignación de una variable

- La declaración de una variable es la parte en la que definimos.

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         String nombre; //Declaración  
4     }  
5 }
```

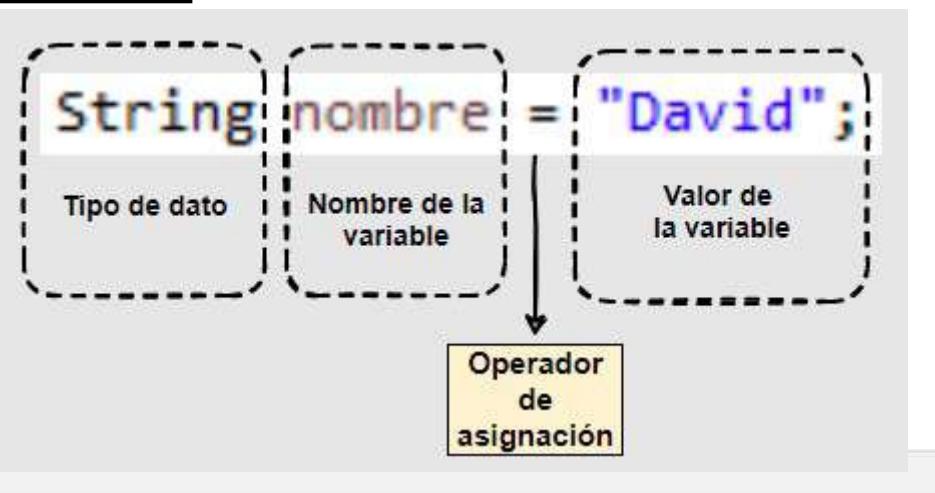
- Una vez declarada una variable, ya podemos asignar un valor a dicha variable. Vamos a verlo:

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         String nombre; //Declaración  
4         nombre = "David"; //Asignación  
5     }  
6 }
```



Declaración y asignación de una variable en una misma instrucción

- Existe la posibilidad de realizar la declaración y la asignación en una misma instrucción. Vamos a ver un ejemplo:



```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         String nombre = "David"; //Declaración + asignación  
4     }  
5 }
```

Uso de una variable

- Ya hemos definido una variable y asignado un valor. Ahora, solamente nos faltaría utilizar dicha variable. Vamos a ver un ejemplo:

The screenshot shows an IDE interface with a code editor and a terminal window.

Code Editor (HelloWorld.java):

```
String nombre = "David", apellidos = "Bernal González";
System.out.println(nombre + " " + apellidos);
```

A callout box points from the line `System.out.println(nombre + " " + apellidos);` to the **Console** tab, with the text **Mostrando el valor almacenado en una variable**.

Console Tab:

```
<terminated> Main [Java Application]
David Bernal González
```

Code Editor (HelloWorld.java):

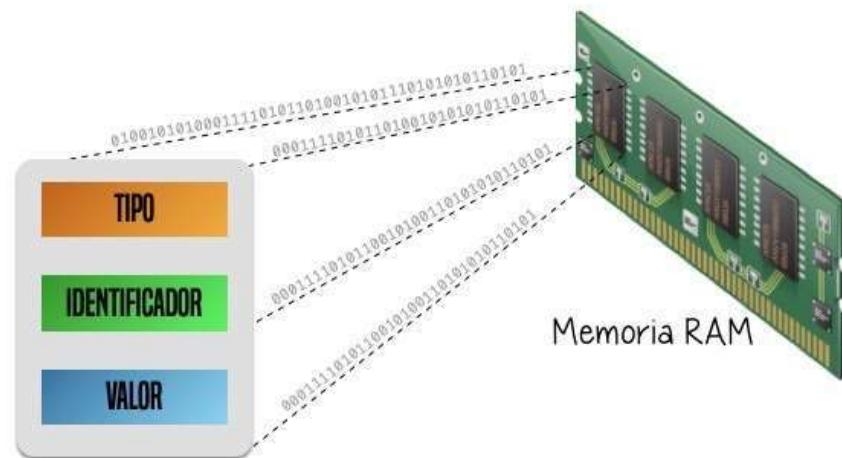
```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         String nombre = "David"; // Declaración + asignación
4         System.out.println("Hola "+ nombre + "!"); // Uso de la variable
5     }
6 }
```

Console Tab:

```
<terminated> HelloWorld [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (17 abr. 2022 22:00:45 – 22:00:45) [pid: 20012]
Hola David!
```

83

Uso de una variable



```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         String nombre = "David"; // Declaración + asignación  
4         System.out.println("Hola " + nombre + "!"); // Uso de la variable  
5     }  
6 }
```

84

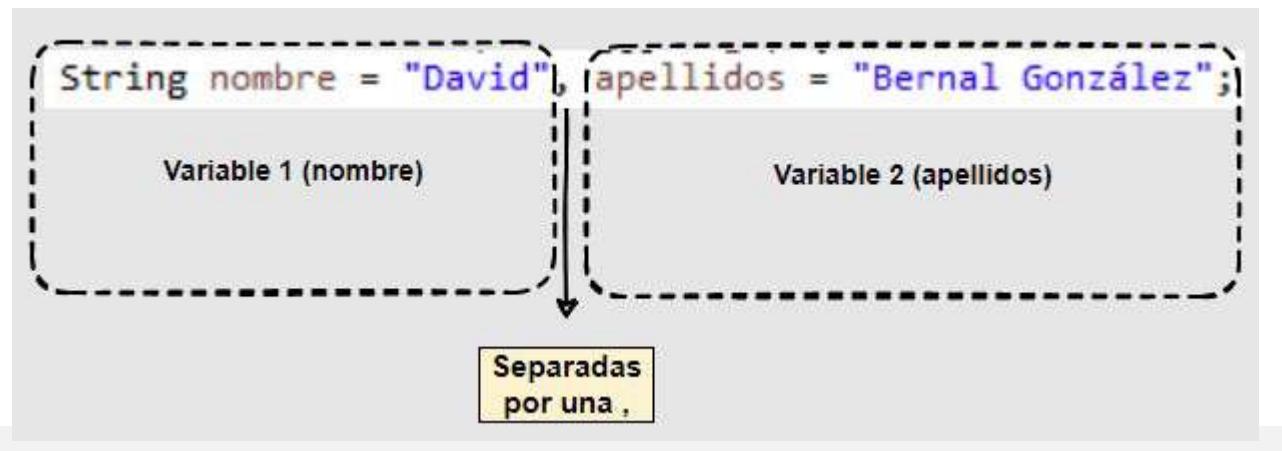
Problems Javadoc Declaration Console X

<terminated> HelloWorld [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (17 abr. 2022 22:00:45 – 22:00:45) [pid: 20012]

Hola David!

Declaración y asignación de varias variables en una misma instrucción

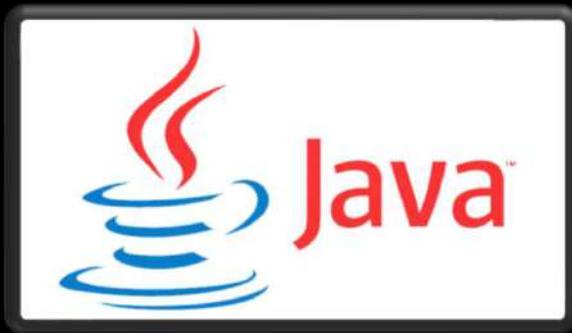
- También, existe la posibilidad de realizar la declaración y la asignación de varias variables en una misma instrucción. Vamos a ver un ejemplo:



HelloWorld.java ×

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         //Declaración + asignación de varias variables  
4         String nombre = "David", apellidos = "Bernal González";  
5     }  
6 }
```

¿PORQUE LAS VARIABLES RECIBEN DICHO NOMBRE?



¿Porque se llaman variables?

- El motivo por el que las variables reciben dicho nombre es porque pueden cambiar a lo largo del flujo del programa.
Vamos a ver un ejemplo:

The screenshot shows a Java code editor with a file named `HelloWorld.java`. The code defines a `HelloWorld` class with a `main` method. Inside the `main` method, the variable `nombre` is initially set to "Desconocido". It is then printed to the console. After a comment, its value is modified to "David", and it is printed again. The output window shows the two println statements and their corresponding outputs: "En este punto, la variable nombre tiene un valor de: Desconocido" and "En este punto, la variable nombre tiene un valor de: David".

```
String nombre = "Desconocido";
System.out.println(nombre);
```

Valor inicial de la variable

```
nombre = "David";
System.out.println(nombre);
```

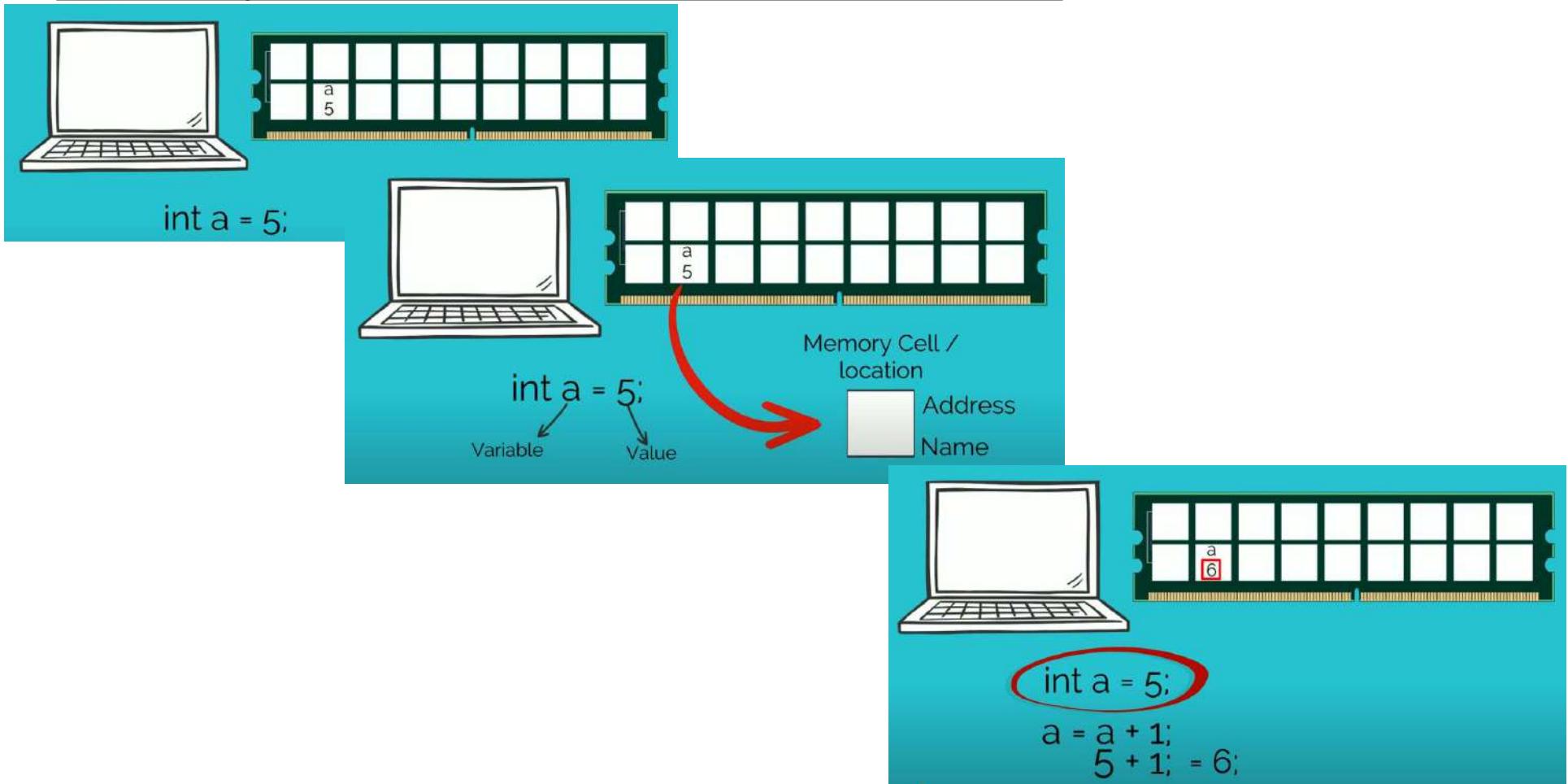
Nuevo valor de la variable

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         // Valor inicial de la variable nombre
4         String nombre = "Desconocido";
5         System.out.println("En este punto, la variable nombre tiene un valor de: " + nombre);
6         // Modificación del valor de la variable nombre
7         nombre = "David";
8         System.out.println("En este punto, la variable nombre tiene un valor de: " + nombre);
9     }
10 }
```

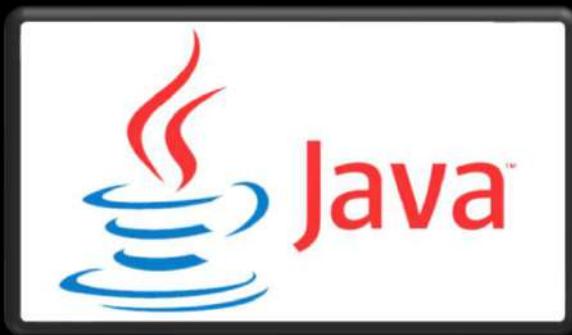
Problems Javadoc Declaration Console <terminated> HelloWorld [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (17 abr. 2022 21:57:39 - 21:57:39) [pid: 24372]

En este punto, la variable nombre tiene un valor de: Desconocido
En este punto, la variable nombre tiene un valor de: David

¿Porque se llaman variables?



¿DÓNDE SE ALMACENAN REALMENTE LAS VARIABLE?



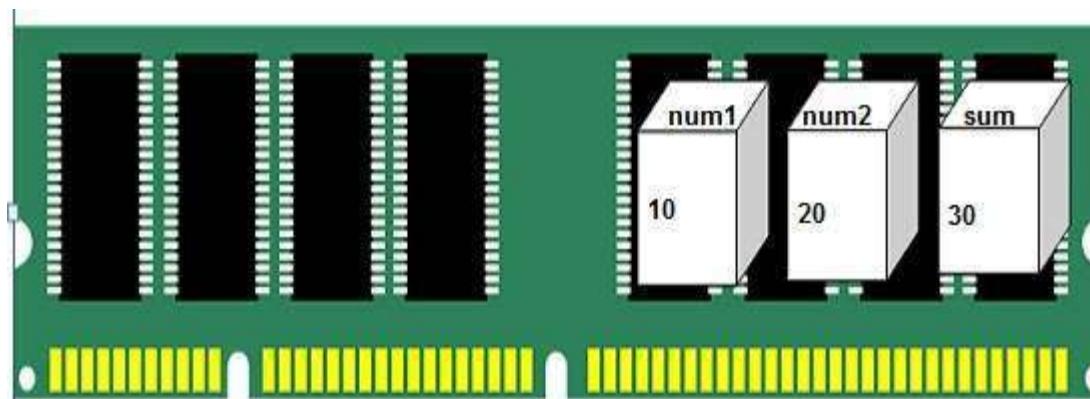
¿Donde se almacena una variable?

- Estás “especie de cajas” en las que almacenamos los datos, son pequeños espacios en memoria que reservamos dentro de la memoria volátil (RAM, ROM...). Habitualmente, la memoria volátil más utilizada suele ser la memoria RAM.
- La principal diferencia entre una memoria volátil y no volátil, es que, en las memorias volátiles cuando finalicemos la ejecución del programa, sistema operativo, etc., ya sea porque ha finalizado la ejecución de un programa, porque se ha interrumpido el suministro eléctrico, etc., perderemos el contenido almacenado en dichas. En cambio, en las memorias no volátiles (discos duros, cds...) si por ejemplo almacenamos una imagen, esa imagen en principio si no se ha dañado la unidad, debería ser accesible tras un reinicio o un corte eléctrico.



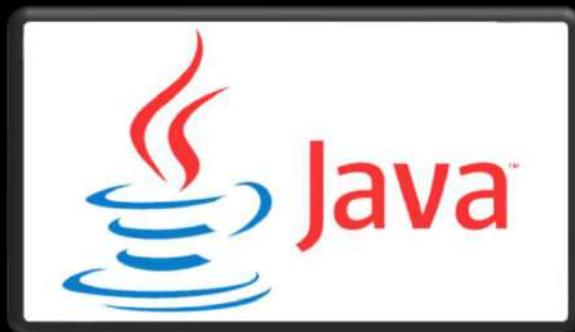
¿Donde se almacena una variable?

- Las variables, es decir, “esta especie de cajas” en las que almacenamos los datos, son espacios que reservamos en las memorias volátiles y no siempre tendrán el mismo tamaño. Como pasa con las cajas en la vida real, en función de lo que queremos almacenar en su interior, en este caso del tipo de dato que queremos utilizar en la variable (“esta especie de cajas”), utilizaremos un determinado tipo de caja (dato) u otra.



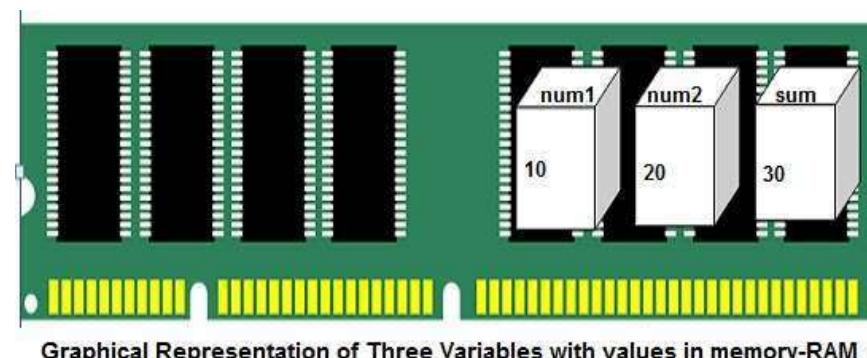
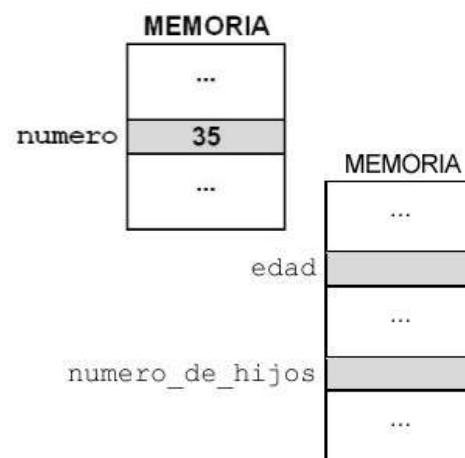
Graphical Representation of Three Variables with values in memory-RAM

DIFERENCIAS ENTRE “CAJA” Y TIPOS DE “CAJA”

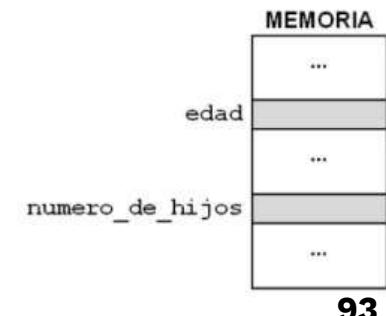


caja=variable, tipos de cajas=tipo de dato

- Las variables, es decir, “esta especie de cajas” en las que almacenamos los datos, son espacios que reservamos en las memorias volátiles; y no siempre tendrán el mismo tamaño.
- Como pasa con las cajas en la vida real, en función de lo que queremos almacenar en su interior, en este caso del tipo de dato que queremos utilizar en la variable (“esta especie de cajas”), utilizaremos un determinado tipo de caja (dato) u otra.
 - Por ejemplo, no será igual la caja utilizar para guardar un número 123235347897349867243298 que un carácter.

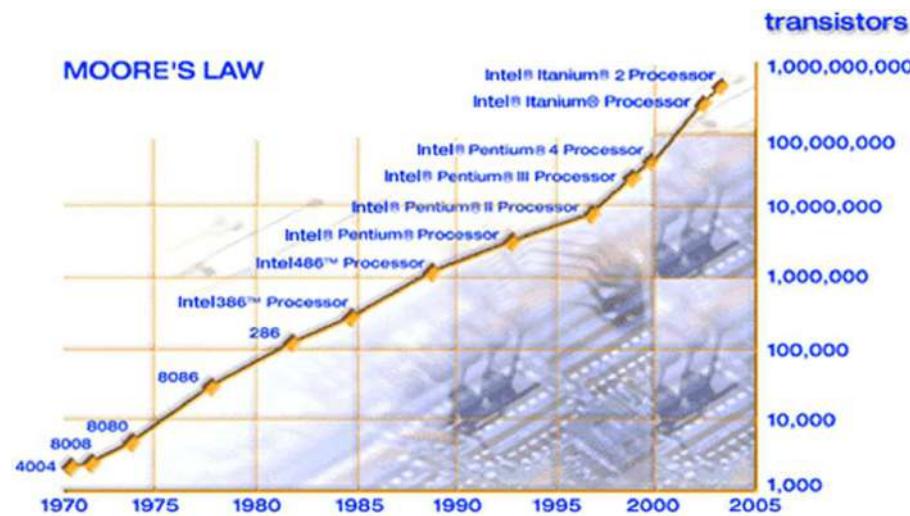


– Las variables no tienen por qué estar contiguas en la memoria.



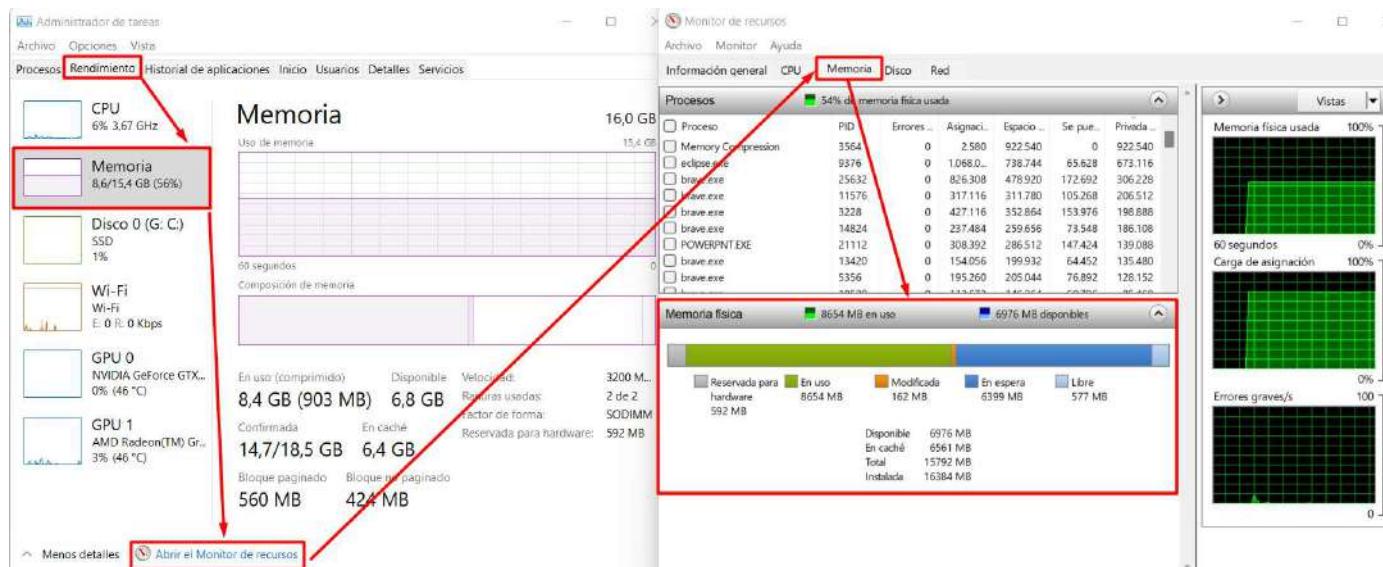
Tipos de “caja” (=tipos de datos)

- Tal y como explica la famosa "Ley de Moore" los velocidad en la que se han producido avances en el mundo de la informática han provocado que la capacidad de procesamiento del hardware y el tamaño de memorias haya aumentado hasta tal punto en que ya prácticamente no hay problemas de almacenamiento.

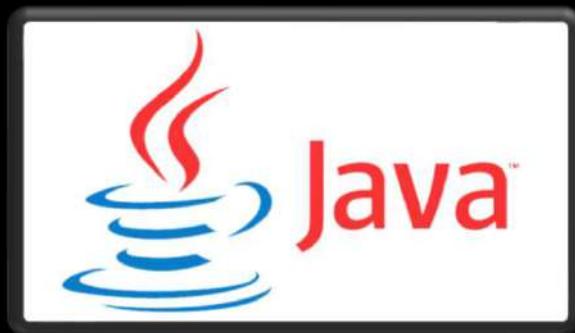


Tipos de “caja” (=tipos de datos)

- Pese a ello, hay que tener en cuenta que cada variable cuenta, y hay que ser consciente con ello. Ya que, estás ocupan un cierto espacio en nuestra memoria y habría que intentar coger la que mejor se adapte a dicho dato.
- Para saber el uso que estamos utilizando de la memoria (volátil) RAM, en nuestro sistema pulsamos CONTROL + ALT + SUPRIMIR, vamos a administrador de tareas y hacemos lo siguiente:

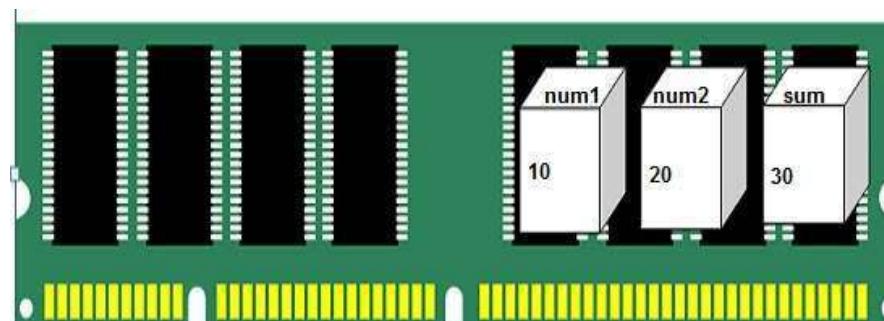
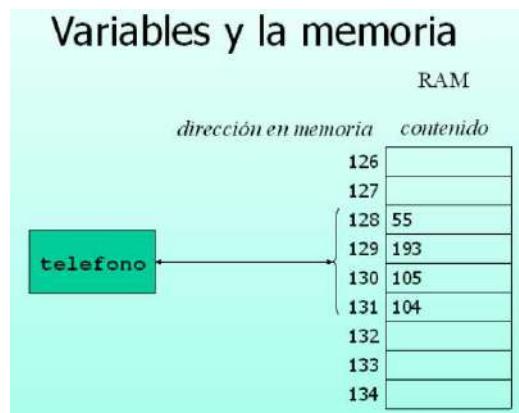


¿QUÉ SON LOS IDENTIFICADORES?



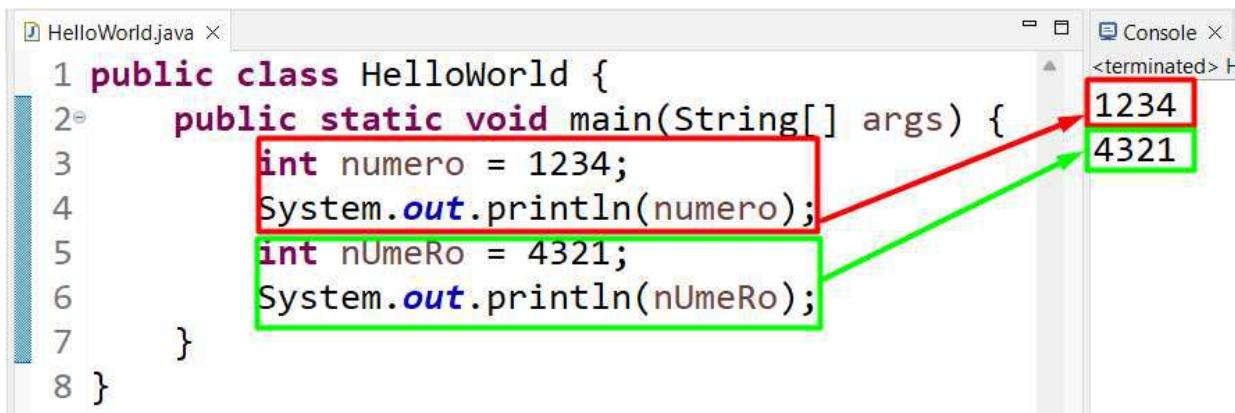
¿Qué es un identificador? ¿Para que sirve?

- Un identificador es un nombre mediante el cual asociamos una variable (o un método, una clase, etc) de nuestro código hacia un espacio de memoria.
- Gracias a los identificadores, podemos acceder/escribir sobre una o varias direcciones que hemos reservado en nuestra memoria volátil.
 - Por ejemplo, en este caso, la variable teléfono, ha reservado 4 direcciones en memoria en las que ha almacenado el teléfono: 55193105104
 - En el segundo ejemplo, hemos declarado tres variables num1, num2 y sum.



Características de los identificadores

- Algunas características de los identificadores son:
 - Java es KeySensitive, es decir, distingue mayúsculas y minúsculas. Por lo que, no será lo mismo si un identificador contiene solamente minúsculas que si contiene mayúsculas pese a que tengan el mismo nombre:



```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         int numero = 1234;
4         System.out.println(numero);
5         int nUmeRo = 4321;
6         System.out.println(nUmeRo);
7     }
8 }
```

- Los identificadores deben comenzar con una letra, o un símbolo de dólar \$ o una barra baja _

Características de los identificadores

- Los identificadores deben comenzar con una letra, o un símbolo de dólar \$ o una barra baja _ . El resto de letras pueden ser letras, números, etc, aunque se aconseja el no utilizar ñ, letras con tildes, etc.
 - Vamos a ver unos ejemplos:

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         int numero = 1234;
4         System.out.println(numero);
5         int $numero = 4321;
6         System.out.println($numero);
7         int _numero = 4312;
8         System.out.println(_numero);
9     }
10 }
```

Console X
<terminated>
1234
4321
4312

- En cambio, si intentamos utilizar caracteres que no son admitidos vemos que no aparece una X en roja junto al número de línea y nos subraya la parte que no le gusta al IDE. De hecho, si nos podemos encima, inclusive nos dice que eliminamos los números o los caracteres que no

99

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         int 123numero = 1234;
4         int @asdf;
5     }
6 }
```

Syntax error on token "@", delete this token
Press F2 for focus

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         int numero = 1234;
4         int asdf;
5     }
6 }
```

Características de los identificadores

- En el caso anterior, hemos visto que aparecen una lightbulbs (“bombillitas”) junto al número de instrucción.
- Si nos ponemos encima nos aparecerá un mensaje de “The value of the local variable is not used”.
- Si hacemos click en la bulb, vemos que nos aparecen algunas maneras de solucionar el problema

The screenshot shows two instances of an IDE interface. The top instance displays the Java code:

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         int numero = 1234;  
4         int asdf;  
5     }  
6 }
```

The line "int asdf;" has a red square icon next to the number 4 in the margin. The bottom instance shows the same code, but the line "int numero = 1234;" has a red square icon next to the number 3 in the margin. A tooltip appears over the icon at line 3, reading "The value of the local variable numero is not used". Below the code, a context menu is open at the tooltip location, listing several options:

- Remove 'numero' and all assignments
- Remove 'numero', keep assignments with side effects
- Change type of 'numero' to 'var'
- Convert local variable to field
- Inline local variable
- Rename in file (Ctrl+2, R)
- Rename in workspace
- Split variable declaration
- Add @SuppressWarnings 'unused' to 'numero'
- Add @SuppressWarnings 'unused' to 'main()'
- Configure problem severity

Características de los identificadores

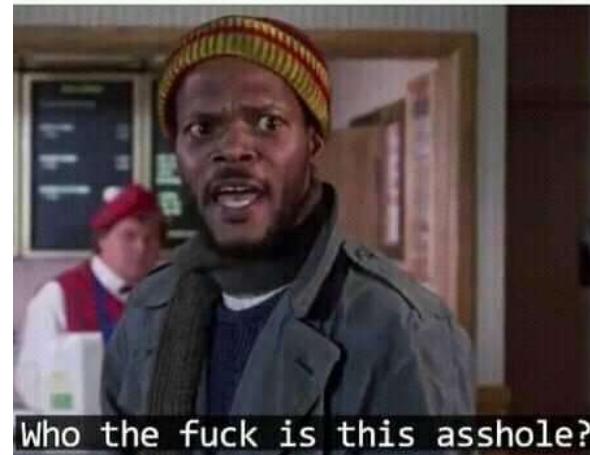
- Este tipo de bulb de advertencia no son un error sino bulbs de advertencia que nos muestra el IDE (Eclipse en este caso) en la que nos informa que no tiene sentido ocupar/reservar un espacio en memoria y asignar un valor un valor si después no lo vamos a utilizar. Lo único que estamos haciendo es consumir recursos “a lo tonto”. Pero, pese a ello, el programa compilará sin problema. Explicaremos estos bulbs más a detalle más adelante.

error	
warning	
alert	

A screenshot of the Eclipse IDE interface. On the left, there's a small window titled "error" with three rows: "error" (red X), "warning" (yellow exclamation mark), and "alert" (yellow exclamation mark). To the right, a Java file named "HelloWorld.java" is open. The code contains two lines of code: "public class HelloWorld" and "public static void". Below the code, a tooltip from a yellow warning bulb says: "The value of the local variable numero is not used".

Me: *Introduces new variable and never uses it*

Compiler:



Cuando declaras una variable, y nunca la utilizas



Variable name convection

- Existen muchas convenciones de nombres para utilizar en los identificadores de nuestras variables, métodos, clases, etc.



snake_case

Pros: Concise when it consists of a few words.
Cons: Redundant as hell when it gets longer.
`push_something_to_first_queue, pop_what, get_whateve`



PascalCase

Pros: Seems neat.
`GetItem, SetItem, Convert, ...`
Cons: Barely used. (why?)



camelCase

Pros: Widely used in the programmer community.
Cons: Looks ugly when a few methods are n-word
`push, reserve, beginBuilding, ...`



skewer-case

Pros: Easy to type.
`easier-than-capitals, easier-than-underscore, ...`
Cons: Any sane language freaks out when you try it.



SCREAMING_SNAKE_CASE

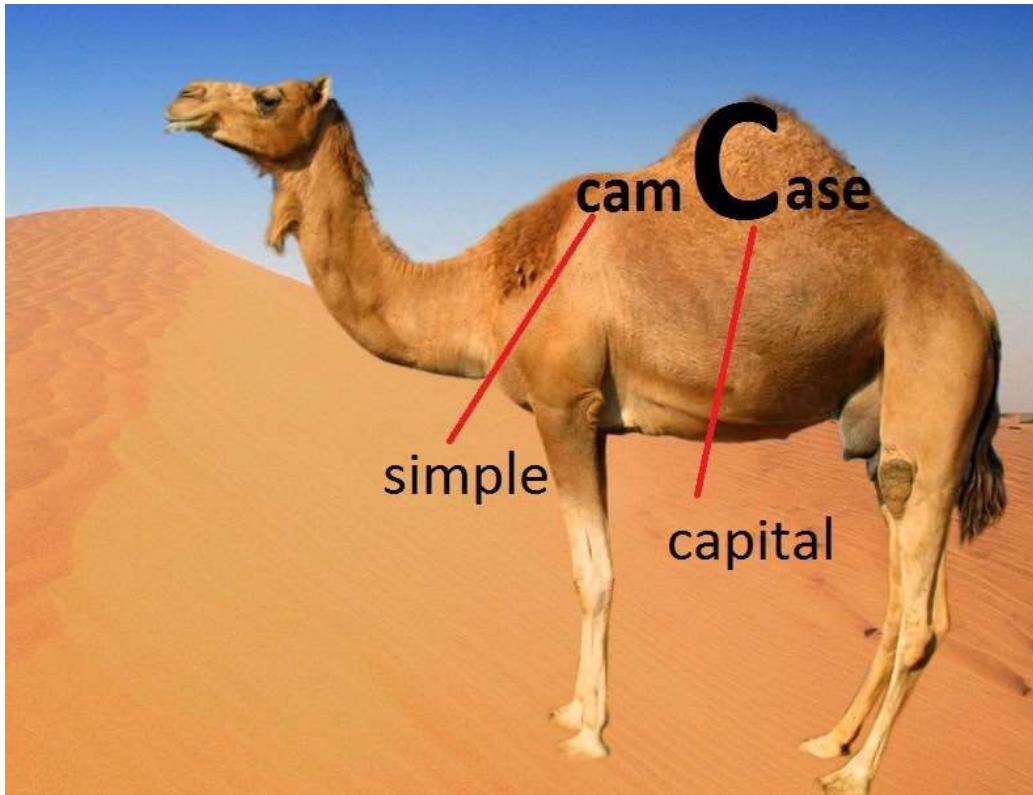
Pros: Can demonstrate your anger with text.
Cons: Makes your eyes deaf.
`LOOK_AT_THIS, LOOK_AT_THAT, LOOK_HERE_YOU_MORON, ...`



nocase

Pros: Looks professional.
Cons: Misleading af.
`supersexyhippotalamus, bool penisbig, ...`

Variable name convection



Variable name convection



Programmers:



Variable name convection



HOW I SPEND MY TIME:



This happens too often

Variable name convection

When I'm searching for a meaningful variable name



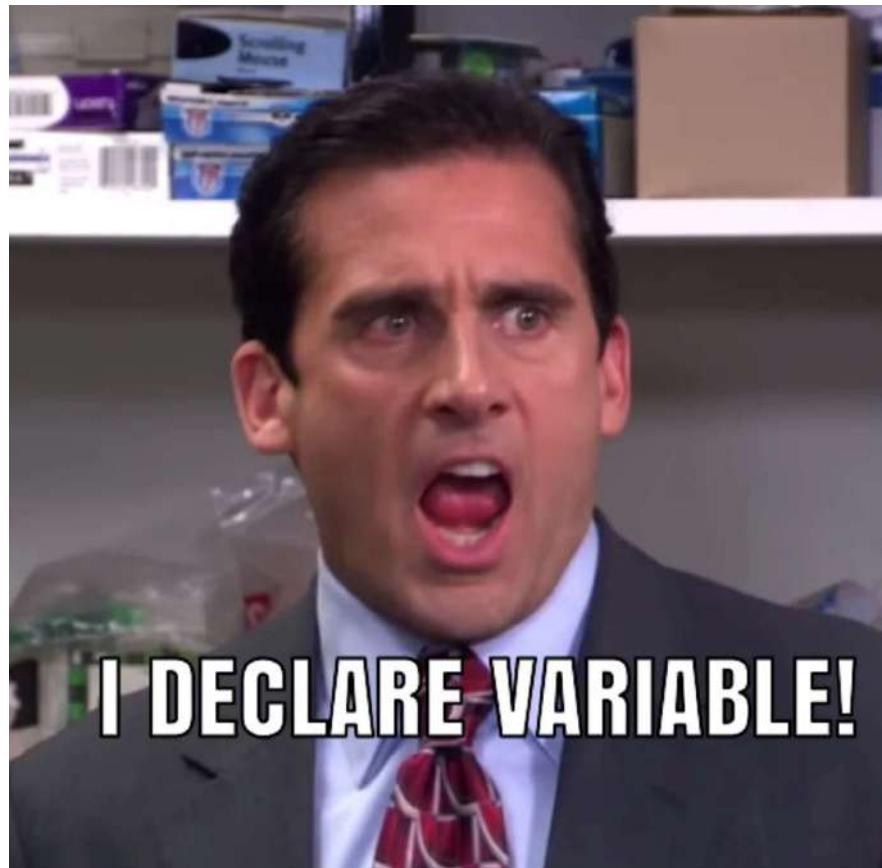
Variable name convection

En Java, las utilizadas usualmente son:

- Notación del camello: la primera palabra en minúscula y el primer carácter de la segunda en mayúscula.
 - Por ejemplo: nombreCompleto.
 - Se utiliza sobre funciones y métodos
- Notación pascal: todas las palabras tendrán la primera letra en mayúscula. El resto de letras en minúsculas.
 - Por ejemplo: NombreCompleto.
 - Se utiliza sobre clases, interfaces y namespaces
- Notación SCREAMING_SNAKE_CASE: todas las palabras van en mayúsculas al completo y deben ir separadas con una barra baja _ entre ellas.
 - Por ejemplo: NOMBRE_COMPLETO
 - Se utiliza sobre constantes.

Funciones y métodos	camelCase()
Clases	PascalCase()
Interfaces	PascalCase()
Namespaces	PascalCase()
Constantes	SCREAMING_SNAKE_CASE





ESTRUCTURAS DE DATOS PRIMITIVAS





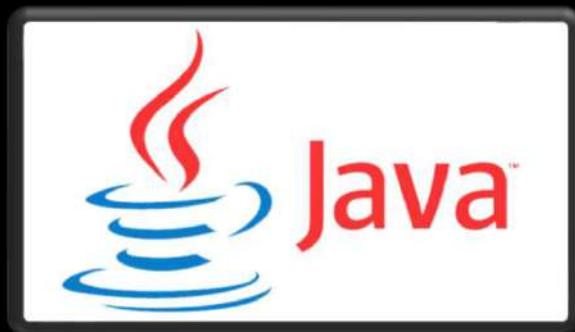
Primitive Data Types



Civilized Data Types

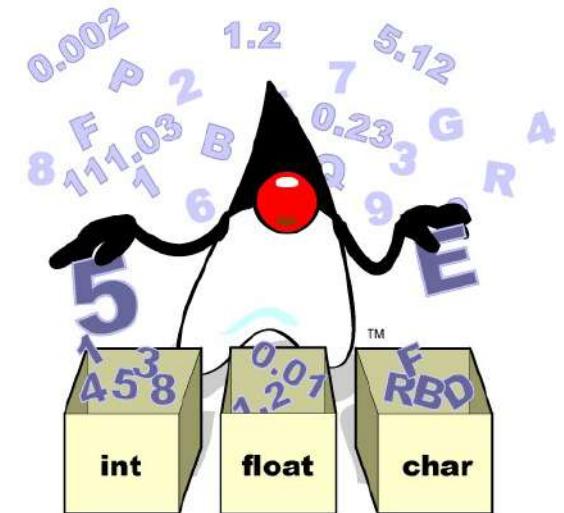
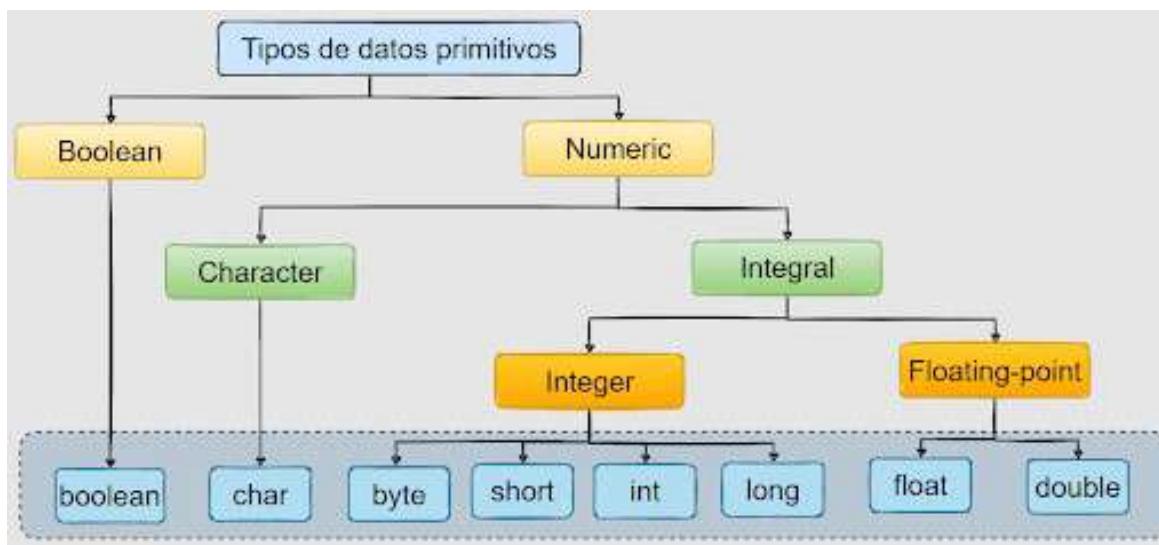


TIPOS DE DATOS PRIMITIVOS



Tipos de datos primitivos

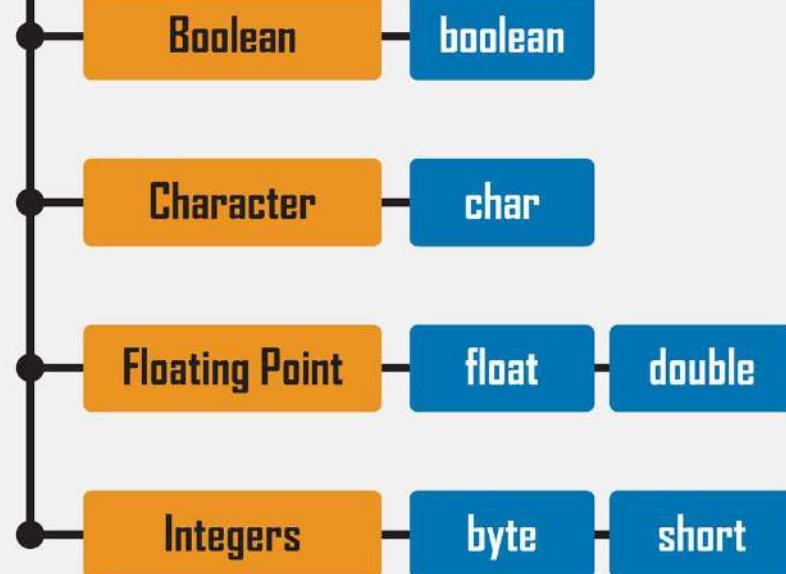
- En función del tipo de dato que queramos almacenar en una variable, utilizamos un tipo de dato u otro.
- Dentro de los datos primitivos en los que queremos centrarnos en este apartado, tenemos los siguientes tipos:



Java Data Types

DATA TYPES

Primitive



Non-Primitive/Reference

Annotations

Arrays

Strings

Classes

Interfaces

Enums

Objects

Tipos de datos primitivos

- Tabla resumen de tipos de datos primitivos:

Tipos de datos primitivos				
Nombre	Tipo	Tamaño	Valor por defecto	Rango
boolean	Lógico	1 bit	false	true-false
char	Carácter	16 bits	0	Unicode (del 0 al 65.535)
byte	Número entero	8 bits	0	-128 a 127
short	Número entero	16 bits	0	-32.768 a 32.767
int	Número entero	32 bits	0	-2.147.483.648 al 2.147.483.647
long	Número entero	64 bits	0	-9.223.372.036.854.775,808 a 9.223.372.036.854.775,807
float	Número real	32 bits	0.0	1.4E-45 a 3.4028235E38
double	Número real	64 bits	0.0	4.9E-324 a 1.7976931348623157E308

114

- Los datos amarillos son los genéricos dentro de los tipos numéricos enteros (int) y real (double)

Tipos de datos primitivos

- Tabla resumen de tipos de datos primitivos:

Java's primitive types

Ordered by descending upper range limit

CATEGORIZATION	NAME	UPPER RANGE	LOWER RANGE	BITS
Floating point	double	Really huge positive	Really huge negative	64
	float	Huge positive	Huge negative	32
Integral	long	9,223,372,036,854,775,807	-9,223,372,036,854,775,808	64
	int	2,147,483,647	-2,147,483,648	32
	char	65,535	0	16
	short	32,767	-32,768	16
	byte	127	-128	8
Boolean	boolean	No numeric equivalent	true or false	1

Tipos de datos primitivos

- Cuando acabemos de ver los tipos de datos, ya sabremos utilizar, es decir, añadiremos a nuestro repertorio todas estas palabras reservadas:

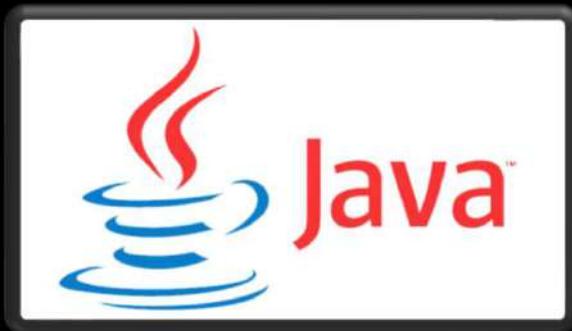
abstract	default	goto	package	this
assert	do	if	private	throw
boolean	double		protected	throws
break	else	import	public	transient
byte	enum	instanceof	return	true
case	extends	int	short	try
catch	false	interface	static	void
char	final	long	strictfp	volatile
class	finally	native	super	while
continue	float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

Aportaciones de:

lenguaje c, lenguaje c++, lenguaje java

Nota: goto y const se mantienen aunque no se utilizan.

BOOLEAN



Dato primitivo lógico: boolean default value

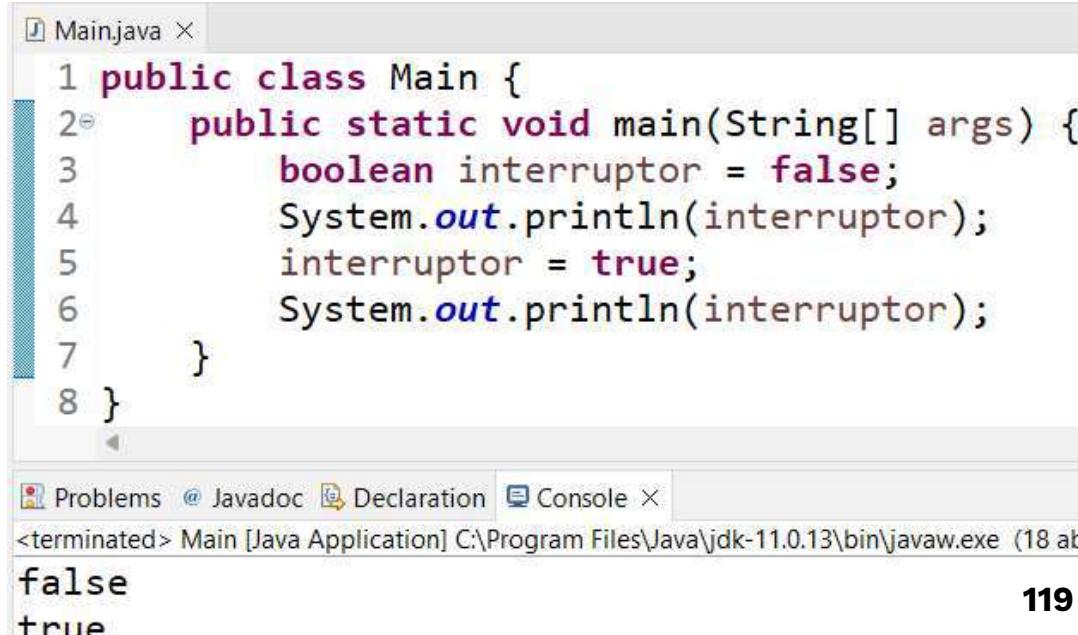
Tipos de datos primitivos				
Nombre	Tipo	Tamaño	Valor por defecto	Rango
boolean	Lógico	1 bit	false	true-false

- El default value en los tipos de datos boolean, es fals. Es decir, si no especificamos un valor e intentamos usar la variable, vemos como el IDE nos indica que tenemos que inicializar la variable asignando false como default value.

```
Main.java ×
1 public class Main {
2     public static void main(String[] args) {
3
4     }
5 }
6
```

Dato primitivo lógico: boolean, rango de valores

Tipos de datos primitivos				
Nombre	Tipo	Tamaño	Valor por defecto	Rango
boolean	Lógico	1 bit	false	true-false



The screenshot shows a Java development environment. The code editor window displays the following Java code:

```
Main.java
1 public class Main {
2     public static void main(String[] args) {
3         boolean interruptor = false;
4         System.out.println(interruptor);
5         interruptor = true;
6         System.out.println(interruptor);
7     }
8 }
```

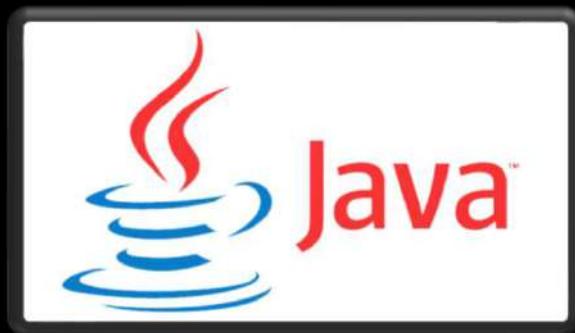
The console window below shows the output of the program:

```
false
true
```

At the bottom right of the interface, the number "119" is displayed.

- El rango de los boolean (1 bit) es el más pequeño de todos, simplemente, dentro de una variable de tipo booleano podemos almacenar los valores true or false.

CHAR



120

¿Qué es UNICODE?

Tipos de datos primitivos

Nombre	Tipo	Tamaño	Valor por defecto	Rango
char	Carácter	16 bits	0	Unicode (del 0 al 65.535)

- Java fue diseñado para usarse a nivel mundial. Por ello, necesita utilizar un juego de caracteres que pueda representar caracteres de todos los idiomas del mundo. Por ese motivo, se decidió que Java no utilizase ASCII, sino UNICODE como sistema de codificación de caracteres.
- Si nos metemos en https://en.wikipedia.org/wiki/List_of_Unicode_characters podemos ver que tenemos distintos caracteres

¿Qué es UNICODE?

- Si nos metemos en <https://unicode-table.com/> podemos ver que tenemos distintos caracteres. En mi caso, me basaré en la almohadilla #:

The screenshot shows the Unicode Table website interface. On the left, there's a binary grid where the cell for code point U+0023 is highlighted with a red box. The character itself, '#', is displayed prominently in the center. To the right, the character is identified as '# Almohadilla' and described as 'signo de Libra , picadillo , sombreado , octothorpe'. Below this, the hex code 'U+0023' is also highlighted with a red box. A table titled 'Codificación' (Encoding) provides the byte representation for various encodings:

Codificación	hex	dec (bytes)	dec	binary
UTF-8	23	35	35	00100011
UTF-16BE	00 23	0 35	35	00000000 00100011
UTF-16LE	23 00	35 0	8960	00100011 00000000
UTF-32BE	00 00 00 23	0 0 0 35	35	00000000 00000000 00000000 00100011
UTF-32LE	23 00 00 00	35 0 0 0	587202560	00100011 00000000 00000000 00000000

Dato primitivo lógico: char default value

Tipos de datos primitivos				
Nombre	Tipo	Tamaño	Valor por defecto	Rango
char	Carácter	16 bits	0	Unicode (del 0 al 65.535)

0 Cero

U+0030

- El default value en el tipo de dato char es 0 (que sería equivalente a utilizar \u0030). Es decir, si no especificamos un valor e intentamos usar la variable, vemos como el IDE nos indica que tenemos que inicializar la variable asignando 0 como default value.

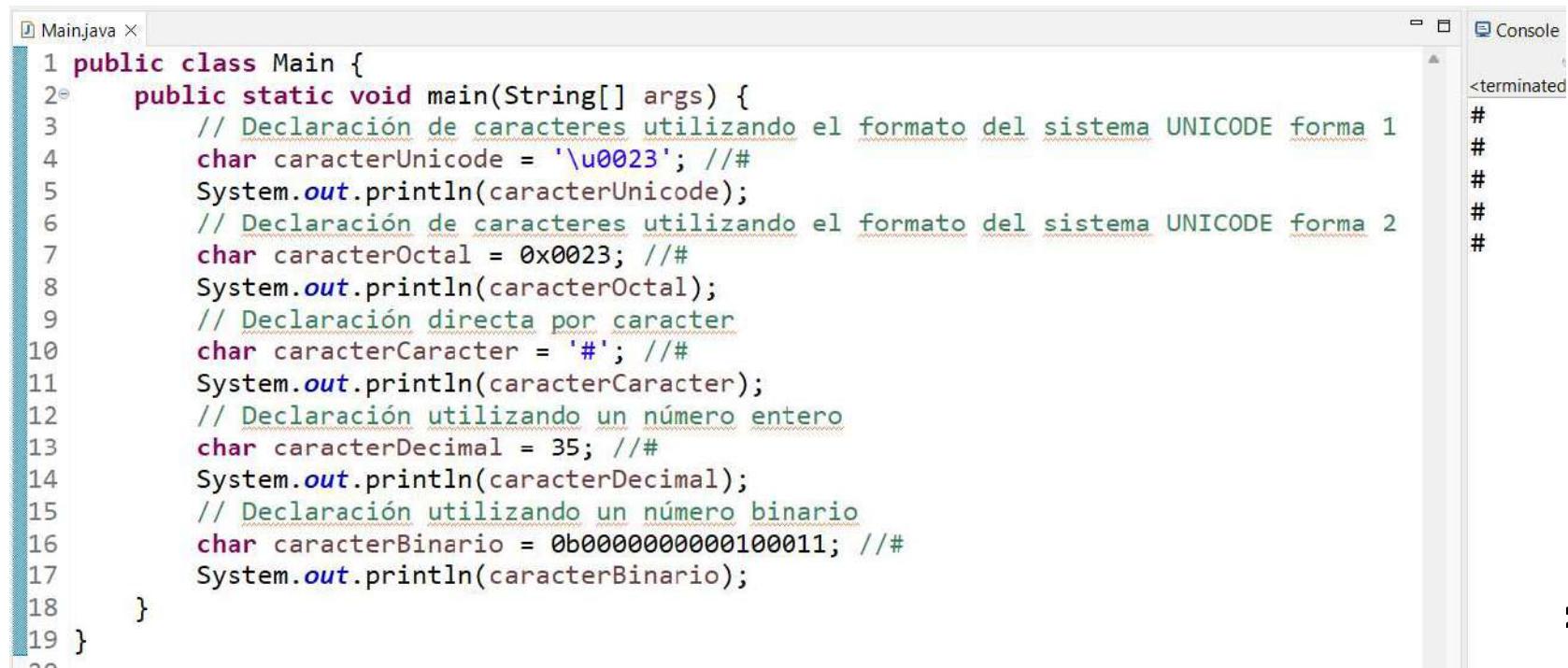
Main.java

```
1 public class Main {  
2     public static void main(String[] args) {  
3     }  
4 }  
5  
6
```

The screenshot shows a Java code editor with a file named "Main.java". The code contains a single-line main method. The code editor has a status bar at the bottom with tabs for "Problems", "Issues", "Declaration", and "Console".

Maneras de asignar un valor a un char

- Existen muchas maneras de declarar un char. La principal peculiaridad es que en algunos casos, utiliza comillas simples ". Vamos a verlas:

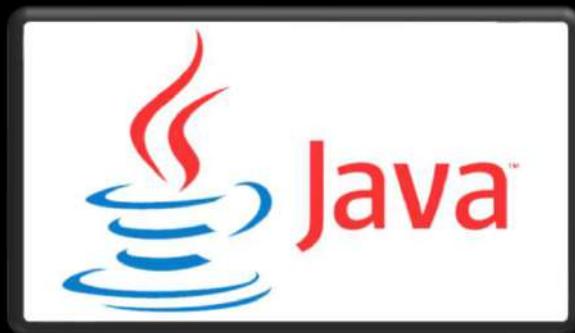


The screenshot shows a Java IDE interface with two main panes. The left pane, titled 'Main.java', contains the following Java code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         // Declaración de caracteres utilizando el formato del sistema UNICODE forma 1  
4         char caracterUnicode = '\u0023'; //#  
5         System.out.println(caracterUnicode);  
6         // Declaración de caracteres utilizando el formato del sistema UNICODE forma 2  
7         char caracterOctal = 0x0023; //#  
8         System.out.println(caracterOctal);  
9         // Declaración directa por carácter  
10        char caracterCaracter = '#'; //#  
11        System.out.println(caracterCaracter);  
12        // Declaración utilizando un número entero  
13        char caracterDecimal = 35; //#  
14        System.out.println(caracterDecimal);  
15        // Declaración utilizando un número binario  
16        char caracterBinario = 0b000000000100011; //#  
17        System.out.println(caracterBinario);  
18    }  
19}
```

The right pane, titled 'Console', shows the output of the program, which consists of five '#' characters, each preceded by a line number from 1 to 5.

FAMILIA DE LOS ENTEROS: BYTE, SHORT, INT Y LONG



COMENZAMOS CON EL “CHIQUITÍN” BYTE

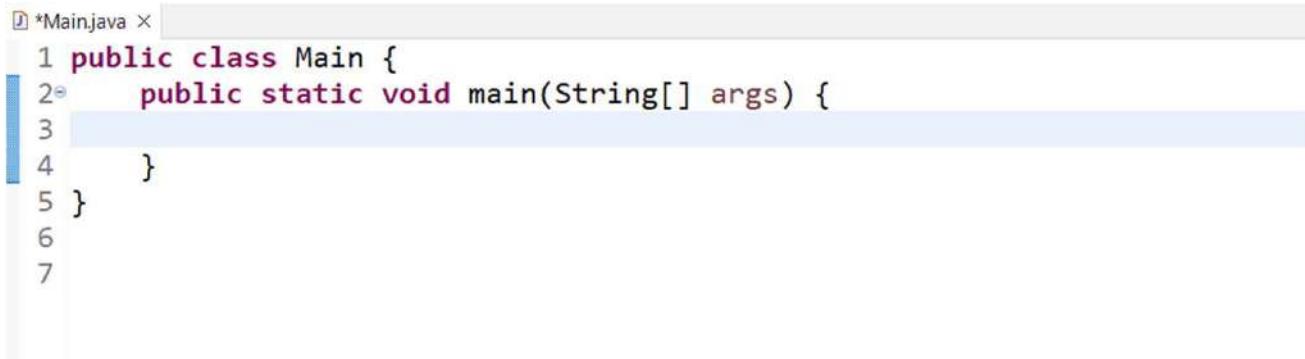


Dato primitivo entero: byte

Tipos de datos primitivos

Nombre	Tipo	Tamaño	Valor por defecto	Rango
byte	Número entero	8 bits	0	-128 a 127

- El default value en el tipo de dato byte es 0. Es decir, si no especificamos un valor e intentamos usar la variable, vemos como el IDE nos indica que tenemos que inicializar la variable asignando 0 como default value.



```
1 public class Main {  
2     public static void main(String[] args) {  
3     }  
4 }  
5  
6  
7
```

Dato primitivo entero: byte

- El rango de byte (8 bit) es el más pequeño de los enteros. Y, simplemente, dentro de una variable de tipo byte podemos almacenar los valores del -128 a 127.

Tipos de datos primitivos				
Nombre	Tipo	Tamaño	Valor por defecto	Rango
byte	Número entero	8 bits	0	-128 a 127

The screenshot shows a Java development environment. On the left, the code editor displays a file named 'Main.java' with the following content:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         byte numero = 125;  
4         System.out.println(numero);  
5     }  
6 }
```

On the right, there is a terminal window titled 'Console' with the output: '125'. The status bar at the bottom right of the terminal window says '<terminate>'.

Dato primitivo entero: byte

- Si definimos un número por debajo de -128 o por encima de +127, estamos realizando un desbordamiento del rango y cuando realicemos la compilación del programa aparecerá un error.

The screenshot shows a Java IDE interface with two tabs: 'Main.java' and 'Console'. In the code editor, line 3 contains the assignment statement: 'byte numero = 130;'. A tooltip appears over this line with the message 'Type mismatch: cannot convert from int to byte'. Below the code editor, the 'Console' tab displays a compilation error: 'Exception in thread "main" java.lang.Error: Unresolved compilation problem: Type mismatch: cannot convert from int to byte at Main.main(Main.java:3)'. A red arrow points from the error message in the console back to the tooltip in the code editor.

```
1 public class Main {  
2     public static void main(String[] args) {  
3         byte numero = 130;  
4         System.out.println(numero);  
5     }  
6 }  
7  
8
```

Console X

<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (18 abr. 2022 23:10:45 – 23:10:46) [pid: 25776]
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
Type mismatch: cannot convert from int to byte
at Main.main(Main.java:3)

Tipos de datos primitivos

Nombre	Tipo	Tamaño	Valor por defecto	Rango
byte	Número entero	8 bits	0	-128 a 127

Dato primitivo entero: byte

- Por defecto, cuando trabajamos con la familia de los números enteros, si el número está fuera del rango del tipo de dato, si os fijáis, Java apunta siempre hacia el tipo de dato int (integer). Por eso decimos, que int, es el tipo de dato genérico para los números enteros.

byte	Número entero	8 bits	0	-128 a 127
short	Número entero	16 bits	0	-32.768 a 32.767
int	Número entero	32 bits	0	-2.147.483.648 al 2.147.483.647
long	Número entero	64 bits	0	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

- De hecho, si intentamos hacerla compilación nos confirma lo que acabamos de comentar:

The screenshot shows an IDE interface with two tabs: 'Main.java' and 'Console'. In the code editor ('Main.java'), line 3 contains the assignment: 'byte numero = 130;'. A tooltip appears over the number '130' with the text 'Type mismatch: cannot convert from int to byte' and '2 quick fixes available: Add cast to 'byte'' and 'Change type of 'numero' to 'int''. In the 'Console' tab, the output is: '<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (18 abr. 2022 23:10:45 - 23:10:46) [pid: 25776] Exception in thread "main" java.lang.Error: Unresolved compilation problem: Type mismatch: cannot convert from int to byte at Main.main(Main.java:3)'.

130

Dato primitivo entero: byte

- Si definimos un número por debajo de -128 o por encima de +127, estamos realizando un desbordamiento del rango y cuando realicemos la compilación del programa aparecerá un error.
- Existe la posibilidad de castea (transformar) ese 130 en un valor de un byte. El problema es que el rango va del -127 al 128. Al transformar un 130, lo que hace es llegar al 128 y seguir sumando números entrando la parte negativa del rango. Por ello, se llaman rangos circulares.

The screenshot shows a Java code editor window titled "Main.java". The code defines a class Main with a main method. Inside the main method, there is a line of code: "byte numero = 130;". A tooltip appears over this line with the text "Type mismatch: cannot convert from int to byte". Below the tooltip, two quick fix options are shown: "Add cast to 'byte'" and "Change type of 'numero' to 'int'".

```
1 public class Main {  
2     public static void main(String[] args) {  
3         byte numero = 130;  
4     }  
5 }  
6  
7
```

The screenshot shows a Java code editor window titled "Main.java". The code defines a class Main with a main method. Inside the main method, there is a line of code: "byte numero = (byte) 130;". To the right of the editor, a "Console" window is open, showing the output: "-126".

```
1 public class Main {  
2     public static void main(String[] args) {  
3         byte numero = (byte) 130;  
4         System.out.println(numero);  
5     }  
6 }
```

Casteando el desbordamiento de un byte y explicando el concepto de rangos circulares

- Aunque, si lo pensamos, tendría más sentido el utilizar un tipo de dato que permita un rango superior, evitando así que desborde o tener que hacer un casteo hacia un rango superior. Por ello, es recomendable no realizar estos casteos (o no realizarlos hacia tipos de datos que sean más pequeños) ya que harán que se utilicen los rangos circulares y los cuales en el caso anterior transformaban un 130, en un -126.

The image shows three separate Java code snippets, each with a red box highlighting a specific line of code:

- Top Editor:** Shows a `byte` variable assignment. A tooltip appears over the value `130`, suggesting to "Add cast to 'byte'" or "Change type of 'numero' to 'int'".
- Middle Editor:** Shows an `int` variable assignment.
- Bottom Editor:** Shows a `short` variable assignment.

SHORT

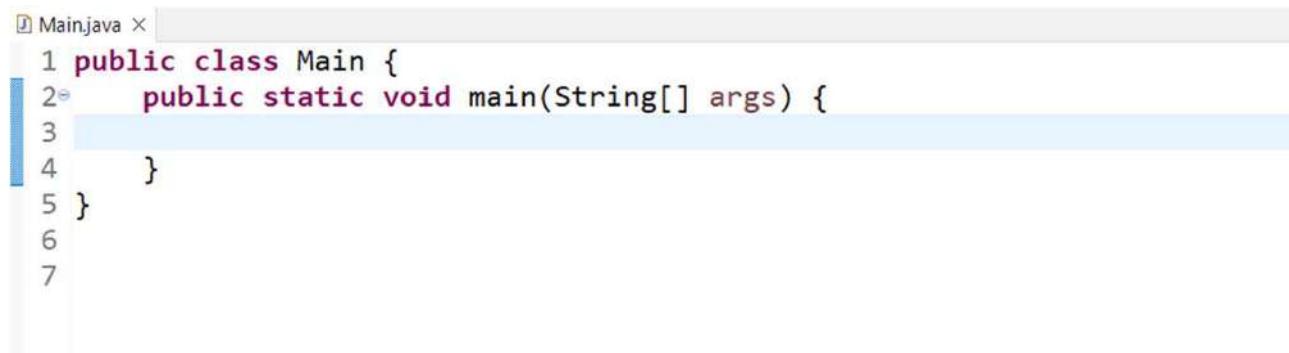


Dato primitivo lógico: short

Tipos de datos primitivos

Nombre	Tipo	Tamaño	Valor por defecto	Rango
short	Número entero	16 bits	0	-32.768 a 32.767

- El default value en los tipos de datos short, es 0. Es decir, si no especificamos un valor e intentamos usar la variable, vemos como el IDE nos indica que tenemos que inicializar la variable asignando 0 como default value.



```
Main.java ×
1 public class Main {
2     public static void main(String[] args) {
3
4     }
5 }
6
7
```

Dato primitivo entero: short, rango de valores

- El tipo de dato short ocupa 16 bits, y por tanto, el doble de bits que byte. Y, simplemente, dentro de una variable de tipo byte podemos almacenar los valores del -32.768 al 32.767.

Tipos de datos primitivos				
Nombre	Tipo	Tamaño	Valor por defecto	Rango
short	Número entero	16 bits	0	-32.768 a 32.767

The screenshot shows an IDE interface with two tabs: 'Main.java' and 'Console'. The 'Main.java' tab contains the following code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         short numero = -32000;  
4         System.out.println(numero);  
5     }  
6 }
```

The 'Console' tab shows the output: <terminated> M -32000. This demonstrates that a short variable can hold the value -32000, which is outside the range of a byte (-128 to 127).

Dato primitivo entero: short, desbordando el rango

- Si definimos un número por debajo de -32.768 o por encima de +32.767, estamos realizando un desbordamiento del rango y cuando realicemos la compilación del programa aparecerá un error.

```
Main.java x
1 public class Main {
2     public static void main(String[] args) {
3         short numero = 32768;
4         System.out.println(numero);
5     }
6 }
Console x
<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (19 abr. 2022 11:54:18 - 11)
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Type mismatch: cannot convert from int to short
                at Main.main(Main.java:3)
```

Tipos de datos primitivos				
Nombre	Tipo	Tamaño	Valor por defecto	Rango
short	Número entero	16 bits	0	-32.768 a 32.767

Dato primitivo entero: short, desbordando el rango

- Por defecto, cuando trabajamos con la familia de los números enteros, si el número está fuera del rango del tipo de dato, si os fijáis, Java apunta siempre hacia el tipo de dato int (integer). Por eso decimos, que int, es el tipo de dato genérico para los números enteros.

byte	Número entero	8 bits	0	-128 a 127
short	Número entero	16 bits	0	-32.768 a 32.767
int	Número entero	32 bits	0	-2.147.483.648 al 2.147.483.647
long	Número entero	64 bits	0	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

- De hecho, si intentamos hacerla compilación nos confirma lo que acabamos de comentar:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         short numero = 32768;  
4         System.out.println(numero);  
5     }  
6 }  
7  
8
```

Console output:
<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (19 abr. 2022 11:54:18 - 11:54:20) [pid: 6840]
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
Type mismatch: cannot convert from int to short
at Main.main(Main.java:3)

Casteando el desbordamiento de un short y explicando el concepto de rangos circulares

- Ya sabemos lo que tenemos que hacer para castear y lo que va a pasar. Vamos a verlo:
- Aunque, tendría más sentido el utilizar un tipo de dato que no desborde evitando así el que un 130, se transforme en un -126.

The screenshot shows an IDE interface with two panes. The top pane displays a Java file named 'Main.java' with the following code:1 public class Main {
2 public static void main(String[] args) {
3 short numero = 32768;
4 System.out.println(numero);
5 }
6 }A tooltip appears over the line 'short numero = 32768;' with the message 'Type mismatch: cannot convert from int to short'. It contains two quick fix options: 'Add cast to 'short'' (highlighted with a red box) and 'Change type of numero' to 'int''. The bottom pane shows the 'Console' tab with the output '-32768'.

INT



Dato primitivo lógico: INT default value

Tipos de datos primitivos				
Nombre	Tipo	Tamaño	Valor por defecto	Rango
int	Número entero	32 bits	0	-2.147.483.648 al 2.147.483.647

- El default value en los tipos de datos int, también es 0. Es decir, si no especificamos un valor e intentamos usar la variable, vemos como el IDE nos indica que tenemos que inicializar la variable asignando 0 como default value.

```
Main.java ×
1 public class Main {
2     public static void main(String[] args) {
3
4     }
5 }
6
```

The screenshot shows a Java code editor window titled "Main.java". The code contains a single class definition with a main method. The first two lines of the code ("public class Main {" and "public static void main(String[] args) {") are highlighted with a light blue background, indicating they are selected or being edited. The line numbers 1 through 6 are visible on the left side of the code editor.

Dato primitivo entero: short, rango de valores

- El tipo de dato int ocupa 32 bits, y por tanto, el dos veces más grande que short (16 bits) y cuatro veces lo de byte (8 bits). Y, simplemente, dentro de una variable de tipo int podemos almacenar los valores del -2.147.483.648 al 2.147.483.647.

byte	Número entero	8 bits	0	-128 a 127
short	Número entero	16 bits	0	-32.768 a 32.767
int	Número entero	32 bits	0	-2.147.483.648 al 2.147.483.647
long	Número entero	64 bits	0	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

The screenshot shows an IDE interface with two tabs: 'Main.java' and 'Console'. The 'Main.java' tab contains the following code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int numero = 283958736;  
4         System.out.println(numero);  
5     }  
6 }
```

The 'Console' tab shows the output: <terminated> Main 283958736, indicating that the value 283958736 was printed to the console.

Dato primitivo entero: int, desbordando el rango

- Por defecto, cuando trabajamos con la familia de los números enteros, si el número está fuera del rango del tipo de dato, si os fijáis, Java apunta siempre hacia el tipo de dato int (integer). Por eso decimos, que int, es el tipo de dato genérico para los números enteros.

byte	Número entero	8 bits	0	-128 a 127
short	Número entero	16 bits	0	-32.768 a 32.767
int	Número entero	32 bits	0	-2.147.483.648 al 2.147.483.647
long	Número entero	64 bits	0	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

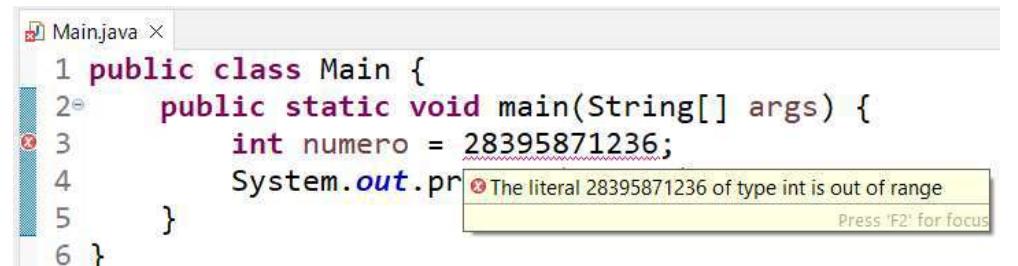
- De hecho, si intentamos hacerla compilación nos confirma lo que acabamos de comentar.

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int numero = 28395871236;  
4         System.out.println(numero);  
5     }  
6 }
```

<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (19 abr. 2022 12:04:03 - 12:04:05) [pid: 24492]
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
The literal 28395871236 of type int is out of range
at Main.main(Main.java:3)

El desbordamiento de un int no nos ofrece opción de casteo

- En este caso, no nos aparece opción ni de castear por defecto. ¿Por qué no? Porque tal y como decimos el tipo genérico en la familia de los enteros es int. Pero si no indicamos que es de un tipo superior long añadiendo una L al final. Java no sabe que desborda al integer pero no entiende muy bien que tiene que hacer con ese número. Y se queda como en “shock”

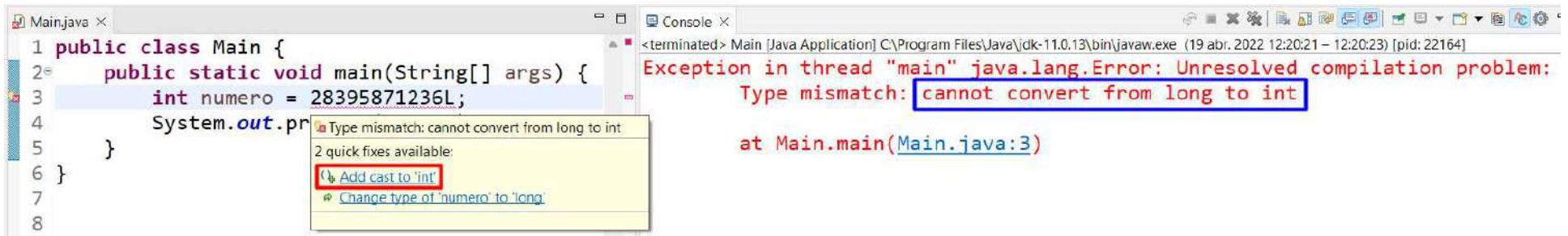


A screenshot of an IDE showing a Java file named Main.java. The code is:1 public class Main {
2 public static void main(String[] args) {
3 int numero = 28395871236;
4 System.out.pr
5 }
6 }The line `int numero = 28395871236;` has a red X icon next to it. A tooltip window appears over the line with the text: `The literal 28395871236 of type int is out of range`. Below the code editor are four images of people looking shocked.



¿Y ahora que?

- Si añadimos una L al final del número que desbordaba el int, ahora si que tenemos un tipo de dato long. Y, podemos ver que el mensaje a cambiado y que además, ya nos ofrece la posibilidad de realizar un casteo a un int:

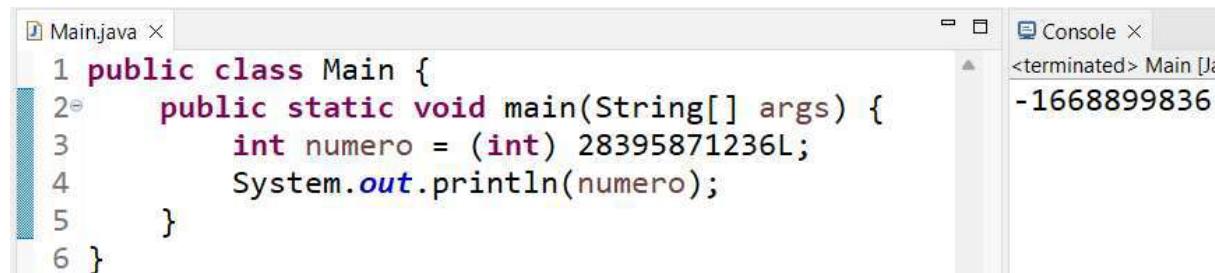


```
Main.java X Console X
1 public class Main {
2     public static void main(String[] args) {
3         int numero = 28395871236L;
4         System.out.println(numero);
5     }
6 }
7
8
```

Exception in thread "main" java.lang.Error: Unresolved compilation problem:
Type mismatch: cannot convert from long to int
at Main.main(Main.java:3)

Type mismatch: cannot convert from long to int
2 quick fixes available:
Add cast to 'int'
Change type of 'numero' to 'long'

- Y ahora si, que se nos aplica el casting y vemos que se realiza la conversión y que se aplica el rango circular:



```
Main.java X Console X
1 public class Main {
2     public static void main(String[] args) {
3         int numero = (int) 28395871236L;
4         System.out.println(numero);
5     }
6 }
```

-1668899836

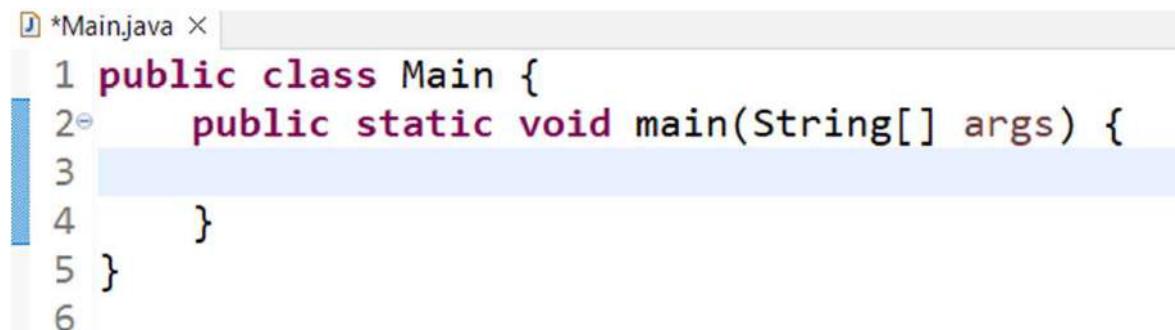
LONG



Dato primitivo entero: LONG default value

Tipos de datos primitivos				
Nombre	Tipo	Tamaño	Valor por defecto	Rango
long	Número entero	64 bits	0	-9.223.372.036.854.775,808 a 9.223.372.036.854.775,807

- El default value en los tipos de datos LONG, también es 0. Es decir, si no especificamos un valor e intentamos usar la variable, vemos como el IDE nos indica que tenemos que inicializar la variable asignando 0 como default value.



```
1 public class Main {  
2     public static void main(String[] args) {  
3     }  
4 }  
5 }  
6 }
```

Dato primitivo entero: long, rango de valores

- El tipo de dato long ocupa 64 bits, y por tanto, es el mayor de toda la familia de enteros. Y, simplemente, dentro de una variable de tipo long podemos almacenar los valores del -9.223.372.036.854.775,808 a 9.223.372.036.854.775,807.
- Además, vemos que es necesario utilizar la L en mayúsculas o minúsculas

The screenshot shows an IDE interface with two panes. On the left, the code editor displays a Main.java file with the following code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         long numero = 92233720368547;  
4         System.out.println(numero);  
5     }  
6 }
```

A tooltip appears over the line `long numero = 92233720368547;`, stating: "The literal 92233720368547 of type int is out of range".

On the right, the console pane shows the output of the program: "92233720368547".

Below the code editor is a table comparing primitive integer types:

byte	Número entero	8 bits	0	-128 a 127
short	Número entero	16 bits	0	-32.768 a 32.767
int	Número entero	32 bits	0	-2.147.483.648 al 2.147.483.647
long	Número entero	64 bits	0	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

Dato primitivo entero: long, desbordando el rango

- Por defecto, cuando trabajamos con la familia de los números enteros, si el número está fuera del rango del tipo de dato, si os fijáis, Java apunta siempre hacia el tipo de dato int (integer). Por eso decimos, que int, es el tipo de dato genérico para los números enteros.

byte	Número entero	8 bits	0	-128 a 127
short	Número entero	16 bits	0	-32.768 a 32.767
int	Número entero	32 bits	0	-2.147.483.648 al 2.147.483.647
long	Número entero	64 bits	0	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

- Pero si desbordamos un dato long y queremos realizar un casteo y un rango circular deberíamos subir un dato real: float o double.

long	Número entero	64 bits	0	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
float	Número real	32 bits	0.0	1.4E-45 a 3.4028235E38
double	Número real	64 bits	0.0	4.9E-324 a 1.7976931348623157E308

Dato primitivo entero: casteo de double a long aplicando el rango circular

- Si transformo el dato a un double añadiendo un .0 al digito, vemos que ahora si que nos permite realizar el casteo a long:

The screenshot shows an IDE interface with three main windows: a code editor, a console, and a status bar.

In the code editor (Main.java), line 4 contains the assignment: `long numero = 92233720362342342343243238547.0;`. A tooltip appears over the decimal point, stating "Type mismatch: cannot convert from double to long" and offering two quick fixes: "Add cast to 'long'" and "Change type of 'numero' to 'double'".

In the code editor (Main.java), line 4 has been modified to: `long numero = (long) 92233720362342342343243238547.0;`. The tooltip is no longer present.

In the console window, the output is: `<terminated> Main [Java Application] C 9223372036854775807`.

Dato primitivo entero: casteo de long a float aplicando el rango circular

- Si transformo el dato a un float añadiendo una F al final del digito, vemos que ahora si que nos permite realizar el casteo a long:

The screenshot shows a Java code editor with a file named Main.java. The code is as follows:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         long numero = 9223372036854775847F;  
4         System.out.println(numero);  
5     }  
6 }  
7  
8
```

A tooltip appears over the line `long numero = 9223372036854775847F;`, stating "Type mismatch: cannot convert from float to long". It offers two quick fixes:

- (↳ Add cast to 'long')
- (↳ Change type of 'numero' to 'float')

The screenshot shows the Java code editor with the corrected code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         long numero = (long) 9223372036854775847F;  
4         System.out.println(numero);  
5     }  
6 }
```

The code is executed in the console, and the output is:

```
<terminated> Main [Java Application]  
9223372036854775807
```

FAMILIA DE LOS DECIMALES: FLOAT Y LONG



FLOAT



Dato primitivo real: FLOAT

Tipos de datos primitivos

Nombre	Tipo	Tamaño	Valor por defecto	Rango
float	Número real	32 bits	0.0	1.4E-45 a 3.4028235E38

- Si intentamos crear un float, vemos que por ejemplo cuando añadimos un valor sin definir una F (en mayúsculas o minúsculas) el tipo de dato que se utiliza en los enteros por defecto es double. Esto es debido a que en los tipos de datos reales, es decir, los que tienen decimales, el tipo de dato genérico es double. Vamos a ver un ejemplo:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         float num01 = 0.5;  
4     }  
5 }  
6  
7
```

Console ×

<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (20 abr. 2022 16:21:54 - 16:21:55) [pid: 11204]

Exception in thread "main" java.lang.Error: Unresolved compilation problem:
Type mismatch: cannot convert from double to float
at Main.main(Main.java:3)

Type mismatch: cannot convert from double to float
2 quick fixes available:
Add cast to 'float'
Change type of 'num01' to 'double'

Dato primitivo real: FLOAT

- Para solucionar esto, existen varias maneras. La primera sería añadir una f al tipo de dato.

Vamos a verlo:

The screenshot shows an IDE interface with two tabs: 'Main.java' and 'Console'. The 'Main.java' tab contains the following code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         float num01 = 0.5f;  
4         System.out.println(num01);  
5     }  
6 }
```

The value '0.5' is highlighted with a red box. The 'Console' tab shows the output: '<terminated> 0.5'.

- La segunda manera, sería casteando (transformando) el tipo de dato double a float.

Vamos a ver un ejemplo:

The screenshot shows an IDE interface with two tabs: 'Main.java' and 'Console'. The 'Main.java' tab contains the following code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         float num01 = (float) 0.5;  
4         long bits = Double.doubleToLongBits(num01);  
5         System.out.println(Long.toBinaryString(bits));  
6     }  
7 }
```

- El principal problema que tenemos si hacemos un parse de un doble a un float, es que podemos perder algo de datos.

Dato primitivo real: FLOAT

- En este caso, al tener un valor que supera el valor máximo de float, nos aparecerá como resultado infinity. Ya que no puede convertir dicho valor en el valor anterior.

The screenshot shows a Java development environment with two tabs open: 'Main.java' and 'Console'.

Main.java:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         // Wrapper method Double  
4         Double d = Double.MAX_VALUE;  
5         System.out.println(d);  
6  
7         float numFloat = 1.7976931348623157E308f;  
8     }  
9 }
```

A tooltip appears over the float assignment line: "The literal 1.7976931348623157E308f of type float is out of range".

Console:

```
<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (20 abr. 2022 16:46:16 - 16:46:17) [pid: 22008]  
Exception in thread "main" java.lang.Error: Unresolved compilation problem:  
    The literal 1.7976931348623157E308f of type float is out of range  
        at Main.main(Main.java:7)
```

The output in the Console tab shows 'Infinity'.

Dato primitivo real: FLOAT default value

Tipos de datos primitivos				
Nombre	Tipo	Tamaño	Valor por defecto	Rango
float	Número real	32 bits	0.0	1.4E-45 a 3.4028235E38

- Por ello, el default value en los tipos de datos FLOAT, debería de ser 0f y no 0 aunque sea 0 el que se asigna automáticamente. Ya que siendo puristas el genérico que nos asigna Java es el 0 del tipo de dato double y no el de float (aunque en este caso sean lo mismo).

DOUBLE, EL “HERMANO MAYOR” DE LOS DATOS PRIMITIVOS



Dato primitivo real: DOUBLE

- Es el número primitivo mayor de todos LOS PRIMITIVOS. Vamos a ver un ejemplo:

Tipos de datos primitivos				
Nombre	Tipo	Tamaño	Valor por defecto	Rango
double	Número real	64 bits	0.0	4.9E-324 a 1.7976931348623157E308

The screenshot shows an IDE interface with two main panes. On the left, the 'Main.java' file is open, displaying the following Java code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         double num = 1234567891232132.1234;  
4         System.out.println(num);  
5     }  
6 }
```

On the right, the 'Console' pane shows the output of the program: '1.234567891232132E15'. This output is displayed in scientific notation, which is the standard way to represent such large numbers.

- Si queremos trabajar con un tipo mayor tendríamos que buscar otro tipo no primitivo que cubriera nuestras necesidades.

FLOAT VS DOUBLE



FLOAT vs DOUBLE

- La principal diferencia entre float y double es que double tiene tanto la parte entera como la decimal el doble de grande que float.

Tipos de datos primitivos				
Nombre	Tipo	Tamaño	Valor por defecto	Rango
float	Número real	32 bits	0.0	1.4E-45 a 3.4028235E38
double	Número real	64 bits	0.0	4.9E-324 a 1.7976931348623157E308

- Por ejemplo, centrándonos en la parte decimal float acepta hasta 9 decimales, Y, en cambio, double acepta hasta 18 decimales. Vamos a verlo:

The screenshot shows a Java IDE interface with two panes. On the left, the code editor displays a file named 'Main.java' with the following content:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         float numFloat = 0.124567891245678912456789f;  
4         System.out.println("Float: " + numFloat);  
5  
6         double numDouble = 0.12456789123456789123456789;  
7         System.out.println("Double: " + numDouble);  
8     }  
9 }
```

On the right, the 'Console' pane shows the output of the program:

```
<terminated> Main [Java Application] C:\Program Files  
Float: 0.12456789  
Double: 0.12456789123456789
```

Two red arrows point from the highlighted floating-point numbers in the code editor to their corresponding outputs in the console. The first arrow points from the line 'float numFloat = 0.124567891245678912456789f;' to the output 'Float: 0.12456789'. The second arrow points from the line 'double numDouble = 0.12456789123456789123456789;' to the output 'Double: 0.12456789123456789'.

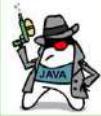


Ejercicio

- Define grupos realizados mediante comentarios en lo que separaremos los tipos primitivos en función de si son lógicos, enteros, reales... Y pon un ejemplo para cada uno de los datos e imprime el valor por la consola.

Nombre	Tipo
boolean	Lógico
char	Carácter
byte	Número entero
short	Número entero
int	Número entero
long	Número entero
float	Número real
double	Número real

```
Main.java x
1 package interfaces;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         // Lógicos
7
8         // Caracter
9
10        // Numericos enteros
11
12        // Número reales
13
14        // Constantes
15
16    }
17 }
```



Ejercicio

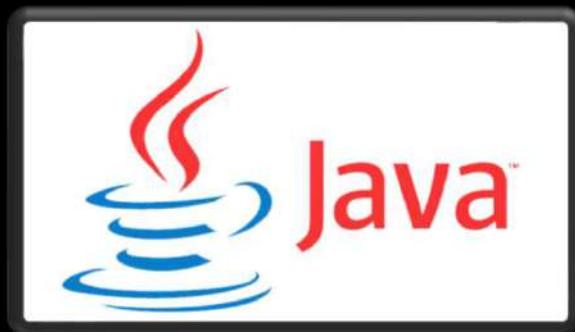
- Haz un ejemplo de rango circular sobre todos los tipos de números enteros.
 - En el caso de byte, el número deberá de desbordar 5 posiciones
 - En el caso de short, el número deberá de desbordar 25 posiciones
 - En el caso de int, el número deberá de desbordar 10000 posiciones
 - En el caso de long, el número deberá de desbordar 20000 posiciones.

byte	Número entero
short	Número entero
int	Número entero
long	Número entero

CONSTANTES



¿QUÉ ES UNA CONSTANTE?



¿Qué son las constantes?

- Una constante es una variable que tiene un valor inmutable. Es decir, que no cambiará al o largo de la vida/ejecución del programa.
- Para definir una constante utilizamos la palabra reservada final.
- Además, es recomendable seguir la convención de identificadores (name convention) SCREAMING_SNAKE_CASE para realizar la definición de los identificadores de nuestras constantes para ver de forma sencilla que se trata de una constante y no de una variable. Evitando así el intento de reasignar un nuevo valor..
- Vamos a ver un ejemplo:



SCREAMING_SNAKE_CASE
Pros: Can demonstrate your anger with text.
Cons: Makes your eyes deaf.
LOOK_AT_THIS, LOOK_AT_THAT, LOOK_HERE_YOU_MORON, ...

```
Main.java x
1 public class Main {
2     public static void main(String[] args) {
3         final String NOMBRE_COMPLETO = "David Bernal González";
4         System.out.println(NOMBRE_COMPLETO);
5     }
6 }
```

Console x
<terminated> Main [Java Application] C:\Pr
David Bernal González

¿Qué son las constantes?

- La principal diferencia entre las constantes y las variables es la siguiente:

The screenshot shows an IDE interface with several windows:

- Main.java** (Editor): Displays the Java code for a Main class. Line 5 contains the assignment `NOMBRE_COMPLETO = "ASDFDASFSA";`, which is highlighted with a red box.
- Console**: Shows the output of the Java application. It includes the command run, the path to the Java executable, and the timestamp. Below that, it displays an error message: "Exception in thread "main" java.lang.Error: Unresolved compilation problem: The final local variable NOMBRE_COMPLETO cannot be assigned. at Main.main(Main.java:5)".
- Problems**: A list of errors and warnings. It shows one error: "The final local variable NOMBRE_COMPLETO cannot be assigned. It must be blank and not using a compound assignment". This error is also highlighted with a red box and has a red arrow pointing from the editor window to the Problems window.

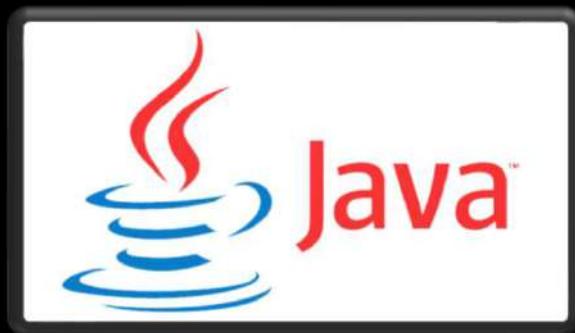
- Si nos fijamos, vemos que no podemos reasignar el valor. Por lo que únicamente la utilizamos cuando queramos definir un valor que sea inmutable.

Ejercicio



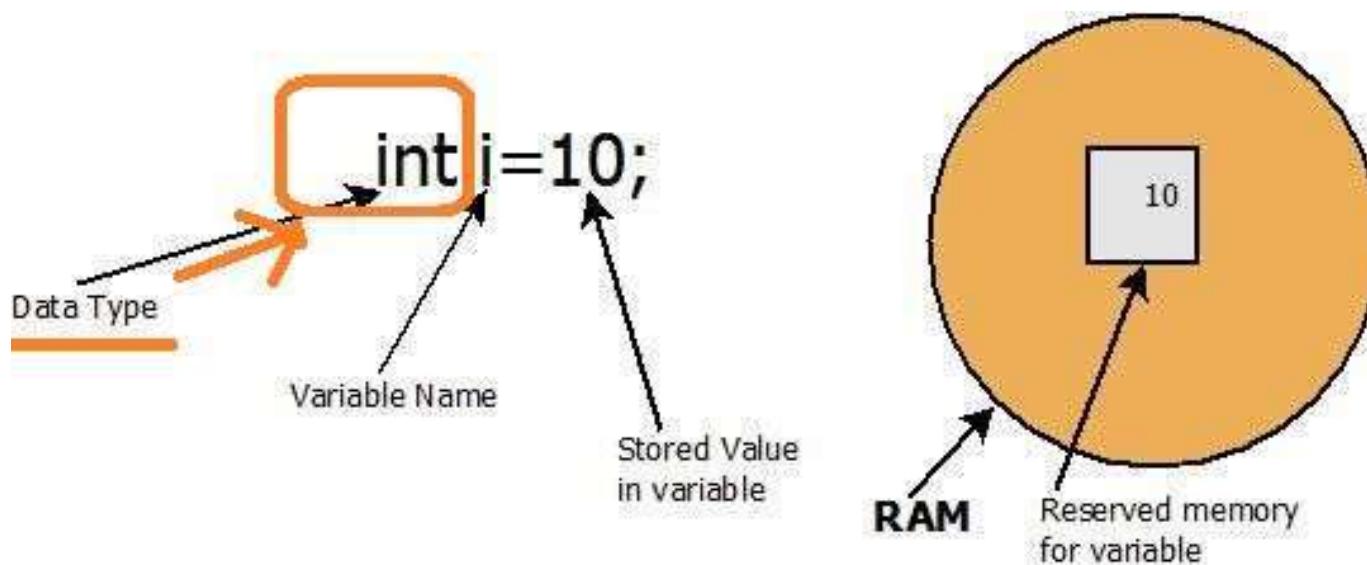
- Basándote en el ejercicio anterior en el que hemos declarado una variable para cada tipo de dato primitivo. Transforma dichas variables a constantes.

¿QUÉ TIPO DE DATO UTILIZO?



¿Qué tipo de dato utilizo?

- Para guardar una variable lo recomendable sería que utilizáramos el tipo de dato que mejor se adapte a nuestro dato. Por ejemplo, si queremos guardar un valor del 1 al 100, ¿Qué tipo de dato primitivo utilizarías?

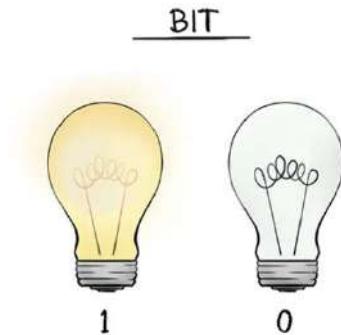


CALCULANDO EL RANGO DE UN TIPO DE DATO MANUALMENTE



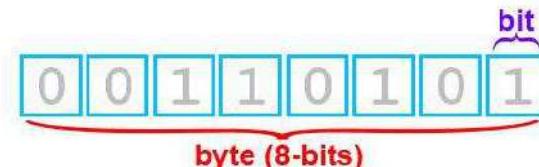
Calculando el rango de un tipo de dato manualmente

- Los ordenadores solamente “hablan”/entienden el lenguaje binario. El lenguaje binario, solamente trabaja con unos (1) y ceros (0). Por lo que estos ceros y unos será la manera mediante a la cual enviaremos una serie de impulsos eléctricos al computador que nos permitirán comunicaremos con el dispositivos.
 - Cero significa ausencia de electricidad y un 1 presencia.
- Un bit es la unidad mínima del binario y por tanto, también de la informática. Ya que la informática trabaja sobre el lenguaje binario. Y acepta los valores 0 (apagado) o 1 (encendido)



Calculando el rango de un tipo de dato manualmente

- En binario, además de la unidad mínima bit. Tenemos otras unidades como por ejemplo:
 - Byte que equivale a 8 bits
 - Kilobyte (Kb) equivale a 1024 bytes, etc.



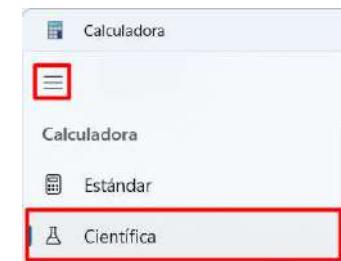
Medida	Simbología	Equivalencia	Equivalente en Bytes
byte	b	8 bits	1 byte
kilobyte	Kb	1024 bytes	1 024 bytes
megabyte	MB	1024 KB	1 048 576 bytes
gigabyte	GB	1024 MB	1 073 741 824 bytes
terabyte	TB	1024 GB	1 099 511 627 776 bytes
Petabyte	PB	1024 TB	1 125 899 906 842 624 bytes
Exabyte	EB	1024 PB	1 152 921 504 606 846 976 bytes
Zetabyte	ZB	1024 EB	1 180 591 620 717 411 303 424 bytes
Yottabyte	YB	1024 ZB	1 208 925 819 614 629 174 706 176 bytes
Brontobyte	BB	1024 YB	1 237 940 039 285 380 274 899 124 224 bytes
Geophyte	GB	1024 BB	1 267 650 600 228 229 401 496 703 205 376 bytes

Calculando el rango de un tipo de dato manualmente

- El sistema binario, es también conocido como base 2 debido a que por cada bit solamente tiene 2 opciones (0 y 1).
- Por ejemplo:
 - Para calcular el rango de un bit elevamos la base 2 a la cantidad de bits(1). Por ejemplo:
 - $2 \text{ (base binaria)}^1 \text{ (bits)} = 2$ por lo que nos ofrece en total 2 valores en su rango. En este caso, un 1 y un 0
 - Para calcular el rango de un byte elevamos la base 2 a la cantidad de bits(8)
 - $2 \text{ (base binaria)}^8 \text{ (bits)} = 256$ por lo que ofrece en total 256 valores en su rango.



Calculadora
Aplicación



$$2^1 = 2$$

$$2^8 = 256$$

Calculando el rango de un tipo de dato manualmente

Nombre	Tipo	Tamaño	Valor por defecto	Rango
byte	Número entero	8 bits	0	-128 a 127

- Dentro del $2^8=256$ total de los 256 valores que tiene un rango byte, el rango se subdivide en dos grandes bloques:
 - La parte positiva del rango (del 0 en adelante), incluye todo lo que no tenga un - delante.
 - La parte negativa del rango del -1 hacia delante, incluye todo lo que tenga un - delante.
- Para separar ambos rangos debemos dividir los valores totales del rango entre 2. Vamos a ver un ejemplo:

$$256 \div 2 =$$

128

- Total de valores del tipo de dato byte $2(\text{base binaria})^8(\text{bits})=256$
- Si dividimos los valores totales del tipo de dato entre 2 vemos que sale el total de los valores del rango positivo y negativo
 - El rango negativo empieza en el -128 y acaba en el -1
 - Al rango positivo le restamos -1 ya que al empezar en el 0 acaba en el 127

175



Ejercicio

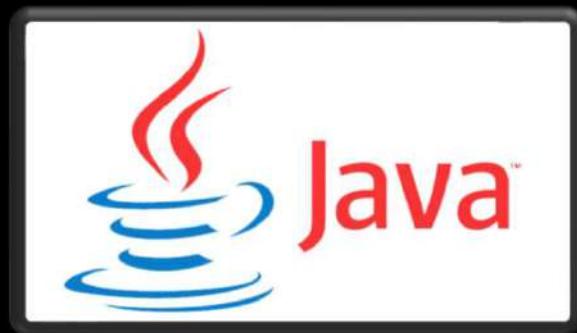
- Calcula el rango de todos los tipos de datos primitivos (en un fichero de Word). Veras que tanto char y boolean tienen solamente positivos. Y los números (al tener necesidades distintas), se subdividen en valores positivos y negativos.
 - $2(\text{base binaria})^X \text{ (bits)} = Y(\text{total del rango})$
 - $Y(\text{total del rango})/2 = Z(\text{valores para positivos y negativos})$

Tipos de datos primitivos				
Nombre	Tipo	Tamaño	Valor por defecto	Rango
boolean	Lógico	1 bit	false	true-false
char	Carácter	16 bits	0	Unicode (del 0 al 65.535)
byte	Número entero	8 bits	0	-128 a 127
short	Número entero	16 bits	0	-32.768 a 32.767
int	Número entero	32 bits	0	-2.147.483.648 al 2.147.483.647
long	Número entero	64 bits	0L o 0l	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
float	Número real	32 bits	0.0f o 0.0F	1.4E-45 a 3.4028235E38
double	Número real	64 bits	0.0d o 0.0D	4.9E-324 a 1.7976931348623157E308

OPERADORES EN JAVA



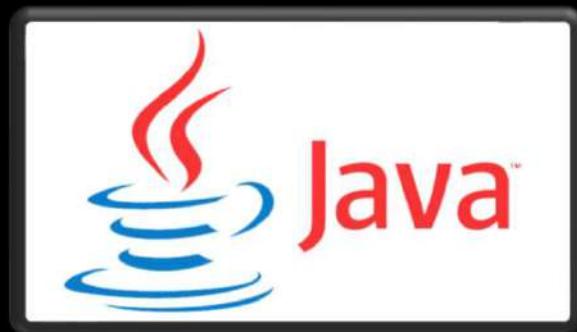
TIPOS DE OPERADORES



Operadores en Java

- Dentro de los operadores, tenemos distintos operadores en función de sobre cuantos datos u operandos realice una función. Y los principales son:
 - Unarios: cuando operan sobre un solo dato/operando
 - Binario: cuando operan sobre dos datos/operandos
 - Ternario: cuando opera sobre tres datos/operandos
- También, podemos clasificar los operadores en función de la relación
 - Operador de asignación
 - Operadores aritméticos: básicos, incrementales, combinados
 - Operadores de relación
 - Operadores lógicos o booleanos
 - Operador condicional
 - Operadores de bit
 - Operadores de concatenación de cadenas

PIRÁMIDE DE JERARQUÍA DE OPERACIONES



Pirámide de jerarquía de operaciones:

- Las matemáticas y la programación comparten las mismas prioridades en lo que se conoce como jerarquía de operaciones. Vamos a verlas:



OPERADOR DE ASIGNACIÓN

=



Operador de asignación =

- El = es el operador de asignación.

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
=	Operador asignación	n = 4	n vale 4

- Vamos a ver un ejemplo:

```
public class Main {  
    public static void main(String[] args) {  
        int numero = 1; // Operador de asignación (=)  
    }  
}
```

OPERADOR DE CONCATENACIÓN DE CADENAS

+



Operador de concatenación de cadenas +

- El operador es un operador binario que nos permitirá realizar concatenaciones de cadenas.

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
+	Operador concatenación	"Hola" + "Juan"	"HolaJuan"

- Vamos a ver un ejemplo:

The screenshot shows a Java development environment with two windows. On the left, the code editor displays the Main.java file with the following content:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         String nombre = "David";  
4         String apellidos = "Bernal González";  
5         System.out.println(nombre + " " + apellidos);  
6     }  
7 }
```

On the right, the console window shows the output of the program: "David Bernal González".

Ejercicio de concatenación de cadenas

- Define las siguientes variables: nombre, apellido, calle, municipio, código postal, sexo (tipo char), correo electrónico y concaténalas haciendo saltos de líneas, tabulaciones... Como consideres imprimiendo todos los valores en un solo sysout.

OPERADORES ARITMETICOS BÁSICOS

+ , - , -(símbolo) , +(símbolo) , * , / , %



Operadores aritméticos:

- Tenemos distintos operadores aritméticos. Posteriormente veremos un ejemplo de un uso de cada uno de ellos. Pero de momento vamos a ver dichos operadores:

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
-	operador unario de cambio de signo	-4	-4
+	Suma	2.5 + 7.1	9.6
-	Resta	235.6 - 103.5	132.1
*	Producto	1.2 * 1.1	1.32
/	División (tanto entera como real)	0.050 / 0.2 7 / 2	0.25 3
%	Resto de la división entera	20 % 7	6

Operadores aritméticos: +

- El + se puede utilizar como:
 - Operador unario: se utiliza para definir que el signo del operando es positivo. Aunque, todo operando que no tenga un menos será positivo. Por lo que, no es necesario definirlo ya que por defecto todo lo que no tenga un - delante será positivo.
 - Operador binario: cuando queremos realizar una suma entre dos operandos.

The screenshot shows three Java code snippets in a code editor and their results in a terminal window.

- Code 1:** Prints the value of `numero`. The variable is initialized with `+1`, so the output is `1`.

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int numero = +1;  
4         System.out.println(numero);  
5     }  
6 }
```

- Code 2:** Prints the value of `numero`. The variable is initialized with `1`, so the output is `1`.

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int numero = 1;  
4         System.out.println(numero);  
5     }  
6 }
```

- Code 3:** Adds `num1` and `num2` and prints the result. The output is `5`.

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int num1 = 2;  
4         int num2 = 3;  
5         int result = num1 + num2;  
6         System.out.println(result);  
7     }  
8 }
```

Operadores aritméticos: -

- El - se puede utilizar como:
 - Operador unario: se utiliza para definir que el signo del operando es negativo.
 - Operador binario: cuando queremos realizar una resta entre dos operandos.

The screenshot shows a Java development environment. The code editor displays the following Java code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int numero = -1;  
4         System.out.println(numero);  
5     }  
6 }
```

To the right of the code editor is a terminal window showing the output: -1.

The screenshot shows a Java development environment. The code editor displays the following Java code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int num1 = 5;  
4         int num2 = 3;  
5         System.out.println(num1-num2);  
6     }  
7 }
```

To the right of the code editor is a terminal window showing the output: 2.

Operadores aritméticos: *, / y %

- El * se puede utilizar como:
 - Operador binario: para realizar una multiplicación entre dos operandos.
- El / se puede utilizar como:
 - Operador binario: para realizar una división entre dos operandos.
- El % se puede utilizar como:
 - Operador binario: para obtener el resto de una división entre dos operandos.

The screenshot shows three code snippets in a Java IDE:

- Top Snippet:** A Java class named Main with a main method. It declares two integer variables num1 and num2, both initialized to 5 and 3 respectively. It then prints the result of their multiplication (15) to the console.
- Middle Snippet:** A Java class named Main with a main method. It declares two integer variables num1 and num2, both initialized to 5 and 3 respectively. It then prints the result of their division (1) to the console.
- Bottom Snippet:** A Java class named Main with a main method. It declares two integer variables num1 and num2, both initialized to 5 and 3 respectively. It then prints the result of their modulus operation (2) to the console.

PROBLEMAS CON LOS OPERADORES ARITMÉTICOS



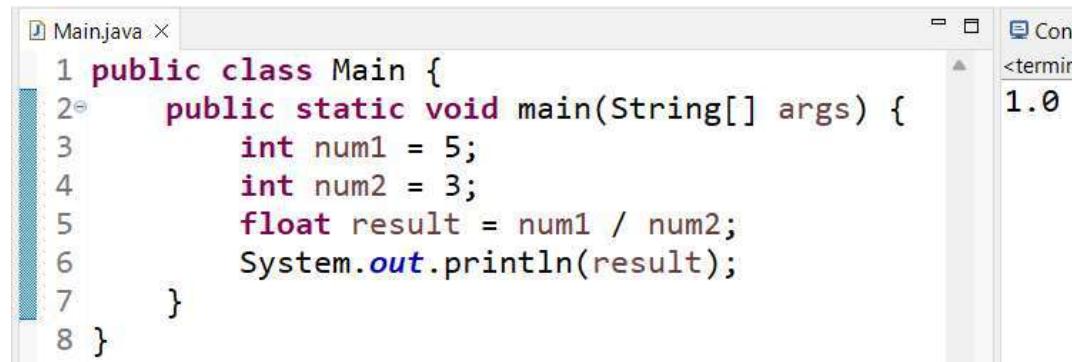
Problemas con operadores aritméticos

- En determinadas situaciones como por ejemplo restos o divisiones, es importante que tengamos en cuenta que la selección de un tipo de dato u otro, será determinante para obtener un resultado u otro. Vamos a ver unos ejemplos:
 - Si divido dos enteros y guardo el resultado en otro tipo de dato entero, el resultado será un entero. Por lo que, si por ejemplo divido 5 entre 3 vemos que el resultado es 1. Ya que, realiza la operación con enteros y desecha los decimales. En cambio, si hacemos dicha operación con una calculadora, vemos que el resultado es diferente.

The image displays two side-by-side windows. On the left is a Java code editor with a file named 'Main.java'. The code contains the following:1 public class Main {
2 public static void main(String[] args) {
3 int num1 = 5;
4 int num2 = 3;
5 int result = num1 / num2;
6 System.out.println(result);
7 }
8 }A cursor is positioned over the line 'int result = num1 / num2;'. To the right of the code editor is a calculator window titled 'Calculadora' set to 'Científica' mode. The display shows the calculation '5 ÷ 3 =' followed by the result '1,66666666666666666666666666666667'.

Problemas con operadores aritméticos

- Si intento realizar esto mismo con los operandos definidos sobre el tipo de dato entero (int), pero almaceno el resultado en un float, la operación realmente se realizará sobre enteros por lo que el resultado será 1. Y posteriormente, guardaremos dicho resultado sobre un float, pero, realmente el resultado de la operación será 1. Y, posteriormente, dicho número entero 1 se transformará a float obteniendo como resultante 1.0.



The screenshot shows a Java development environment. On the left, the code editor displays a file named 'Main.java' with the following content:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int num1 = 5;  
4         int num2 = 3;  
5         float result = num1 / num2;  
6         System.out.println(result);  
7     }  
8 }
```

On the right, there is a terminal window labeled 'Cons' with the output: '<termin> 1.0'. This demonstrates that the division of two integers results in an integer (1), which is then cast to a float, resulting in 1.0.

Problemas con operadores aritméticos

- Lo correcto que cuando realicemos divisiones y restos siempre utilicemos casteos o bien el mismo tipo de dato para todos los campos:

The image shows two side-by-side Java code editors and their respective consoles. On the left, the code uses integer variables num1 and num2, resulting in a floating-point division output of 1.6666666. On the right, the code uses float variables num1 and num2, resulting in a floating-point division output of 1.6666666.

```
>Main.java x Main.java x
1 public class Main {
2     public static void main(String[] args) {
3         int num1 = 5;
4         int num2 = 3;
5         float result = (float) num1 / num2;
6         System.out.println(result);
7     }
8 }
```

```
Console x <terminated> Main
1.6666666
```

```
>Main.java x Main.java x
1 public class Main {
2     public static void main(String[] args) {
3         float num1 = 5f;
4         float num2 = 3f;
5         float result = num1 / num2;
6         System.out.println(result);
7     }
8 }
```

```
Console x <terminated> Main
1.6666666
```

- Evitando así que se produzcan situaciones como las siguientes:

The image shows a Java code editor on the left with code that performs a modulus operation between an integer and a float, resulting in a floating-point remainder of 1.5999999. To the right, a scientific calculator application shows the division of 5 by 3, which is displayed as 1,66666666666666666666666666666667, illustrating the loss of precision due to type mismatch.

```
>Main.java x Main.java x
1 public class Main {
2     public static void main(String[] args) {
3         int num1 = 5;
4         float num2 = 1.7f;
5         float result = num1 % num2;
6         System.out.println(result);
7     }
8 }
```

```
Console x <terminated> Main
1.5999999
```

```
Calculadora
Científica
5 ÷ 3 =
1,66666666666666666666666666666667
```

Ejercicios de operadores aritméticos:

- Réstale a 10 5
- Haz una suma de 55 y 45
- Pon el número 3 en negativo
- Multiplica 5 por 5
- Haz una división de 25 entre 7 haciendo que el resultado sea un número entero
- Haz una división de 25 entre 7 haciendo que el resultado sea un número real
- Calcula el resto de dividir 25 entre 4

OPERADORES ARITMÉTICOS INCREMENTALES

++ y --

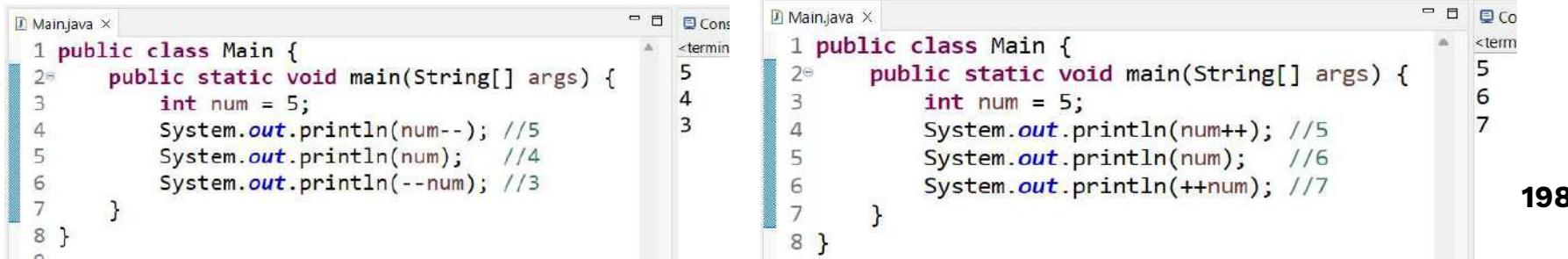


Operadores aritméticos incrementales: ++ y --

- Los operadores aritméticos incrementales son operadores unarios (ya que afectan a un solo operando) y nos permiten incrementar (con `++`) o decrementar (`--`) el valor de una variable.

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
<code>++</code>	Incremento <code>i++</code> primero se utiliza la variable y luego se incrementa su valor <code>++i</code> primero se incrementa el valor de la variable y luego se utiliza	<code>4++ a=5; b=a++; a=5; b=++a;</code>	<code>5 a vale 6 y b vale 5 a vale 6 y b vale 6</code>
<code>--</code>	decremento	<code>4--</code>	<code>3</code>

- Vamos a ver un ejemplo:



```
Main.java x Cons <termin
1 public class Main {
2     public static void main(String[] args) {
3         int num = 5;
4         System.out.println(num--); //5
5         System.out.println(num); //4
6         System.out.println(--num); //3
7     }
8 }
```

```
Main.java x Cons <termin
1 public class Main {
2     public static void main(String[] args) {
3         int num = 5;
4         System.out.println(num++); //5
5         System.out.println(num); //6
6         System.out.println(++num); //7
7     }
8 }
```

Ejercicios de Operadores aritméticos incrementales: ++ y --

- Define una variable global de tipo entero llamada numero inicializada con un valor de 0
- Haz una función que cada vez que llamemos a la función incrementar, incremente en un valor de la variable numero en 1 número en la misma instrucción
- Haz otra función llamada decrementar que reduzca el valor de un variable en 2 números en la siguiente instrucción

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
++	Incremento i++ primero se utiliza la variable y luego se incrementa su valor ++i primero se incrementa el valor de la variable y luego se utiliza	4++ a=5; b=a++; a=5; b=++a;	5 a vale 6 y b vale 5 a vale 6 y b vale 6
--	decremento	4--	3

OPERADORES ARITMÉTICOS COMBINADOS



200

Operadores aritméticos combinados

- Los operadores aritméticos combinados nos permiten combinar un operador aritmético con el operador de asignación (`=`). Los operadores que podemos utilizar son:
- Vamos a ver unos ejemplos:

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
<code>+=</code>	Suma combinada	<code>a+=b</code>	<code>a=a+b</code>
<code>-=</code>	Resta combinada	<code>a-=b</code>	<code>a=a-b</code>
<code>*=</code>	Producto combinado	<code>a*=b</code>	<code>a=a*b</code>
<code>/=</code>	División combinada	<code>a/=b</code>	<code>a=a/b</code>
<code>%=</code>	Resto combinado	<code>a%=b</code>	<code>a=a%b</code>

The screenshot shows a Java development environment. On the left, the code editor displays a file named Main.java with the following content:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         float num = 5f;  
4         System.out.println(num); //5.0  
5         System.out.println(num+=5); //10.0  
6         System.out.println(num-=5); //5.0  
7         System.out.println(num*=5); //25.0  
8         System.out.println(num/=2); //12.5  
9         System.out.println(num%=3.35); //2.45  
10    }  
11 }
```

On the right, the console window shows the output of the program:

```
<terminated>  
5.0  
10.0  
5.0  
25.0  
12.5  
2.45
```

A small '01' is visible in the bottom right corner of the slide.

Ejercicios de operadores aritméticos combinados

- Define una variable numérica entera de tipo integer y asignale un valor de 25.
 - 1. Utiliza e operador de suma combinada para sumar 5 al valor actual.
 - 2. Utiliza el operador de resta combinada para restar 15 al valor actual
 - 3. Utiliza el operador de producto combinado para multiplicar por 2 el valor actual
 - 4. Utiliza el operador de división combinada para dividir entre 2 el valor actual
 - 5. Utiliza el operador de resto combinado para saber el resto del valor actual dividido entre 3.

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
<code>+=</code>	Suma combinada	<code>a+=b</code>	<code>a=a+b</code>
<code>-=</code>	Resta combinada	<code>a-=b</code>	<code>a=a-b</code>
<code>*=</code>	Producto combinado	<code>a*=b</code>	<code>a=a*b</code>
<code>/=</code>	División combinada	<code>a/=b</code>	<code>a=a/b</code>
<code>%=</code>	Resto combinado	<code>a%=b</code>	<code>a=a%b</code>

OPERADORES DE RELACIÓN

`==, !=, <, >, <=, >=`



Operadores de relación

- Los operadores de relación nos permiten realizar comparaciones entre tipos primitivos u otros tipos de datos. Estas comparaciones tendrán siempre como resultado un valor booleano.

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
<code>==</code>	igual que	<code>7 == 38</code>	false
<code>!=</code>	distinto que	<code>'a' != 'k'</code>	true
<code><</code>	menor que	<code>'G' < 'B'</code>	false
<code>></code>	mayor que	<code>'b' > 'a'</code>	true
<code><=</code>	menor o igual que	<code>7.5 <= 7.38</code>	false
<code>>=</code>	mayor o igual que	<code>38 >= 7</code>	true

- Importante, en los operadores `<=` y `>=` si os fijáis, el igual siempre va en el lado derecho de la comparación.**

Operadores de relación: == y !=

- Vamos a ver unos ejemplos de los operadores de igualdad y de distintos que:

The screenshot shows a Java development environment with two panes. The left pane, titled 'Main.java', contains the following Java code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int num1 = 5, num2 = 5, num3 = 3;  
4  
5         // Operador de igualdad  
6         System.out.println(num1 + " ¿Es igual a " + num2 + "? " + (num1 == num2)); //true  
7         System.out.println(num2 + " ¿Es igual a " + num3 + "? " + (num2 == num3)); //false  
8         //Operador de no es igual  
9         System.out.println(num1 + " ¿No es igual a " + num2 + "? " + (num1 != num2)); //false  
10        System.out.println(num2 + " ¿No es igual a " + num3 + "? " + (num2 != num3)); //true  
11    }  
12 }
```

The right pane, titled 'Console', shows the output of the program:

```
<terminated> Main [Java Application] C:\Program Files\Java\Java SE Development Kit 8\bin>  
5 ¿Es igual a 5? true  
5 ¿Es igual a 3? false  
5 ¿No es igual a 5? false  
5 ¿No es igual a 3? true
```

- **Importante, no confundir el == con el operador de asignación =**

Operadores de relación: >, >=, < y <=

- Vamos a ver unos ejemplos de los operadores mayor que y mayor o igual que:

The screenshot shows a Java IDE interface. On the left, the code editor window titled "Main.java" contains the following Java code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int num1 = 5, num2 = 5, num3 = 3;  
4  
5         // Mayor que >  
6         System.out.println(num1 + " ¿Es mayor que " + num2 + "? " + (num1 > num2)); //false  
7         System.out.println(num2 + " ¿Es mayor que " + num3 + "? " + (num2 > num3)); //false  
8         //Mayor o igual >=  
9         System.out.println(num1 + " ¿Es mayor o igual que " + num2 + "? " + (num1 >= num2));//true  
10    }  
11 }
```

On the right, the "Console" window shows the output of the program:

```
<terminated> Main [Java Application] C:\Program File  
5 ¿Es mayor que 5? false  
5 ¿Es mayor que 3? true  
5 ¿Es mayor o igual que 5? true
```

- Vamos a ver unos ejemplos con los operadores menor que y menor o igual que:

The screenshot shows a Java IDE interface. On the left, the code editor window titled "Main.java" contains the following Java code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int num1 = 5, num2 = 5, num3 = 3;  
4  
5         // Menor que <  
6         System.out.println(num1 + " ¿Es menor que " + num2 + "? " + (num1 < num2)); //false  
7         System.out.println(num3 + " ¿Es menor que " + num2 + "? " + (num3 < num2)); //false  
8         //Menor o igual <=  
9         System.out.println(num1 + " ¿Es menor o igual que " + num2 + "? " + (num1 <= num2));//true  
10    }  
11 }
```

On the right, the "Console" window shows the output of the program:

```
<terminated> Main [Java Application] C:\Program Fil  
5 ¿Es menor que 5? false  
3 ¿Es menor que 5? true  
5 ¿Es menor o igual que 5? true
```

Ejercicios de operadores de relación

- Define 3 variables enteras num1, num2 y num3, num4 con los siguientes valores 1, 2, 3, 2
 - Comprueba si el valor de num4 es igual al de num2
 - Comprueba si el valor de num2 es igual al de num3
 - Comprueba si el valor de num1 es distinto a num3
 - Comprueba si el valor de num2 es distinto a num2
 - Comprueba si el valor de num4 es menor que num3
 - Comprueba si el valor de num3 es menor que num1
 - Comprueba si el valor de num1 es mayor que num3
 - Comprueba si el valor de num2 es mayor que num4
 - Comprueba si el valor de num2 es mayor que num3
 - Comprueba si el valor de num2 es igual o menor que num3
 - Comprueba si el valor de num2 es igual o menor que num4
 - Comprueba si el valor de num2 es igual o menor que num1
 - Comprueba si el valor de num1 es mayor o igual que num2
 - Comprueba si el valor de num2 es mayor o igual que num1
 - Comprueba si el valor de num2 es igual o mayor que num2

OPERADORES LÓGICOS O BOOLEANOS

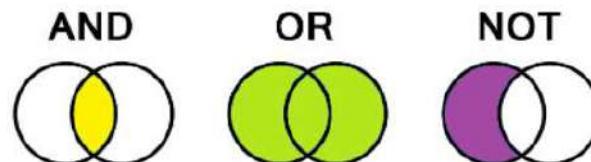
!, |, ^, &, ||, &&



Operadores lógicos o booleanos

- Los operadores booleanos, realizan operaciones sobre tipos de datos booleanos y nos permiten obtener como resultado un valor booleano. Vamos a ver los diferentes operadores:

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
!	Negación - NOT (unario)	<code>!false !(5==5)</code>	true false
	Suma lógica – OR (binario)	<code>true false (5==5) (5<4)</code>	true true
^	Suma lógica exclusiva – XOR (binario)	<code>true ^ false (5==5) (5<4)</code>	true true
&	Producto lógico – AND (binario)	<code>true & false (5==5) & (5<4)</code>	false false
	Suma lógica con cortocircuito: si el primer operando es true entonces el segundo se salta y el resultado es true	<code>true false (5==5) (5<4)</code>	true true
&&	Producto lógico con cortocircuito: si el primer operando es false entonces el segundo se salta y el resultado es false	<code>false && true (5==5) && (5<4)</code>	false false



Operadores lógicos o booleanos: ! y | (OR)

- Vamos a comenzar con el operador de negación ! y el operador de suma lógica |:

The screenshot shows a Java IDE interface with the following components:

- Main.java**: The source code for a Java class named Main. It contains methods for printing the results of logical operations.
- Bitwise OR (|):** A truth table for the Bitwise OR operation (|). It shows the result for all combinations of inputs A and B (0 or 1).

A	B	A B
1	1	1
1	0	1
0	1	1
0	0	0
- Logical OR (||):** A truth table for the Logical OR operation (||). It shows the result for all combinations of inputs A and B (true or false).

A	B	A B
T	T	T
T	F	T
F	T	T
F	F	F
- Console**: The output window showing the results of the printed statements. It lists the output for each row of the tables and then for the bitwise OR examples.

```
1 public class Main {  
2     public static void main(String[] args) {  
3         boolean interruptor = false;  
4  
5         // Operador de negación !  
6         System.out.println(!interruptor); // Invierte el valor del boolean  
7         // Operador de suma lógica |  
8         System.out.println(false | true); // false + true = true  
9         System.out.println(true | false); // true + false = true  
10        System.out.println(true | true); // true + true = true  
11        System.out.println(false | false); // false + false = false  
12  
13        System.out.println(0 | 1); // 0 + 1 = 1  
14        System.out.println(1 | 0); // 1 + 0 = 1  
15        System.out.println(1 | 1); // 1 + 1 = 1  
16        System.out.println(0 | 0); // 0 + 0 = 0  
17    }  
18 }
```

Console Output:

```
true  
true  
true  
true  
false  
1  
1  
1  
0
```

Operadores lógicos o booleanos: | (OR)

- Lo que realmente hace el operador es realizar sumas de la siguiente manera:

1001 1100=1000				
DECIMAL	BINARIO			
9	1	0	0	1
12	1	1	0	0
13	1	0	0	1

Bitwise OR (|):

A	B	A B
1	1	1
1	0	1
0	1	1
0	0	0

Logical OR (||):

A	B	A B
T	T	T
T	F	T
F	T	T
F	F	F

The screenshot shows a Java code editor with a file named Main.java. The code defines a class Main with a main method. It declares two integer variables num1 and num2, initializes them to 9 and 12 respectively, and then prints their bitwise OR result. Following this, it demonstrates parsing a binary string "1101" back into a decimal integer using Integer.parseInt.

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int num1 = 9; // 1001 = 9  
4         int num2 = 12; //1100 = 12  
5         //           1101 = 13  
6         System.out.println(num1 | num2); // 1001 + 1100 = 1  
7  
8         // Parsing the binary value (1101) to decimal  
9         String binaryString="1101";  
10        int decimal=Integer.parseInt(binaryString,2);  
11        System.out.println(decimal);  
12    }  
13 }
```

Operadores lógicos o booleanos: ^ (XOR)

- Operador suma lógica exclusiva XOR (binario) será true solamente cuando uno de ellos sea true y el otro sea false. Vamos a ver unos ejemplos:

Bitwise NOT (~):

A	$\sim A$
1	0
0	1

Logical NOT (!):

A	$\sim A$
T	F
F	T

A	B	$A \wedge B$
1	1	0
1	0	1
0	1	1
0	0	0

A	B	$A \wedge B$
T	T	F
T	F	T
F	T	T
F	F	F

```
Main.java X:
1 public class Main {
2     public static void main(String[] args) {
3         // Operador de suma lógica |
4         System.out.println(false ^ true); // false + true = true
5         System.out.println(true ^ false); // true + false = true
6         System.out.println(true ^ true); // true + true = false
7         System.out.println(false ^ false); // false + false = false
8
9         System.out.println(0 ^ 1); // 0 + 1 = 1
10        System.out.println(1 ^ 0); // 1 + 0 = 1
11        System.out.println(1 ^ 1); // 1 + 1 = 0
12        System.out.println(0 ^ 0); // 0 + 0 = 0
13    }
14 }
```

```
Console X
<terminated> I
true
true
false
false
1
1
0
0
```

Operadores lógicos o booleanos: ^ (XOR)

- Lo que realmente hace el operador es realizar sumas de la siguiente manera:

1001^1100=1000				
DECIMAL	BINARIO			
9	1	0	0	1
12	1	1	0	0
5	0	1	0	1

Bitwise XOR (^):

A	B	A^B
1	1	0
1	0	1
0	1	1
0	0	0

Logical XOR (^):

A	B	A^B
T	T	F
T	F	T
F	T	T
F	F	F

The screenshot shows a Java IDE interface. On the left, a code editor window titled "Main.java" contains the following Java code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int num1 = 9; //1001  
4         int num2 = 12; //1100  
5         int result = num1 ^ num2; //0101  
6         System.out.println(result); // 1001^1100=0101  
7     }  
8 }
```

To the right of the code editor is a terminal window showing the output of the program: "5".

Operadores lógicos o booleanos: & (AND)

- Lo que realmente hace el producto lógico & su funcionamiento es el siguiente:

1001&1100=1000				
DECIMAL	BINARIO			
9	1	0	0	1
12	1	1	0	0
8	1	0	0	0

Bitwise AND (&):

A	B	A&B
1	1	1
1	0	0
0	1	0
0	0	0

Logical AND (&&):

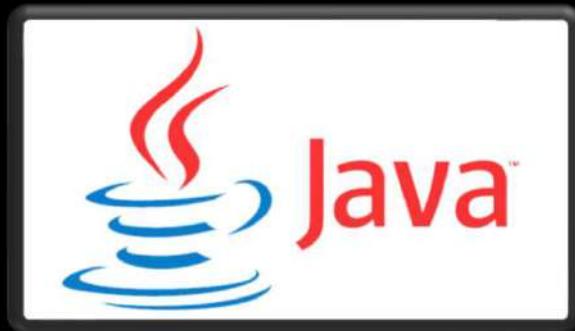
A	B	A&&B
T	T	T
T	F	F
F	T	F
F	F	F

```
Main.java ×
1 public class Main {
2     public static void main(String[] args) {
3         int num1 = 9; //1001
4         int num2 = 12; //1100
5         int result = num1 & num2; //1000
6         System.out.println(result); // 1001^1100=1000
7     }
8 }
```

Ejercicios de operadores lógicos/booleanos

- Define 3 variables booleanas bool1, bool2, bool3 y bool4 con los valores true, false, false
 - Compara si bool1 es igual al valor invertido de bool2
 - Compara si bool2 es igual al valor invertido de bool3

SUMA LÓGICA II CON CORTOCIRCUITO



Operadores lógicos o booleanos: || (OR)

- Si copiamos el siguiente código:

The screenshot shows a Java code editor window titled "Main.java". The code is as follows:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println(true || true);  
4     }  
5 }
```

A tooltip is displayed over the line "System.out.println(true || true);". The tooltip title is "Dead code" (highlighted with a red box). It contains three quick fix options:

- Remove
- @ Add @SuppressWarnings 'unused' to 'main()'
- Configure problem severity

At the bottom right of the tooltip, it says "Press 'F2' for focus".

- Vemos como el IDE (Eclipse) nos indica que tenemos Dead code. Esto nos puede ayudar a entender mejor el funcionamiento de un suma lógica con cortocircuito.

Operadores lógicos o booleanos: || (OR)

- El operador | (con un solo |) evaluará todas las condiciones. En cambio, el operador || con un doble | trabaja con lo que se conoce como cortocircuito. El motivo es el rendimiento.
- Ya que si una de las condiciones devuelve true, inmediatamente después del primer true se devolverá el resultado true omitiendo el análisis del resto de condiciones.

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
!	Negación - NOT (unario)	!false !(5==5)	true false
	Suma lógica – OR (binario)	true false (5==5) (5<4)	true true
^	Suma lógica exclusiva – XOR (binario)	true ^ false (5==5) ^ (5<4)	true true
&	Producto lógico – AND (binario)	true & false (5==5) & (5<4)	false false
	Suma lógica con cortocircuito: si el primer operando es true entonces el segundo se salta y el resultado es true	true false (5==5) (5<4)	true true
&&	Producto lógico con cortocircuito: si el primer operando es false entonces el segundo se salta y el resultado es false	false && true (5==5) && (5<4)	false false

Bitwise OR (|):

A	B	A B
1	1	1
1	0	1
0	1	1
0	0	0

Logical OR (||):

A	B	A B
T	T	T
T	F	T
F	T	T
F	F	F

Operadores lógicos o booleanos: || (OR)

- Por ello, si por ejemplo, hacemos lo siguiente:

The screenshot shows an IDE interface with a code editor and a console window. The code editor contains the following Java code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println(false || true || false);  
4     }  
5 }
```

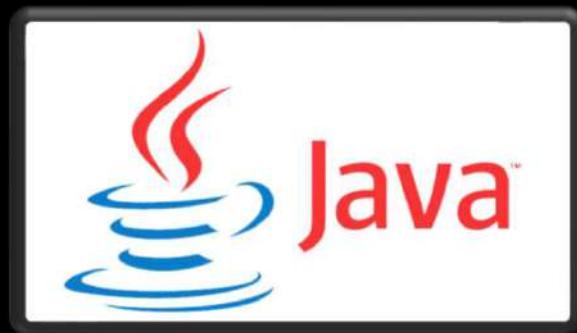
The console window shows the output: **true**. A tooltip is displayed over the line `false || true || false`, indicating it is **Dead code** (código muerto). The tooltip also lists three quick fixes available:

- Remove
- @ Add @SuppressWarnings('unused') to 'main()'
- Configure problem severity

Press 'F2' for focus

- Podemos ver que, a partir del primer true, nos vuelve a decir que tenemos Dead code (código muerto).

PRODUCTO LÓGICO && CON CORTOCIRCUITO



220

Operadores lógicos o booleanos: && (AND)

- Si copiamos el siguiente código:

The screenshot shows the Eclipse IDE interface. On the left is the code editor with a file named "Main.java". The code contains a single-line print statement: `System.out.println(false && true);`. To the right is the "Console" view, which displays the output: `false`. A yellow tooltip box is overlaid on the code editor, indicating "Dead code" and providing three quick fixes: "Remove", "Add @SuppressWarnings 'unused' to 'main()'", and "Configure problem severity".

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println(false && true);  
4     }  
5 }  
6  
7  
8  
9
```

- Vemos como el IDE (Eclipse) nos indica que tenemos Dead code. Esto nos puede ayudar a entender mejor el funcionamiento de un producto lógico con cortocircuito.

Operadores lógicos o booleanos: && (AND)

- El operador & (con un solo &) evaluará todas las condiciones. En cambio, el operador && con un doble & trabaja con lo que se conoce como cortocircuito. El motivo es el rendimiento.
- Ya que, si una de las condiciones devuelve false, inmediatamente después del primer false se devolverá el resultado false omitiendo el análisis del resto de condiciones.

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
!	Negación - NOT (unario)	!false !(5==5)	true false
	Suma lógica – OR (binario)	true false (5==5) (5<4)	true true
^	Suma lógica exclusiva – XOR (binario)	true ^ false (5==5) ^ (5<4)	true true
&	Producto lógico – AND (binario)	true & false (5==5) & (5<4)	false false
	Suma lógica con cortocircuito: si el primer operando es true entonces el segundo se salta y el resultado es true	true false (5==5) (5<4)	true true
&&	Producto lógico con cortocircuito: si el primer operando es false entonces el segundo se salta y el resultado es false	false && true (5==5) && (5<4)	false false

Bitwise AND (&):

A	B	A&B
1	1	1
1	0	0
0	1	0
0	0	0

Logical AND (&&):

A	B	A&&B
T	T	T
T	F	F
F	T	F
F	F	F

Operadores lógicos o booleanos: || (OR)

- Por ello, si por ejemplo, hacemos lo siguiente:

The screenshot shows an IDE interface with two tabs: 'Main.java' and 'Console'. The code in Main.java is:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println(true && false && true);  
4     }  
5 }
```

The 'Console' tab shows the output: 'false'. A tooltip is displayed over the line 'System.out.println(true && false && true);', indicating it is 'Dead code' with three quick fixes available:

- ✖ Remove
- ⓘ Add @SuppressWarnings 'unused' to 'main()'
- ⚙ Configure problem severity

Press 'F2' for focus

- Podemos ver que a partir del primer false, nos vuelve a decir que tenemos Dead code (código muerto).

OPERADOR CONDICIONAL

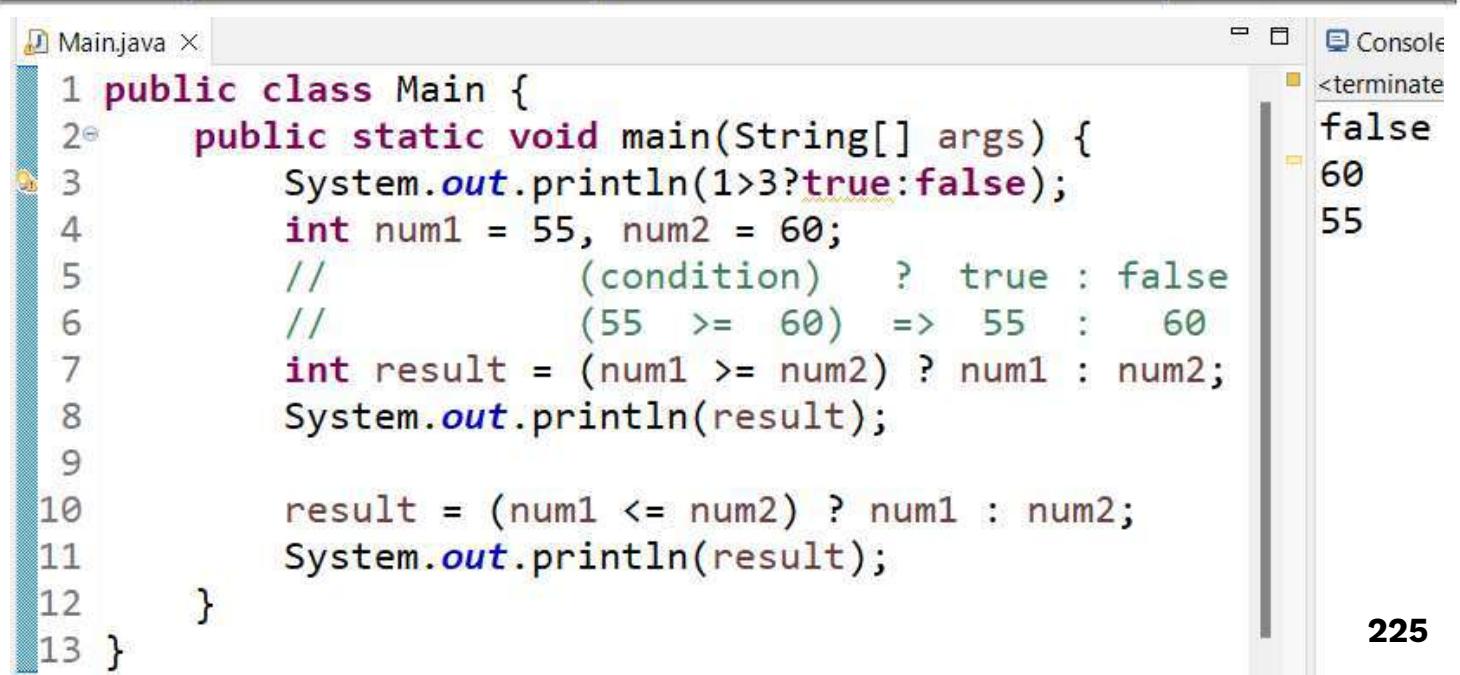
?:



Operador condicional: ?:

- Nos permite devolver valores en función de una expresión lógica. Vamos a ver unos ejemplos:

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
?:	operador condicional	a = 4; b = a == 4 ? a+5 : 6-a; b = a > 4 ? a*7 : a+8;	b vale 9 b vale 12



The screenshot shows an IDE interface with a code editor and a console window. The code editor contains a Java file named Main.java with the following content:

```
Main.java x
1 public class Main {
2     public static void main(String[] args) {
3         System.out.println(1>3?true:false);
4         int num1 = 55, num2 = 60;
5         // (condition) ? true : false
6         // (55 >= 60) => 55 : 60
7         int result = (num1 >= num2) ? num1 : num2;
8         System.out.println(result);
9
10        result = (num1 <= num2) ? num1 : num2;
11        System.out.println(result);
12    }
13 }
```

The console window shows the output of the program:

```
Console
<terminate
false
60
55
```

Ejercicios de operadores lógicos/booleanos

- Calcula si un número es par o impar mediante al operador condicional. Y, finalmente, imprime “par” o “impar” por terminal en función del número introducido.
- Realiza un programa que nos imprima blanco o negro en función de un Math.random mediante al operador booleano.

OPERADORES DE BITS

~, |, ^, &, <<, >>, >>>



Operadores de bits:

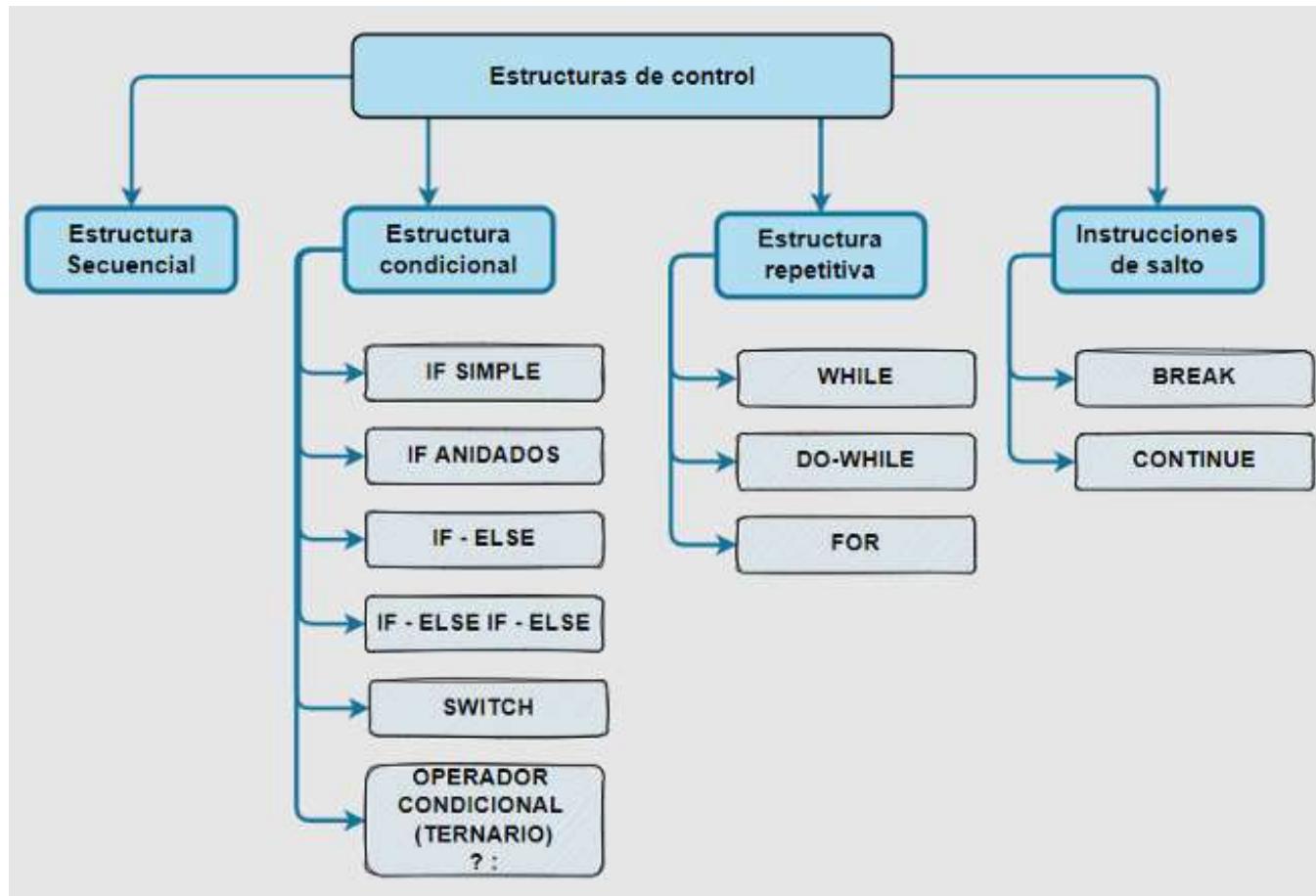
- Los operadores de bits se utilizan para realizar la manipulación de bits individualmente.
- También son conocidos como operadores lógicos de bit, y también son conocidos como bitwise o operaciones lógicas de bit.

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
<code>~</code>	Negación ó complemento binario (unario)	<code>~12</code>	-13
<code> </code>	Suma lógica binaria – OR (binario)	<code>12 10</code>	8
<code>^</code>	Suma lógica exclusiva – XOR (binario)	<code>12 ^ 10</code>	6
<code>&</code>	Producto lógico binario – AND (binario)	<code>12 & 10</code>	14
<code><<</code>	Desplaza a la izquierda los bits del 1º operando tantas veces como indica el 2º operando (por la derecha siempre entra un cero)	<code>7<<2 -7<<2</code>	28 -28
<code>>></code>	Desplaza a la derecha los bits del 1º operando tantas veces como indica el 2º operando (por la izquierda entra siempre el mismo bit más significativo anterior)	<code>7>>2 -7>>2</code>	1 -2
<code>>>></code>	Desplaza a la derecha los bits del 1º operando tantas veces como indica el 2º operando – sin signo (por la izquierda entra siempre un cero).	<code>7>>>2 -7>>>2</code>	1 1073741822

ESTRUTURAS DE CONTROL



Estructuras de control

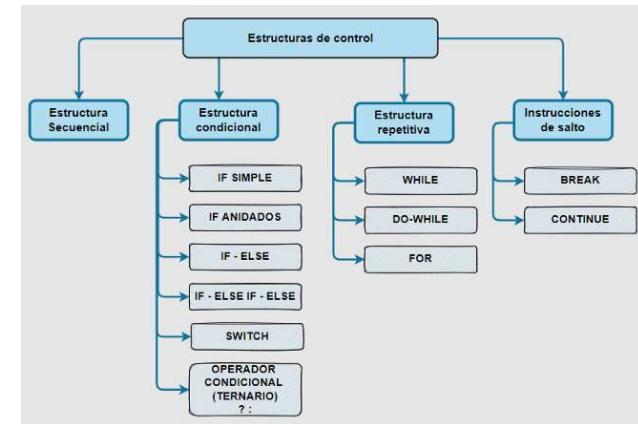




231

Estructuras de control

- Las estructuras de control, pueden ser de varios tipos:
 - **Estructuras secuenciales:** ejecutan las instrucciones una a una de arriba abajo.
 - **Estructuras condicionales:** dependen de una o varias condiciones y nos permiten ejecutar y/o no ejecutar un bloque de instrucciones que será englobado entre {} en función de si dichas instrucciones son verdaderas o falsas.
 - **Estructuras repetitivas:** permiten ejecutar un bloque de instrucciones varias veces.
 - **Instrucciones de salto:** permiten terminar la iteración y/o el bucle al completo.
- Para obtener más información, os aconsejo la lectura del siguiente artículo que escribí para Medium: <https://medium.com/@davidbernalgonzalez/programaci%C3%B3n-declarativa-en-java-786751686e70>



EJERCICIOS DE CONDICIONALES



Ejercicios de estructuras condicionales

- 1. Realiza un programa que a partir de un número aleatorio (0 o 1) muestre un valor booleano true or false e imprima rojo o negro.
 - Para ello, primeramente utiliza ifs (unidireccionales) sin else.
 - Posteriormente, pásalo a if-else (bidireccional)
 - Y, finalmente, pásalo a un operador ternario
- 2. Realiza un programa que muestre la temperatura y la categoría de un clima en función de la temperatura que le pasemos:
 - Para ello, primeramente utiliza ifs (unidireccionales) sin else
 - Y, posteriormente, utiliza la estructura if-elseif-else

```
Main.java X
1 package entradaSalida;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         double temperatura = 26.5;
7     }
8 }
```

Console X Problems Debug Shell
<terminated> Main () [Java Application] C:\Program Files\Java\jdk
Climas cálidos
Tipo Desértico

Temperatura	Categoría del clima	Tipo de clima
menor de 10 °C	Climas frios	Polar - Cuando es menor que 5 Alta montaña - Cuando es mayor o igual que 5 pero menor que 10
igual o mayor que 10 °C y menor que 20 °C	Climas templados	Oceánico - Cuando es mayor o igual que 10 pero menor que 13,5 Mediterráneo - Cuando es mayor o igual que 13,5 pero menor que 16,5 Continental - Cuando es mayor o igual que 16,5 pero menor que 20
igual o mayor que 20 °C	Climas cálidos	Ecuatorial - Cuando es mayor o igual que 20 pero menor que 23,5 Tropical - Cuando es mayor o igual que 23,5 pero menor que 26,5 Desértico - Cuando es mayor o igual a 26,5

Ejercicios de estructuras condicionales

- 3. Crea un programa mediante a un Switch case que cuando introduzcamos un valor del 1 al 9 en formato numérica imprima dicho número a texto.
- 4. Transforma el código del siguiente if-elseif-else
 - <https://gist.githubusercontent.com/DavidBernalGonzalez/c6bc57a00fccc4dbb5558faaa0031207/raw/fb0e9c38b0c86dc320b5bcc85ca1ba36ba3af81c/Main.java> a un operador ternario:

Main.java

```
1 package entradaSalida;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         double numero = 9;
```

Console X Problems Debug Shell

```
<terminated> Main (1) [Java Application] C:\Program Files\Java\j...  
Nueve
```

Main.java

```
1 public class Main {
2     public static void main(String[] args) {
3         double number = 55;
4
5         if (number > 0) {
6             System.out.println("¡Es positivo!");
7         } else if (number < 0) {
8             System.out.println("¡Es negativo!");
9         } else {
10             System.out.println("¡Es cero, na de ná!");
11         }
12     }
13 }
```

Console X

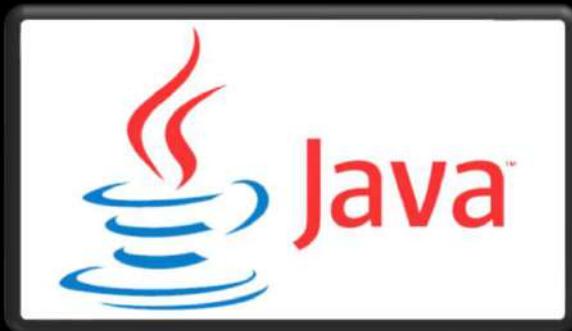
```
<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (28 abr. 2022 17:47:19 - 17  
¡Es positivo!
```

Ejercicios de estructuras condicionales

- 5. Realiza el siguiente árbol mediante a bucles:

```
**
 ***
 ****
 *****
 ******
 ******
 *****
 ****
 ***
```

EJERCICIOS DE ESTRUTURAS REPETITIVAS

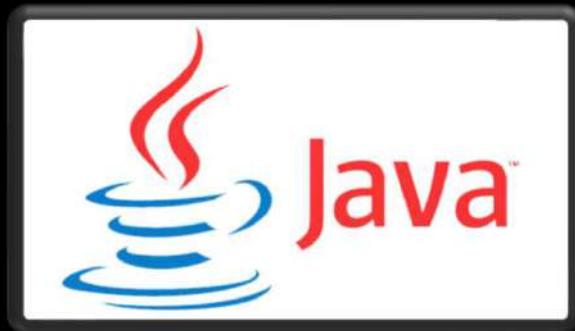


Ejercicios de estructuras repetitivas

- 1. Realiza un programa que muestre solamente los números que sean divisible por 2.
- 2. Escribe un programa que a partir de un array de números por pantalla, muestre los que sean divisibles por 3.
 - El ejercicio se realizará con los bucles FOR y WHILE
- 3. Realiza un bucle que salude al menos una vez (independientemente de que se cumpla una condición o no). Y que posteriormente, envíe tantos saludos como veces le indiquemos en una variable.
 - El ejercicio se realizará con los bucles FOR , WHILE y DO WHILE
- 4. Haz un bucle que vaya eliminando las letras del abecedario hasta que solamente tengamos una A. Y, ves guardando dichas letras en otra variable y una vez llegues a la A, vuelve a hacer el inverso mostrando AB, ABC...

ZYWXVUTSRQPONMLKJIHGfedcba
YWVXVUTSRQPONMLKJIHGfedcba
WXVUTSRQPONMLKJIHGfedcba
XVUTSRQPONMLKJIHGfedcba
VUTSRQPONMLKJIHGfedcba
UTSRQPONMLKJIHGfedcba
TSRQPONMLKJIHGfedcba
SRQPONMLKJIHGfedcba
RQPONMLKJIHGfedcba
QPONMLKJIHGfedcba
PONMLKJIHGfedcba
ONMLKJIHGfedcba
NMLKJIHGfedcba
MLKJIHGfedcba
LKJIHGfedcba
KJIHGfedcba
JIHGfedcba
IHGFEDCBA
HGFEDCBA
GFEDCBA
FEDCBA
EDCBA
DCBA
CBA
BA
A

EJERCICIOS DE INSTRUCCIONES DE SALTO



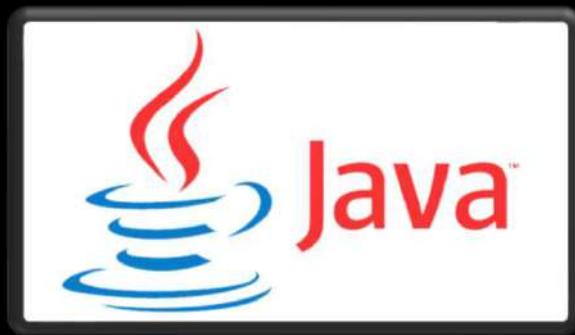
Ejercicios de instrucciones de salto

- 1. Realiza un programa que si el bucle llega a la 3^a iteración (loop) te heche fuera del bucle.
 - Deberás de realizar el programa mediante a los bucles FOR y WHILE.
- Realiza un programa que muestre un número (que salga de la iteración) y un texto (cualquiera) por ejemplo Iteración. Y si el bucle llega a la 5^a iteración (loop) no imprima el mensaje y te heche fuera del loop pero no del bucle.

FUNCIONES/MÉTODOS EN JAVA

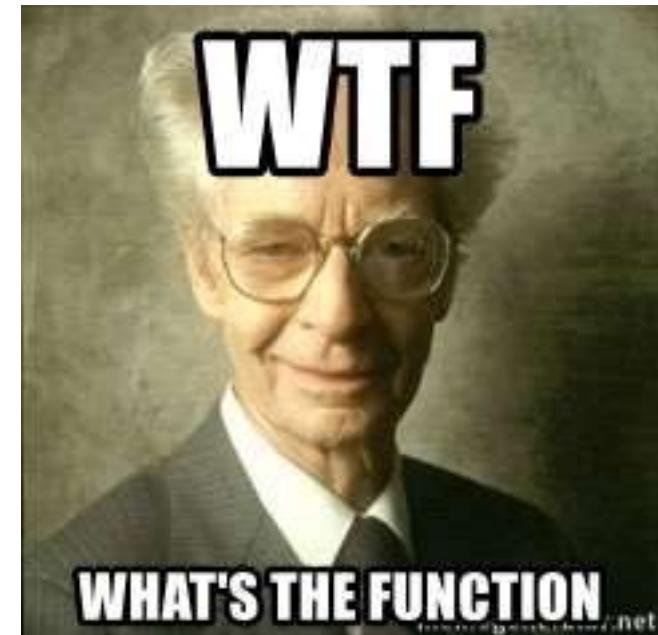


¿QUÉ SON LAS FUNCIONES?



¿Qué son las funciones?

- Las funciones nos permiten dividir el código en distintos bloques delimitados por {} lo que se conoce como programación funcional.
- Los principales beneficios de la programación funcional son:
 - La reutilización
 - Reducir el código duplicado
 - Estructurar mejor el código de nuestros programas, etc.



TIPOS DE FUNCIONES



Tipos de funciones

- Partes de un función:

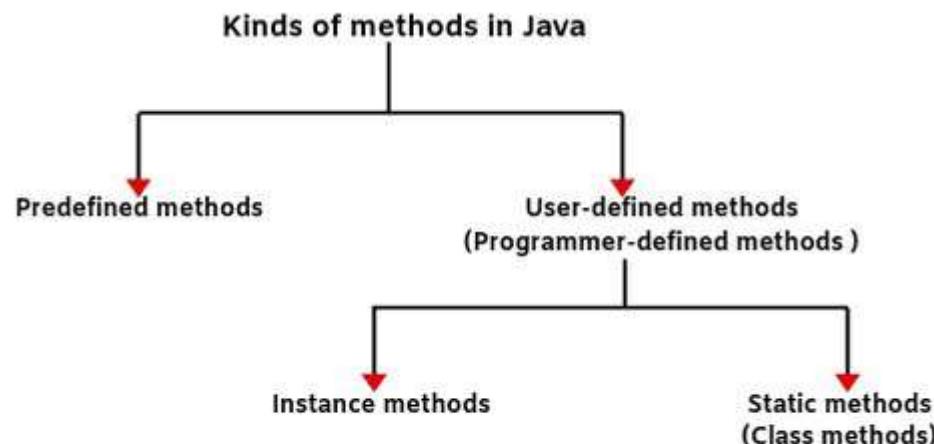
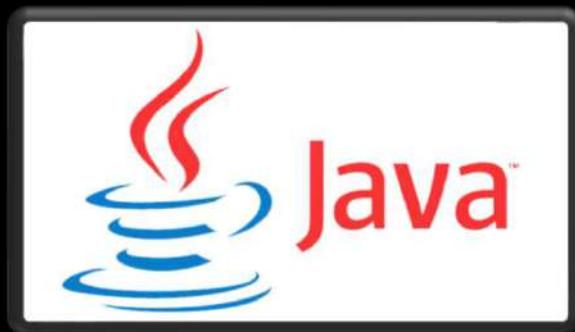


Fig: Basic kinds of methods in Java

¿FUNCIONES CON O SIN STATIC?



Ejemplo de functions con static y sin

- Dentro de las funciones tenemos:
 - Las funciones sin static: necesitan crear la clase a excepción que dicha clase sea static
 - Las funciones con static: podemos acceder a ellas directamente.
- En este caso en particular ambas funciones retornan void (vacío).
- Para ejecutar la función es necesario realizar la llamada de dicha función.
Vamos a ver un ejemplo:

The screenshot shows an IDE interface with two panes. The left pane, titled 'Main.java', contains Java code. The right pane, titled 'Console', shows the output of the program. The code in 'Main.java' is as follows:

```
1 package com.main;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Main main = new Main();
7         main.saludarA("javer@s!");
8
9         saludoGenerico();
10    }
11
12    // Forma 1: Cuando NO asignamos static al método
13    public void saludarA(String nombre){ ///
14        System.out.println("Hola " + nombre);
15    }
16
17    // Forma 2: Cuando asignamos static al método
18    public static void saludoGenerico(){
19        System.out.println("Hola!");
20    }
21 }
```

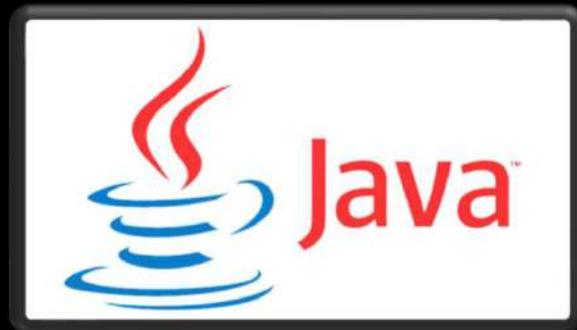
The 'Console' pane shows the output: 'Hola javer@s!' followed by 'Hola!'. The line 'main.saludarA("javer@s!");' is highlighted with a red box, and the entire block starting with 'Forma 1' is also highlighted with a red box. The line 'System.out.println("Hola!");' is highlighted with a blue box, and the entire block starting with 'Forma 2' is highlighted with a blue box.

Ejemplo de functions con static y sin

- **Forma 1 (sin static):** necesita realizar la creación del objeto para poder utilizar sus métodos. Aunque la palabra más correcta para referirnos al proceso de crear un objeto en una clase es instanciar, que es sacar un objeto de la Clase. La clase sería el molde del que sacamos dichos objetos.
- **Forma 2 (con static):** no necesita realizar la creación (instanciación) del objeto para poder utilizar los métodos. Por lo que podemos acceder directamente a ellos.

```
Main.java
1 package entradaSalida;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         // Forma 1
7         Main main = new Main(); //Instanciamos es decir declaramos/construimos la clase a la que pertenece dicho método
8         main.saludarA("David"); //Llamamos al método correspondiente
9     }
10
11     // Forma 2: Cuando el método tiene static
12     saludоГenerico();
13 }
14
15     // Forma 1: Cuando NO asignamos static al método
16     public void saludarA(String nombre){ ///
17         System.out.println("Hola " + nombre);
18     }
19
20     // Forma 2: Cuando asignamos static al método
21     public static void saludоГenericо(){
22         System.out.println("Hola!");
23     }
24 }
```

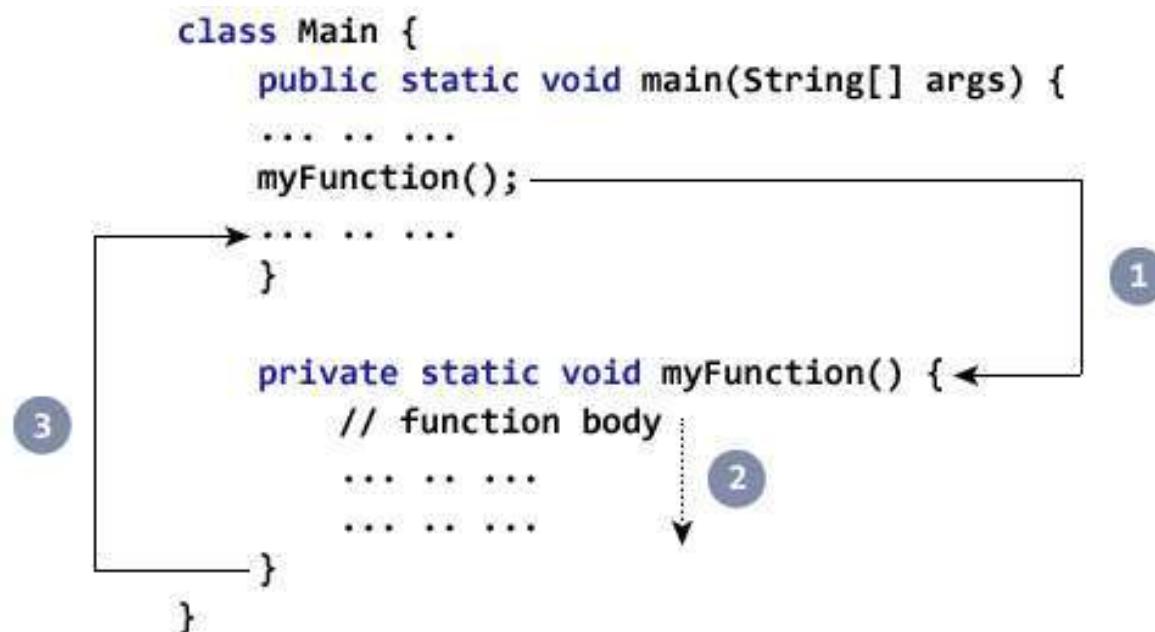
LLAMANDA A UNA FUNCIÓN



Ejemplo de llamada a una función

- Para ejecutar la función es necesario realizar la llamada de dicha función.

Vamos a ver un ejemplo:



FUNCTION CON RETURN



Function con return

- El return nos permite definir lo que nos va a devolver una función.
- La función devuelve el resultado pero no imprime nada salvo que hagamos un sysout, etc.
- Vamos a ver un ejemplo:

The screenshot shows an IDE interface with three panes. The left pane displays the code for `Main.java`. The middle pane shows the `Main.java` file with some code highlighted in red. The right pane shows the `Console` output.

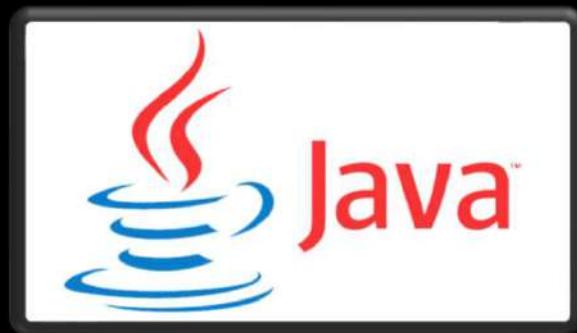
```
1 package com.main;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Main main = new Main();
7         main.saludarA("javer@s!");
8
9         saludoGenerico();
10    }
11
12    public String saludarA(String nombre) { // return "Hola " + nombre;
13    }
14
15
16    // Forma 2: Cuando asignamos static al método
17    public static String saludoGenerico() { return "Hola!";
18    }
19
20 }
```

```
1 package com.main;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Main main = new Main();
7         System.out.println(main.saludarA("javer@s!"));
8
9         System.out.println(saludoGenerico());
10    }
11
12    public String saludarA(String nombre) { // return "Hola " + nombre;
13    }
14
15
16    // Forma 2: Cuando asignamos static al método
17    public static String saludoGenerico() {
18        return "Hola!";
19    }
20 }
```

`<terminated> Main (2)`

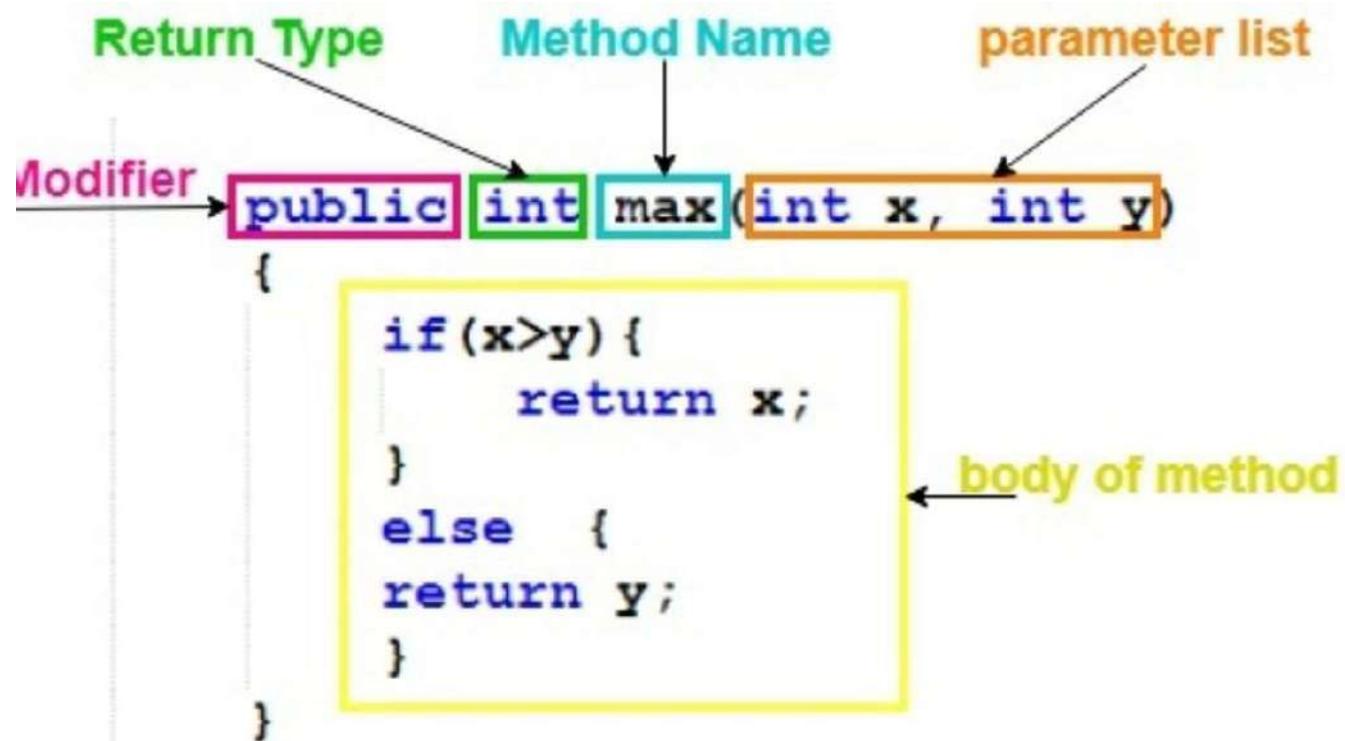
Hola javer@s!
Hola!

PARTES DE UNA FUNCIÓN



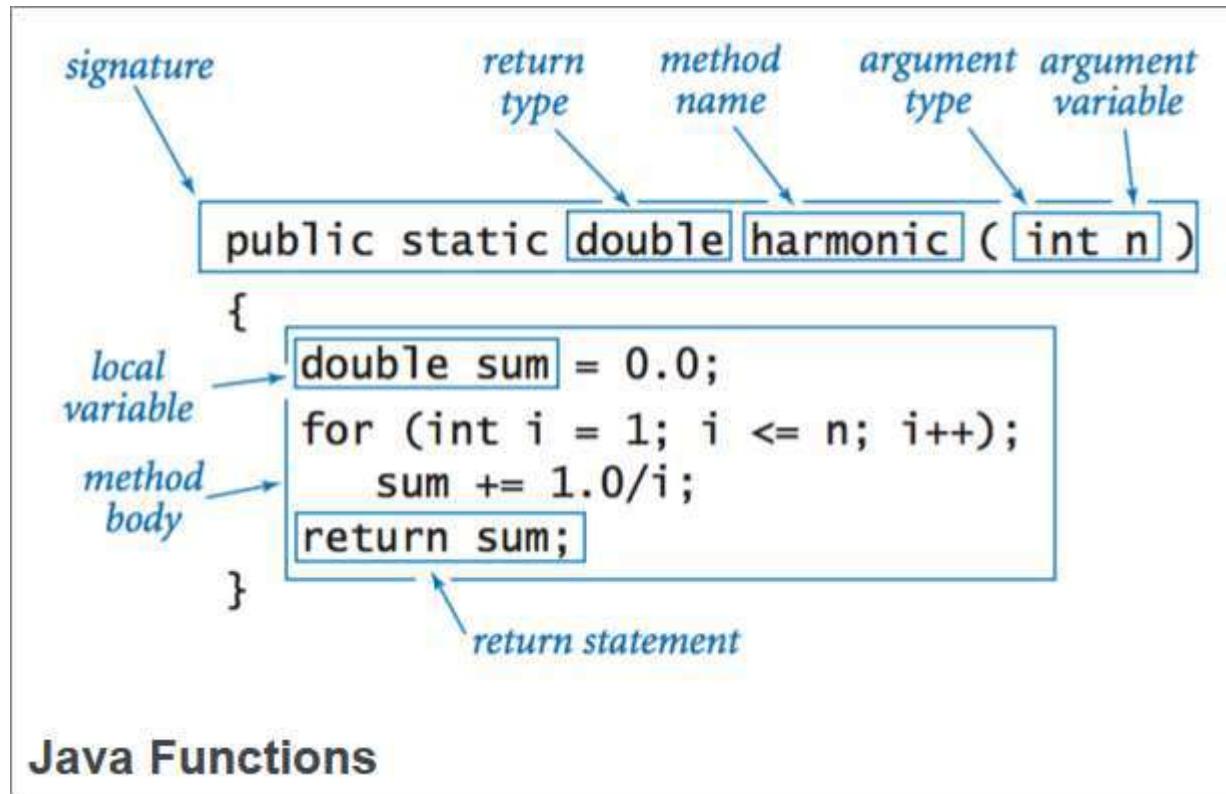
Partes de una función

- Partes de un función:



Partes de una función

- Partes de un función:



FUNCIONES CON VARIOS RETURNS



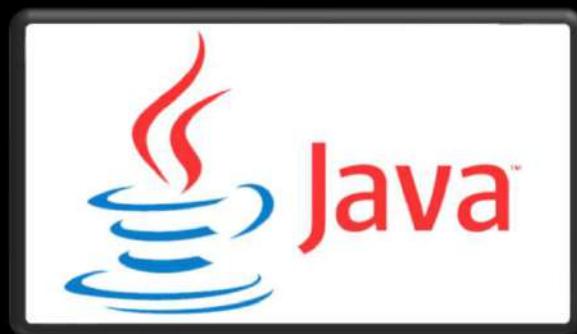
Funciones con varios returns

- Existe la posibilidad de romper una función en distintos puntos. Vamos a ver un ejemplo:

The screenshot shows an IDE interface with two windows. On the left, the code editor window titled 'Main.java X' displays Java code. On the right, the 'Console X' window shows the output of the program. A red box highlights the line 'System.out.println(main.saludarA("javer@s!"));' and a blue box highlights the line 'System.out.println(main.saludarA(""));'. Red arrows point from these highlighted lines to their corresponding outputs in the console: 'Hola javer@s!' and 'Hola desconocido' respectively.

```
1 package com.main;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Main main = new Main();
7         System.out.println(main.saludarA("javer@s!"));
8         System.out.println(main.saludarA(""));
9     }
10
11     public String saludarA(String nombre) {
12         if(nombre.isEmpty() || nombre.isBlank()) {
13             return "Hola desconocido";
14         } else {
15             return "Hola " + nombre;
16         }
17     }
18 }
```

INSTRUCCIONES DESPUÉS DE UN RETUN



Instrucciones después de un return

- Una vez tenemos un return “se acaba el mundo”, es decir, una vez se retorna un valor se sale de la función. Por ello, si intentamos meter algo después de un return, vemos que aparece un mensaje de Unreachable code (código inalcanzable) y que cuando ejecutamos el programa no compilará bien

The screenshot shows an IDE interface with two windows. On the left is the code editor window titled "Main.java" containing the following Java code:

```
1 package com.main;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Main main = new Main();
7         System.out.println(main.saludarA("javer@s!"));
8     }
9
10    public String saludarA(String nombre) {
11        return "Hola " + nombre;
12        System.out.println("Test");
13    }
14 }
15
16
17
```

A tooltip is displayed over the line "System.out.println("Test");" at line 12, stating "Unreachable code" with "1 quick fix available" and a "Remove" option.

On the right is the "Console" window showing the output of the program's execution:

```
<terminated> Main (2) [Java Application] C:\Exception in thread "main"
Unreachable code
at com.main.Main.s
at com.main.Main.m
```

Ejercicios de funciones

- 1. Haz una función llamada `javaOrNull()` y que si le pasamos `java` como parámetro retorne `java` y en caso contrario, retorne `null`.
- 2. Realiza una función llamada `login` que acepte dos parenteros y en función de el usuario es “`admin`” y la pasword “`1234`” o no. Retorné “`Login true`” o “`Login false`”;
- 3. Realiza una función que aleatoriamente calcule si un número es par o impar y finalmente nos retorne `true` (cuando sea par) o `false` (cuando sea impar).



A screenshot of a Java console window titled "Console". The window shows the command "java" followed by "null" on a new line, indicating the output of a function call.

MENSAJES DE CONSOLA CON COLORES



Antes de las interfaces graficas (CLI)

- En los inicios de la informática era muy común que las aplicaciones se ejecutaran bajo la Command Line Interface (CLI), es decir, sobre la terminal/línea de comandos.
- Con el tiempo, y la evolución de la informática, se empezaron a añadir colores a las terminales mediante a las secuencias de escape ANSI. Con esto, siempre que el Terminal fuera compatible con dicha secuencias de escape, podríamos ver autenticas “obras de arte” y hasta juegos de terminal.



¿Qué son las secuencias de escape ANSI?

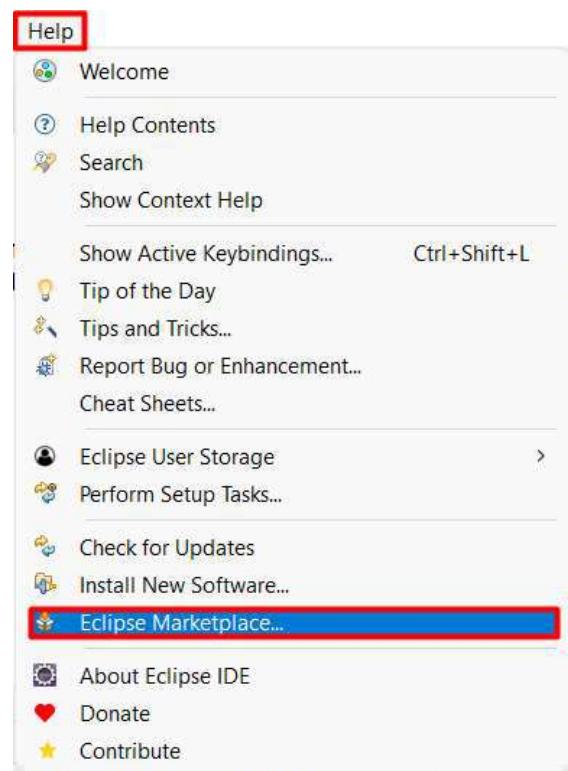
- Mediante a la secuencias de escape ANSI se puede enviar cierta información de control a la consola la cual nos permitirá cambiar ciertos atributos sobre el texto representado. Los atributos modificables son:
 - El estilo del texto: normal, claro, subrayado, parpadeante, inverso y oculto
 - El color del fondo: rojo, amarillo, verde, azul...
 - El color del texto: rojo, amarillo, verde, azul...
- El código de escape ANSI puede ser representado de distintas maneras:
 - \033 (octal)
 - \u001b (unicode)

CONFIGURANDO ECLIPSE PARA QUE SOPORTE ANSI



¿Qué son las secuencias de escape ANSI?

- Para que los colores se visualicen en Eclipse necesitamos instalar el plugin ANSI Escape in Console. Para ello, realizamos lo siguiente:



A screenshot of the Eclipse Marketplace interface. The search bar at the top contains 'ANSI escape'. Below it, the 'ANSI Escape in Console' plugin is listed. It has a green icon with a white letter 'A', a rating of 451, and 74,3K installs. A button labeled 'Installed' is visible. The plugin description states: 'This Eclipse plugin interprets the ANSI escape sequences to color the console output. It works for output text with escape sequences directly from Java, Groovy,...'. Below the description, there is a link to 'more info'. At the bottom of the screenshot, a preview of the Eclipse Console shows the text 'Texto Color rojo' in red, demonstrating the plugin's functionality.

265

APLICANDO FORMATOS EN TEXTOS CON ANSI



Formateo

- Dentro de los códigos ANSI, tenemos distintos códigos para realizar determinadas acciones sobre un texto. Vamos a ver un ejemplo:

Style	Code
Bold	\x1B[1m
Faint	\x1B[2m
<i>Italic</i>	\x1B[3m
<u>Underlined</u>	\x1B[4m
Inverse	\x1B[7m
Strikethrough	\x1B[9m

The screenshot shows a Java application running in an IDE. The code in Main.java prints two lines of text to the console using System.out.println. The first line uses the \x1B[1m code to make the text bold, and the second line uses the \x1B[4m code to make the text underlined.

```
Main.java X
1 public class Main {
2     public static void main(String[] args) {
3         System.out.println("\x1B[1mTexto negrita (bold)\x1B[0m");
4         System.out.println("\x1B[4mTexto subrayado (underline)\x1B[0m");
5     }
6 }
```

Console X
<terminated> Main [Java Application] C:\Program File
Texto negrita (bold)
Texto subrayado (underline)

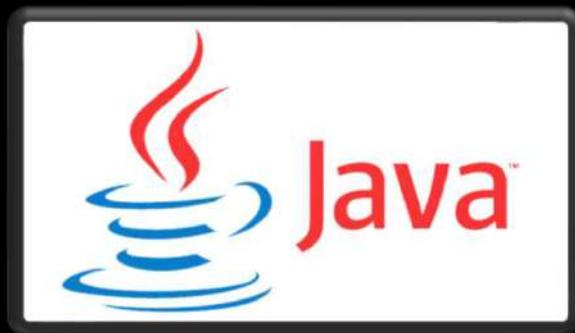
CATEGORIAS DE COLORES ANSI



¿Qué son las secuencias de escape ANSI?

- Cuando queremos añadir colores con colores en ANSI, tenemos varias maneras de hacerlo. Vamos a ver las principales opciones que podemos utilizar para trabajar con colores mediante a ANSI y sus principales diferencias:
 - 8 colores: los colores son muy reducidos, solamente tenemos 8. Y, estos colores, son primarios cuyos colores destacan por ser bastante “agresivos” (puros).
 - 16 de colores: nos ofrece 8 colores también, aunque dichos colores no son primarios sino brillantes. Por lo que, son más suaves. Aunque, como en la versión de 8 colores solamente tenemos 8 colores.
 - 256 colores: nos ofrece muchos colores (256 colores) por lo que es la manera más completa de trabajar con colores en ANSI.

TRABAJANDO CON 8 COLORES



Background y text color con 8 colores

- Dentro de los códigos ANSI, cuando trabajamos con 8 colores, tenemos 8 códigos distintos para trabajar con colores. Estos colores utilizan distintos números para determinar si queremos modificar el background o el color de un texto. Vamos a verlos :
 - Foreground Color: utiliza los valores del 30 al 39
 - Background Color: utiliza los valores del 40 al 49
 - Reset: utilizan el 0

The screenshot shows a Java IDE interface. On the left, the code editor displays a file named Main.java with the following content:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         // Ejemplo con formato Unicode: \u001b  
4         System.out.println("\u001b[0;31mTexto de color rojo\u001b[0m");  
5         System.out.println("Texto sin formato");  
6         System.out.println("\u001b[0;41mTexto subrayado\u001b[0m");  
7         // Ejemplo con formato Octal: \033  
8         System.out.println("\033[0;32mTexto de color rojo\033[0m");  
9         System.out.println("Texto sin formato");  
10        System.out.println("\033[0;42mTexto subrayado\033[0m");  
11    }  
12}
```

On the right, the console window shows the output of the program. It displays several lines of text with different colors and styles:

```
Texto de color rojo  
Texto sin formato  
Texto subrayado  
Texto de color rojo  
Texto sin formato  
Texto subrayado
```

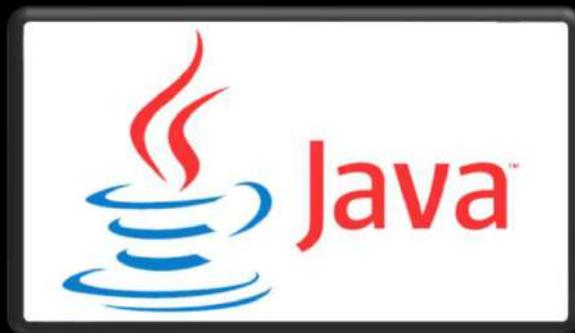
8-16 Colors

Color Name	Foreground Color Code	Background Color Code
Black	30	40
Red	31	41
Green	32	42
Yellow	33	43
Blue	34	44
Magenta	35	45
Cyan	36	46
White	37	47
Default	39	49
Reset	0	0

Partes de un código ANSI

- La estructura de un código ANSI está formada por:
 - El código mediante al cual le indicamos al compilador que vamos a escribir una secuencia ANSI
 - Se pueden utilizar \033 o \u001
 - Seguidamente, ponemos un [para separar la zona en la que indicamos que vamos a trabajar con una secuencia ASCII de la zona en la que definimos los grupos (números)
 - Posteriormente, en la zona de grupos: se definen uno o varios números separados por un ;
 - Y, finalmente, se escribe un carácter m para indicar al compilador que vamos a cerrar la escape
- \033[1; 31; 42m
- \u001[1; 31; 42m

TRABAJANDO CON 16 COLORES



Background y text color con 16 colores

- Dentro de los códigos ANSI, tenemos distintos códigos para referirnos a los colores en función de si queremos trabajar con el color de fondo o el de la fuente:
 - Foreground Color:** utiliza los valores del 90 al 97
 - Background Color:** utiliza los valores del 100 al 107
 - Reset:** utilizan el 0

Color Name	Foreground Color Code	Background Color Code
Bright Black	90	100
Bright Red	91	101
Bright Green	92	102
Bright Yellow	93	103
Bright Blue	94	104
Bright Magenta	95	105
Bright Cyan	96	106
Bright White	97	107

The screenshot shows a Java development environment. On the left, the code editor displays a file named Main.java with the following content:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         //Set foreground color.  
4         System.out.println("\u033[94mTrabajando con 16 colors\u033[0m");  
5         //Set background color.  
6         System.out.println("\u033[103mTrabajando con 16 colors\u033[0m");  
7         //Set foreground color + set background color.  
8         System.out.println("\u033[95;106mTrabajando con 16 colors\u033[0m");  
9     }  
10 }
```

On the right, the console window shows three lines of output text in different colors: cyan, magenta, and blue.

TRABAJANDO CON 256 COLORES (8BITS)



Background y text color con 256 colores

- Dentro de los códigos ANSI, tenemos distintos códigos para referirnos a los colores en función de si queremos trabajar con el color de fondo o el de la fuente:

- Foreground Color:**
utiliza los valores del 0 al 255
- Background Color:**
utiliza los valores del 0 al 255
- Reset:** utilizan el 0

\$ ansi --color-codes													
Standard:		0	1	2	3	4	5	6	7				
Intense:		8	9	10	11	12	13	14	15				
16	17	18	19	20	21	34	35	36	37	38	39		
52	53	54	55	56	57	70	71	72	73	74	75		
88	89	90	91	92	93	106	107	108	109	110	111		
124	125	126	127	128	129	142	143	144	145	146	147		
160	161	162	163	164	165	178	179	180	181	182	183		
196	197	198	199	200	201	214	215	216	217	218	219		
22	23	24	25	26	27	40	41	42	43	44	45		
58	59	60	61	62	63	76	77	78	79	80	81		
94	95	96	97	98	99	112	113	114	115	116	117		
130	131	132	133	134	135	148	149	150	151	152	153		
166	167	168	169	170	171	184	185	186	187	188	189		
202	203	204	205	206	207	220	221	222	223	224	225		
28	29	30	31	32	33	46	47	48	49	50	51		
64	65	66	67	68	69	82	83	84	85	86	87		
100	101	102	103	104	105	118	119	120	121	122	123		
136	137	138	139	140	141	154	155	156	157	158	159		
172	173	174	175	176	177	190	191	192	193	194	195		
208	209	210	211	212	213	226	227	228	229	230	231		
Grays:		232	233	234	235	236	237	238	239	240	241	242	243
\$		244	245	246	247	248	249	250	251	252	253	254	255

Background y text color con 256 colores

- Para trabajar con 256 colores, podemos utilizar las siguientes maneras:
- Vamos a ver un ejemplo con cada una de ellas:

34	35	36	37	38	39
70	71	72	73	74	75
106	107	108	109	110	111
142	143	144	145	146	147
178	179	180	181	182	183
214	215	216	217	218	219
40	41	42	43	44	45
76	77	78	79	80	81
112	113	114	115	116	117
148	149	150	151	152	153
184	185	186	187	188	189
220	221	222	223	224	225

Color	Font code	Background code
Any palette color (with V in [0-255])	\x1B[38;5;Vm	\x1B[48;5;Vm
Any RGB color (with values in [0-255])	\x1B[38;2;R;G;Bm	\x1B[48;2;R;G;Bm

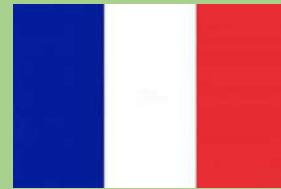
```
Main.java x
1 public class Main {
2     public static void main(String[] args) {
3         // 256 colors: PALETTE COLOR FORMAT
4         System.out.println("\033[38;5;43mTexto con palette color y 256 colores\033[0m");
5         System.out.println("\033[48;5;74mTexto trabajando con 256 colores\033[0m");
6         // 256 colors: RGB COLOR FORMAT
7         System.out.println("\033[38;5;153;204;255mTexto con RGB y 256 colores\033[0m");
8         System.out.println("\033[48;5;153;204;255mTexto con RGB y 256 colores\033[0m");
9     }
10 }
```

Console x
<terminated> Main [Java Application] C:\Program Files\Java\jdk
Texto trabajando con 256 colores
Texto trabajando con 256 colores
Texto trabajando con 256 colores
Texto trabajando con 256 colores

Hex: # 99CCFF
Red: 153
Green: 204
Blue: 255

Ejercicios de banderas con colores ANSI

- Define constantes para los valores RESET, RED, YELLOW... Tanto para el tipo de color de fondo (background), colores de texto (color), tipos de textos, etc.
- Dibuja las siguientes banderas con los códigos de colores ANSI:



- Pon nombres a cada una de las banderas en subrayado, negrita y un subrayado.
- Haz una función que te transforme un texto a un formato pasándole parámetros de background, color de texto y tipo de texto.
- Haz una función que limpie el formato de un texto pasándole un string como parámetro.

Bucles de colores

- Haz un “arcoíris” bucle que aleatoriamente aplica un color de texto, de background y un formato al texto (cursiva, negrita, italic...).

Pintando el árbol

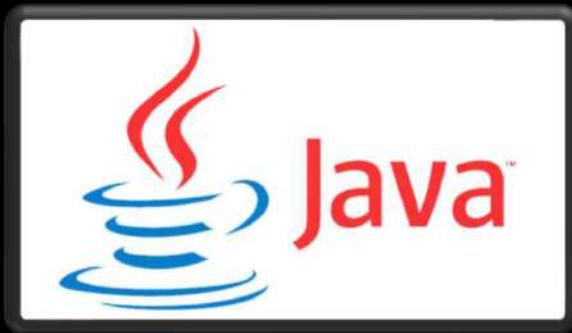
- Pinta el “arbolito” del ejercicio anterior:



WRAPPERS CLASS (TIPO OBJETO)

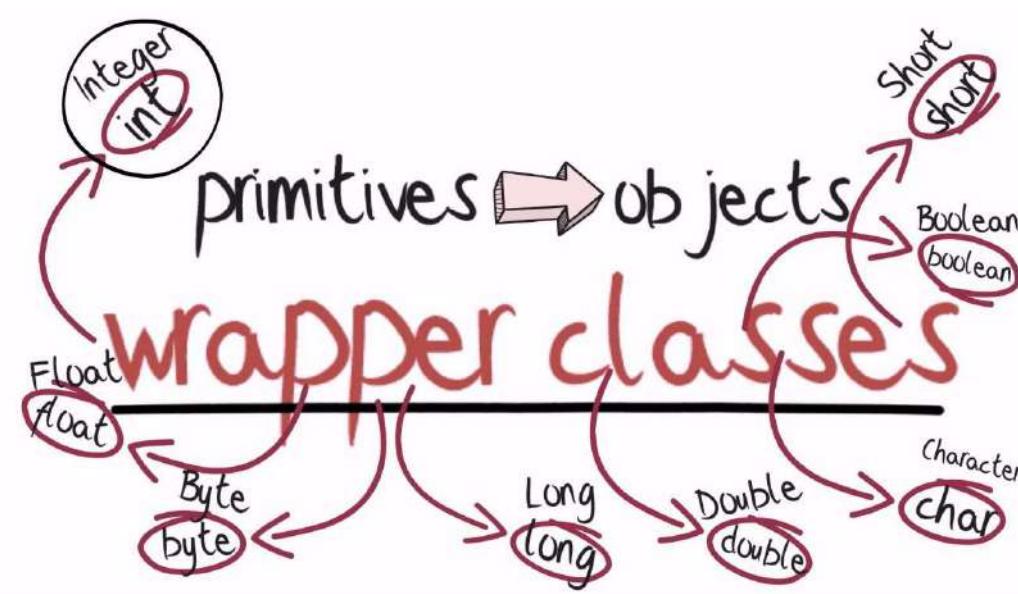


¿QUÉ SON LOS WRAPPERS?

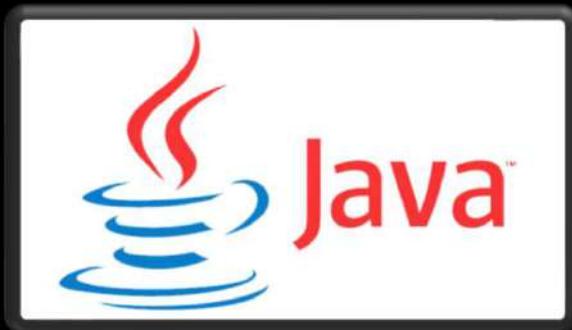


¿Qué son los Wrappers?

- Los wrappers, también son conocidos como envoltorios ya que envuelven un tipo primitivo con la finalidad de transformar un dato en tipos primitivos (que no tiene métodos) a un tipo Objeto (con sus correspondientes métodos).



DIFERENCIAS ENTRE TIPOS PRIMITIVOS Y WRAPPERS



Los tipos primitivos no tienen métodos

Tipos de datos primitivos				
Nombre	Tipo	Tamaño	Valor por defecto	Rango
boolean	Lógico	1 bit	false	true-false

- Si trabajamos directamente con un tipo de dato primitivo como por ejemplo boolean, y no con su clase Wrapper, Boolean. Podemos ver que no tenemos métodos y que, por tanto, no se nos permitirá trabajar con métodos ya que estos no estarán disponibles. Solamente las podremos utilizar para asignar un valor a una variable.

The screenshot shows a Java code editor with a file named "Main.java". The code is as follows:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         boolean interruptor = false;  
4         System.out.println(interruptor.);  
5     }  
6 }
```

The line "System.out.println(interruptor.);" is highlighted in blue, indicating it is being typed. A tooltip "No Default Proposals" appears over the period at the end of the println statement, confirming that no method suggestions are available for the primitive type boolean.

Los tipos no primitivos (objetos) si que tienen métodos

Tipos de datos primitivos				
Nombre	Tipo	Tamaño	Valor por defecto	Rango
boolean	Lógico	1 bit	false	true-false

- El rango de los boolean como solo tiene un tamaño de un bit, solamente puede almacenar dos valores TRUE o FALSE.
- Si queremos utilizar métodos relacionados con los métodos primitivos utilizamos las clases Wrappers que si que tienen métodos disponibles.

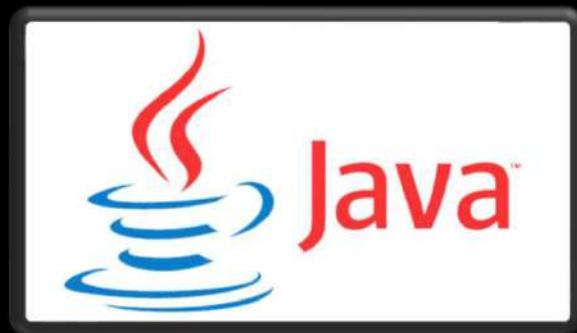
The screenshot shows an IDE interface with a code editor and a Javadoc tooltip. The code editor contains a Main.java file with the following code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println(Boolean.FALSE);  
4         System.out.println(Boolean.TRUE);  
5     }  
6 }
```

The Javadoc tooltip for the Boolean class is displayed above the code editor. It includes methods like compare, getBoolean, hashCode, logicalAnd, logicalOr, logicalXor, parseBoolean, toString, and class. Two specific methods are highlighted with red boxes: FALSE and TRUE. The FALSE method is described as returning Boolean.FALSE and the TRUE method as returning Boolean.TRUE.

The IDE status bar at the bottom shows: Problems, Declaration, Console, <terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (18 abr). The console output shows the printed values: false and true.

TIPOS DE WRAPPERS



Tipos de Wrappers

- Cada tipo primitivo tiene su Wrapper específico. Si nos fijamos, la diferencia entre el tipo primitivo y un Wrapper es que la primera letra del Wrapper siempre irá en mayúscula en mayúscula y la del tipo primitivo no. Vamos a verlas:

Tipos primitivos (no son objetos y por tanto no poseen métodos)	Wrappers (son objeto y por tanto poseen métodos)
byte	Byte
short	Short
int	Integer
long	Long
boolean	Boolean
float	Float
double	Double
char	Character



int

Integer

RENDIMIENTO DE TIPOS PRIMIVOS VS WRAPPERS



¿Por qué no trabajamos directamente con Wrappers?

- Los primitivos han estado presentes desde los orígenes de Java en 1996. A día de hoy, los tipos primitivos siguen utilizándose principalmente por el notable rendimiento que ofrecen.
- Aunque es un tema que ha generado una gran controversia al respecto ya que existen grandes detractores de los tipos primitivos como, por ejemplo, Simon Ritter (responsable de promover el uso de las tecnologías de SUN) pese a jugar un rol muy importante en Oracle no consiguió eliminar quien quería eliminarlos en el lanzamiento de la versión 10 de JDK desde hace tiempo.
- El motivo por el cual siguen siendo usados y que les ha permitido sobrevivir durante tantos años es simple, el rendimiento. Los tipos primitivos no contienen métodos lo que supone un enorme beneficio en lo referente a su rendimiento.

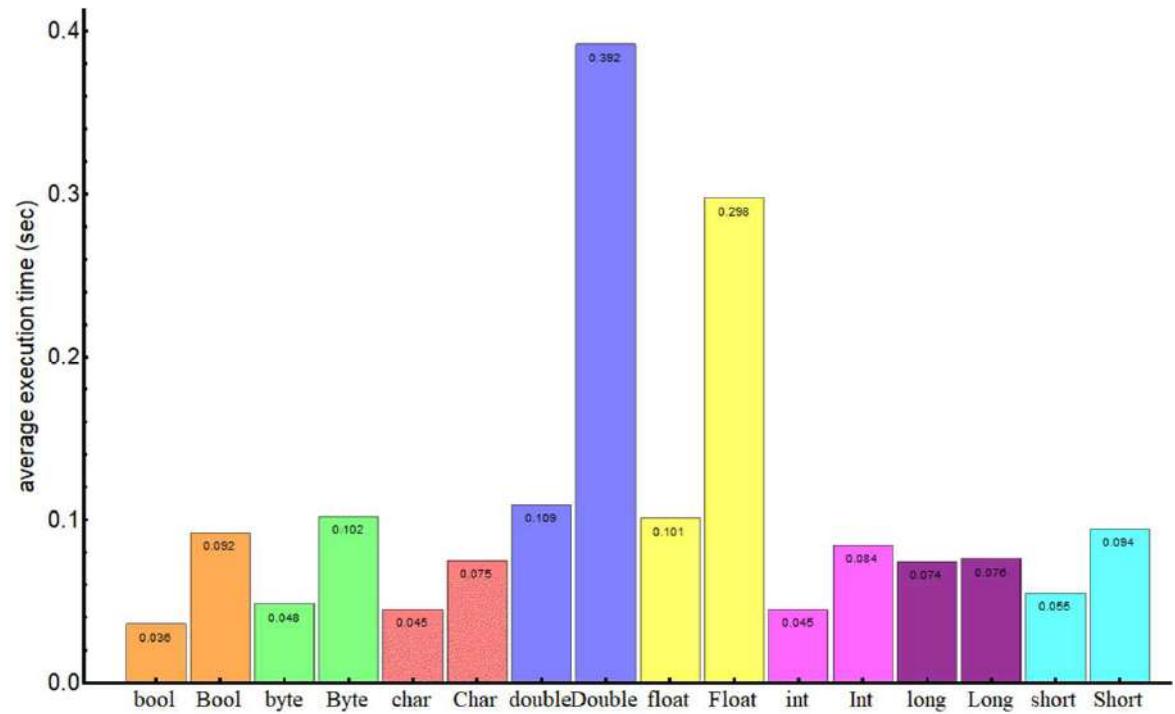
The slide has a red header bar with the text "Vision: Language Features". Below it is a portrait of a man. To the right of the portrait is a bulleted list of Java features:

- Large data support (JDK 9)
 - Large arrays (64 bit support)
- Unified type system (JDK 10+)
 - No more primitives, make everything objects
- Other type reification (JDK 10+)
 - True generics
 - Function types
- Data structure optimizations (JDK 10+)
 - Structs, multi-dimensional arrays, etc
 - Close last(?) performance gap to low-level languages

At the bottom right of the slide is the Oracle logo.

¿Porque Java trabaja con Wrappers y tipos primitivos?

- El motivo por el que los tipos primitivos no tienen métodos es el rendimiento. Tal y como podemos apreciar en la siguiente table, los Wrappers precisamente tienen un rendimiento mucho menor que el que tienen los tipos primitivos. Ese es realmente el motivo real por lo que se trabaja con métodos primitivos.



Los tipos no primitivos (objetos) si que tienen métodos

- He preparado el siguiente ejemplo en el que comparo el rendimiento de un bucle que recorre el rango de los Integers:
 - El primer caso trabaja con el Wrapper de long (Long)
 - El segundo caso trabaja con el tipo primitivo long sin wrappers
- La diferencia entre ambos vemos que es bastante elevada 5536ms en el caso del Wrapper y 630ms en el caso del tipo primitivo.

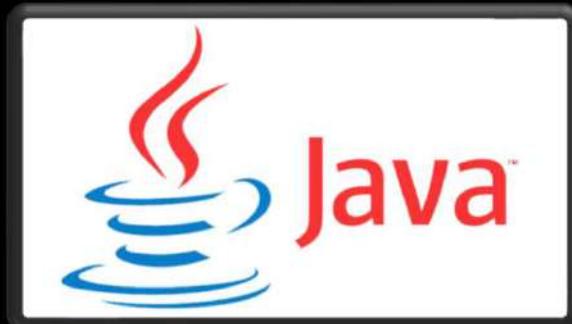
The screenshot shows an IDE interface with two tabs: 'MainJava' and 'Console'. The 'MainJava' tab displays the following Java code:

```
1 public class Main {
2     public static void main(String[] args) {
3         long start = System.currentTimeMillis();
4         long finish;
5         Long noPrimitivo = 0L;
6         for (long i = 0; i < Integer.MAX_VALUE; i++) {
7             noPrimitivo += i;
8         }
9         finish = System.currentTimeMillis();
10        System.out.println("Rendimiento cuando trabajamos con Float (objeto): " + (finish - start) + "ms");
11
12        start = System.currentTimeMillis();
13        long primitivo = 0L;
14        // Integer.MAX_VALUE = 2147483647
15        for (long i = 0; i < Integer.MAX_VALUE; i++) {
16            primitivo += i;
17        }
18        finish = System.currentTimeMillis();
19        System.out.println("Rendimiento cuando trabajamos con float (primitivo): " + (finish - start) + "ms");
20    }
21 }
```

The 'Console' tab shows the output of the program:

```
<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (28 abr, 2022 18:32:30 - 18:32:36) [pid: 22332]
Rendimiento cuando trabajamos con Float (objeto): 5536ms
Rendimiento cuando trabajamos con float (primitivo): 630ms
```

CALCULANDO EL RANGO DE UN TIPO PRIMITIVO CON WRAPPERS





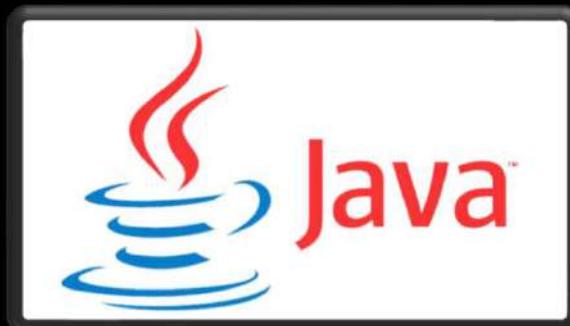
Ejercicio

- Haz un array en el que guardaras Mediante al uso de wrappers y secuencias de escape, haz una tabla que combine ambas tablas. Intenta aplicar colores, formatos, etc.

```
Console X
<terminated> IdentificadoresJava [Java Application]
El tipo de datos: byte
Está formado por: 1 Bytes
Un byte está formado por: 8 Bits
Su Rango va del: -128 a 127

Console X
<terminated> IdentificadoresJava [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (18 oct. 2017 15:43:40)
Rango byte desde: -128 hasta: 127
-----
Rango short desde: -32768 hasta: 32767
-----
Rango int desde: -2147483648 hasta: 2147483647
-----
Rango long desde: -9223372036854775808 hasta: 9223372036854775807
-----
Rango float desde: 1.4E-45 hasta: 3.4028235E38
-----
Rango double desde: 4.9E-324 hasta: 1.7976931348623157E308
```

PROFUNDIZANDO EN LOS WRAPPERS



Los wrappers son objects

- Todos los Wrappers a su vez dependen de Java.lang.Object ya que, como hemos dicho, los Wrappers no dejan de ser objetos y por tanto descienden de la clase Object situada en el package java.lang.
- Si nos metemos en la [API de Java](#), y buscamos el package java.lang, podemos ver que nos aparecen los distintos tipos de Wrappers:

Package java.lang

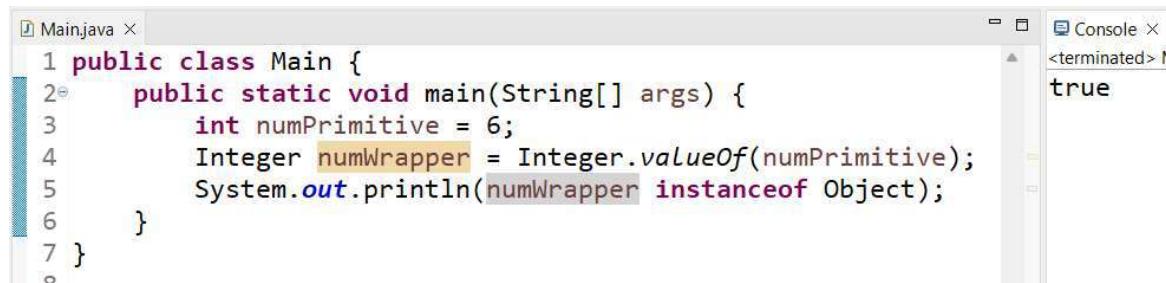
Provides classes that are fundamental to the design of the Java programming language.

See: Description

Class Summary	Description
Boolean	The Boolean class wraps a value of the primitive type boolean in an object.
Byte	The Byte class wraps a value of primitive type byte in an object.
Character	The Character class wraps a value of the primitive type char in an object.
Character.Subset	Instances of this class represent particular subsets of the Unicode character set.
Character.UnicodeBlock	A family of character subsets representing the character blocks in the Unicode specification.
Class<T>	Instances of the class Class represent classes and interfaces in a running Java application.
ClassLoader	A class loader is an object that is responsible for loading classes.
ClassValue<T>	Lazily associate a computed value with (potentially) every type.
Compiler	The Compiler class is provided to support Java-to-native-code compilers and related services.
Double	The Double class wraps a value of the primitive type double in an object.
Enum<E extends Enum<E>>	This is the common base class of all Java language enumeration types.
Float	The Float class wraps a value of primitive type float in an object.
InheritableThreadLocal<T>	This class extends ThreadLocal to provide inheritance of values from parent thread to child thread: when a child thread is created
Integer	The Integer class wraps a value of the primitive type int in an object.
Long	The Long class wraps a value of the primitive type long in an object.
Math	The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square
Number	The abstract class Number is the superclass of classes BigDecimal , BigInteger , Byte , Double , Float , Integer , Long , and Short .

Los wrappers son objects

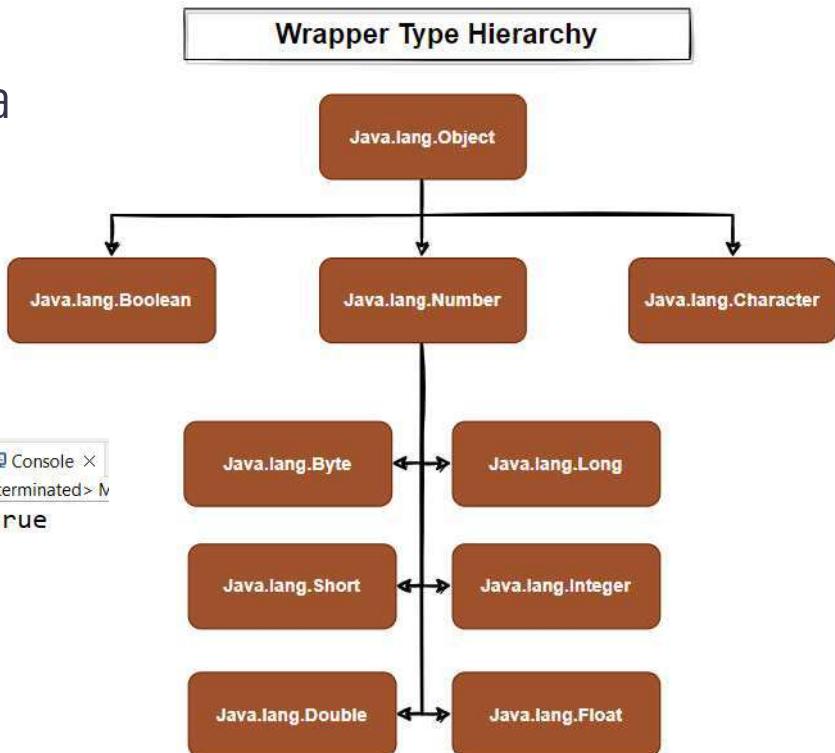
- Os comparto un esquema que preparé para un artículo de la consultora en la que trabajara para la que escribí un artículo al respecto a dicho tema:
<https://profile.es/blog/clases-wrapper-envoltorio-en-java/>



The screenshot shows an IDE interface with two panes. On the left, the code editor displays a file named Main.java with the following content:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int numPrimitive = 6;  
4         Integer numWrapper = Integer.valueOf(numPrimitive);  
5         System.out.println(numWrapper instanceof Object);  
6     }  
7 }
```

On the right, the console pane shows the output: "true".

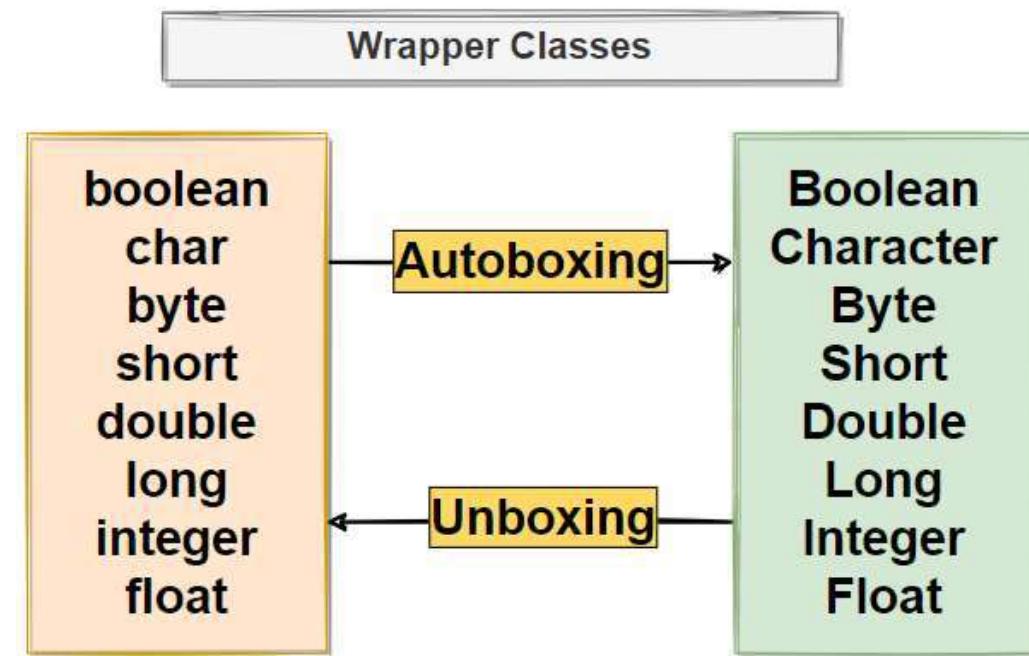


WRAPEANDO Y DESWRAPEANDO

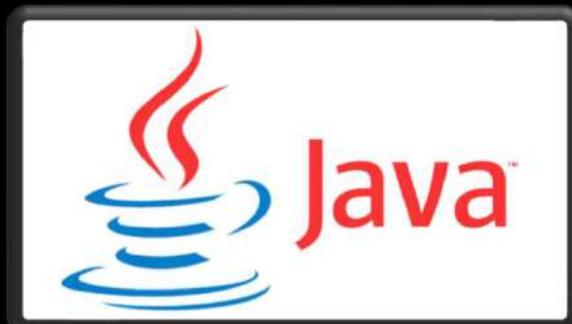


Wrapping y deswrapping

- El **boxing** o **wrapping**, consiste en el proceso de envoltura de un tipo primitivo (sin métodos) hacia un objeto. Concretamente el Wrapper correspondiente a dicho tipo primitivo. Proporcionándole una serie de métodos.
- El **unboxing** o “**deswrapping**”, consiste en el proceso de desenvoltura de un objeto (concretamente de un Wrapper) hacia un tipo primitivo.



AUTOBOXING WRAPPING



300

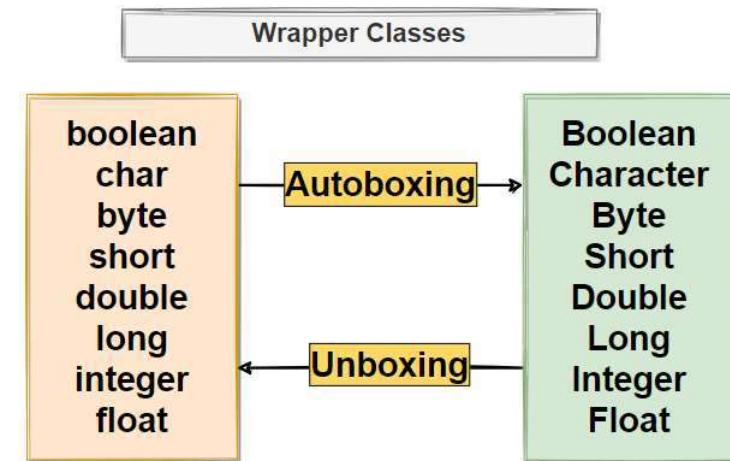
Wrapping autoboxing

- Cuando queremos transformar un tipo primario a un objeto, tenemos varias maneras de hacer dicha envoltura (wrapping).
 - Autoboxing, es un Wrapping automático, y consiste en un proceso de conversión automática realizado internamente por el compilador de java de un tipo primitivo hacia su clase de envoltura (Wrapper).
 - Si hacemos el método `getClass()`, vemos que hemos convertido el tipo primitivo a un objeto.

The screenshot shows a Java IDE interface with two tabs: "Main.java X" and "Console X". The Main.java tab contains the following code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Character caracterWrapped = 'a'; //AutoBoxing  
4         System.out.println(caracterWrapped.getClass().getSimpleName());  
5     }  
6 }
```

The "Character" output from the console is highlighted with a red box. A red arrow points from the line of code `System.out.println(caracterWrapped.getClass().getSimpleName());` in the editor to the highlighted "Character" in the console output.



Wrapping autoboxing

- Internamente, lo que realmente el autoboxing wraping es lo siguiente:

Main.java

```
1 public class Main {  
2     public static void main(String[] args) {  
3         char caracter = 'a'; //AutoBoxing  
4         System.out.println(((Character) caracter).getClass().getName());  
5     }  
6 }
```

Console

```
<terminated> Main [Java Application]  
java.lang.Character
```

- También podríamos castear hacia un Object obteniendo el mismo resultado:

Main.java

```
1 public class Main {  
2     public static void main(String[] args) {  
3         char caracter = 'a'; //AutoBoxing  
4         System.out.println(((Object) caracter).getClass().getName());  
5     }  
6 }
```

Console

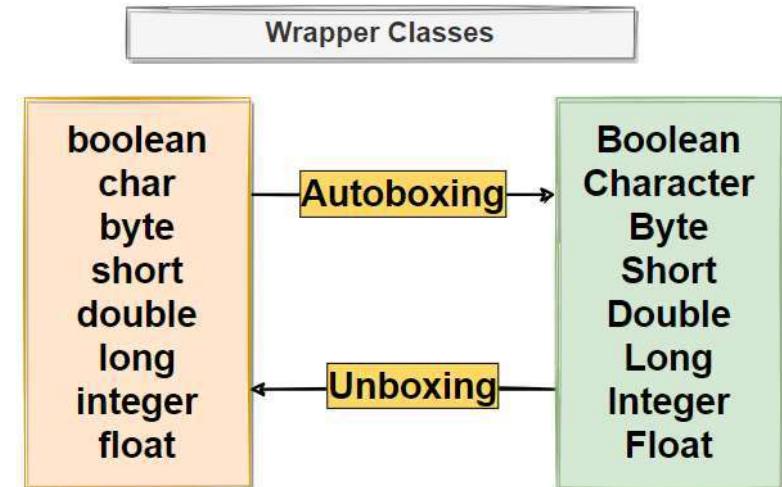
```
<terminated> Main [Java Application]  
java.lang.Character
```

AUTOUNBOXING DESWRAPPING



Wrapping autounboxing

- De la misma forma que podemos hacer un autoboxing para wrapper un tipo primitivo hacia un objeto. Podemos desgrapear un Wrapper hacia un tipo primitivo con autounboxing. Vamos a verlo:

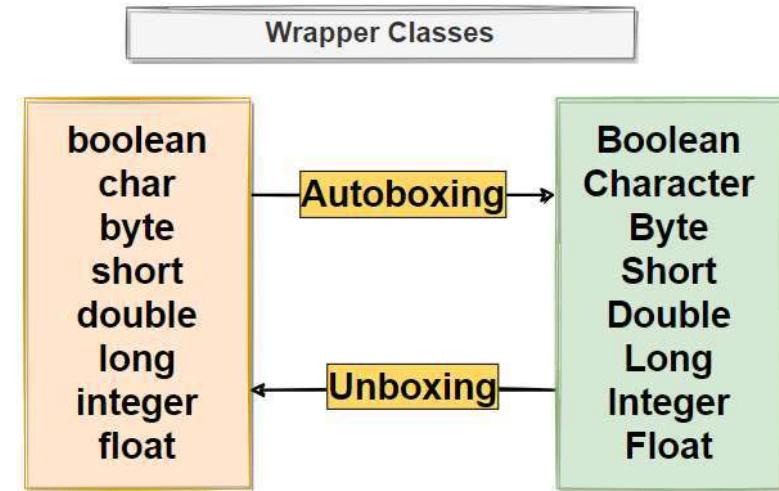


```
Main.java X
1 public class Main {
2     public static void main(String[] args) {
3         Character characterWrapped = 'a'; //AutoBoxing
4         System.out.println(characterWrapped.getClass().getName());
5         char characterUnwrapped= characterWrapped; //AutoUnboxing
6         System.out.println(characterUnwrapped);
7     }
8 }
```

```
Console X
<terminated> Main [Java Application]
java.lang.Character
a
```

Wrapping autounboxing

- De hecho, una vez hacemos el autounboxing, tenemos un tipo primitivo, lo que se traduce en que no tenemos métodos. Por tanto, si intentamos acceder a los métodos, vemos que no nos muestra ningún método:



```
*Main.java ×
1 public class Main {
2     public static void main(String[] args) {
3         Character characterWrapped = 'a'; //AutoBoxing
4         System.out.println(characterWrapped.getClass().getName());
5         char characterUnwrapped= characterWrapped; //AutoUnboxing
6         System.out.println(characterUnwrapped);
7     }
8 }
```

No Default Proposals

WRAPPING MANUAL PROBLEMAS DE TRABAJAR MEDIANTE EL CONSTRUTOR



Wrapping mediante constructor hasta JDK 9

- Hasta la versión 9 de JDK podemos utilizar el constructor del Wrapper, es decir, los constructores de la clase Integer, Boolean, Character... Para construir nuestros elementos. Pero si trabajamos con una versión superior como es mi caso ya no está permitido ya que esto está deprecated (obsoleto). Vamos a verlo:

The screenshot shows the Eclipse IDE interface. On the left, there is a code editor window titled "Main.java" containing the following Java code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Character caracterWrapped = new Character('a');  
4         System.out.println(caracterWrapped);  
5     }  
6 }
```

On the right, there is a terminal window showing the output of running the program:

```
Microsoft Windows [Versión 10.0.22000.613]  
(c) Microsoft Corporation. Todos los derechos reservados  
  
C:\Users\David>java -version  
java version "11.0.13" 2021-10-19 LTS  
Java(TM) SE Runtime Environment 18.9 (build 11.0.13+10-  
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.13+1
```

A tooltip is displayed over the line of code `Character caracterWrapped = new Character('a');` with the message: "The constructor Character(char) is deprecated since version 9". Below the tooltip, three quick fix options are listed:

- @ Add @SuppressWarnings 'deprecation' to 'caracterWrapped'
- @ Add @SuppressWarnings 'deprecation' to 'main()'
- Configure problem severity

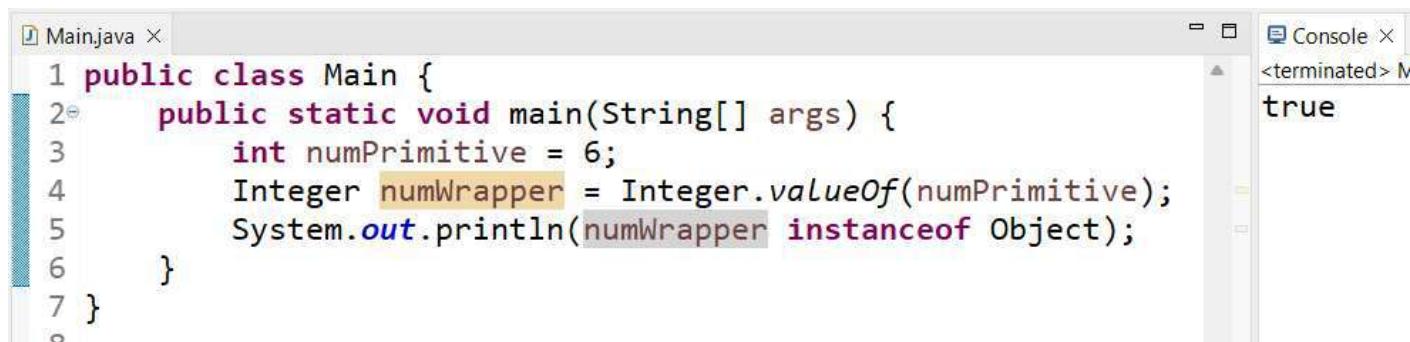
- Pese a que está deprecada, si compilamos vemos que aún funciona, aunque Eclipse nos muestra un mensaje de está deprecada por lo que no es aconsejable su uso.

WRAPPING MANUAL CON VALUEOF



Wrapping manual con valueOf

- Boxing (wrapping manual): es el proceso de conversión no automático que realizamos con el fin de pasar un tipo primitivo a un objeto mediante su clase de envoltura (“Wrapper”).
- Actualmente valueof viene a sustituir a la manera deprecada de hacer un wrapping manual mediante a constructor que a partir de la versión 9 de JDK está deprecada:



The screenshot shows an IDE interface with two panes. On the left, the code editor displays a file named 'Main.java' with the following content:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int numPrimitive = 6;  
4         Integer numWrapper = Integer.valueOf(numPrimitive);  
5         System.out.println(numWrapper instanceof Object);  
6     }  
7 }
```

On the right, the 'Console' pane shows the output of the program: 'true'. This indicates that the variable 'numWrapper' is indeed an instance of the 'Object' class.

- Además, vemos que con instanceof podemos ver si realmente este wrapper es una instancia de Object (una especie de “copia”, es decir que sale de la clase Object).

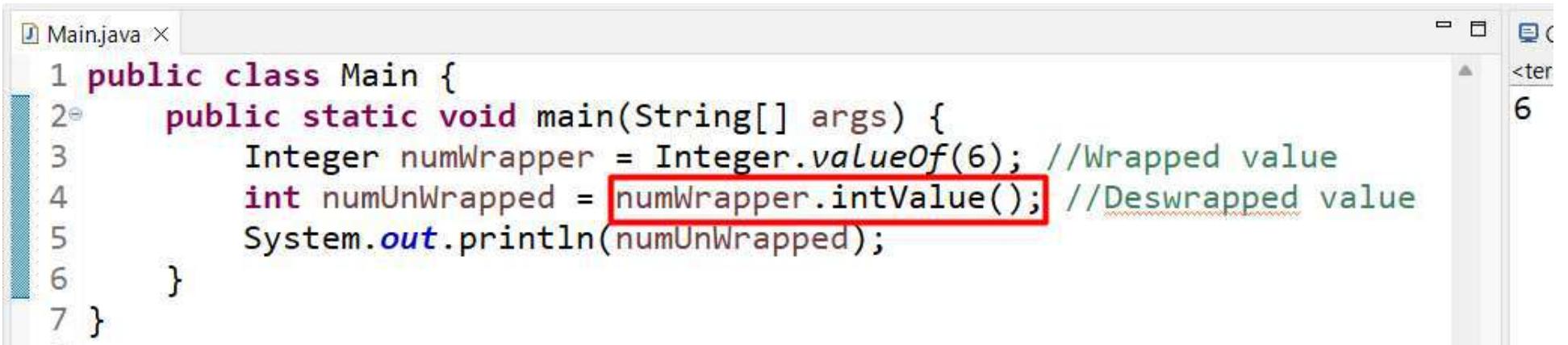
DESWRAPPING MANUAL



310

Deswrapping manual

- Unboxing (deswraping manual): el proceso de conversión no automático que realizamos con el fin de pasar un tipo no primitivo (Wrapper) a un tipo primitivo.



```
Main.java ×
1 public class Main {
2     public static void main(String[] args) {
3         Integer numWrapper = Integer.valueOf(6); //Wrapped value
4         int numUnwrapped = numWrapper.intValue(); //Deswrapped value
5         System.out.println(numUnwrapped);
6     }
7 }
```

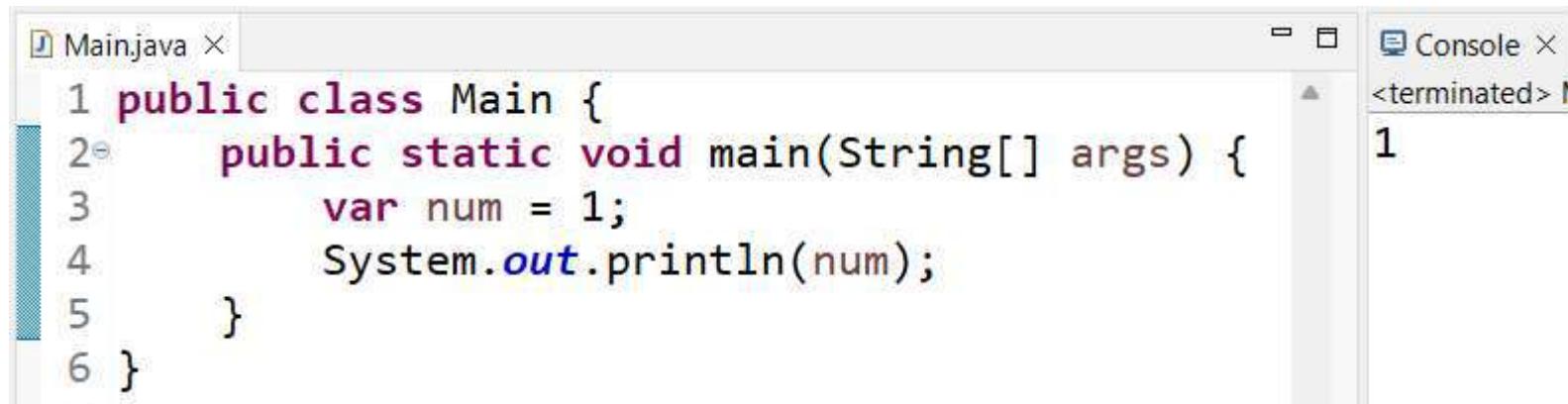
- El proceso transforma un tipo de dato primitivo hacia su clase Wrapped se conoce como embalaje (boxing). En cambio, si el proceso transforma un objeto de tipo Wrapped a un tipo primitivo se conoce como desembalaje (unboxing).

JDK 10 AÑADE VAR Y LA INFERENCIA DE TIPOS



VAR (Local-Variable Type Inference)

- A partir de JDK 10 se introduce una alternativa a los tipos primitivos/genéricos.
- Esta manera de trabajar con variables (var) utiliza la inferencia de tipos.
- Y permite el uso del tipo reservado var para realizar la declaración de variables.
Vamos a ver un ejemplo:



The screenshot shows an IDE interface with two main panes. On the left, the code editor displays a Java file named 'Main.java' with the following content:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         var num = 1;  
4         System.out.println(num);  
5     }  
6 }
```

On the right, the 'Console' pane shows the output of the program: the number '1'. This demonstrates that the variable 'num' is inferred to be of type int.

VAR (Local-Variable Type Inference)

- Hasta ahora, éramos nosotros los que especificamos el tipo de variable a declarar. En cambio, con el uso de var, y gracias a la inferencia de tipos, es el compilador el encargado de dictaminar el tipo de un variable que ha sido declarar sin realizar una declaración explícita de su tipo.

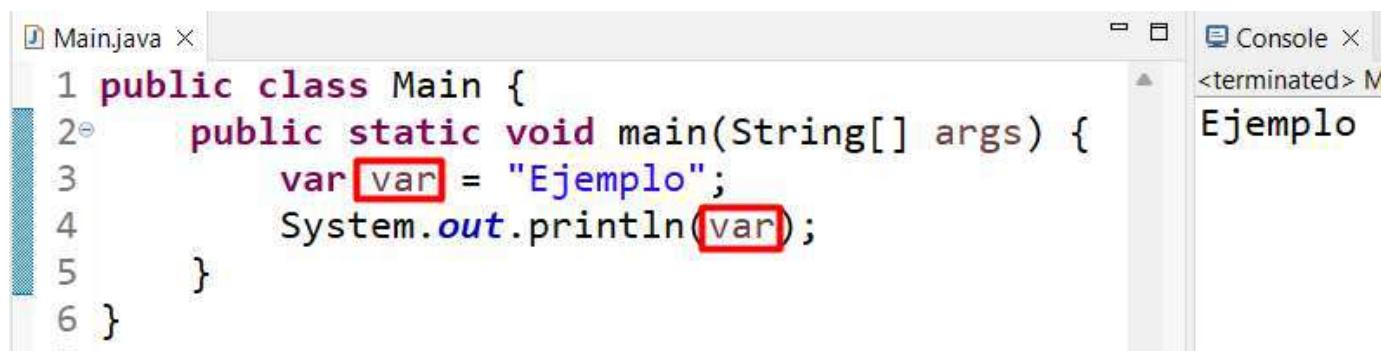
The screenshot shows an IDE interface with two tabs: 'Main.java X' and 'Console X'. The Main.java tab contains the following code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         var var = "Ejemplo";  
4         System.out.println(var);  
5     }  
6 }
```

The variable 'var' is highlighted with a red box. The Console tab shows the output: <terminated> M Ejemplo

VAR (Local-Variable Type Inference)

- Var no es una palabra clave o Keyword, sino un nombre de tipo reservado, por lo que al no ser una palabra reservada, podrá usarse para definir el nombre de variable y/o paquetes. El siguiente código es valido:



The screenshot shows an IDE interface with two tabs: 'Main.java X' and 'Console X'. The Main.java tab contains the following code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         var var = "Ejemplo";  
4         System.out.println(var);  
5     }  
6 }
```

The word 'var' is highlighted in red in both the code and the console output. The console tab shows the output: '<terminated> M Ejemplo'.

- Aunque se puede utilizar, no es recomendable utilizar var como identificador. Ya que, lo recomendable es no hacerlo. Evitando así posibles conflictos en el futuro con nuestro código en las nuevas versiones de JDK

**VAR INFIERE EL TIPO
PERO NO PERMITE
MODIFICARLO
POSTERIORMENTE**



VAR (Local-Variable Type Inference)

- Lo que si que no podemos hacer es modificar el tipo de valor una vez ha sido inferido. Ya que el valor de asigna de forma estáticamente y no dinámicamente:

The screenshot shows an IDE interface with two main panes. On the left, the code editor displays `Main.java` with the following content:

```
1 package com.main;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         var ejemplo= "Ejemplo";
7         System.out.println(ejemplo.getClass().getName());
8         ejemplo= 1234;
9         System.out.println(ejemplo);
10    }
11 }
12
13
14
```

A tooltip is displayed over the assignment `ejemplo= 1234;`, indicating a "Type mismatch: cannot convert from int to String". It also shows a quick fix option: "Change type of 'ejemplo' to 'int'".

On the right, the console window shows the output of the program:

```
<terminated> Main (2) [Java]
java.lang.String
```

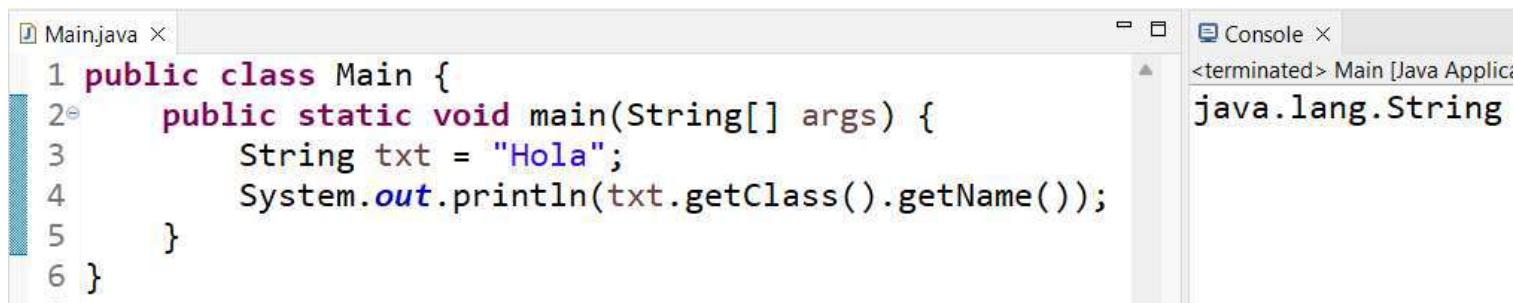
The page number **317** is located in the bottom right corner.

OBTENIENDO EL TIPO DE DATO



getClass.getName() sobre Objetos

- De la misma manera que en javascript teníamos el typeof para sacar el tipo de dato. En Java siempre y que no trabajemos con los métodos primitivos, podemos sacar el tipo de dato sobre el que estamos trabajando mediante a los métodos getClass() y getName(). Si por ejemplo trabajamos con la clase String vemos que podemos sacar el tipo de dato sin problemas:



The screenshot shows an IDE interface with two main windows. On the left, the 'Main.java' editor pane displays the following Java code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         String txt = "Hola";  
4         System.out.println(txt.getClass().getName());  
5     }  
6 }
```

On the right, the 'Console' pane shows the output of the application's execution. It displays the text 'java.lang.String' in black font, indicating the class name of the object 'txt'.

getClass.getName() sobre tipos primitivos

- El principal problema de Java es que los tipos de datos primitivos no tienen métodos. Por lo que, si intentamos hacer lo mismo sobre una variable primitiva, por ejemplo int, vemos que nos aparece un error.
- Ya que realmente, desde el intellisense ni siquiera nos muestra métodos. Ya que los tipos primitivos no tienen la posibilidad ni si quiera de obtener los métodos:

A screenshot of an IDE showing a Java code editor. The code is as follows:

```
1 ic class Main {  
2 public static void main(String[] args) {  
3     int num = 123;  
4     System.out.println(num.getClass().getName());  
5 }  
6
```

The line `System.out.println(num.getClass().getName());` is highlighted in blue. A tooltip appears over the `getClass()` method call, stating: `Cannot invoke getClass() on the primitive type int`. The cursor is positioned at the end of the line.

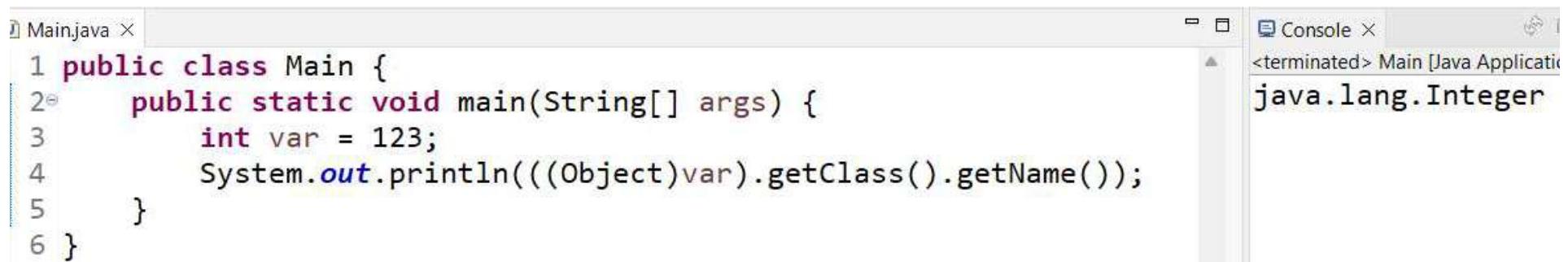
A screenshot of an IDE showing a Java code editor. The code is as follows:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int num = 123;  
4         System.out.println([num.]);  
5     }  
6 }
```

The line `System.out.println([num.]);` is highlighted in blue. A tooltip appears over the opening parenthesis, stating: `No Default Proposals`. The cursor is positioned at the end of the line.

VAR (Local-Variable Type Inference)

- Para realizar saber que tipo de dato primitivo vamos a utilizar, deberíamos, realmente, esto mismo desde un tipo primitivo deberíamos de realizar lo siguiente:



The screenshot shows an IDE interface with two panes. The left pane, titled 'Main.java', contains the following Java code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int var = 123;  
4         System.out.println(((Object)var).getClass().getName());  
5     }  
6 }
```

The right pane, titled 'Console', shows the output of the program: 'java.lang.Integer'. This demonstrates that the variable 'var' is inferred to be of type Integer, even though it is declared as int.

Ejercicio de tipos de datos

- Ejercicio 1:
 - Declara una variable de tipo Long, byte, short, Integer.
 - ¿Puedes obtener la clase a la que pertenecen todas ellas?
 - Ante el caso de que no pudieras, explica algunas maneras de resolver esto de las siguientes maneras:
 - Castear para identificar a Wrapper debería de pertenecer
 - Envolver el valor con un Wrapper de alguna de las maneras para posteriormente acceder a su valor
- Ejercicio 2:
 - Haz un programa que en un función del tipo de dato que le pasemos por parámetros (String, int, Long, etc.) identifique a que Wrapper pertenece o bien si es un tipo de dato primitivo a que Wrapper pertenecería cuando se realice la conversión (casteo) a Object.

CONVERSIONES



¿Que son las conversiones?

- Cuando queremos asignar el valor asignado a un tipo de dato en otro tipo de dato, existe la posibilidad que estos tipos de datos no sean compatibles.
- Si los tipos de datos son compatibles, Java realizará la conversión (automáticamente) lo que se conoce como conversión automática de tipos. En caso contrario, cuando los tipos no sean compatibles, no se podrá realizar esta autoconversión y deberemos de realizarla nosotros manualmente un casting o conversión explícita.

The screenshot shows a Java code editor with the following code:

```
Main.java X
1 public class Main {
2     public static void main(String[] args) {
3         // Conversión automática de tipos: solamente se puede utilizar sobre tipos compatibles
4         byte numByte = 123;
5         short numShort01 = numByte;
6         // Ejemplo de intento de conversión no automática: requerirá de un casting o conversión explícita
7         int numInt = 1235432787;
8         short numShort02 = numInt;
9     }
10 }
```

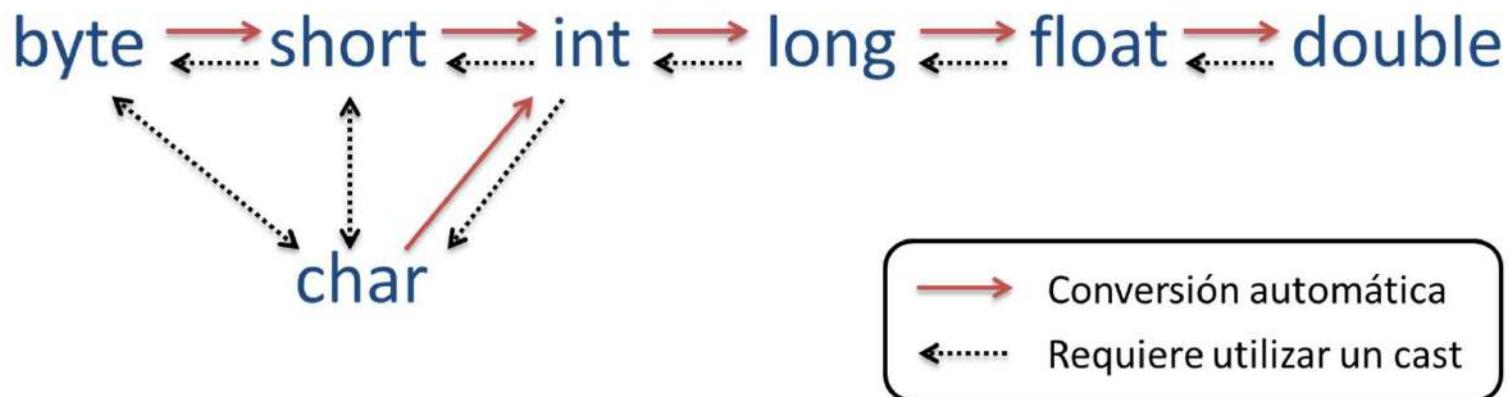
A tooltip is displayed over the line `short numShort02 = numInt;` with the message: "Type mismatch: cannot convert from int to short". It also says "3 quick fixes available:" and lists three options:

- Add cast to 'short'
- Change type of 'numShort02' to 'int'
- Change type of 'numInt' to 'short'

The IDE interface includes tabs for "Main.java" and "Console", and a status bar showing "324".

¿Que son las conversiones?

- Este esquema relata muy bien las direcciones hacia las que podemos realizar las conversiones automáticas o las que requieren el uso de cast:



- Dentro de las conversiones existen dos tipos:
 - Conversión de tipos automática (ampliación)
 - Casting o conversión manual (reducción)
- Y estas operaciones pueden ser realizadas por: asignación, casting o Wrapper o promoción aritmética.

CONVERSIÓN AUTOMÁTICA (AMPLIACIÓN)



Conversión automática por asignación

- Este esquema relata muy bien las direcciones hacia las que podemos realizar las conversiones automáticas o las que requieren el uso de cast:

byte -> short -> int -> long -> float -> double

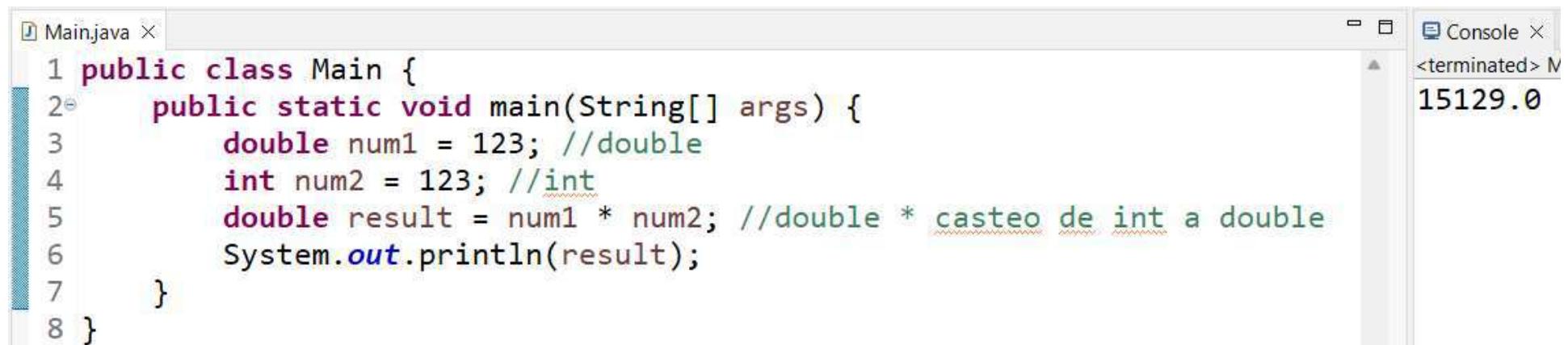
The screenshot shows a Java application named 'Main.java' in a code editor. The code defines a main method that initializes an integer variable 'numInteger' to 25. It then demonstrates automatic type conversion by assigning 'numInteger' to a long variable 'numLong' and printing its value as an int, long, and float. The terminal window shows the output: 'Valor Int 25', 'Valor Long 25', and 'Valor Float 25.0'.

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int numInteger = 25;  
4  
5         //Conversión automática  
6         long numLong = numInteger;  
7  
8         //conversion automática de tipo  
9         float numFloat = numLong;  
10        System.out.println("Valor Int "+ numInteger);  
11        System.out.println("Valor Long "+ numLong);  
12        System.out.println("Valor Float "+ numFloat);  
13    }  
14 }  
15 }
```

Tipo de origen	Tipo de destino
byte	short, int, long, float, double
short	int, long, float, double
char	int, long, float, double
int	long, float, double
long	double
float	float, double
double	-

Conversión automática por promoción aritmética

- También existe la posibilidad de realizar el casteo de un int (num2) a un double automáticamente cuando realizamos operaciones aritméticas. Vamos a ver un ejemplo:



The screenshot shows a Java development environment with two tabs: 'Main.java X' and 'Console X'. The Main.java tab contains the following code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         double num1 = 123; //double  
4         int num2 = 123; //int  
5         double result = num1 * num2; //double * casteo de int a double  
6         System.out.println(result);  
7     }  
8 }
```

The Console tab shows the output: <terminated> N 15129.0

CONVERSIÓN MANUAL (EXPLÍCITA)



Conversiones manuales

- Son más delicadas que como ya vimos si el tipo de dato que queremos convertir es mayor que el tipo de dato que guardará la transformación, estamos volcando más “agua” (datos) de los que puede albergar dicho tipo de dato. Y en determinadas situaciones se puede perder cierta información o alterar dicha información por lo que es menor recomendable el realizar este tipo de conversiones.

double -> float -> long -> int -> short -> byte

- Un ejemplo de ello, podría ser el siguiente en el que convertimos un double (número real) hacia un int (número entero) y vemos como perdemos la parte decimal:

Tipo de origen	Tipo de destino
byte	char
short	byte, char
char	byte, short
int	byte, short, char
long	byte, short, char, int
float	byte, shot, char, int, long
double	byte, short, char, int, long, float

The screenshot shows a Java development environment with two panes. The left pane displays the code for Main.java:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         double numDouble = 99999.12345;  
4         System.out.println(numDouble);  
5         int numInt = (int) numDouble;  
6         System.out.println(numInt);  
7     }  
8 }
```

The right pane shows the output in the Console tab:

```
99999.12345  
99999
```

Conversión manual por casting “casteo”

- Cuando queremos meter un tipo de dato más grande en un tipo de dato más pequeño realizamos lo que se conoce como conversión explícita. Vamos a ver un ejemplo:

double → float → long → int → short → byte

The screenshot shows a Java IDE interface with two panes. The left pane, titled 'Main.java', contains the following code:

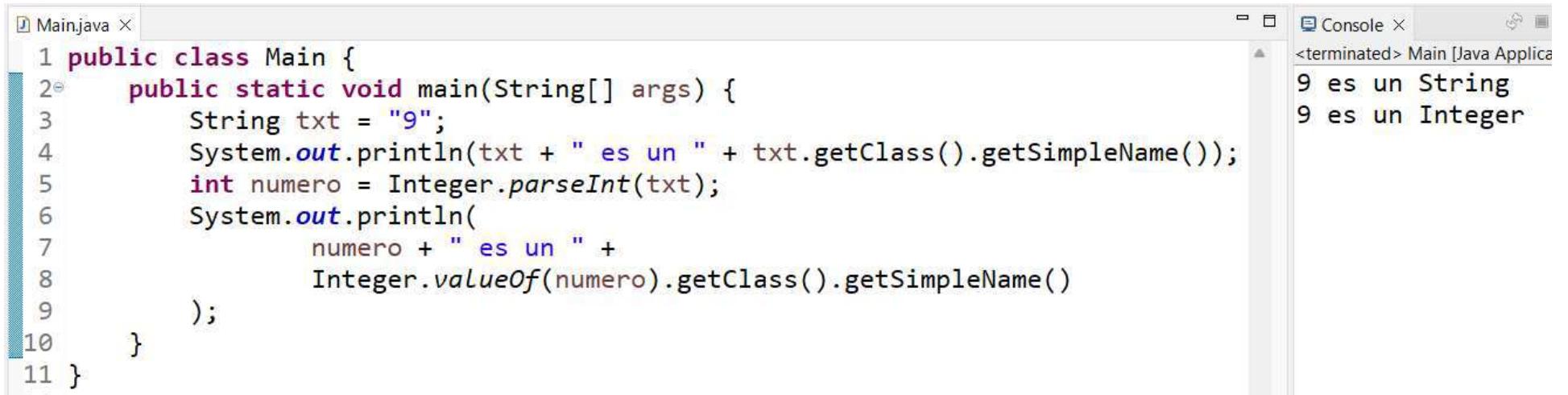
```
1 public class Main {  
2     public static void main(String[] args) {  
3         double numDouble = Double.MAX_VALUE;  
4         System.out.println(numDouble);  
5         float numFloat = (float) numDouble;  
6         System.out.println(numFloat);  
7         long numLong = (long) numDouble;  
8         System.out.println(numLong);  
9         int numInt = (int) numDouble;  
10        System.out.println(numInt);  
11        short numShort = (short) numDouble;  
12        System.out.println(numShort);  
13        byte numByte = (byte) numDouble;  
14        System.out.println(numByte);  
15    }  
16 }
```

The right pane, titled 'Console', shows the output of the program:

```
<terminated> Main [Java Application] C:\Pro  
1.7976931348623157E308  
Infinity  
9223372036854775807  
2147483647  
-1  
-1
```

Conversión manual con Wrappers

- También podemos utilizar las clases Wrappers para convertir por ejemplo un String a un integer. Vamos a verlo:



The screenshot shows an IDE interface with two windows. On the left, the code editor window titled "Main.java" contains the following Java code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         String txt = "9";  
4         System.out.println(txt + " es un " + txt.getClass().getSimpleName());  
5         int numero = Integer.parseInt(txt);  
6         System.out.println(  
7             numero + " es un " +  
8             Integer.valueOf(numero).getClass().getSimpleName()  
9         );  
10    }  
11 }
```

On the right, the "Console" window shows the output of the program:

```
<terminated> Main [Java Application]  
9 es un String  
9 es un Integer
```

Ejercicio de conversiones

- 1. Declara un tipo de dato double y ves haciendo una escalera de conversiones dato primitivo a dato primitivo hasta llegar a byte.
- 2. Declara un tipo de dato byte y ves realizando conversiones ascendentes de dato primitivo a dato primitivo hasta llegar a double.
- 3. Declara un tipo de dato byte e inicializa la variable a 100. Modifica el valor de la variable multiplicando su valor x 2 ¿Qué está pasando? ¿Cómo lo arreglarías?

SYSTEM.OUT.PRINTF



¿PARA QUE SIRVE PRINTF?

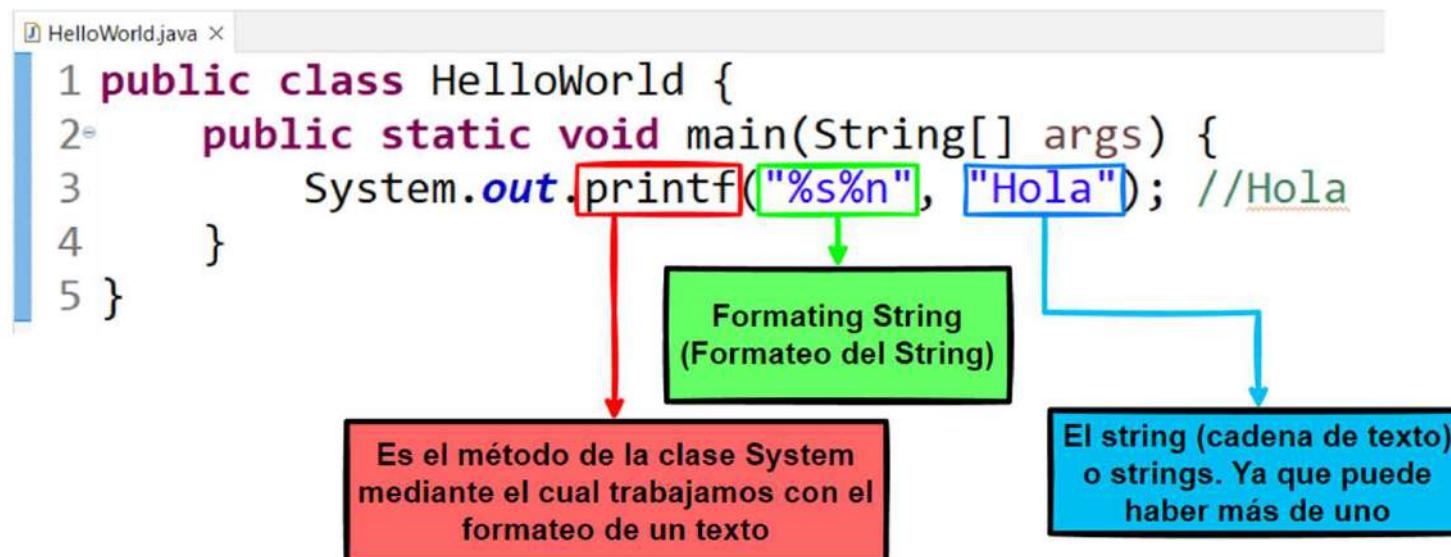


¿Para que nos sirve el método printf de la clase System?

- Anteriormente, hemos hablado de la clase System, concretamente de como hacer el output mediante a out/error junto al método print, println.
- La función printf es una abreviatura de print formatted y nos permite dar formato, es decir, formatear un mensaje utilizando una "cadena de formato" la cual incluirá una serie de instrucciones que definirán el contenido y la forma en la que estructurará dicha cadena antes de finalmente, mostrarla por pantalla.
- <https://gist.github.com/DavidBernalGonzalez/d4651847a6bf57ab64dc7264499b5218>

¿Para que nos sirve el método printf de la clase System?

- Vamos a ver un ejemplo:



EXPLICANDO ALGUNOS FORMATOS DE SALIDA DE PRINTF



Printf

- printf -> Se utiliza para dar formato a la salida en la consola. Valores aceptados:

- Secuencias de escape:

- \n Nueva línea
 - \t tabulador horizontal
 - \r Retorno de carro (es como pulsar la tecla de quitar un espacio siempre y que lo haya en dicha línea clara)
 - \\ Para escribir barra diagonal inversa
 - \" para escribir una doble comilla

- Caracteres de conversión:

- %s Se utilizar para seleccionar un String (texto) respectando su formato actual (mayúsculas o minúsculas)
 - %S Se utilizar para seleccionar un String (texto) y formatearlo en mayúsculas
 - %c Se utiliza para seleccionar a un carácter Unicode respectando su formato actual (mayúscula o minúscula)
 - %C Se utiliza para seleccionar a un carácter Unicode y formatearlo en mayúscula
 - %d Se utiliza para seleccionar a números enteros, es decir, sin decimales, base decimal
 - %f Se utiliza para seleccionar a un numero real (de tipo double o float) con punto fijo. Por ejemplo 13.141516
 - %n Se utiliza para añadir nueva línea.
 - %t Se utiliza para añadir una tabulación horizontal
 - %r Se utiliza para añadir un backspace (retorno de carro) es como pulsar la tecla de quitar un espacio siempre y que lo haya en dicha línea clara
 - %% Se utiliza para añadir un % al mensaje de la consola

Printf

Specifier	Explanation
%c	Format characters
%d	Format decimal (integer) numbers (base 10)
%e	Format exponential floating-point numbers
%f	Format floating-point numbers
%i	Format integers (base 10)
%o	Format octal numbers (base 8)
%s	Format string of characters
%u	Format unsigned decimal (integer) numbers
%x	Format numbers in hexadecimal (base 16)
%n	add a new line character

PRINTF SOBRE UN SOLO VALOR



Printf: sobre un solo valor

- Para empezar a entender printf vamos a empezar a trabajar sobre una solo valor:

The screenshot shows an IDE interface with two panes. The left pane, titled 'Main.java', contains the following Java code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         // Trabajando con un solo valor  
4         System.out.printf("%s%n", "Hola"); //Hola  
5         System.out.printf("%S%n", "hola"); //HOLA  
6         System.out.printf("%d%n", 10); //10  
7         System.out.printf("%d%n", 1234); //1234  
8         System.out.printf("%f%n", 10.1234); //10.123400  
9         System.out.printf("%.2f%n", 10.1234F); //10.12  
10        System.out.printf("%.3f%n", 10.1234); //10.123  
11  
12        System.out.printf("\\\"%s\\\"%n", "Hola"); //\"Hola"  
13    }  
14 }
```

The right pane, titled 'Console', shows the output of the code execution:

Output
Hola
HOLA
10
1234
10,123400
10,12
10,123
"Hola"

AÑADIENDO CONTENIDO “HARDCODEADO” SOBRE UN PRINTF



Printf: añadiendo contenido a un solo valor

- Ahora, vamos a añadir contenido “hardcodeado” (introducido a mano) a nuestro valor desde este printf:

The screenshot shows a Java development environment with two windows. On the left is the code editor titled "Main.java X" containing the following Java code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         // Añadiendo contenido al valor  
4         System.out.printf("%d = cinco%n", 5); //5 = cinco  
5         System.out.printf("%+d%n", 10); //+10  
6         System.out.printf("%+d%n", 10); //+10  
7         System.out.printf("%d%%%n", 20); //20%  
8     }  
9 }
```

On the right is the "Console X" window titled "<terminated> Main" showing the output of the program:

```
5 = cinco  
+10  
+10  
20%
```

ALTERANDO EL ORDEN CORRELATIVO DE LOS VALORES DE PRINTF



Printf: sobre varios valores

- Cuando no utilizamos dólares, estamos utilizando el orden correlativo que siguen los valores del printf. Primero la variable1, después la variable2, la variable3...
- Si queremos seleccionar una variable en concreto realizamos lo siguiente:
 - - %1\$ Nos permite indicar que queremos hacer referencia a la PRIMERA variable. Debe ir seguida del tipo por ejemplo %1\$s para un String
 - - %2\$ Nos permite indicar que queremos hacer referencia a la SEGUNDA variable. Debe ir seguida del tipo por ejemplo %1\$s para un String

The screenshot shows an IDE interface with two main panes. On the left, the code editor displays a Java file named Main.java. The code contains several System.out.printf() statements demonstrating various printf format specifiers. On the right, the console window shows the output of the program, which consists of five lines of text corresponding to the printf calls.

```
Main.java ×
1 public class Main {
2     public static void main(String[] args) {
3         // Trabajando con varios valores a la vez
4         System.out.printf("%1$s=%2$f%n", "PI", 3.141516); // PI=3.141516
5         System.out.printf("%2$f%1$s%n", "=PI", 3.141516); // 3.141516=PI
6         System.out.printf("%3$S, %2$s, %1$s%n", "NOMBRE", "APELLIDO1", "APELLIDO2"); // APELLIDO2, APELLIDO1, NOMBRE
7         System.out.printf("%1$d x %1$d = %2$d%n", 5, 5 * 5); // 5 x 5 = 25;
8         System.out.printf("%1$.2f x %1$.2f = %2$.2f%n", 1.1234, 1.1234 * 1.1234); // 1.12 x 1.12 = 1.26
9     }
10 }
```

```
Console × Debug Shell Problems
<terminated> Main [Java Application] C:\Program Files\Ja
PI=3,141516
3,141516=PI
APELIDO2, APELLIDO1, NOMBRE
5 x 5 = 25
1,12 x 1,12 = 1,26
```

ESPACIADO ENTRE VALORES DE PRINTF



Printf: espaciado entre valores

- Existe la posibilidad de dejar espacios en blanco o 0 cuando formateamos un fichero. Para ello, utilizamos:
 - %10f Completará el resto de valores hasta llegar a 10 caracteres con espacios.
 - %010f Completará el resto de valores hasta llegar a 10 caracteres con ceros 0.

The screenshot shows an IDE interface with two panes. On the left is the code editor for a file named Main.java, containing the following code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         // Ocupamos 10 espacios en total. Como solo tenemos 4 digitos los otros 6 serán  
4         // espacios. Por lo que el resultado sera: +1.12  
5         System.out.printf("%+10.2f%n", 1.1234);  
6         // Ocupamos 8 espacios en total. Como solo tenemos 6 digitos los otros 2 serán  
7         // espacios. Por lo que el resultado sera: 1.1234  
8         System.out.printf("%.8.2f%n", 1.1234);  
9         // Ocupamos 20 espacios en total. Como solo tenemos 4 digitos los otros 16 serán  
10        // 0. Por lo que el resultado sera:0000000000000001,12  
11        System.out.printf("%020.2f%n", 1.1234);  
12        // Ocupamos 20 espacios en total. Como solo tenemos 4 digitos los otros 16 serán  
13        // 0. Por lo que el resultado sera:0000000000000001,12  
14        System.out.printf("%020.2f%n", 1.1234);  
15        // Ocupamos 10 espacios en total. Como solo tenemos 6 digitos dejamos 4  
16        // espacios a la izquierda  
17        System.out.printf("%10s%n", "Total:");  
18        // Ocupamos 10 espacios en total. Como solo tenemos 6 digitos dejamos 4  
19        // espacios a la derecha "Total: "  
20        System.out.printf("%-10s%n", "Total:");  
21    }  
22}
```

On the right is the console window showing the output of the program:

```
<terminated> Main [Java Application]  
+1,12  
1,12  
0000000000000001,12  
0000000000000001,12  
Total:  
Total: |
```

Ejercicios de printf

Ejercicio 1: Haz que la siguiente instrucción: System.out.printf("", "NOMBRE", "APELLIDO1", "APELLIDO2"); Devuelva el contenido en el siguiente formato (con tabulaciones) NOMBRE APELLIDO1 APELLIDO2

Ejercicio 2: Haz que la siguiente instrucción: System.out.printf("", "NOMBRE", "apellido1", "apellido2"); Devuelva el contenido en el siguiente formato (son espacios) NOMBRE apellido1apellido2

Ejercicio 3: Haz que la siguiente instrucción: System.out.printf("", "Nombre", "apellido1", "apellido2"); Devuelva el contenido en el siguiente formato (son espacios) apellido2, apellido1, Nombre

Ejercicio 4: Haz que la siguiente instrucción: System.out.printf("", "22"); Devuelva el contenido en el siguiente formato (dejar espacios) " 22"

Ejercicio 5: Haz que la siguiente instrucción: System.out.printf("", "22"); Devuelva el contenido en el siguiente formato (dejar ceros): "0000000022"

Ejercicio 6: Haz que la siguiente instrucción: System.out.printf("", "17.1829327"); Devuelva el contenido: (dejar ceros): "000017.18"

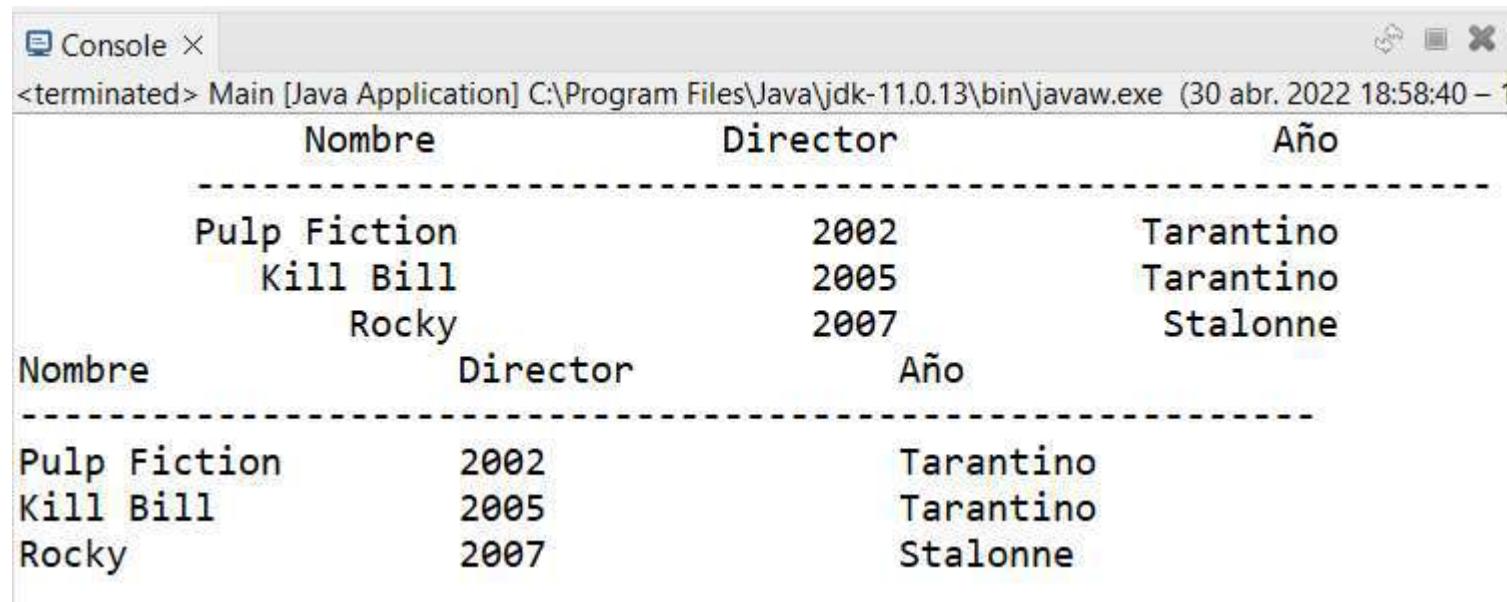
Ejercicio 7: Haz que la siguiente instrucción: System.out.printf("", "Hola ", "Mundo"); Devuelva el contenido: "HolaMundo (utiliza retornos de carros)

CREANDO TABLAS CON PRINTF



Printf: estructura de tablas

- Printf puede ser muy útil para estructurar información en tablas. Vamos a ver un ejemplo:
- Código: <https://gist.github.com/DavidBernalGonzalez/37cf4bf9d92057dcb51d864b8f646d>



The screenshot shows a Java application window titled "Console". The output window displays two tables generated by the printf method. The first table has columns "Nombre", "Director", and "Año". The second table has columns "Nombre", "Director", and "Año". Both tables list three movies: Pulp Fiction, Kill Bill, and Rocky, along with their directors (Tarantino or Stalone) and release years (2002, 2005, 2007). The tables are separated by a blank line.

Nombre	Director	Año
Pulp Fiction	2002	Tarantino
Kill Bill	2005	Tarantino
Rocky	2007	Stalone

Nombre	Director	Año
Pulp Fiction	2002	Tarantino
Kill Bill	2005	Tarantino
Rocky	2007	Stalone

Ejercicio de tabla con printf

- Basándote en el código del ejemplo anterior, haz una tabla en la que se deberá de almacenar la información de alumnos que deberán de tener en su interior los siguientes datos:
 - Nombre
 - Apellidos
 - Curso
 - Año
- Estructura los datos para que se vean de ambas maneras:

The screenshot shows a Java application window titled "Console". The output in the console is as follows:

```
<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (30 abr. 2022 18:58:40 - 1)
```

Nombre	Director	Año
Pulp Fiction	2002	Tarantino
Kill Bill	2005	Tarantino
Rocky	2007	Stalone

Nombre	Director	Año
Pulp Fiction	2002	Tarantino
Kill Bill	2005	Tarantino
Rocky	2007	Stalone

PACKAGES DE JAVA

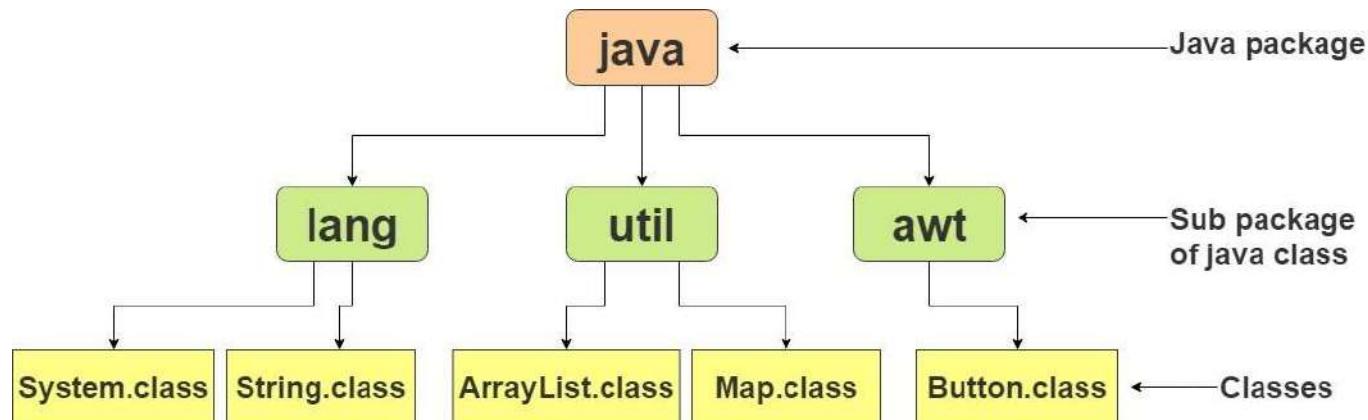


¿QUÉ SON LOS PACKAGE DE JAVA?



¿Que es un package?

- Los paquetes de Java nos permiten realizar grupos en los que contener nuestras clases, interfaces... O incluso de hasta otros packages, que serán conocidos como subpackages.
- Ya que, cuando un paquete está dentro de otro paquete se conocen como sub packages.

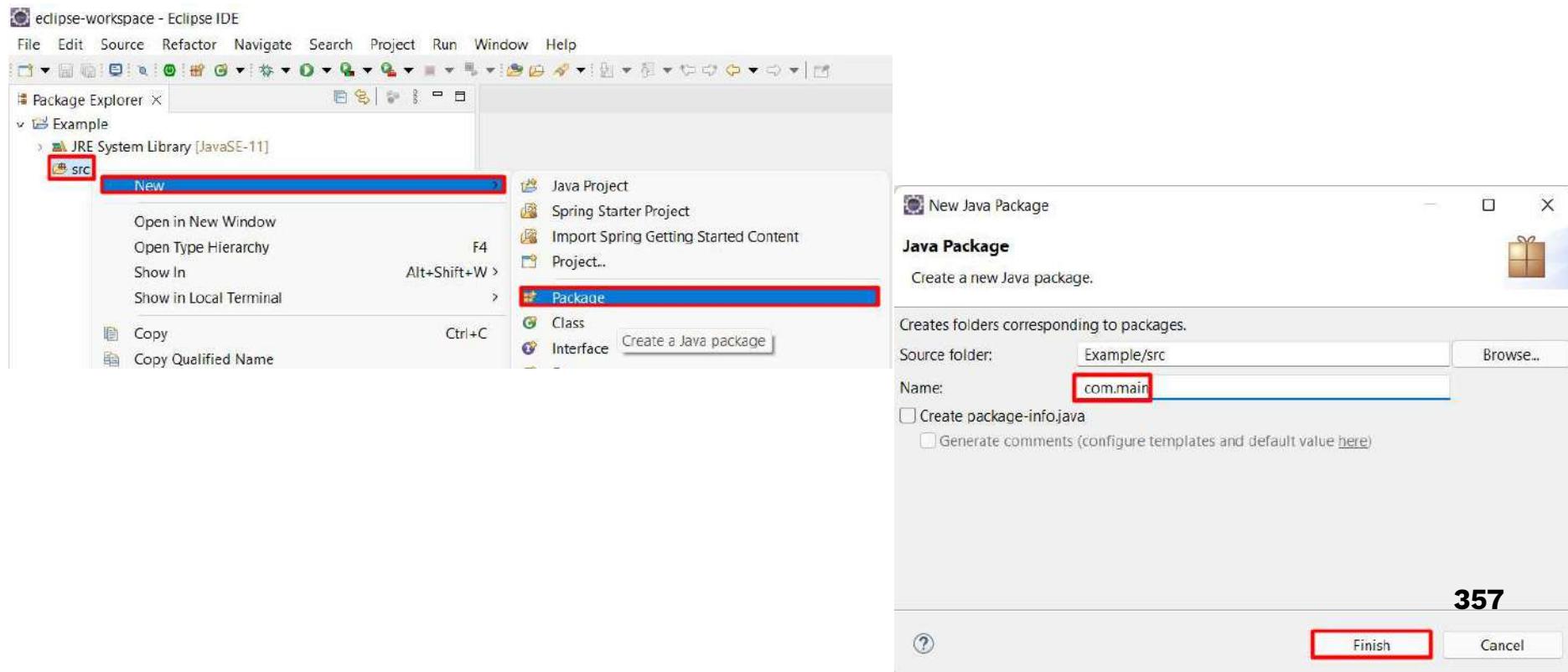


CREANDO UN PACKAGE EN JAVA



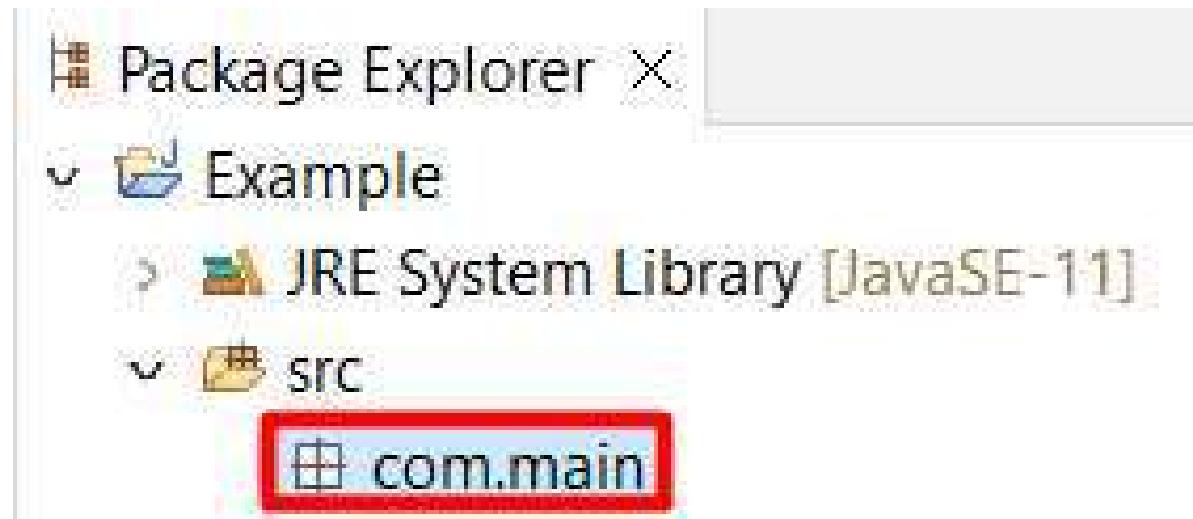
¿Como se crea un paquete en Java?

- Para crear paquetes en Java, simplemente debemos hacer lo siguiente:



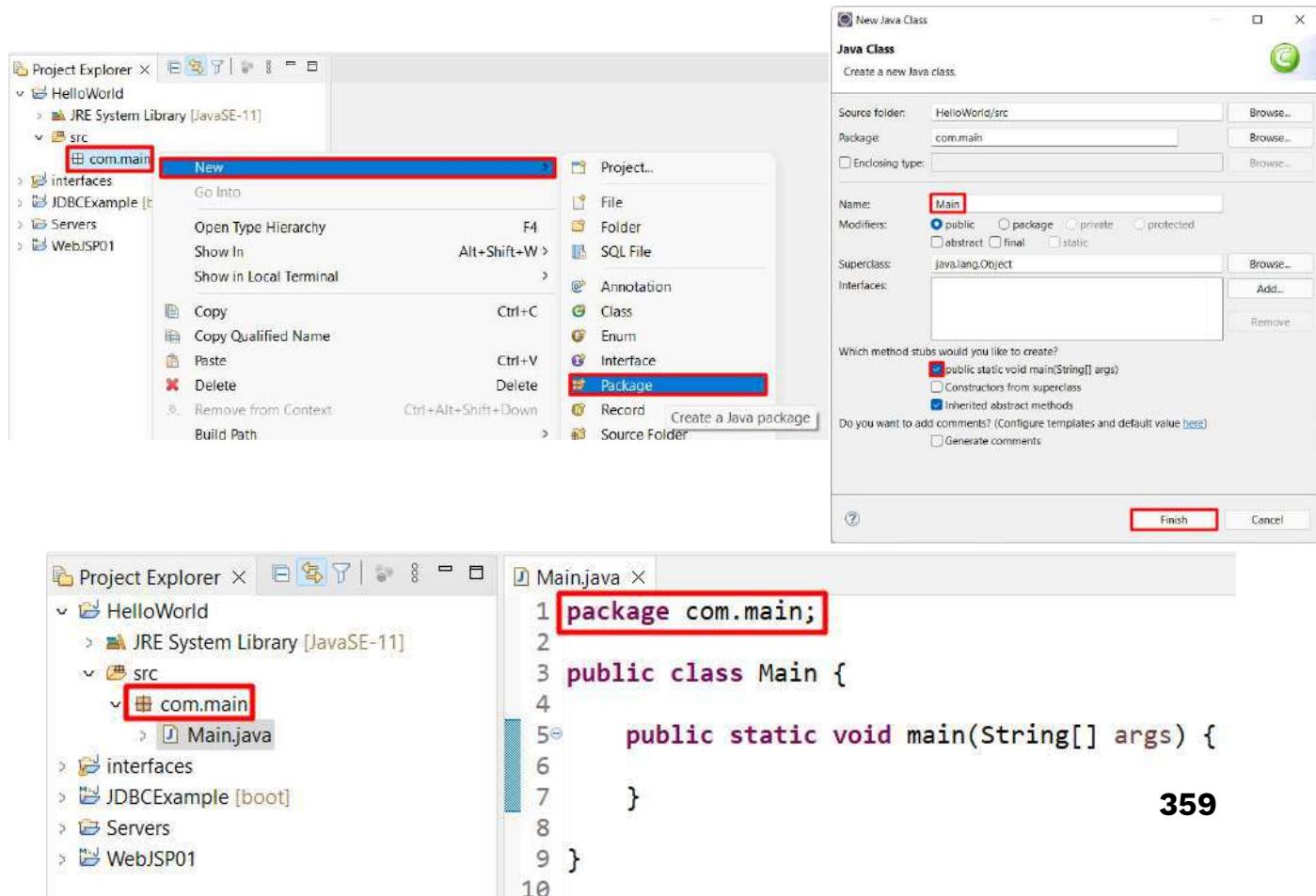
Package vacío

- Por defecto, el package aparece nos crea una especie de caja de color blanco. Debido a que el package aún está vacío. Es el mecanismo que tiene Eclipse para informarnos de que el package está vacío. Es decir, que no contiene nada en su interior.



Package con “chicha” (contenido)

- Si añado una clase main dentro del package, puedo ver que ahora si que el package cambia de color a marrón y se asemeja a una especie de caja. Y deja de ser una especie de caja de “cristal transparente”. Vamos a verlo:



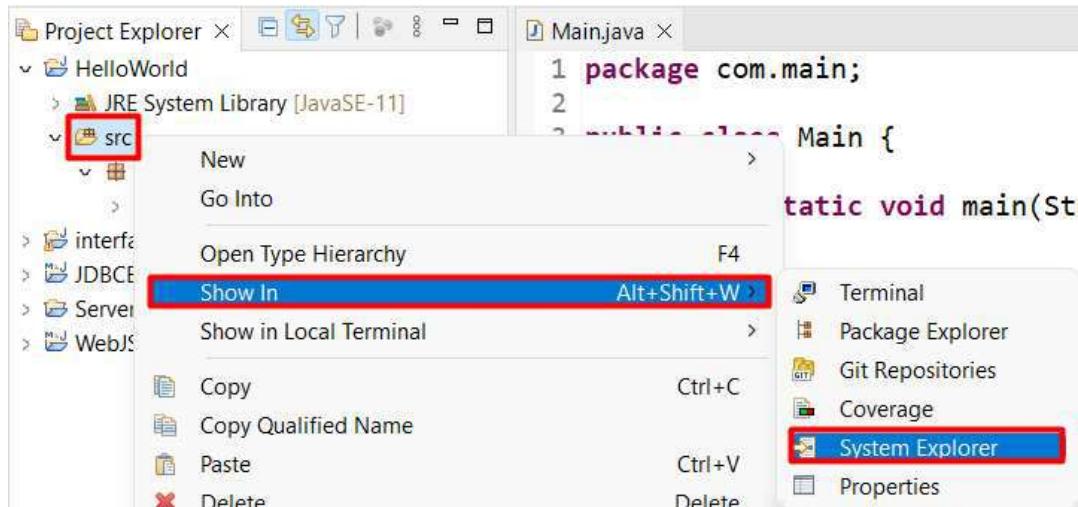
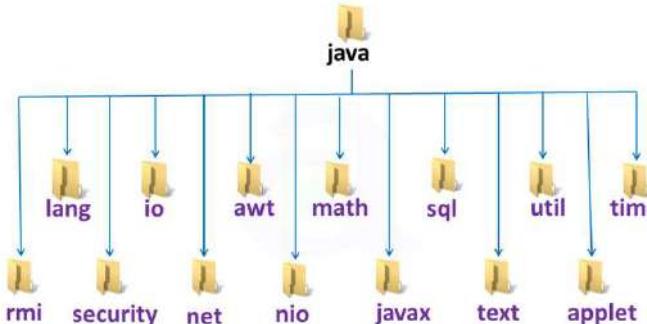
359

EXPLICANDO EL SISTEMA DE FICHEROS DE LOS PACKAGES



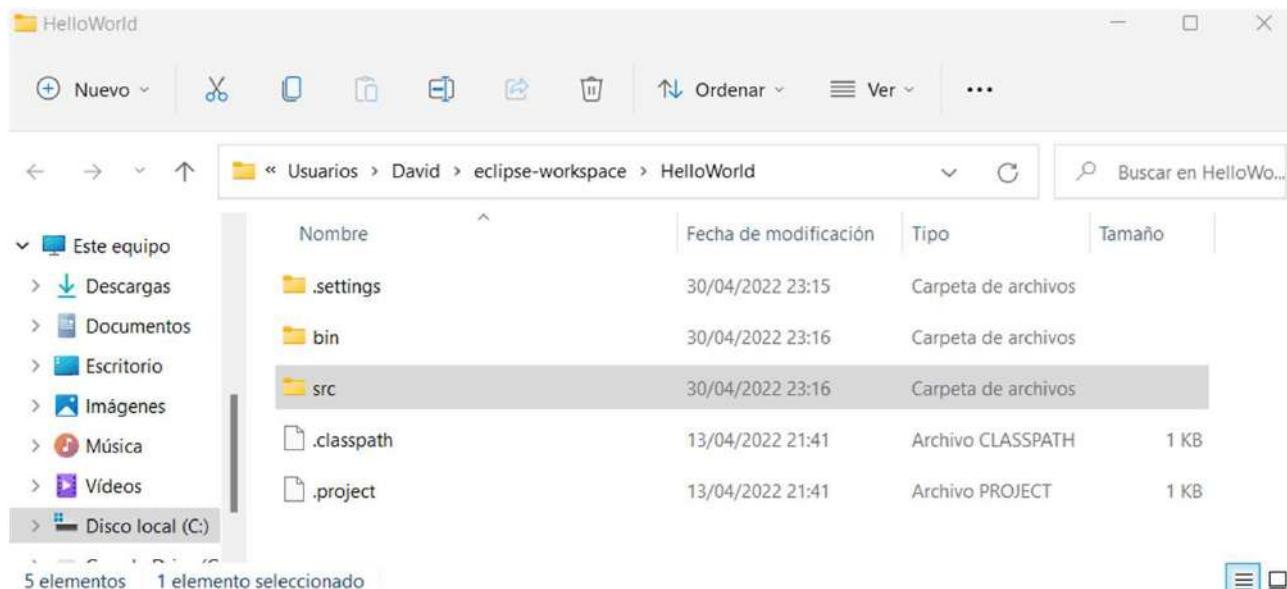
Directarios de un package

- Los packages realmente no son más que directorios en los que agrupamos nuestros ficheros de Java tales como clases, interfaces, etc.
- De hecho, si desde el directorio src (source) pulsamos sobre Show In > System Explorer:



Directorios de un package

- Si nos fijamos, dentro del HelloWorld, tenemos el directorio src (source) que es sobre el directorio el que hemos creado los packages. Y cada uno de los nombres que hemos definidos en el nombre del package separados por un punto es un directorio independiente:



EXPLICANDO EL DEFAULT PACKAGE



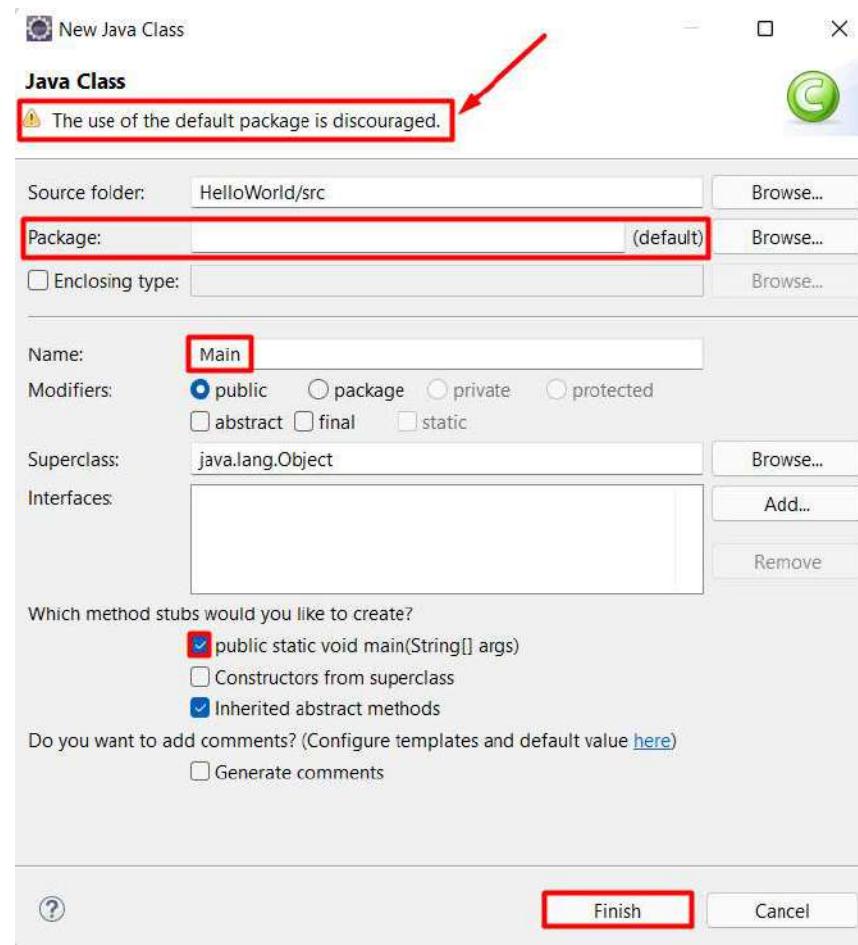
¿Qué es el default package?

- Cuando creamos una clase si no le especificamos un package, esta trabajará sobre lo que se conoce como default package. Vamos a ver un ejemplo:
- Para ello, primeramente vamos a src y creamos una Clase:



¿Qué es el default package?

- Dentro de la clase, si no definimos un package, estaremos utilizando el default package y podemos ver como Eclipse nos arroja un mensaje desaconsejando que trabajemos sobre el default package:
- Si omitimos la advertencia y pulsamos finish:



TIPOS DE IMPORTS



Tipos de imports

- Dentro de los packages podemos realizar distintos tipos de imports:
 - `import nombrePackage.nombreClase` para importar únicamente la clase nombreClase del package nombrePackage.
 - `import nombrePackage.*` para todas las clases del package nombrePackage.
- Si solamente necesitamos trabajar con una clase, es mejor realizar el import sobre dicha clase que sobre el package al completo. Ya que, sino, consumiremos más recursos.

PROBLEMAS CON EL DEFAULT PACKAGE



Problemas con el default package

- Vemos que el fichero Main.java se ha situado sobre el package default package:

The screenshot shows a Java development environment with the following details:

- Project Explorer:** Shows a project named "HelloWorld" with a "src" folder containing a "default package" which contains "Main.java" and "com.java" which contains "HelloWorld.java". It also lists "interfaces" and "JDBCExample [boot]".
- Main.java:** Contains the following code:

```
1 import com.java.*;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         HelloWorld helloworld = new HelloWorld();
7         helloworld.saludar();
8     }
9 }
```
- HelloWorld.java:** Contains the following code:

```
1 package com.java;
2
3 public class HelloWorld {
4     public void saludar() {
5         System.out.println("Hola mundo");
6     }
7 }
```
- Console:** Displays the output "Hola mundo".

- El principal problema de esto, es que no podemos hacer referencia a un package que trabaje con default package. En cambio, si que lo podemos realizar sobre un package que tenga no sea un package predefinido:

Problemas con el default package

- El principal problema de esto, es que no podemos hacer referencia a un package que trabaje con default package. En cambio, si que lo podemos realizar sobre un package que tenga no sea un package predefinido.
- Vamos a ver un ejemplo de como trabajar con la clase HelloWorld.java del package com.java en nuestro método main situado en el default package:

The screenshot shows an IDE interface with the following components:

- Project Explorer:** Shows a project named "HelloWorld" with a "src" folder containing a "com.java" package and a "HelloWorld.java" file.
- Main.java:** A code editor window showing the following code:

```
1 import com.java.*;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         HelloWorld helloWorld = new HelloWorld();
7         helloWorld.saludar();
8     }
9 }
```
- HelloWorld.java:** A code editor window showing the following code:

```
1 package com.java;
2
3 public class HelloWorld {
4     public void saludar() {
5         System.out.println("Hola mundo");
6     }
7 }
```
- Console:** At the bottom, the output window shows the text "Hola mundo".

Red boxes highlight the package imports in Main.java, the class definition in Main.java, the package declaration in HelloWorld.java, and the output "Hola mundo" in the console. Red arrows point from the highlighted code in Main.java to the corresponding parts in HelloWorld.java.

Problemas con el default package

- Otra opción sería importar únicamente la clase específica sobre la que queremos trabajar de la siguiente manera:

The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer view displays a project named 'HelloWorld' with a 'src' folder containing 'Main.java' and 'HelloWorld.java'. The 'HelloWorld.java' file is currently selected. In the center, the code editor shows 'Main.java' with the following code:

```
1 import com.java.HelloWorld;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         HelloWorld helloWorld = new HelloWorld();
7         helloWorld.saladar();
8     }
9 }
```

In the code editor, the 'HelloWorld.java' file is also visible on the right, showing:

```
1 package com.java;
2
3 public class HelloWorld {
4     public void saladar() {
5         System.out.println("Hola mundo");
6     }
7 }
```

A red arrow points from the 'HelloWorld.java' entry in the Project Explorer to the 'import com.java.HelloWorld;' statement in the 'Main.java' code editor.

- Si intentamos importar el default package en la clase HelloWorld.java al no tener un nombre definido dicho package no será permitido

Problemas con el default package

- Si quisiéramos arreglar esto, deberíamos de crear un package y mover los ficheros a dicho package para poder realizar el import en el otro

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with a tree view. A file named **Main.java** is selected and highlighted with a red box.
- Main.java Content:** Displays the code:

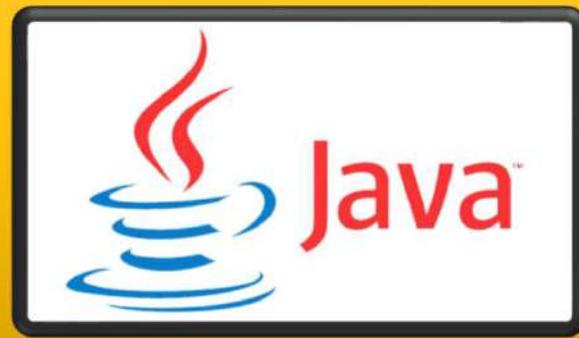
```
1 package com.main;
2 import com.java.HelloWorld;
3
4 public class Main {
5
6     public static void main(String[] args) {
7         HelloWorld helloWorld = new HelloWorld();
8         helloWorld.saludar();
9     }
10 }
```
- HelloWorld.java Content:** Displays the code:

```
1 package com.java;
2
3 import com.main.*;
4
5 public class HelloWorld {
6     public void saludar() {
7         System.out.println("Hola mundo");
8     }
9 }
10 }
```
- Problems View:** Shows several errors:
 - Main - com.main** (red box)
 - DomainCombiner - java.security**
 - DomainLoadStoreParameter - java.security**
 - ProtectionDomain - java.security**

Ejercicio de packages

- Realiza dos packages:
 - El primer package contendrá la clase Main.
 - El segundo package contendrá:
 - La clase HolaMundo que contendrá un método saludar() que nos retornará un Hola Mundo.
 - La clase HastaNunqui contendrá un método patada() que nos retornará ¡Hasta Nunqui!
- Situándonos en Main, haz un import individual de HolaMundo ¿Qué esta pasando? ¿Funciona?
- ¿Qué pasa si intentamos llamar a HastaNunqui sin importar el fichero? Explica las dos maneras distintas mediante a las cuales podemos solucionar esto.

RECUSIVIDAD





¿Qué es la recursividad?

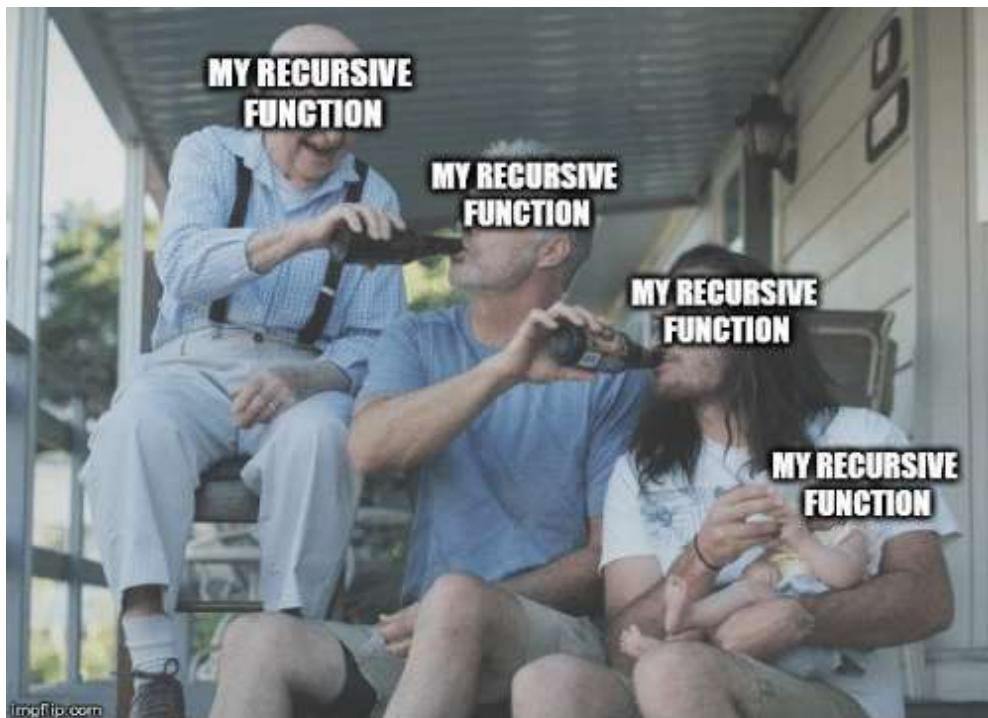
La recursividad es el concepto de llamar a un función o método dentro de si mismo.

Cuando creamos un método recursivo, debemos de tener en cuenta que este tiene que terminar en algún momento. Ya que sino, estaríamos realizando un bucle infinito. Para ello, debemos de introducir una condición dentro del método en la que analicemos una determinada circunstancia para finalmente salir de esta “recursión” que tendrá un comportamiento similar a un bucle.

Vamos a ver un ejemplo:

```
Main.java X
1 public class Main {
2     int num = 0; // Variable global
3
4     public static void main(String[] args) {
5         Main m = new Main();
6         m.saludar(); //Llamada al método recursivo
7     }
8
9     public void saludar() {
10        System.out.println("Hola");
11        num++;
12
13        if(num!=5) {
14            //Aquí se produce la "magia" de la recursividad
15            saludar();
16        }
17    }
18 }
```

The screenshot shows an IDE interface with two panes. The left pane displays the Java code for a class named Main. The right pane shows the output in the Console window, which consists of five lines of text: "Hola", each followed by a new line. A red rectangular box highlights the line "m.saludar(); //Llamada al método recursivo". Another red box highlights the recursive call "saludar();" inside the "if(num!=5)" block. A third red box highlights the annotation "//Aquí se produce la "magia" de la recursividad".



My Recursive Function





Ejercicios de recursividad



- 1. Recorre el siguiente array de forma recursiva
 - ```
String[] tecnologies = {"Markdown", "Regexp", "HTML", "CSS", "JS", "SQL", "Java"};
```
- 2. Recorre el array anterior hasta llegar a "HTML":
- 3. Fibonacci de forma recursiva hasta que un número pasado por parámetro dentro de la función recursiva supere el siguiente número recursivo:
  - Serie fibonacci: 1 1 2 3 5 8 13 21 34 55
- 4. Invierte la palabra supercalifragilisticosespialidoso mediante a una función recursiva
- 5. Suma los números de un número pasado por parametros de forma recursiva.

# **PROGRAMACIÓN ORIENTADA A OBJETOS**



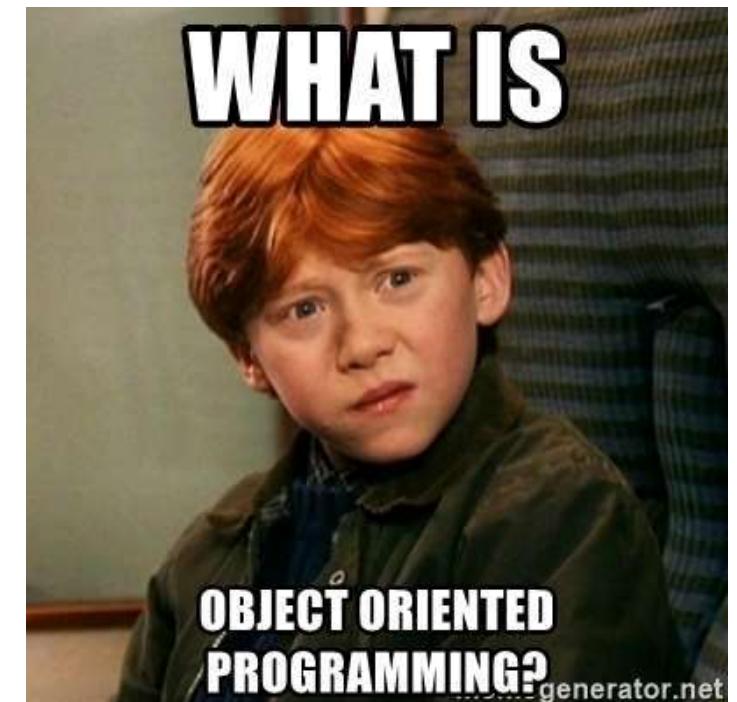


# ¿QUÉ SON LOS PARADIGMAS DE PROGRAMACIÓN?

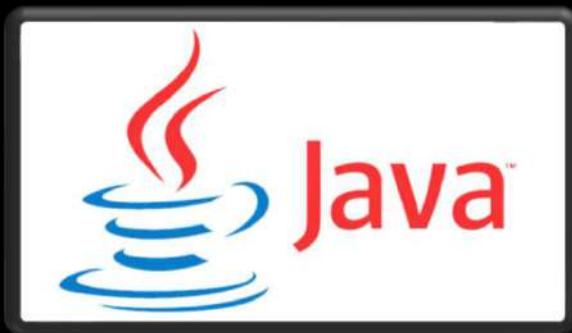


# ¿Qué son los paradigmas de programación?

- Un paradigma es una forma de afrontar la construcción de código de software. Es decir, los diferentes estilos de programación que nos permitirán resolver un problema.
  - No hay paradigmas mejores ni peores
  - Todos tienen sus ventajas e inconvenientes
- Hay distintos paradigmas. Algunos de ellos son:
  - Programación Orientada a Objetos (POO): crea objetos a partir de clases los cuales tendrán características y comportamientos diferentes.
  - Programación estructurada: nos permite utilizar ciclos, condicionales, etc.
  - Programación funcional: nos permite dividir el código de nuestro programa en pequeñas tareas que son contenidas dentro de funciones



# ¿QUÉ ES LA PROGRAMACIÓN ORIENTADA A OBJETOS?



# ¿Qué es la programación orientada a objetos?



- Detrás de la programación orientada a objetos (un paradigma de programación) hay una filosofía. Y java es un lenguaje que abraza fuertemente a dicha filosofía.
- La programación orientada a objetos se basa en la creación de una clase (una especie de molde) mediante el cual instanciamos (creamos) los objetos que tendrán atributos y comportamientos distintos.
- La POO se basa en 4 pilares:
  - Abstracción
  - Encapsulamiento
  - Herencia
  - Polimorfismo



# Explicando las 4 columnas



- Programación orientada a objetos se basa en 4 patas/columnas:
  - **Abstracción:** Es el proceso en el que se define los atributos y los métodos de una clase.
  - **Encapsulamiento:** Protege la información de manipulaciones no autorizadas.
  - **Polimorfismo:** De la misma orden a varios objetos para respondan de diferentes maneras.
  - **Herencia:** Las clases hijo heredan atributos y métodos de las clases padre.

# ¿QUÉ ES UN CLASE?





# ¿Qué es una clase?

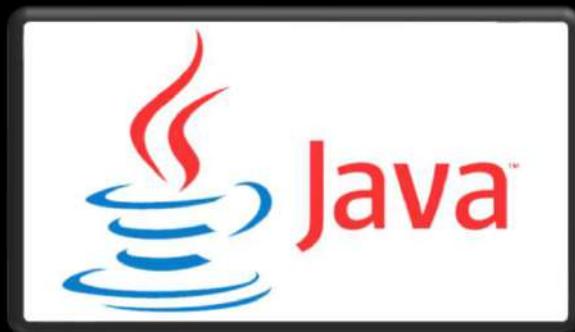
- La clase es la estructura en la que definiremos nuestros atributos (estructuras de datos) y métodos. Es decir, el molde que define las características que tendrán nuestros objetos una vez los instanciemos a partir de dicha clase.
- **Por tanto, una clase es una plantilla para definir elementos (objetos).**



|                  |             |
|------------------|-------------|
| Persona          | Nombre      |
| nombre           | Atributos   |
| apellido materno |             |
| apellido paterno |             |
| sexo             |             |
| edad             |             |
| comer()          | Operaciones |
| beber()          |             |
| dormir()         |             |



# PILARES DE LA POO

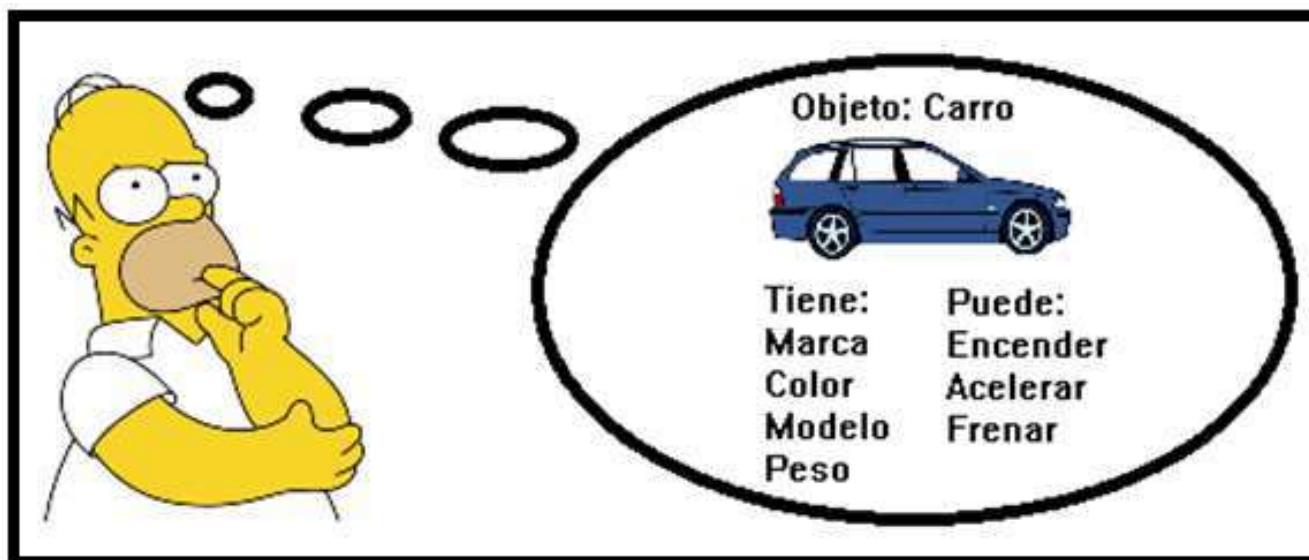


# ABSTRACCIÓN

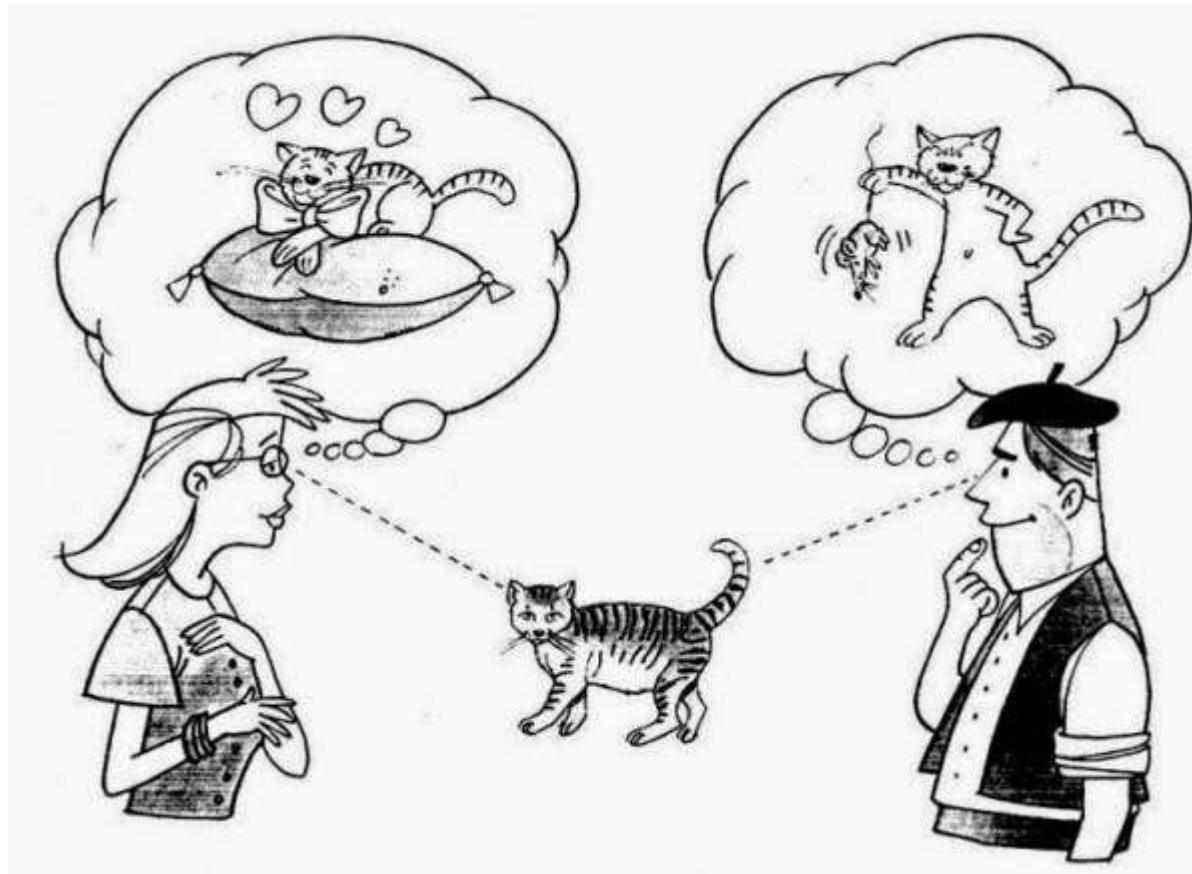


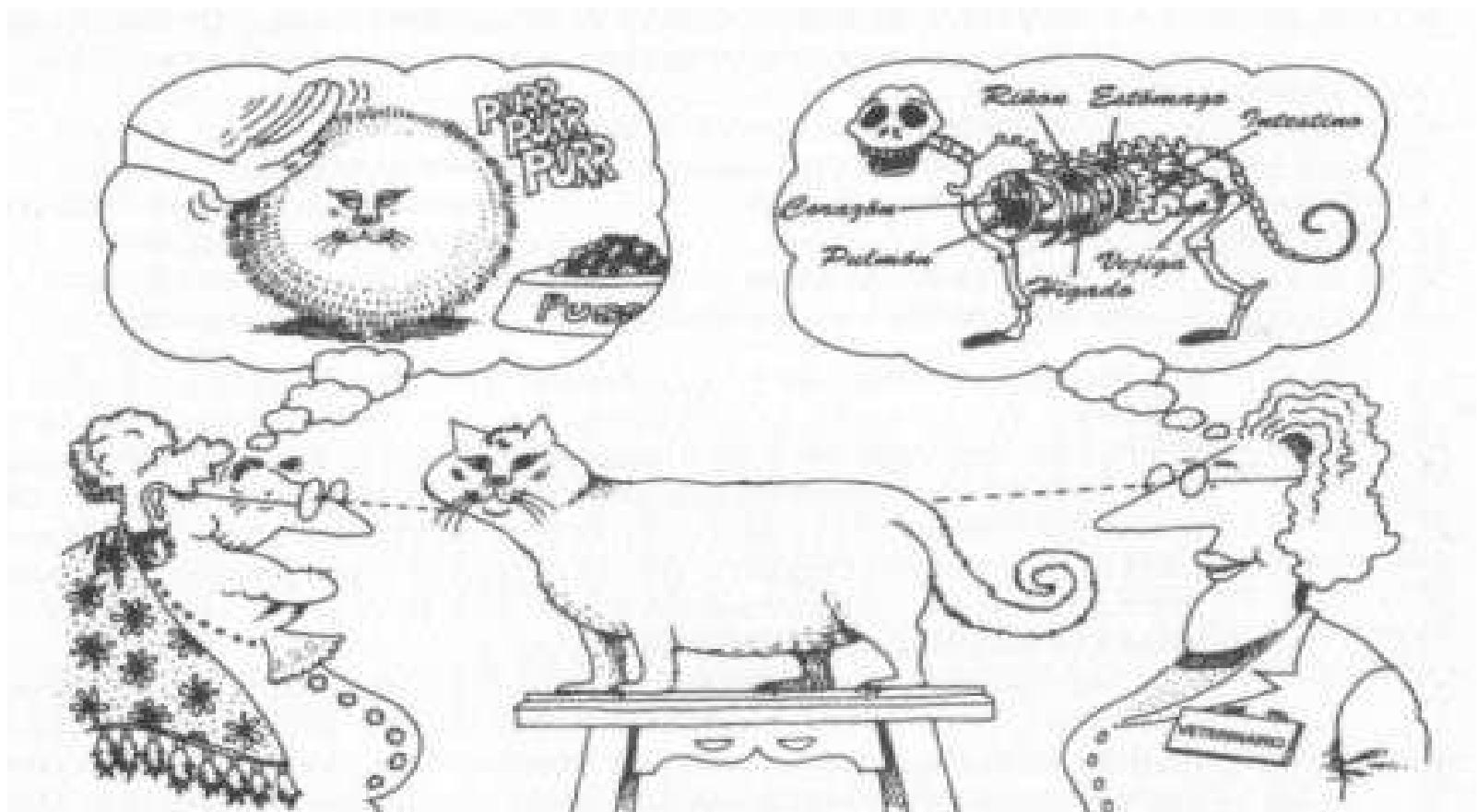
# POO: Abstracción

- La abstracción consiste en el proceso en el que nosotros como desarrolladores debemos de abstraernos con la finalidad de definir las características (propiedades y métodos) que pueden ser importantes a la hora de crear nuestros objetos.
- Por ejemplo, si queremos crear un coche debemos pensar si...



- Cada persona puede abstraerse de una manera diferente. Ya que abstracción es algo que cada persona hace internamente:





La abstracción se centra en las características esenciales de algún objeto, en relación a la perspectiva del observador.

# ¿QUÉ ES UN OBJETO?





# ¿Qué es un objeto?

- Los objetos son los elementos que instanciamos (creamos) a partir de la clase que podría servirnos como una “especie de un molde”. Los objetos, habitualmente tienen propiedades diferentes entre ellos, aunque también podrían existir dos objetos con las mismas propiedades.
- Una de las características de la POO es que los objetos estén separados pero que puedan comunicarse entre sí.

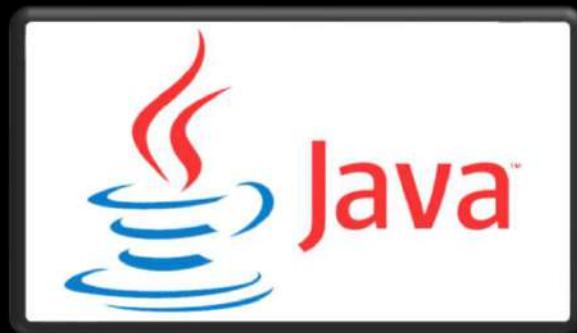
Clase

| Perro      |
|------------|
| • Nombre   |
| • Raza     |
| • Altura   |
| • Comer()  |
| • Dormir() |
| • Ladrar() |

Objeto



# CREANDO UNA CLASE E INSTANCIANDO OBJETOS



# Maneras de crear una clase

- Una vez nos hemos abstraído y hemos pensado que métodos ya podemos crear nuestra Clase, debemos definir sus atributos y métodos.
- Para ello, podemos:
  - Definir la nueva clase en el mismo archivo que de otra clase (no es lo más recomendable)
  - Definir la nueva clase en un fichero externo (es lo más recomendable)

# Compartiendo clase

- Definir la nueva clase en el mismo archivo que de otra clase (no es lo más recomendable).  
Pero si queremos hacerlo realizamos lo siguiente:

The screenshot shows an IDE interface with two windows. On the left, the code editor window titled 'Main.java' displays Java code. On the right, the 'Console' window shows the output of running the application.

**Main.java**

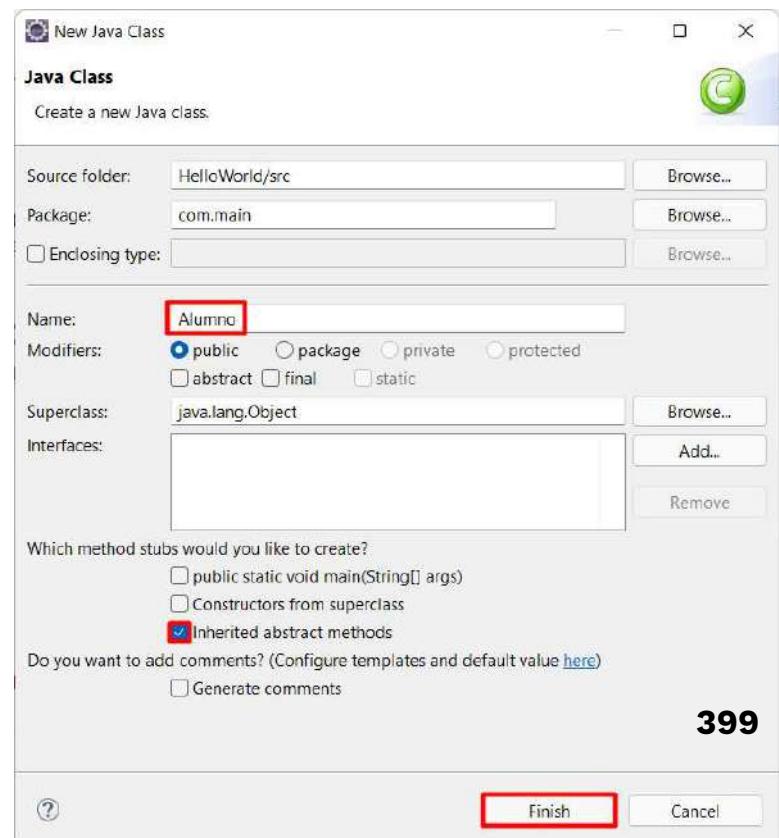
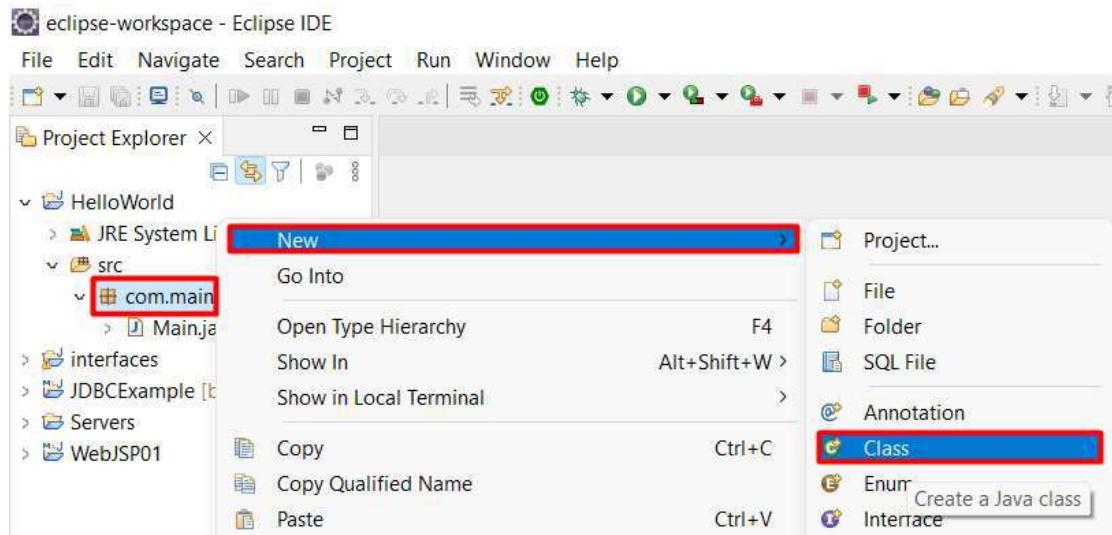
```
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 // Instanciando a la clase Alumno
6 Alumno alumno = new Alumno();
7 //Añadiendo valores a las propiedades
8 alumno.nombre = "David";
9 alumno.apellidos = "Bernal González";
10 //Llamando a los métodos
11 alumno.getInfo();
12 }
13 }
14
15 class Alumno{
16 //Atributos
17 String nombre;
18 String apellidos;
19 String telefono;
20 // Métodos
21 public void getInfo(){
22 System.out.println(this.nombre + " " + this.apellidos + " " + this.telefono);
23 }
24 }
```

**Console**

```
<terminated> Main (2) [Java Application] C:\Pr
David Bernal González null
```

# En un fichero individual

- Definir la nueva clase en un fichero externo (es lo más recomendable). Para ello, realizamos lo siguiente:



# En un fichero individual

- Una vez creada la clase, moverlos los atributos y métodos y ya está:

eclipse-workspace - HelloWorld/src/com/main/Alumno.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer X Main.java X Alumno.java X

Main.java

```
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 // Instanciando a la clase Alumno
6 Alumno alumno = new Alumno();
7 //Añadiendo valores a las propiedades
8 alumno.nombre = "David";
9 alumno.apellidos = "Bernal González";
10 //Llamando a los métodos
11 alumno.getInfo();
12 }
13 }
14 }
```

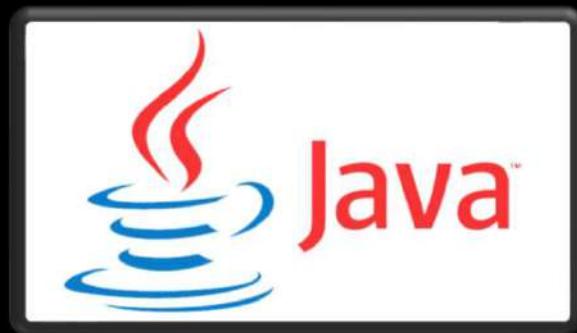
Alumno.java

```
1 package com.main;
2
3 public class Alumno {
4 //Atributos
5 String nombre;
6 String apellidos;
7 String telefono;
8 // Métodos
9 public void getInfo(){
10 System.out.println(this.nombre + " " + this.apellidos
11 + " " + this.telefono);
12 }
13 }
14 }
```

Console X

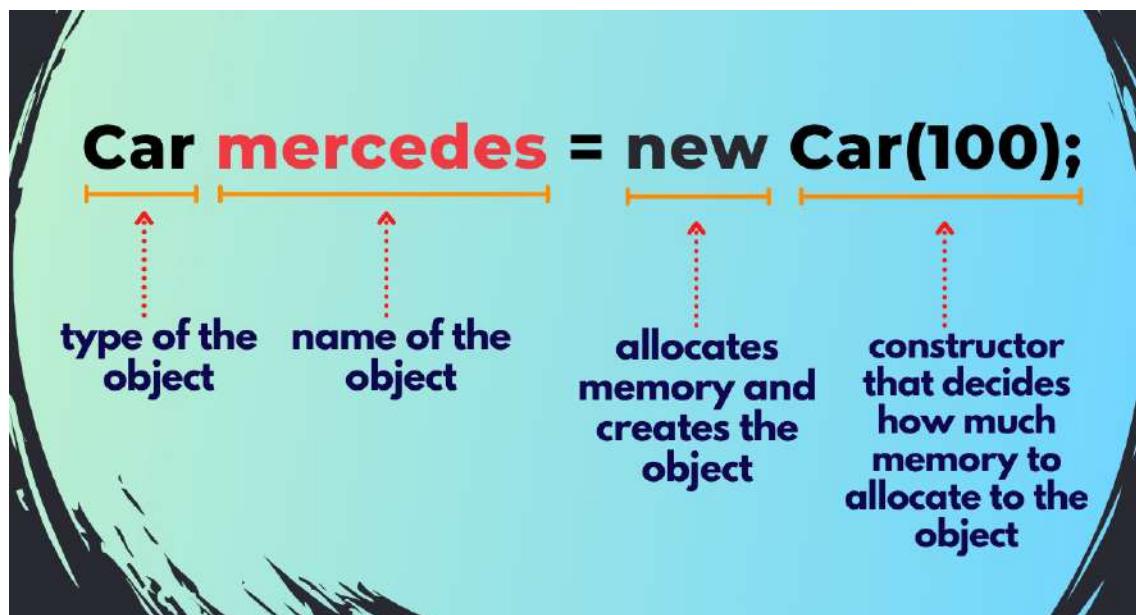
```
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe
David Bernal González null
```

# ¿QUÉ ES LA INSTANCIACIÓN?



# Entendiendo la instancia

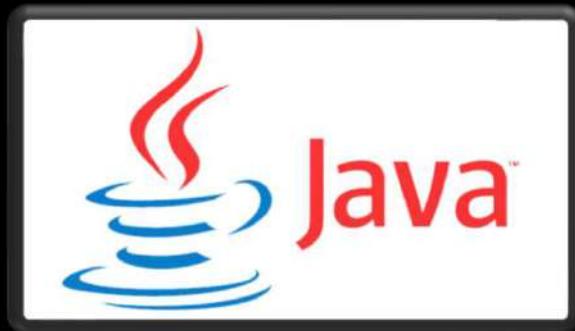
- La instancia es el proceso de creación de un objeto que se creará a partir de una clase. Vamos a ver un ejemplo:



# ENCAPSULAMIENTO



# ¿QUÉ ES EL ENCAPSULAMIENTO?



404

# POO: Encapsulamiento

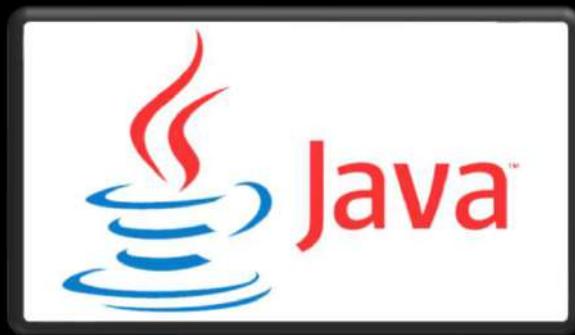
- El encapsulamiento nos permite limitar/restringir el acceso a unos atributos/propiedades de un objeto con la finalidad de evitar que estos atributos puedan ser modificados desde cualquier lugar.
- Actualmente, estamos realizando lo siguiente y vemos que podemos acceder a dichos métodos sin problema ya que no tenemos ningún modificador de acceso aplicado sobre dichos atributos:

```
//Añadiendo valores a las propiedades
alumno.nombre = "David";
alumno.|
```

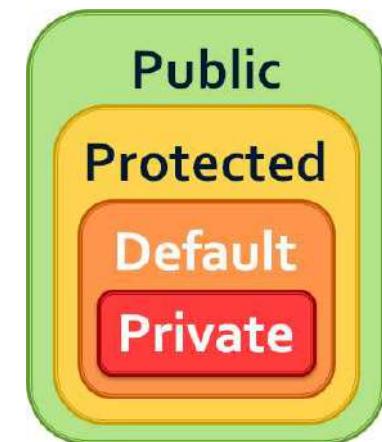
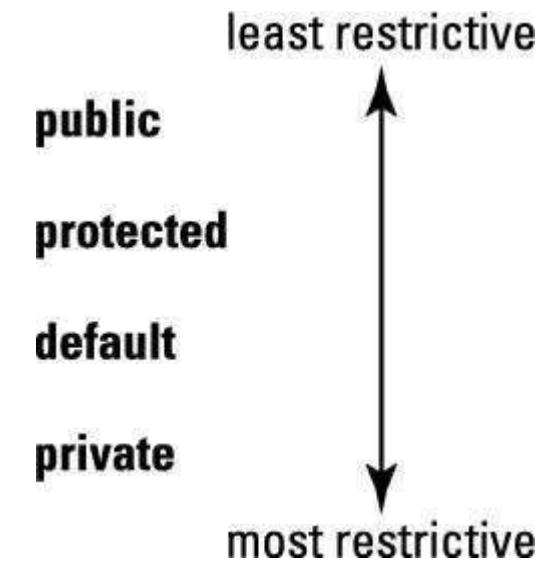
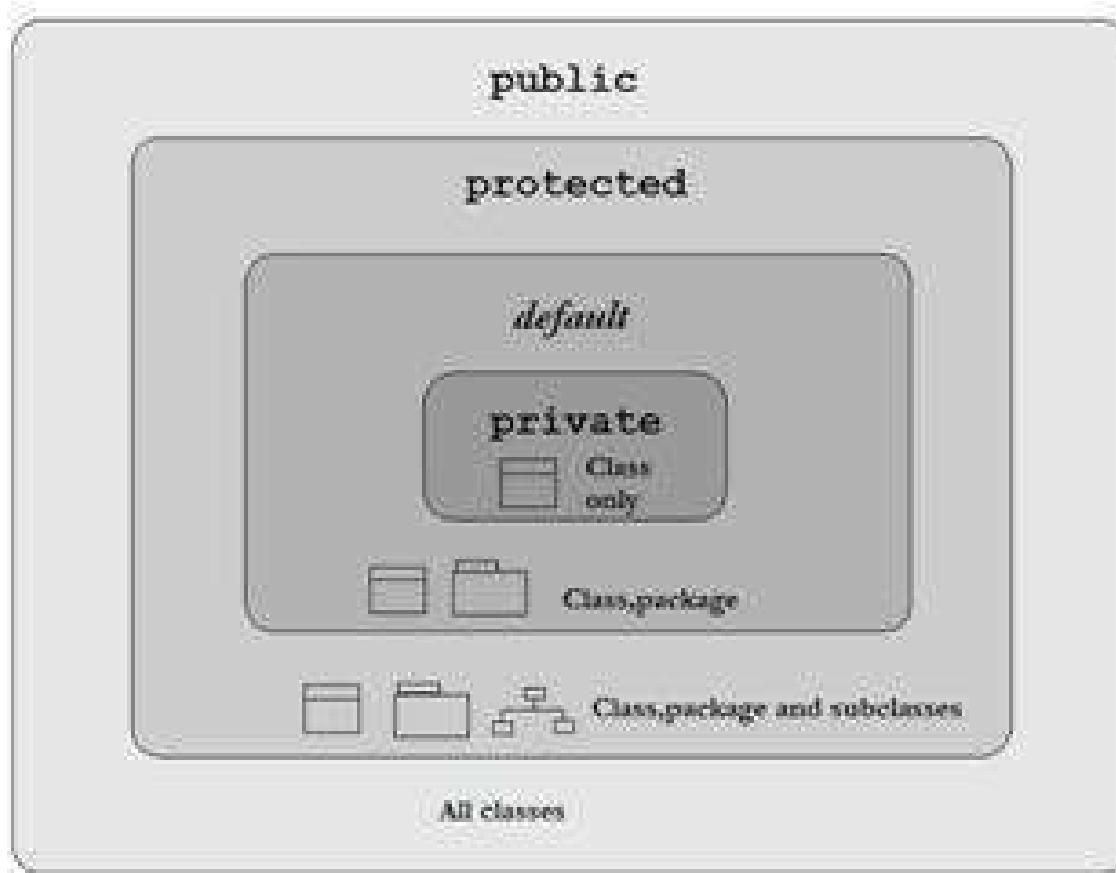
apellidos : String - Alumno  
nombre : String - Alumno  
telefono : String - Alumno  
equals(Object obj) : boolean - Object  
getClass() : Class<?> - Object

```
Alumno.java ×
1 package com.main;
2
3 public class Alumno {
4 //Atributos
5 String nombre;
6 String apellidos;
7 String telefono;
8
9 // Metodos
10 public void getInfo(){
11 System.out.println(this.nombre +
12 " " + this.apellidos +
13 " " + this.telefono);
14 }
15}
```

# MODIFICADORES DE ACCESO



# POO: Encapsulamiento



# POO: Encapsulamiento

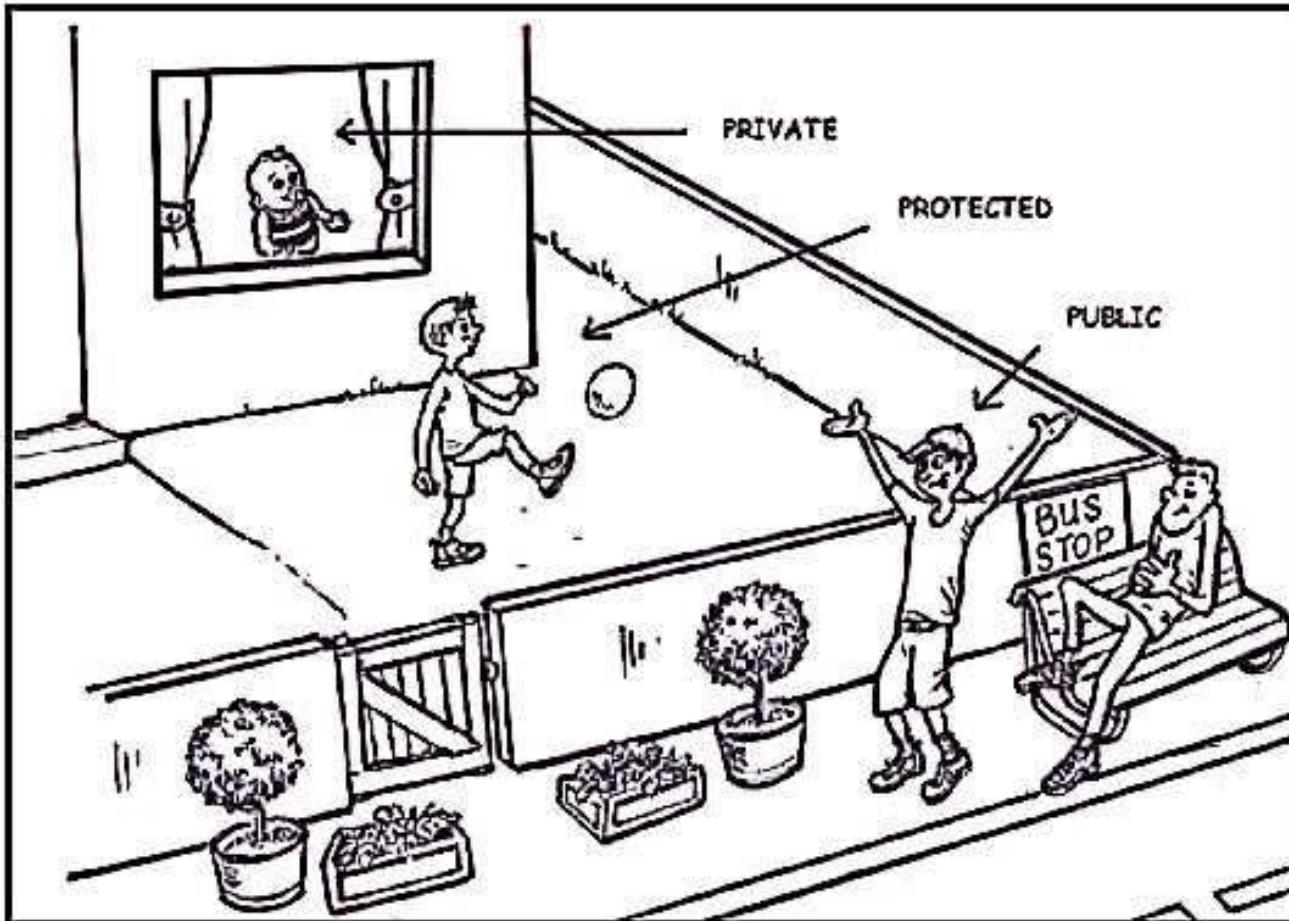
## Access Specifiers in Java

|                   |               | public | private | protected | default |
|-------------------|---------------|--------|---------|-----------|---------|
| Same Package      | Class         | YES    | YES     | YES       | YES     |
|                   | Sub class     | YES    | NO      | YES       | YES     |
|                   | Non sub class | YES    | NO      | YES       | YES     |
| Different Package | Sub class     | YES    | NO      | YES       | NO      |
|                   | Non sub class | YES    | NO      | NO        | NO      |

# POO: Encapsulamiento

| Modifier  | Class | Package | Subclass | Global |
|-----------|-------|---------|----------|--------|
| Public    | Yes   | Yes     | Yes      | Yes    |
| Protected | Yes   | Yes     | Yes      | No     |
| Default   | Yes   | Yes     | No       | No     |
| Private   | Yes   | No      | No       | No     |

# POO: Encapsulamiento



# POO: Encapsulamiento

- Por defecto, el modificador de acceso de los atributos si no declaramos nada es default. Que es accesible desde un mismo paquete. Pero si modificamos el modificar de los atributos de nuestra clase Alumno a private, vemos que dichos atributos no son accesibles desde fuera de nuestra clase:

The screenshot shows an IDE interface with two panes. On the left is a file tree for a project named 'HelloWorld' under 'src/com.main'. It contains 'Alumno.java' and 'Main.java'. On the right is the code editor for 'Alumno.java'.

```
1 package com.main;
2
3 public class Alumno {
4 //Atributos
5 private String nombre;
6 private String apellidos;
7 private String telefono;
8 // Métodos
9 public void getInfo(){
10 System.out.println(this.nombre +
11 " " + this.apellidos +
12 " " + this.telefono);
13 }
14 }
```

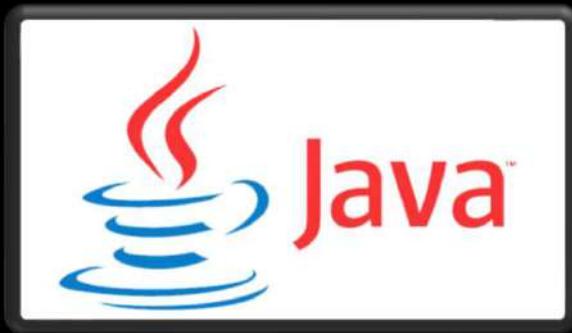
In the code editor, the three private attributes (`nombre`, `apellidos`, and `telefono`) are highlighted with red boxes. In the bottom-left corner of the IDE, there is a tooltip with the following message:

**The field Alumno.nombre is not visible**  
2 quick fixes available:  
Change visibility of 'nombre' to 'package'  
Create getter and setter for 'nombre'...  
Press 'F2' for focus

# **GETTERS Y SETTERS**



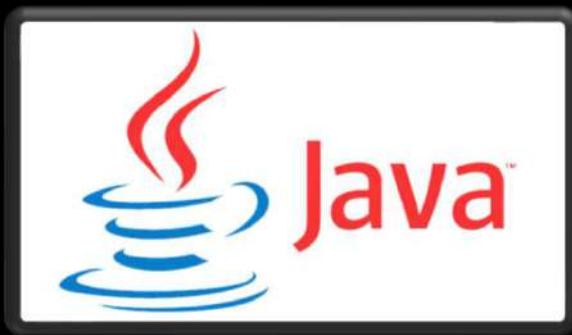
# ¿QUÉ SON LOS GETTERS Y SETTERS?



# ¿Qué son los Getters y Setters?

- Entonces ¿Cómo accedemos a los atributos? Desde sus:
  - **Getters**: para obtener el valor actual de un atributo encapsulado desde fuera de la clase a la que pertenece.
  - **Setters**: para modificar el valor actual de un atributo encapsulado de una clase desde fuera de la clase a la que pertenece.
- Los getters y setters de Java son utilizados habitualmente sobre las clases con la finalidad de obtener (get) o modificar (set) los valores de un atributo que esté encapsulado y al cual no podamos acceder por ejemplo desde fuera de la clase.

# CREACIÓN MANUAL DE GETTERS Y SETTERS



# Generando Getters y Setters: manualmente

- Getters: para obtener el valor actual de un atributo encapsulado desde fuera de la clase a la que pertenece.
- Setters: para modificar el valor actual de un atributo encapsulado de una clase desde fuera de la clase a la que pertenece.
  - Vamos a ver un ejemplo:

```
Main.java X
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 // Instanciando a la clase Alumno
6 Alumno alumno = new Alumno();
7 //Añadiendo valores a las propiedades
8 alumno.setNombre("David");
9 System.out.println(alumno.getNombre());
10 }
11 }
```

```
Alumno.java X
1 package com.main;
2
3 public class Alumno {
4 //Atributos
5 private String nombre;
6 private String apellidos;
7 private String telefono;
8 // Métodos
9 public void getInfo(){
10 System.out.println(this.nombre +
11 " " + this.apellidos +
12 " " + this.telefono);
13 }
14
15 public String getNombre() {
16 return this.nombre;
17 }
18
19 public void setNombre(String nombre) {
20 this.nombre = nombre;
21 }
22 }
```

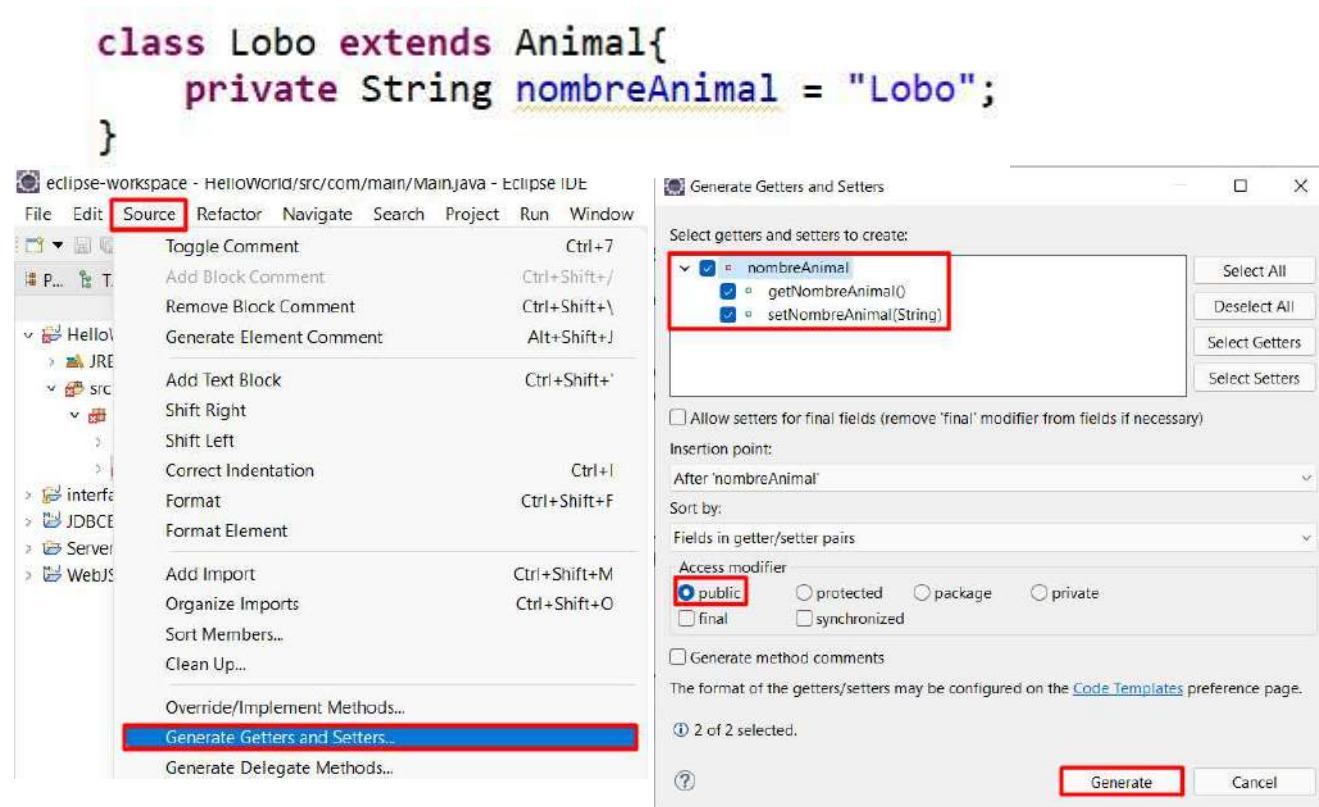
```
Console X
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (1 may. 2022)
David
```

# **CREACIÓN DE GETTERS Y SETTERS CON ECLIPSE**



# Generando Getters y Setters: con Eclipse

- Para generar los getters y setters de forma automática hacemos lo siguiente:



# Generando Getters y Setters: con Eclipse

- Finalmente, podemos ver como se ha generado el get y el set correspondiente al único atributo que tiene actualmente disponible la clase:

```
class Lobo extends Animal{
 private String nombreAnimal = "Lobo";

 public String getNombreAnimal() {
 return nombreAnimal;
 }

 public void setNombreAnimal(String nombreAnimal) {
 this.nombreAnimal = nombreAnimal;
 }
}
```

# **CONSTRUCTORES**



# ¿Qué son los constructores?

- Los constructores nos permiten instanciar (crear) objetos a partir de las clases. Por defecto, Java utiliza un constructor vacío. Vamos a ver un ejemplo:
  - Si utilizamos el constructor con parámetros:

The screenshot shows a Java development environment with two windows. On the left is the code editor for 'Main.java' containing the following code:

```
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 // Animal
6 Animal animal = new Animal("Pollito");
7 animal.makeSound();
8 }
9 }
10
11 class Animal {
12 String nombre = "";
13 public void makeSound() {
14 System.out.println(this.getClass().getName() +
15 "-> Pio Pio...");
```

A red box highlights the line `Animal animal = new Animal("Pollito");`. Another red box highlights the constructor definition `public Animal(String nombre) {` and its body.

On the right is the 'Console' window showing the output of the program:

```
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (2 may, 2022 23:22)
Se ha instanciado un objeto desde el constructor con parametros
com.main.Animal-> Pio Pio...
```

A red arrow points from the highlighted line in the code editor to the corresponding text in the console output.

# ¿Qué son los constructores?

- Los constructores nos permiten instanciar (crear) objetos a partir de las clases. Por defecto, Java utiliza un constructor vacío por lo que definirlo no es necesario. Vamos a ver un ejemplo:
  - Si utilizamos el constructor sin parámetros:

The screenshot shows an IDE interface with two windows. On the left is the code editor for 'Main.java' containing Java code. On the right is the 'Console' window showing the output of the program's execution.

```
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 // Animal
6 Animal animal = new Animal();
7 animal.makeSound();
8 }
9 }
10
11 class Animal {
12 String nombre = "";
13 public void makeSound() {
14 System.out.println(this.getClass().getName() +
15 "-> Pio Pio...");
```

The code editor has several sections highlighted with red boxes:

- A red box surrounds the line `Animal animal = new Animal();` in the `Main` class.
- A red box surrounds the entire `Animal` class definition.
- A red box surrounds the `public Animal()` constructor definition.
- A red box surrounds the `System.out.println` statement in the `makeSound` method.

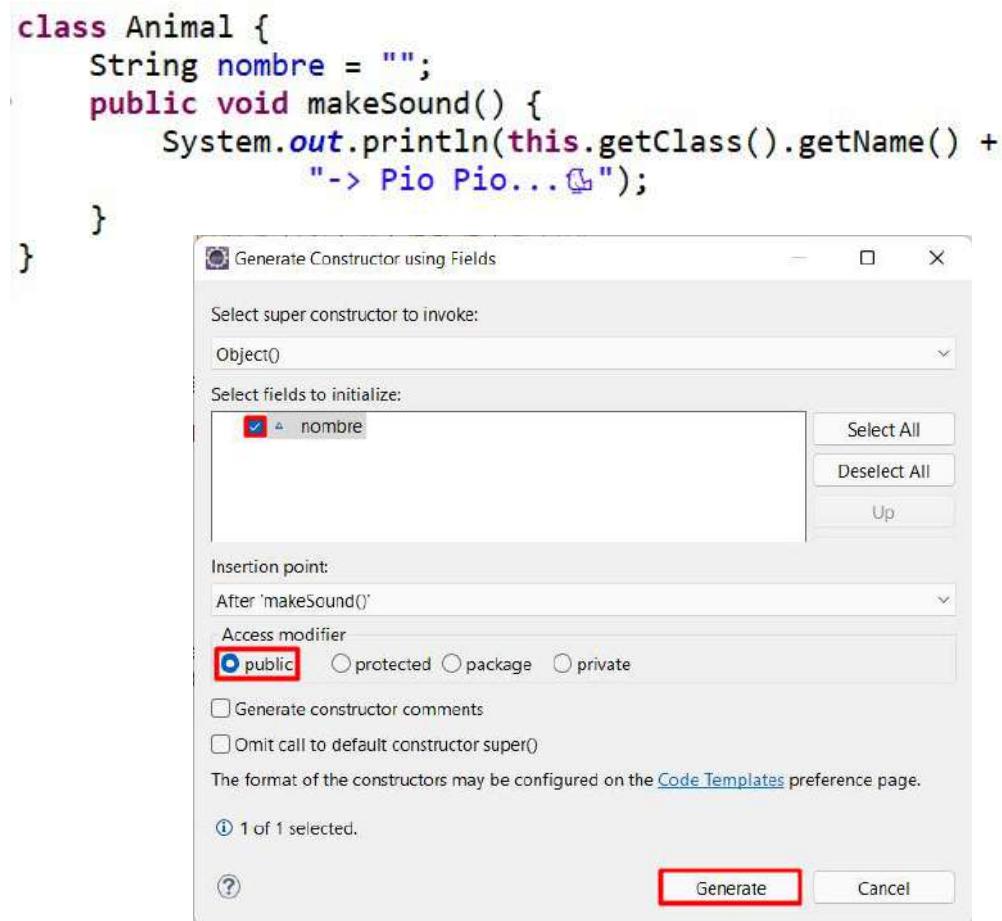
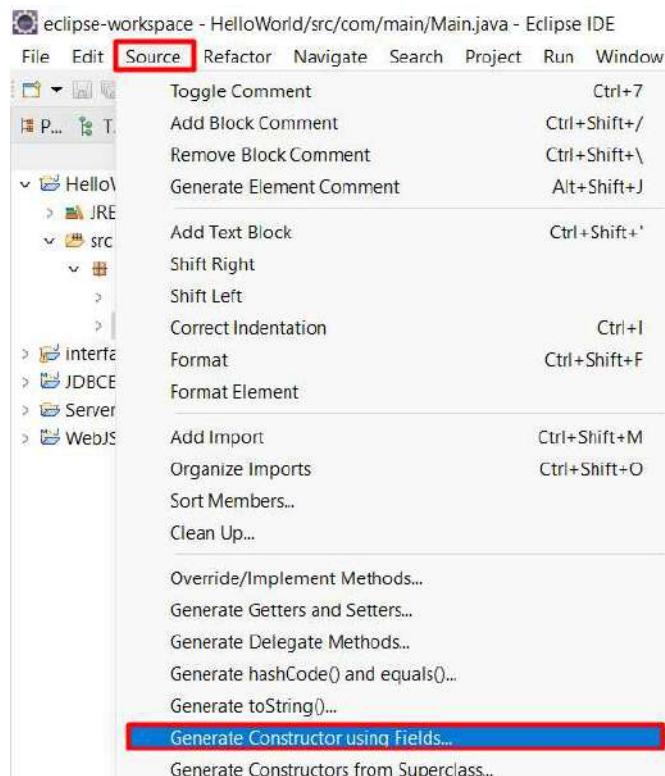
The 'Console' window shows the following output:

```
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (2 may. 2022 23:55)
Se ha instanciado un objeto desde el constructor sin parametros
com.main.Animal-> Pio Pio...
```

Two red arrows point from the highlighted code in the code editor to the corresponding lines in the console output. One arrow points from the `new Animal()` line to the first line of the console output. Another arrow points from the `public Animal()` constructor definition to the second line of the console output.

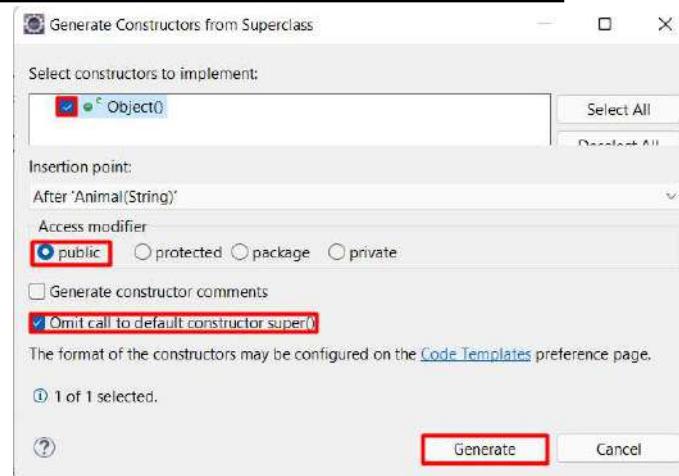
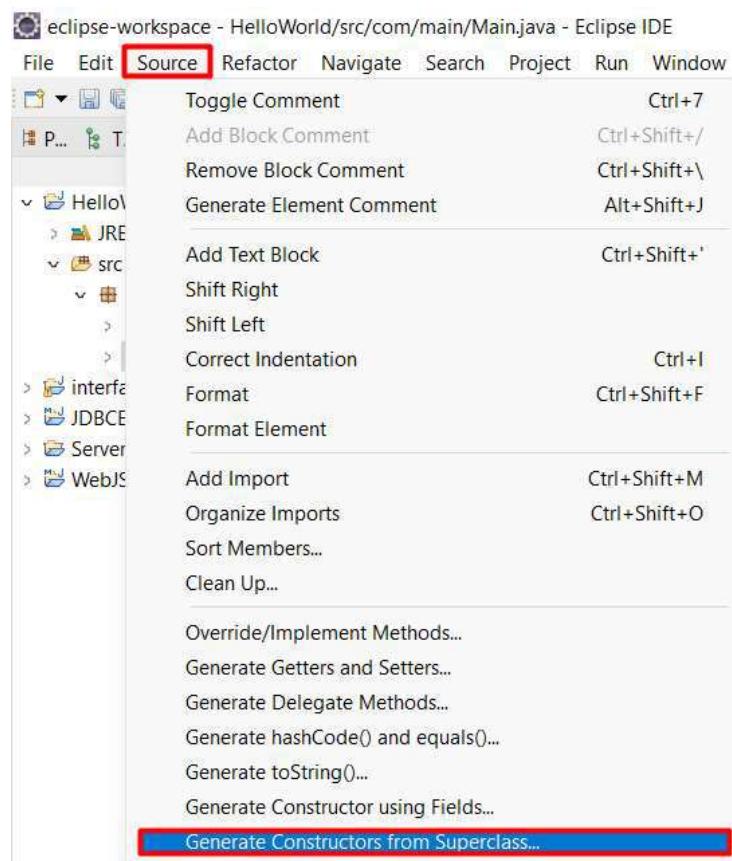
# Autogenerando los constructores

- Basandonos en la siguiente clase:



423

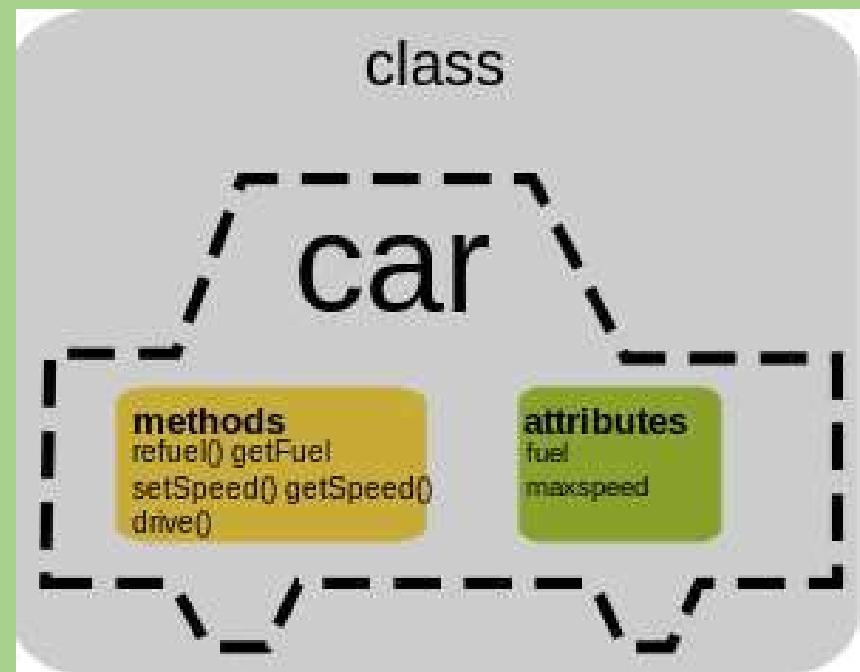
# Autogenerando los constructores



```
class Animal {
 String nombre = "";
 public void makeSound() {
 System.out.println(this.getClass().getName() +
 "-> Pio Pio...");
 }
 public Animal(String nombre) {
 this.nombre = nombre;
 }
 public Animal() {}
}
```

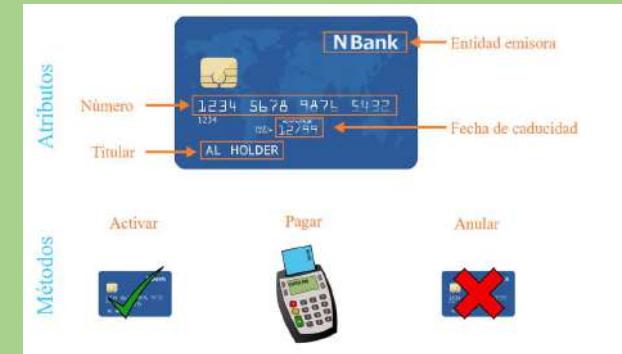
# Ejercicio de clases

- Crea la siguiente clase e instancia 3 objetos.  
Aspectos a tener en cuenta:
  - Los atributos deberán de estar encapsulados con private
  - La clase deberá de tener definidos una serie de métodos (getters y setters) que nos permitirán obtener y modificar los valores.
  - Crea 3 objetos en los que deberás de utilizar todos sus métodos (incluidos los getters y setters).
  - Los objetos deberán disponer de dos constructores. Uno para crearlos vacíos y otro para realizar la creación con todos los atributos.



# Ejercicio de clases

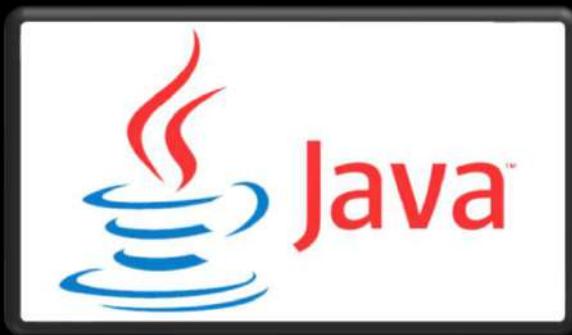
- Analiza el como implementar lo siguiente mediante a programación orientada a objetos de una forma similar a como lo has hecho en el ejercicio anterior:
- En este caso, deberemos de poder pagar solamente si la tarjeta está activa. Por lo que si está anulada no se podrá pagar.
- Y por defecto la tarjeta tendrá 1000 € como saldo. Y conforme se vayan realizando pagos, el crédito se irá reduciendo.
- Si una operación es demasiado supera el crédito actual, no se efectuará y aparecerá un mensaje por pantalla diciendo que el saldo es suficiente.
- Además, tendremos un método que nos permitirá ver el saldo actual.



# HERENCIA



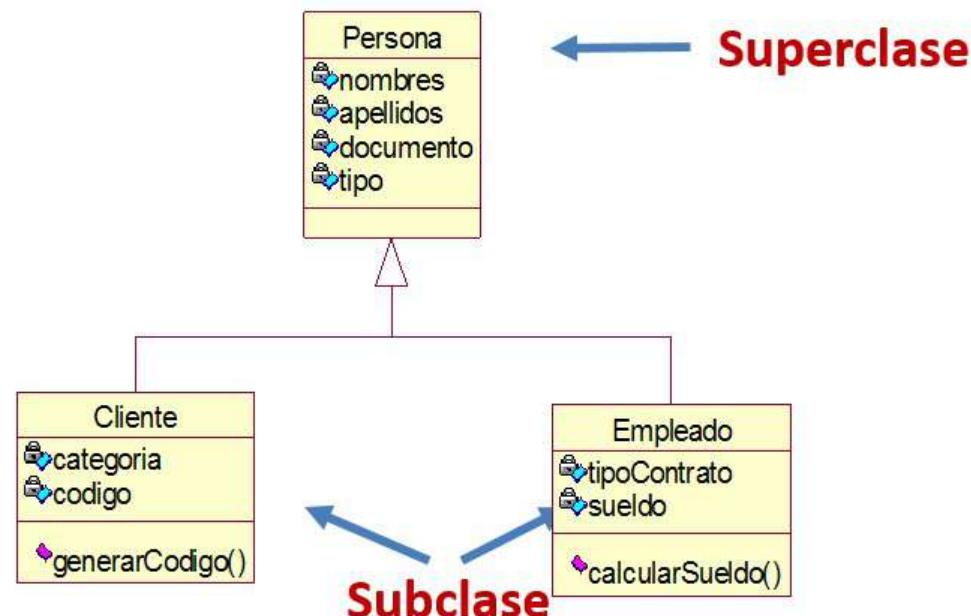
# ¿QUÉ ES LA HERENCIA?





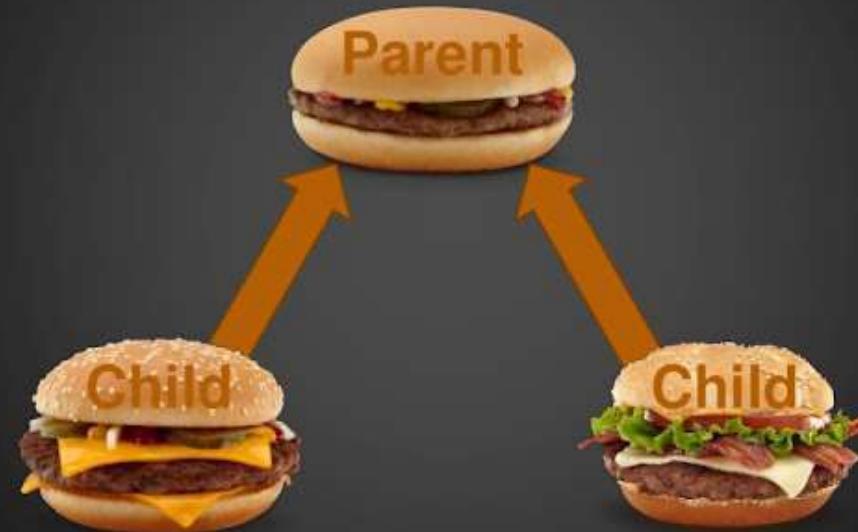
# Herencia

- La herencia nos permite que una clase herede características (atributos y métodos) de otras clase.
- La clase de la que se hereda se conoce como superclase y la clase que hereda como subclase.

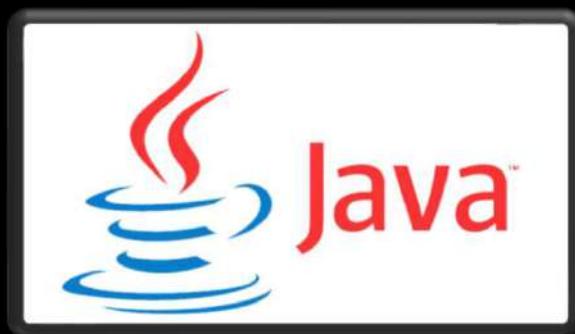


4 Major Principles for OOP

## Inheritance



# EJEMPLO DE HERENCIA



# Ejemplo de herencia

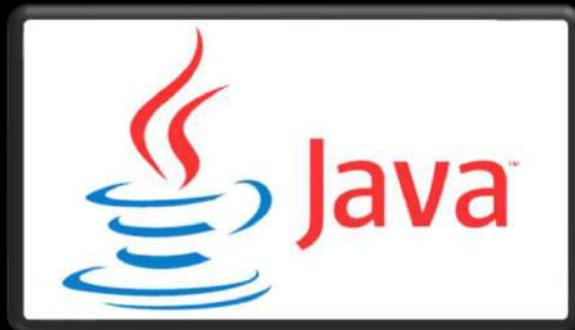
The screenshot shows a Java development environment with two windows. On the left is the code editor window titled 'Main.java X' containing the following Java code:

```
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 // Animal
6 Animal animal = new Animal();
7 animal.makeSound();
8 // Lobo
9 Lobo lobo = new Lobo();
10 lobo.getNombreAnimal();
11 lobo.makeSound();
12 }
13 }
14
15 class Animal {
16 public void makeSound() {
17 System.out.println(this.getClass().getName() + "-> Pio Pio...🐺");
18 }
19 }
20
21 class Lobo extends Animal{
22 private String nombreAnimal = "Lobo";
23
24 public void getNombreAnimal() {
25 System.out.println(this.getClass().getName() + " -> " + nombreAnimal);
26 }
27
28 public void makeSound() {
29 System.out.println(this.getClass().getName() + "-> Auuu...🐺");
30 }
31 }
```

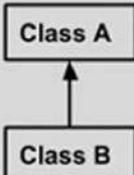
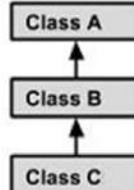
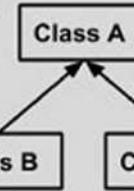
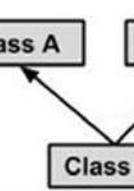
On the right is the 'Console X' window showing the output of the program:

```
<terminated> Main (2) [Java Application] C:\Program
com.main.Animal-> Pio Pio...🐺
com.main.Lobo -> Lobo
com.main.Lobo-> Auuu...🐺
```

# TIPOS DE HERENCIA

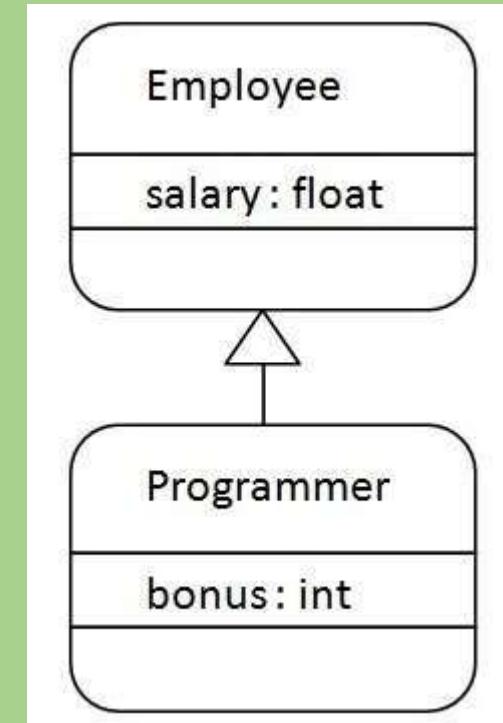


# Tipos de herencia

|                                                                                                                                                                                                                                                                                         |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Single Inheritance</b><br> <pre>public class A {<br/>.....<br/>}<br/>public class B extends A {<br/>.....<br/>}</pre>                                                                              |
| <b>Multi Level Inheritance</b><br> <pre>public class A { ..... }<br/>public class B extends A { ..... }<br/>public class C extends B { ..... }</pre>                                                  |
| <b>Hierarchical Inheritance</b><br> <pre>public class A { ..... }<br/>public class B extends A { ..... }<br/>public class C extends A { ..... }</pre>                                                |
| <b>Multiple Inheritance</b><br> <pre>public class A { ..... }<br/>public class B { ..... }<br/>public class C extends A,B {<br/>.....<br/>}<br/>// Java does not support multiple inheritance</pre> |

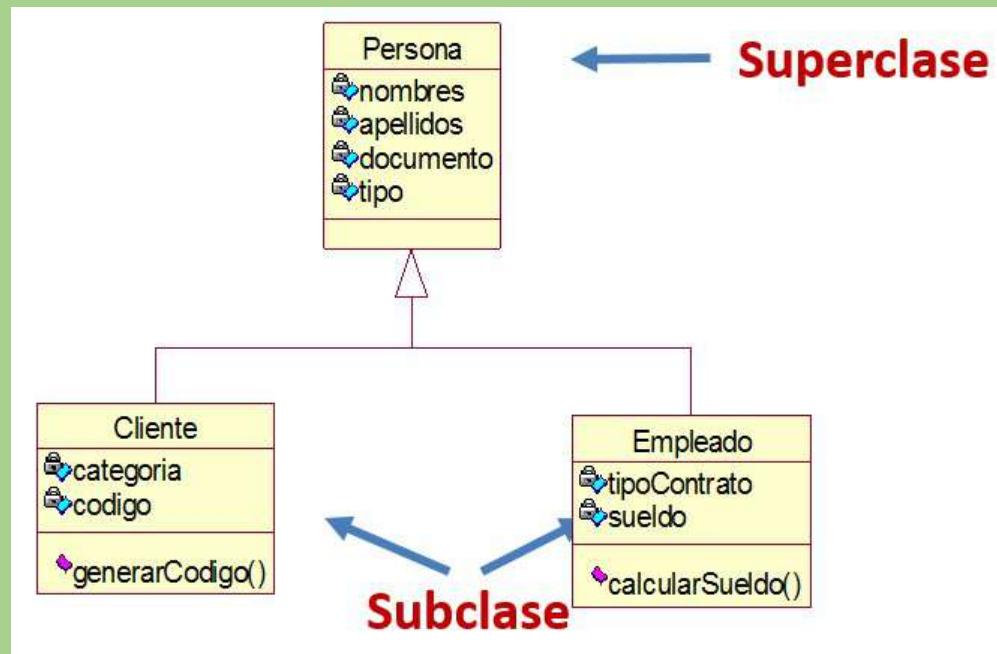
# Ejercicios de herencia

- Realiza la siguiente estructura de clases:
  - Observa que la superclase Employee no tiene métodos, solamente atributos.
  - Observar que las subclases Programmer hereda de Employee (por lo que deberá tener su atributo). Y, además, deberá tener un bonus por ser programador/a 



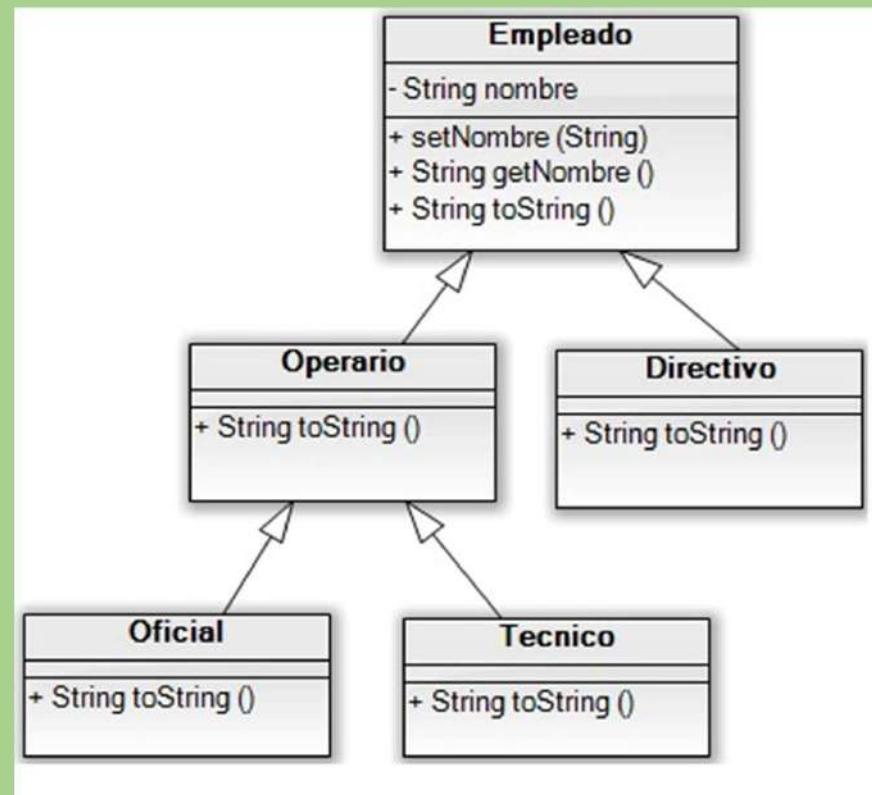
# Ejercicios de herencia

- Realiza la siguiente estructura de clases:
  - Observa que la superclase Persona no tiene métodos, solamente atributos.
  - Observar que las subclases Cliente y Empleado heredan de la superclase Persona y añaden atributos y un método cada una.



# Ejercicios de herencia

- Realiza la siguiente estructura de clases:



# **CLASES ABSTRACTAS**

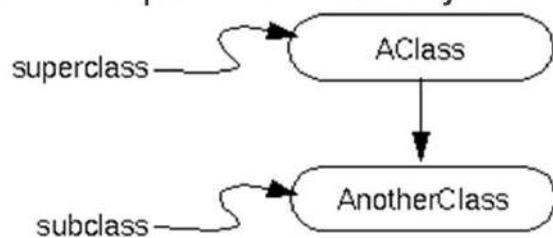


# ¿QUÉ SON LAS CLASES ABSTRACTAS?



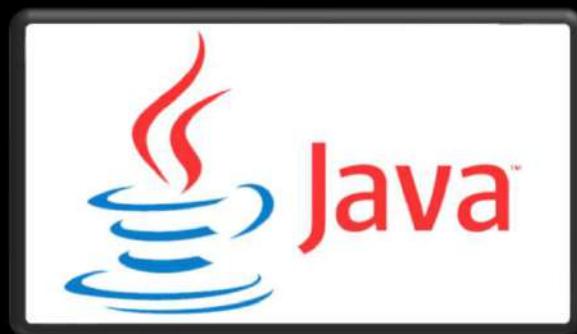
# ¿Qué son las clases abstractas?

- En Java, una clase abstracta es una especie de clase base o superclase la cual no se puede instanciar. Es decir, de la cual no se pueden crear objetos.



- Se utiliza como clase base de la cual heredaran otras clases. Y nos permite declarar métodos abstractos y no abstractos. Pero, si estos métodos son abstractos no podremos realizar la implementación de dichos métodos. Es decir, utilizar las llaves {} y las sentencias en las que habitualmente contenemos las instrucciones de nuestros métodos.
- Una clase abstracta no se puede instanciar pero si se puede heredar y las clases hijas serán las encargadas de agregar la funcionalidad a los métodos abstractos. En el caso de que las clases hijas no definan estos método es debido a que también son clases abstractas.

# CREANDO UNA CLASE ABSTRACTA

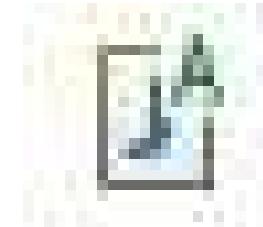
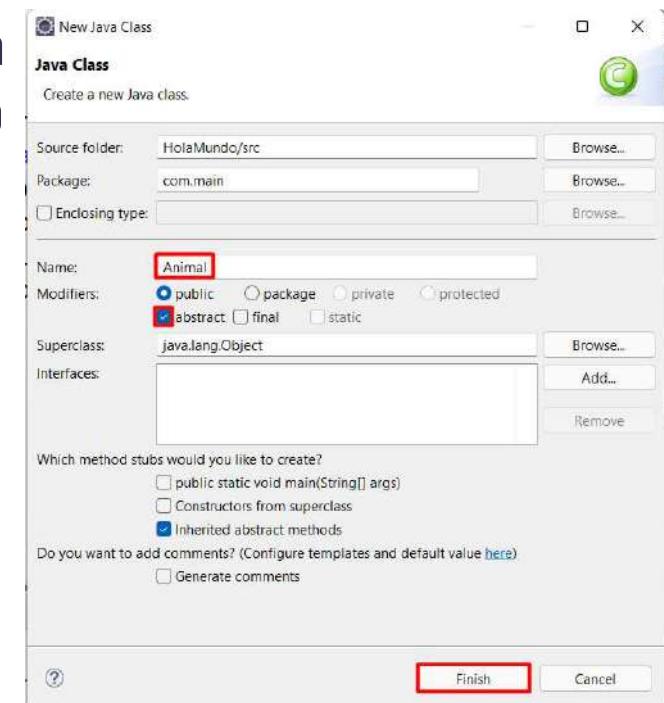


# Creando una clase abstracta

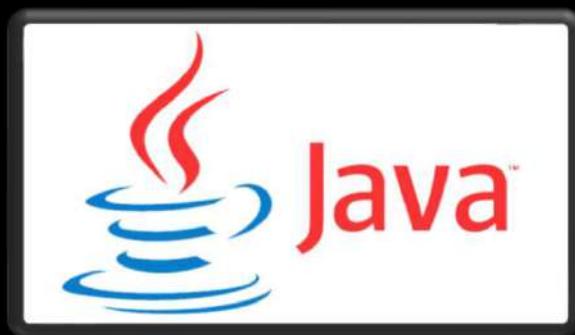
- Para definir una clase abstracta, o bien cuando creamos una clase con New>Class marcamos la pestaña de abstracta. Lo que añadirá a la clase entre el modificador de acceso y la palabra reservada class la palabra reservada abstract.
- O bien también podemos crear una clase normal y escribirle nosotros que sea abstract. Vamos a ver un ejemplo:

```
1 package com.main;
2
3 public abstract class Animal {
4
5 }
```

- Si os fijáis, en el ícono de las clases abstractas aparece una A como la siguiente:



# EJEMPLO DE USO DE CLASES ABSTRACTAS



# Ejemplo de uso de clases abstractas

- Vamos a ver un ejemplo de uso de clases abstractas:

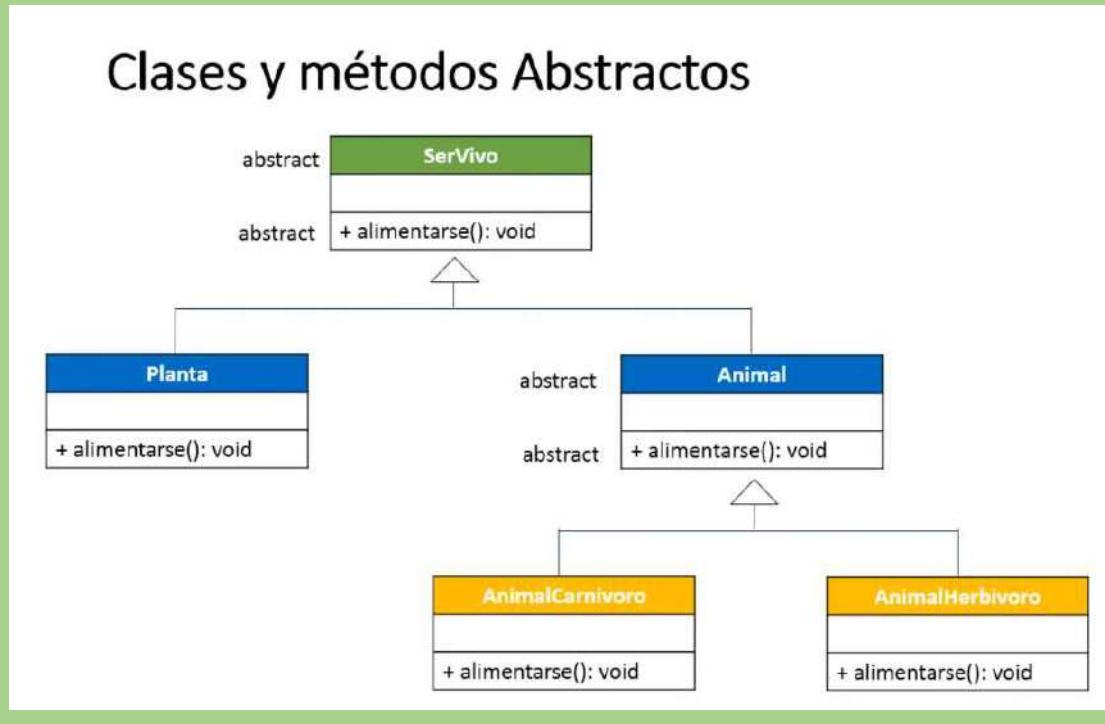
```
Animal.java x
1 package com.main;
2
3 public abstract class Animal {
4 public final int OJOS = 0;
5 private String nombre = "animal";
6
7 // Método abstracto
8 public abstract String sonido(String ruido);
9
10 // Método no abstracto
11 public String getNombre() {
12 return nombre;
13 }
14
15 public void setNombre(String nombre) {
16 this.nombre = nombre;
17 }
18}
```

```
Main.java x
1 package com.main;
2
3 public class Main extends Animal{
4 public static void main(String[] args) {
5 Main m = new Main();
6 // Atributo no privado de la clase abstrata Animal
7 System.out.println(m.OJOS);
8 /* Método GETTER & SETTER no abstracto
9 * heredado de la clase abstracta
10 * Animal. Nos permite acceder a un
11 * atributo encapsulado*/
12 m.setNombre("Lobo");
13 System.out.println(m.getNombre());
14 }
15
16 /*Método abstracto heredado de
17 * la clase abstracta animal*/
18 @Override
19 public String sonido(String ruido) {
20 return null;
21 }
22 }
```

- Una característica importante a destacar de las clases abstractas, es que solamente podemos extender de una clase no de varias.

# Ejercicio de clases abstractas

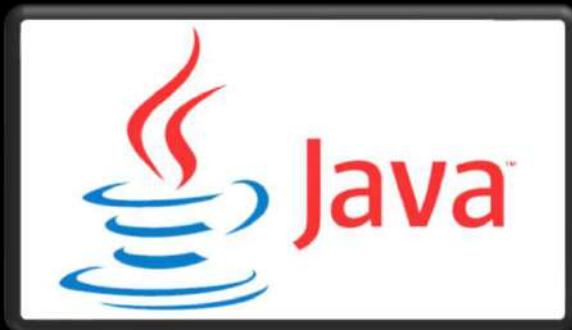
- Realiza lo siguiente:
  - Definiendo en las clases que no son abstractas la información de como se alimentan dichos ser vivos.



# INTERFACES



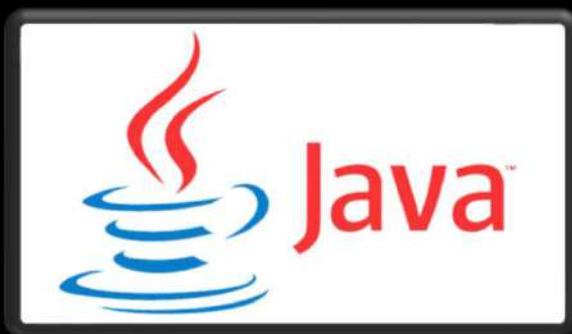
# ¿QUÉ SON LAS INTERFACES?



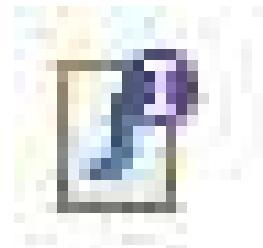
# ¿Qué son las interfaces?

- A diferencia de las clases abstractas, las interfaces solo pueden tener métodos abstractos y constantes. Y es una especie de contrato en el que se detalla una serie de métodos (abstractos) que tendrán que tener la clase que implemente a dicha interface.
- No es obligatorio definir explícitamente que los métodos son public y abstract (se puede omitir) ya que todos deberán de ser public y abstract.
- Pueden tener constantes definidas que se declaran como public, static y final.
- Usualmente cuando creamos una interface usualmente añadimos una I o Impl antes del nombre de la interfaz para que quede se detecte mucho más rápido que es una interface.

# CREANDO UNA INTERFACE



# Declarando una interface



- Para crear una interface hacemos lo siguiente:

The screenshot shows a Java code editor with a file named `IAnimal.java` open. The code defines a package `com.main` and an interface `IAnimal` with two methods: `ruido` and `tipoAnimal`. A context menu is open over the package name `com.main`, with the option `Interface` highlighted. The menu also includes options like `New`, `Go Into`, `Open Type Hierarchy`, `Show In`, `Show in Local Terminal`, `Copy`, `Copy Qualified Name`, and `Paste`.

```
1 package com.main;
2
3 public interface IAnimal {
4 //Por defecto será public static final
5 //Por lo que no hace falta definirlo
6 public static final String CONSTANTE_1 = "Const1";
7 String CONSTANTE_2 = "Const2";
8
9 //Por defecto será public abstract
10 //Por lo que no es necesario definirlo
11 public abstract String ruido(String ruido);
12 String tipoAnimal(String tipo);
13 }
```

# **IMPLEMENTANDO (UTILIZANDO) UNA INTERFACE**



# Implementando una interface

- La implementación de interfaces si que puede multiple inclusive. Vamos a ver un ejemplo:

The screenshot shows an IDE interface with two code editors and a console window.

**IAnimal.java:**

```
package com.main;
public interface IAnimal {
 //Por defecto será public static final
 //Por lo que no hace falta definirlo
 public static final String CONSTANTE_1 = "Const1";
 String CONSTANTE_2 = "Const2";

 //Por defecto será public abstract
 //Por lo que no es necesario definirlo
 public abstract String ruido(String ruido);
 String tipoAnimal(String tipo);
}
```

**Main.java:**

```
package com.main;
public class Main implements IAnimal{
 public static void main(String[] args) {
 Main m = new Main();
 System.out.println(CONSTANTE_1);
 System.out.println(CONSTANTE_2);
 System.out.println(m.ruido("Grr"));
 System.out.println(m.tipoAnimal("Mamifero"));
 }

 @Override
 public String ruido(String ruido) {
 return ruido;
 }

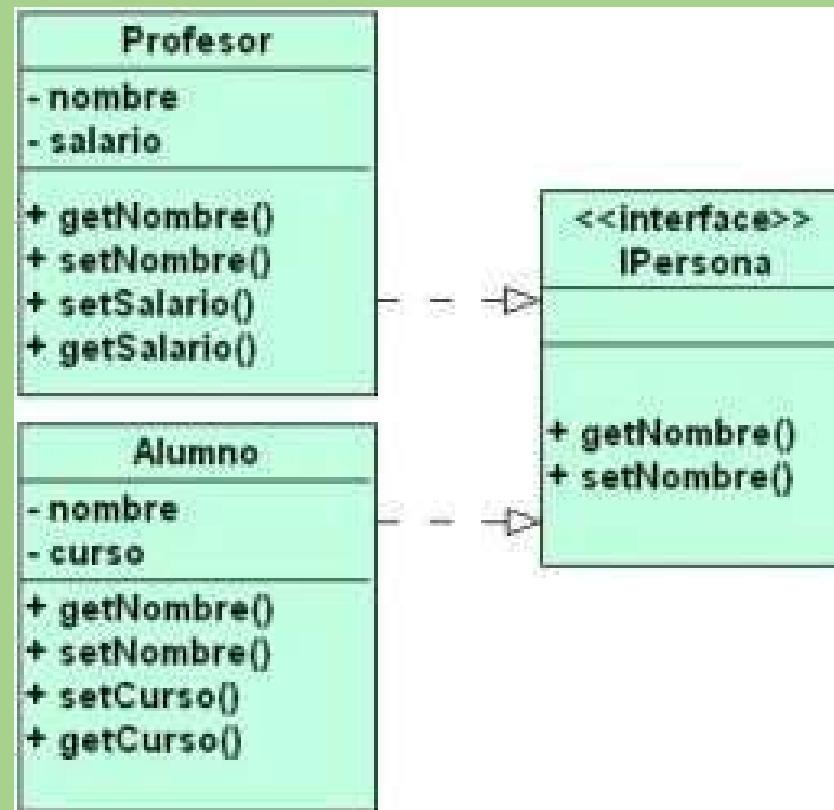
 @Override
 public String tipoAnimal(String tipo) {
 return tipo;
 }
}
```

**Console Output:**

```
Const1
Const2
Grr
Mamifero
```

# Ejercicio de interfaces

- Realiza lo siguiente con una interfaz IPersona:



# **CLASES ABSTRACTAS VS INTERFACES**



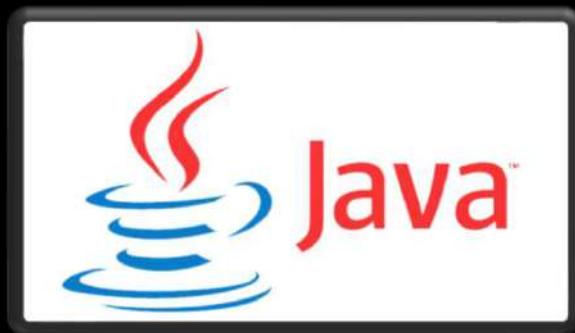
# Diferencias entre clases abstractas e interfaces

| ABSTRACTA                                                                              | INTERFACE                                                    |
|----------------------------------------------------------------------------------------|--------------------------------------------------------------|
| Tiene tanto métodos concretos como métodos abstractos abstractos                       | Solo puede tener métodos abstractos                          |
| Una clase solo puede extender de una única clase abstracta                             | Una clase puede implementar cualquier cantidad de interfaces |
| Puede tener un método constructor                                                      | No puede tener un método constructor                         |
| Los atributos pueden tener cualquier tipo de visibilidad (public, protected o private) | Los atributos únicamente pueden ser públicos                 |
| Pueden tener variables de instancia                                                    | No puede tener variables de instancia                        |

# POLIMORFISMO



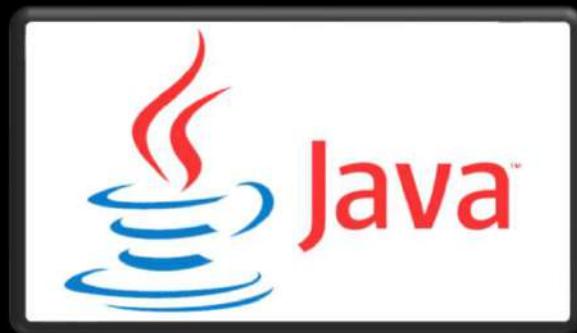
# ¿QUÉ ES EL POLIMORFISMO?



# ¿Qué es el polimorfismo?

- El polimorfismo y la herencia están íntimamente relacionados. La palabra polimorfismo proviene del griego y significa “cualidad que puede tener muchas formas” lo que nos puede ayudar a hacernos una idea de lo que hace dicho concepto.
- Por ejemplo, un método con el mismo nombre, puede ser modificado gracias al polimorfismo el cual nos permite que los métodos con un mismo nombre pueden hacer distintas cosas dependiendo del tipo de objeto que estamos manejando.

# EJEMPLO DE POLIMORFISMO



# Polimorfismo

- Si nos fijamos, en este caso, ambas clases pueden ser de

```
Main.java ×
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 // Animal
6 Animal animal = new Animal();
7 animal.makeSound();
8 Animal animal2 = new Lobo();
9 animal2.makeSound();
10 }
11 }
12
13 class Animal {
14 public void makeSound() {
15 System.out.println(this.getClass().getName() + "-> Pio Pio...");
16 }
17 }
18
19 class Lobo extends Animal{
20 private String nombreAnimal = "Lobo";
21
22 public void getNombreAnimal() {
23 System.out.println(this.getClass().getName() + " -> " + nombreAnimal);
24 }
25 }
```

Console ×  
<terminated> Main (2) [Java Application] C:\Program  
com.main.Animal-> Pio Pio...  
com.main.Lobo-> Pio Pio...

# Polimorfismo

- En este segundo ejemplo, estamos aplicando polimorfismo para modificar los métodos de nuestra clase que extiende de Animal. Vamos a verlo:

The diagram illustrates the execution flow of the Java code. It shows three windows: a code editor for `Main.java`, a terminal window for the `Console`, and a memory dump window.

**Main.java** code:

```
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 // Animal
6 Animal animal = new Animal();
7 animal.makeSound();
8 Animal animal2 = new Lobo();
9 animal2.makeSound();
10 }
11 }
12
13 class Animal {
14 public void makeSound() {
15 System.out.println(this.getClass().getName() + "-> Pio Pio...");
16 }
17 }
18
19 class Lobo extends Animal{
20 private String nombreAnimal = "Lobo";
21 //El metodo makeSound ahora hace Auuu...
22 public void makeSound() {
23 System.out.println(this.getClass().getName() + "-> Auuu...");
```

**Console** output:

```
<terminated> Main (2) [Java Application] C:\Program
com.main.Animal-> Pio Pio...
com.main.Lobo-> Auuu...
```

**Memory Dump** (represented by a blue box):

```
com.main.Animal-> Pio Pio...
com.main.Lobo-> Auuu...
```

Annotations in the code:

- Red boxes highlight the creation of `Animal` and `Lobo` objects and their `makeSound()` calls.
- Blue boxes highlight the `makeSound()` methods in both `Animal` and `Lobo`.
- A red arrow points from the `makeSound()` call in `Main.java` to the `makeSound()` method in `Animal`.
- A blue arrow points from the `makeSound()` call in `Main.java` to the `makeSound()` method in `Lobo`.
- A blue arrow points from the `makeSound()` method in `Animal` to the `makeSound()` method in `Lobo`.

# Polimorfismo

- El principal problema de las instancias que utilizan la clase Animal y no la clase Lobo, es que cuando se crean a partir de un Lobo, este objeto no pueden utilizar el método getNombreAnimal. Vamos a ver un ejemplo:

The screenshot shows a Java development environment with two windows. On the left is the code editor window titled "Main.java x" containing the following Java code:

```
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 // Animal
6 Animal animal = new Animal();
7 animal.makeSound();
8 Lobo lobo = new Lobo();
9 lobo.getNombreAnimal();
10 Animal animal2 = new Lobo();
11 animal2.makeSound();
12 }
13 }
14
15 class Animal {
16 public void makeSound() {
17 System.out.println(this.getClass().getName() + "-> Pio Pio...");
18 }
19 }
20
21 class Lobo extends Animal{
22 private String nombreAnimal = "Lobo";
23
24 public void getNombreAnimal() {
25 System.out.println(this.getClass().getName() + " -> " + nombreAnimal);
26 }
27
28 public void makeSound() {
29 System.out.println(this.getClass().getName() + "-> Auuu...");
```

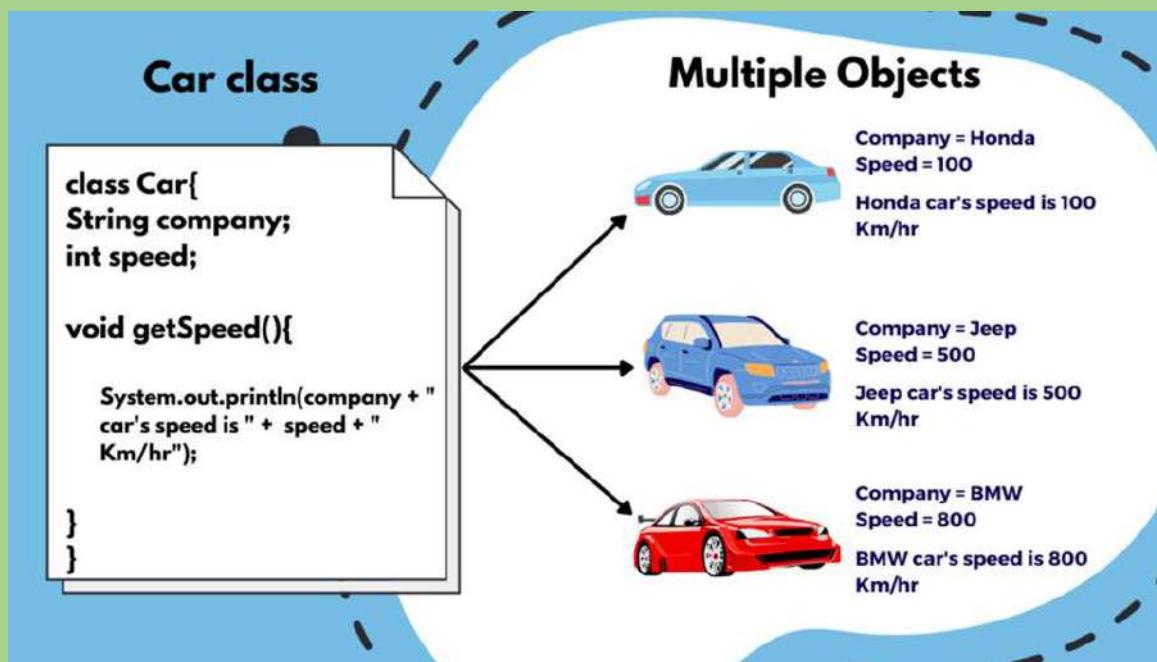
The code editor highlights the line "lobo.getNombreAnimal();". The line "Animal animal2 = new Lobo();" is also highlighted with a blue selection bar.

On the right is the "Console X" window showing the output of the program:

```
<terminated> Main (2) [Java Application] C:\Program
com.main.Animal-> Pio Pio...
com.main.Lobo -> Lobo
com.main.Lobo-> Auuu...
```

# Polimorfismo

- Realiza el siguiente ejemplo:
  - Tienes que poder crear los objetos mediante a un constructor
  - Encapsular los atributos
  - Generarlos los getters y setters (incluido el método getSpeed) con la velocidad correspondiente al vehículo en concreto.
  - Crear los 3 objetos que vemos en la diapositiva



# SCANNER



# Scanner

- La clase Scanner nos permite introducir datos, es decir, realizar la lectura de datos desde la Terminal, ficheros, etc.
- En nuestro caso, vamos a hacer un programa que lea el texto que introducimos desde la terminal. Para ello, primeramente creamos una instancia de la clase Scanner:
  - Scanner sc = new Scanner();
- Lo cual provocará que necesitemos realizar el import de la clase **import java.util.Scanner;**

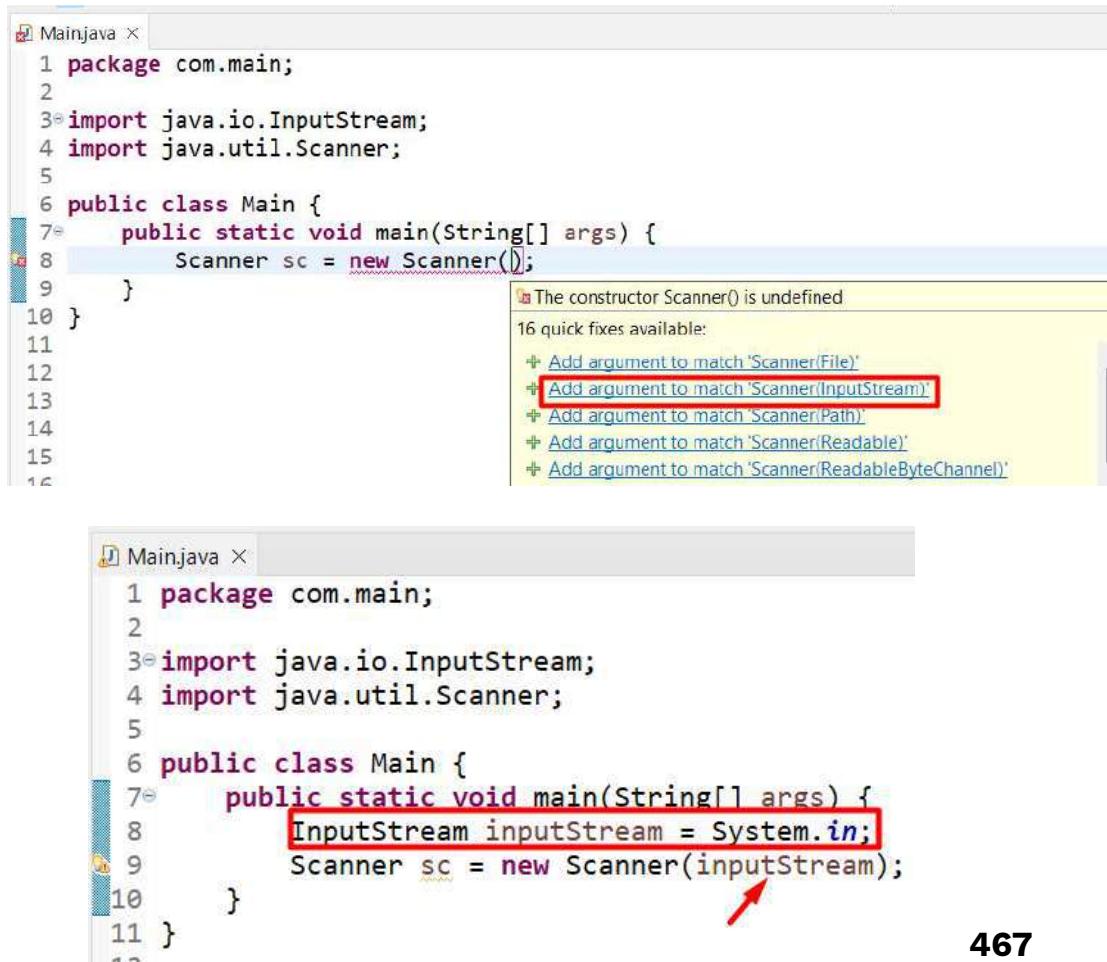
```
Main.java x
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 Scanner sc = new Scanner();
6 }
7 }
```

```
Main.java x
1 package com.main;
2
3 import java.util.Scanner;
4
5 public class Main {
6 public static void main(String[] args) {
7 Scanner sc = new Scanner();
8 }
9 }
```

Import 'Scanner' (java.util)  
Create class 'Scanner'  
Create interface 'Scanner'

# Scanner

- Pero si nos fijamos, la clase Scanner nos continua dando un error. Pese a ello, una de las opciones que nos ofrece es utilizar un InputStream. Aunque, no la vamos a marcar directamente, pero si que vamos a crear un InputStream de tipo System.in.
- Para solucionar esto, una vez hemos creado el InputStream System.in, vamos a añadir dicho inputStream sobre el constructor de nuestra clase Scanner:



```
Main.java
1 package com.main;
2
3 import java.io.InputStream;
4 import java.util.Scanner;
5
6 public class Main {
7 public static void main(String[] args) {
8 Scanner sc = new Scanner();
9 }
10}
11
12
13
14
15
16
```

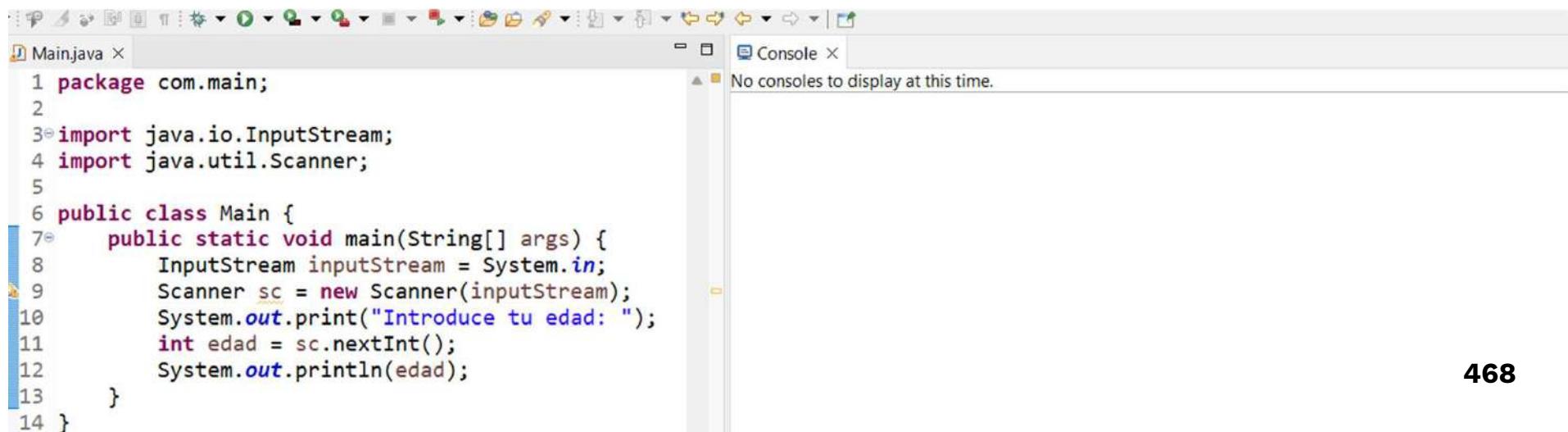
```
Main.java
1 package com.main;
2
3 import java.io.InputStream;
4 import java.util.Scanner;
5
6 public class Main {
7 public static void main(String[] args) {
8 InputStream inputStream = System.in;
9 Scanner sc = new Scanner(inputStream);
10 }
11}
```

# Scanner

En función del tipo de dato que queremos capturar, utilizamos un método sobre la instancia de scanner u otro.

- nextInt() lee un valor int introducido por el usuario;
- nextLong() lee un valor long introducido por el usuario.
- nextShort() lee un valor short introducido por el usuario.
- nextDouble() lee un valor double introducido por el usuario.
- nextByte() lee un valor byte introducido por el usuario.
- nextBoolean() lee un valor boolean introducido por el usuario.
- nextLine() lee un valor String introducido por el usuario.

En mi caso, quiero capturar un entero (int) por lo que utilizaré nextInt(). Vamos a ver un ejemplo:



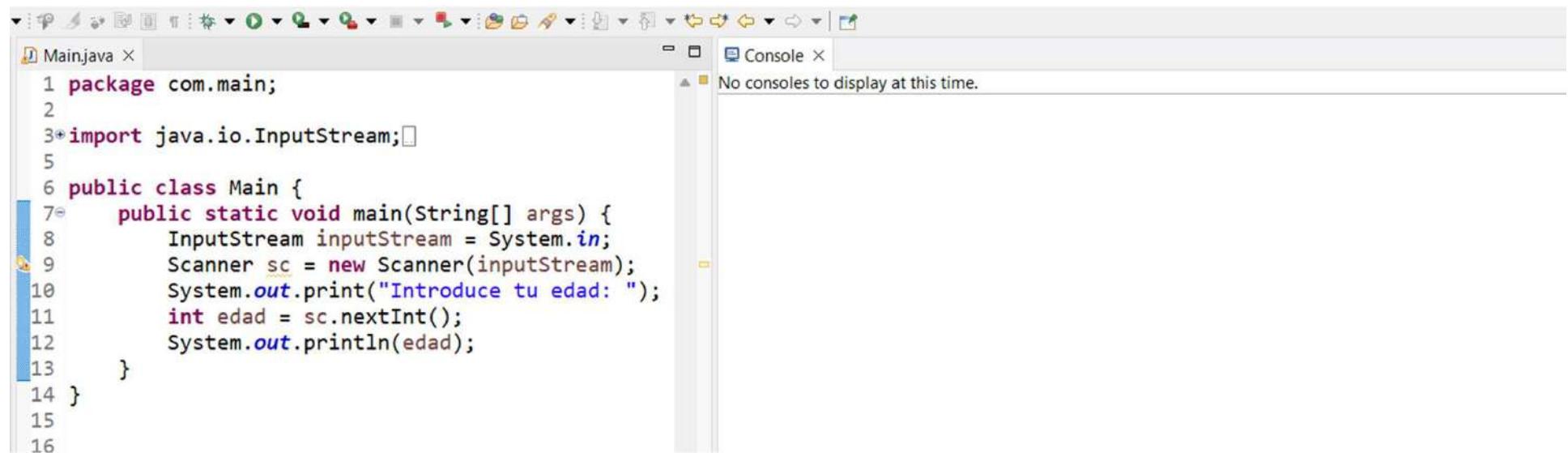
The screenshot shows an IDE interface with two main panes. On the left, the code editor displays a file named 'Main.java' containing the following Java code:

```
1 package com.main;
2
3 import java.io.InputStream;
4 import java.util.Scanner;
5
6 public class Main {
7 public static void main(String[] args) {
8 InputStream inputStream = System.in;
9 Scanner sc = new Scanner(inputStream);
10 System.out.print("Introduce tu edad: ");
11 int edad = sc.nextInt();
12 System.out.println(edad);
13 }
14 }
```

On the right, the 'Console' pane shows the message: "No consoles to display at this time."

# Problemas con Scanner

- ¿Qué pasaría si le pasamos otro tipo de dato que no corresponda con el que hemos definido en el método? Pues que “petaría” el programa, es decir, que se generaría una excepción. Vamos a verlo:



The screenshot shows a Java development environment with the following details:

- Left Panel (Code Editor):** A file named "Main.java" is open. The code is as follows:

```
1 package com.main;
2
3 import java.io.InputStream;
4
5
6 public class Main {
7 public static void main(String[] args) {
8 InputStream inputStream = System.in;
9 Scanner sc = new Scanner(inputStream);
10 System.out.print("Introduce tu edad: ");
11 int edad = sc.nextInt();
12 System.out.println(edad);
13 }
14 }
```

- Right Panel (Console):** A window titled "Console" is present, displaying the message: "No consoles to display at this time."

# Ejercicios de Scanner

- Realiza mediante a Scanner un programa que te vaya pidiendo los distintos tipos de datos y los capture.
  - De momento el programa no tendrá validación de excepciones (ya que eso lo veremos un poco más adelante)
  - Se deben de utilizar todos los tipos de datos. Por ejemplo:
    - Introduce un valor booleano (true/false):
  - Finalmente, se deberá de ir mostrando cada uno de los valores que se han leyendo mediante a Scanner por el Terminal.

# Ejercicios de Scanner: 3 en ralla



- Captura tu nombre y tus dos apellidos (en diferentes líneas) mediante al método Scanner.

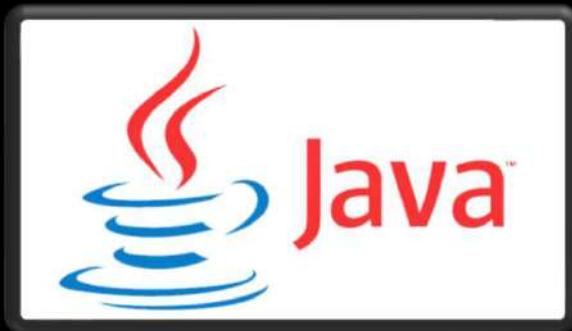
```
1. Humano contra humano
2. Humano contra CPU (El CPU juega como 0)
3. CPU contra CPU
Elige:
1
El jugador que inicia es: X

	1	2	3
1			
2			
3			
Jugador X. Ingresa coordenadas (x,y) para colocar la pieza
Ingresa X: |
```

# **EXCEPCIONES EN JAVA**

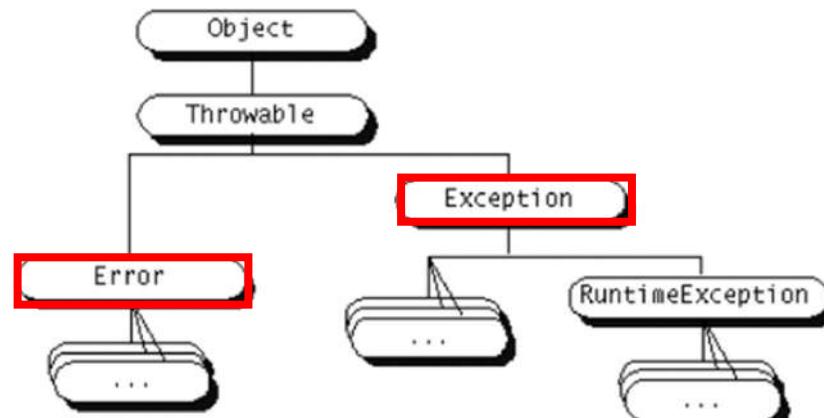
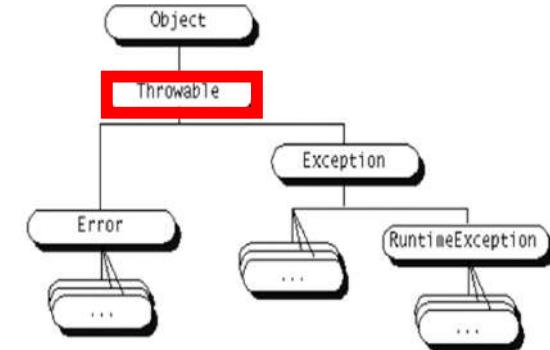


# ¿QUÉ SON LAS EXCEPCIONES?

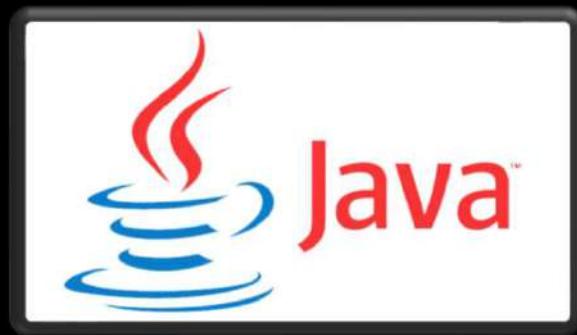


# ¿Qué son las excepciones?

- Una excepción es una situación no esperada durante la ejecución de un programa.
- Todas las excepciones en Java son englobadas dentro de la clase Throwable. Por tanto, cuando se genera una excepción, se genera algún objeto con la excepción correspondiente.
- Las clase Throwable trabaja con dos subclases internamente en función del tipo de excepción que se genere. Estas dos subclases son:
  - Error
  - Exception



# ¿PORQUÉ CONTROLAR LAS EXCEPCIONES?



# ¿Porqué controlar las excepciones?

- Las excepciones detienen el flujo del programa haciendo que no se ejecuten el resto de nuestras instrucciones. Vamos a ver un ejemplo:
  - Si todo va OK se ejecutan las 3 instrucciones:

```
Main.java x
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 System.out.println("Instrucción 1");
6 System.out.println(10/2); //Instrucción 2
7 System.out.println("Instrucción 3");
8 }
9 }
```

```
Console x
<terminated> Main (2)
Instrucción 1
5
Instrucción 3
```

- En cambio, una vez se produce una Excepción (ArithmeticException) el flujo del programa deja de ejecutarse, lo que supone que haya líneas que no se ejecuten:

```
Main.java x
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 System.out.println("Instrucción 1");
6 System.out.println(5/0); //Instrucción 2
7 System.out.println("Instrucción 3");
8 }
9 }
```

```
Console x
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (3 may. 2022 23:33:30 -)
Instrucción 1
Exception in thread "main" java.lang.ArithmaticException: / by zero
 at com.main.Main.main(Main.java:6)
```

# DIFERENCIAS ENTRE ERROR Y EXCEPTION



# Subclases de Throwable: Error vs Exception

- Las diferencias de estas dos subclases de Throwable cuando se produce un error son:

- Subclase Error:** están relacionadas con la JVM (Java Virtual Machine) y no con el programa. Este tipo de excepciones no tienen que ver con nuestro programa, y por tanto, no las podemos controlar ya que escapan de nuestro control. Por tanto, este tipo de excepciones no podrán ser controladas.
- Por ejemplo, cuando la JVM no tiene suficiente memoria RAM disponible en el sistema para poder ejecutar los programas.
- Subclase Exception:** están relacionadas con el código de nuestro programa.
- Por ejemplo, llamar a una posición de un array que aún no esté definida, olvidarse de cerrar un paréntesis o no poner un ; al final de una instrucción.



A screenshot of an IDE interface. On the left, there is a code editor window titled "Main.java" containing the following Java code:

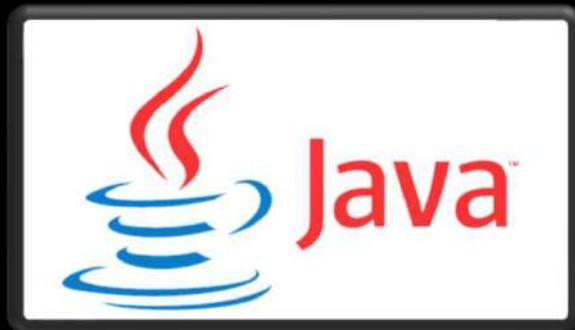
```
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 System.out.println("Ejemplo de error")
6 }
7 }
```

To the right of the code editor is a "Console" window showing the output of the program's execution. The output shows:

```
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (3 may. 2022 21:19:52 – 21:19:54)
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
Syntax error, insert ";" to complete BlockStatements
at com.main.Main.main(Main.java:5)
```

The number "478" is located in the bottom right corner of the IDE interface.

# MANEJO DE EXCEPCIONES CON TRY CATCH



# Capturando la excepción con try catch finally

- La excepciones que son lanzadas (producidas) desde nuestras aplicaciones pueden ser controladas mediante de diferentes maneras. Vamos a comenzar viendo la estructura completa de un try catch (try catch finally):

The screenshot shows a Java development environment with two windows. On the left, the code editor displays `Main.java` with the following content:

```
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 // Instrucciones cuando no hay una excepción
6 try {
7 System.out.println("Instrucción 1");
8 System.out.println(5 / 0); // Instrucción 2
9 System.out.println("Instrucción 3");
10 } catch (Exception e) {
11 // Instrucciones cuando se produce una excepción
12 System.err.println("Se ha producido la siguiente excepción: \n\t" + e);
13 } finally {
14 // Instrucciones que se ejecutan, tanto si hay como sino hay excepciones
15 System.out.println(
16 "Por aquí pasará siempre. Es decir, independientemente\n" +
17 "de si se produce un error o no.");
18 }
19 }
20 }
```

The code uses a try-catch-finally block. The try block contains three print statements. The catch block handles any exception and prints it to standard error. The finally block contains a single print statement that will always execute, regardless of whether an exception was thrown or not.

On the right, the console window shows the output of running the application:

```
Instrucción 1
Se ha producido la siguiente excepción:
java.lang.ArithmetricException: / by zero
Por aquí pasará siempre. Es decir, independientemente
de si se produce un error o no.
```

The output shows the first instruction, the exception message, and the final message, demonstrating the flow of execution through the try-catch-finally block.

# Capturando la excepción con try catch

- Aunque ya hemos visto el try catch al completo, habitualmente es más común ver un try catch sin el bloque finally. Vamos a ver un ejemplo:

The screenshot shows an IDE interface with two panes. The left pane, titled 'Main.java X', contains the following Java code:

```
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 // Instrucciones cuando no hay una excepción
6 try {
7 System.out.println("Instrucción 1");
8 System.out.println(5 / 0); // Instrucción 2
9 System.out.println("Instrucción 3");
10 } catch (Exception e) {
11 // Instrucciones cuando se produce una excepción
12 System.err.println("Se ha producido la siguiente excepción: \n\t" + e);
13 }
14 }
15 }
```

The code uses a try-catch block. The try block contains three print statements: "Instrucción 1", "Instrucción 2", and "Instrucción 3". The catch block contains a single print statement for the exception. A green box highlights the try block, and a red box highlights the catch block.

The right pane, titled 'Console X', shows the execution output:

```
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\java
Instrucción 1
Se ha producido la siguiente excepción:
java.lang.ArithmetricException: / by zero
```

The output shows the first two print statements from the try block, followed by the error message from the catch block, indicating that the division by zero caused an arithmetic exception.

- **IMPORTANTE**, si por algún motivo se produce una excepción dentro del try, TODAS las líneas que no se hayan ejecutado y que pertenezcan al bloque del try no se ejecutarán.

# Try con multiples catchs

- Existe la posibilidad de crear catchs específicos y/o catch genéricos. Los específicos nos permiten capturar solamente una exception específica. En cambio, los genéricos nos permiten capturar varios errores. Vamos a ver un ejemplo:

The screenshot shows a Java IDE interface with two panes. The left pane displays the code for Main.java, and the right pane shows the corresponding Console output.

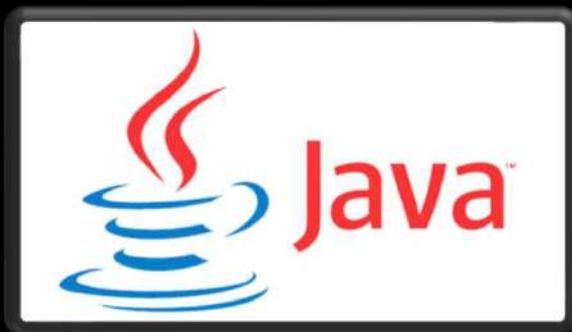
**Main.java Content:**

```
1 package com.main;
2
3 public class Main {
4 private static void printLength(String str) {
5 System.out.println(str.length());
6 }
7
8 public static void main(String[] args) {
9 // Instrucciones cuando no hay una excepción
10 try {
11 System.out.println("Dentro del try");
12 String nombre = null;
13 System.out.println(nombre.length());
14 System.out.println("Saliendo del try");
15 } catch (ArithmetricException aritmeticException) {
16 // Captura las excepciones aritméticas (ArithmetricException)
17 System.err.println("Se ha producido la siguiente excepción: \n\t" +
18 aritmeticException);
19 }
20 catch (Exception exception) {
21 // Captura al resto de excepciones
22 System.err.println("Se ha producido la siguiente excepción: \n\t" +
23 exception);
24 }
25 }
26}
```

**Console Output:**

```
Dentro del try
Se ha producido la siguiente excepción:
java.lang.NullPointerException
```

# MANEJO DE EXCEPCIONES CON TRY CATCH Y THROWS



# Throws: capturando y subiendo exceptions

- throws se utiliza cuando un método puede producir una excepción que el mismo no es capaz de manejar. Mediante a throws junto al nombre de la excepción subirá hacia el siguiente método que en principio debería de ser el que la deba controlar. Vamos a ver un ejemplo:
  - Primeramente, comenzamos con un trows en el que capturamos las excepciones de NullPointerException pero sin tener ningún otro

The screenshot shows an IDE interface with two main panes. On the left, the code editor displays a Java file named Main.java. The code defines a class Main with a static method getLength that throws a NullPointerException if its parameter txt is null. The main method prints three instructions to the console before calling getLength. On the right, the console window shows the execution of the program. It prints 'Instrucción 1' and 'Instrucción 3', then throws a NullPointerException at line 8 of Main.java, which is caught by the main method at line 12.

```
Main.java X
1 package com.main;
2
3 public class Main {
4 public static int getLength(String txt) throws NullPointerException {
5 //Aqui se producirá la excepción y subirá hacia arriba hasta llegar a
6 //algún método que la capture o bien aparecerá un error que detendrá
7 //el flujo de ejecución
8 return txt.length();
9 }
10 public static void main(String[] args) {
11 System.out.println("Instrucción 1");
12 System.out.println(getLength(null));
13 System.out.println("Instrucción 3");
14 }
15 }
```

```
Console X
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (4 ma
Instrucción 1
Exception in thread "main" java.lang.NullPointerException
 at com.main.Main.getLength(Main.java:8)
 at com.main.Main.main(Main.java:12)
```

# Throws: capturando y subiendo exceptions

- throws se utiliza cuando un método puede producir una excepción que el mismo no es capaz de manejar. Mediante a throws junto al nombre de la excepción subirá hacia el siguiente método que en principio debería de ser el que la deba controlar. Vamos a ver un ejemplo:
  - En este ejemplo, vemos que tenemos la llamada al método getLength dentro de un try catch. Por lo que, una vez se produzca la excepción, como el getLength no es capaz de capturarla, la subirá hacia el método main el cual si que es tiene la capacidad de capturarla sin problema.

The screenshot shows an IDE interface with two panes. The left pane, titled 'Main.java X', contains the following Java code:

```
1 package com.main;
2
3 public class Main {
4 public static int getLength(String txt)
5 throws NullPointerException {
6 /*Aqui se producirá la excepción y
7 subirá hacia arriba hasta llegar a
8 algún método que la capture o bien aparecerá
9 un error que detendrá el flujo de ejecución */
10 return txt.length();
11 }
12 public static void main(String[] args) {
13 System.out.println("Instrucción 1");
14 try {
15 System.out.println(getLength(null));
16 }catch (Exception e) {
17 //Aqui se capturará la excepción
18 System.err.println(e);
19 }
20 System.out.println("Instrucción 3");
21 }
22 }
```

The right pane, titled 'Console X', shows the execution output:

```
<terminated> Main (2) [Java Application] C:\Progra
Instrucción 1
java.lang.NullPointerException
Instrucción 3
```

# GENERANDO NUESTRAS PROPIAS EXCEPCIONES



# Generando una excepción con Throw

- Mediante a throw new junto al tipo de excepción que queremos generar, podemos lanzar una excepción nosotros de la misma forma que lo hace Java cuando se produce un error. Vamos a ver un ejemplo:

The screenshot shows a Java development environment with two windows. On the left, the code editor window titled 'Main.java' displays the following Java code:

```
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 System.out.println("Instrucción 1");
6 try {
7 String txt = "Generando una excepción NullPointerException";
8 throw new NullPointerException(txt);
9 }catch (Exception e) {
10 //Aqui se capturará la excepción
11 System.err.println(e);
12 }
13 System.out.println("Instrucción 3");
14 }
15 }
```

On the right, the 'Console' window shows the terminal output:

```
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (4 may. 2022 1:05:03)
Instrucción 1
java.lang.NullPointerException: Generando una excepción NullPointerException
Instrucción 3
```

The red text in the console output, 'java.lang.NullPointerException: Generando una excepción NullPointerException', indicates that a NullPointerException was thrown and caught by the code.

# Ejercicios de Scanner: useLocale

- 1. A partir del ejercicio anterior de introducir valores con todos los tipos de datos, protege dicho código utilizando try catch.
- 2. Analiza como generar excepciones como capturarlas individualmente
  - Por ejemplo como desbordar un array, dividir entre 0, etc.
  - Además, engloba a las instrucciones que puedan generar errores en funciones que contendrán un throw con las excepciones que se pueden producir. Finalmente, haz que dichas excepciones suban el error hacia el main y muestra un System.err con la excepción correspondiente.
- 3. Queremos trabajar con Scanner y poder introducir un número real (con decimales) y necesitamos que investigues como realizarlo:
  - Os aconsejo investigar como utilizar el método useLocale para que acepte tanto , como .
  - Además, también deberás proteger el código evitando así otros errores.

**¿PARA QUÉ DEFINIMOS UN  
ARRAY (STRING ARGS[]) EN LA  
ZONA DE PARÁMETROS DEL  
MÉTODO MAIN?**



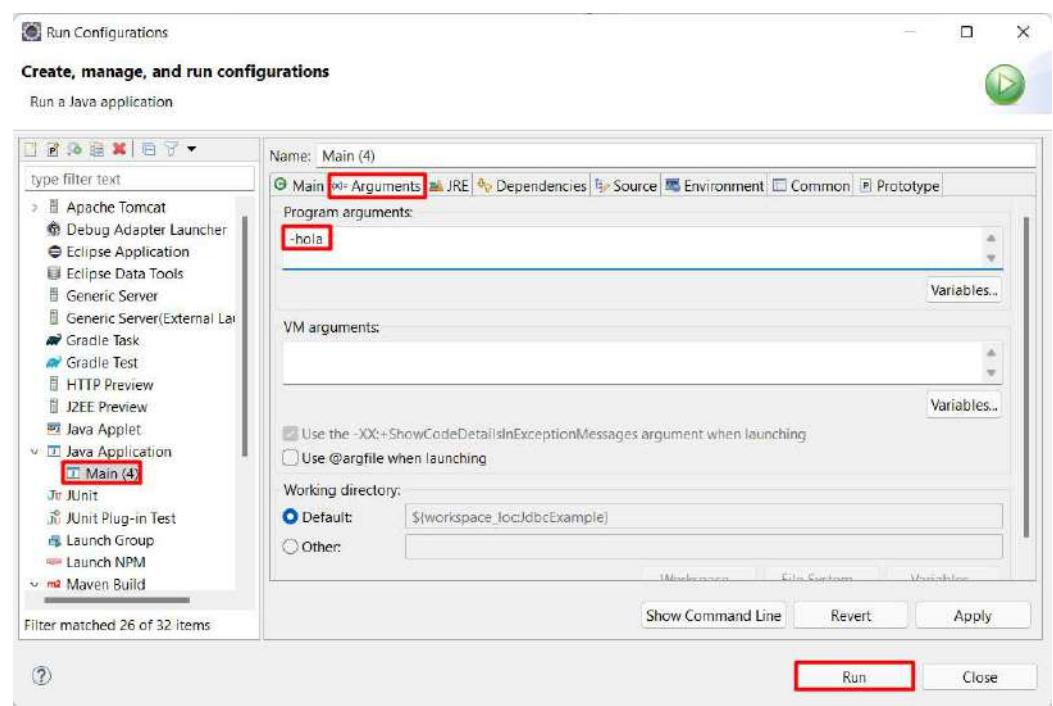
# Array en el método main

- Existe la posibilidad de pasar flags/parámetros al método main. Para ello, realizamos lo siguiente:

The screenshot shows the Eclipse IDE interface. On the left, there's a toolbar with various icons, a list of recent projects (1 Main (4), 2 Tomcat v9.0 Server at localhost, 3 WebJSP01 (1)), and a 'Run As' dropdown menu. The 'Run Configuration...' option is highlighted with a red box. Below the toolbar is the Main.java code editor, which contains the following Java code:

```
1 public class Main {
2 public static void main(String[] args) {
3 System.out.println(args[0]); // Hola
4 }
5 }
```

At the bottom, there's a Console tab showing the output of the run: <terminated> Main (4) [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (6 main) -hola



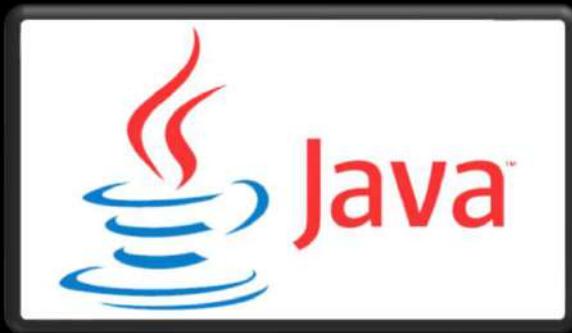
# Ejercicio de argumentos a nuestro programa

- 1. Realiza una programa con una clase Main que compruebe si has introducido parámetros para la ejecución del programa y si el primer parámetro es “Federico” y el segundo “GarcíaLorca” te llevará hacia una clase Lorca.class en la que tendremos un método recitar poema el cual nos imprimirá por pantalla un poema de Lorca. En cambio, si los parámetros introducidos al programa no son correctos irá a la clase NotAuthorized que tendrá un método que nos imprimirá que los parámetros introducidos no son los esperados.
  - Realiza la ejecución desde el IDE y hacer comprueba que funciona
  - Realiza la ejecución desde la consola y comprueba nuevamente que funciona

# ENUMS



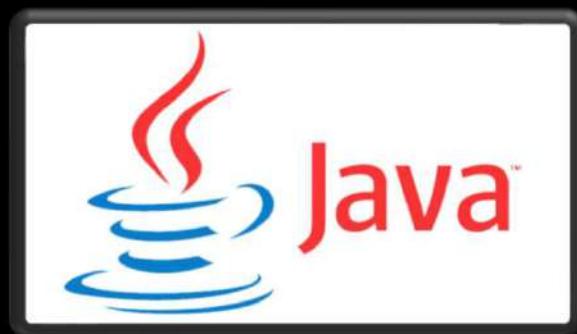
# ¿QUÉ SON LOS ENUMS?



# ¿Qué son los Enums?

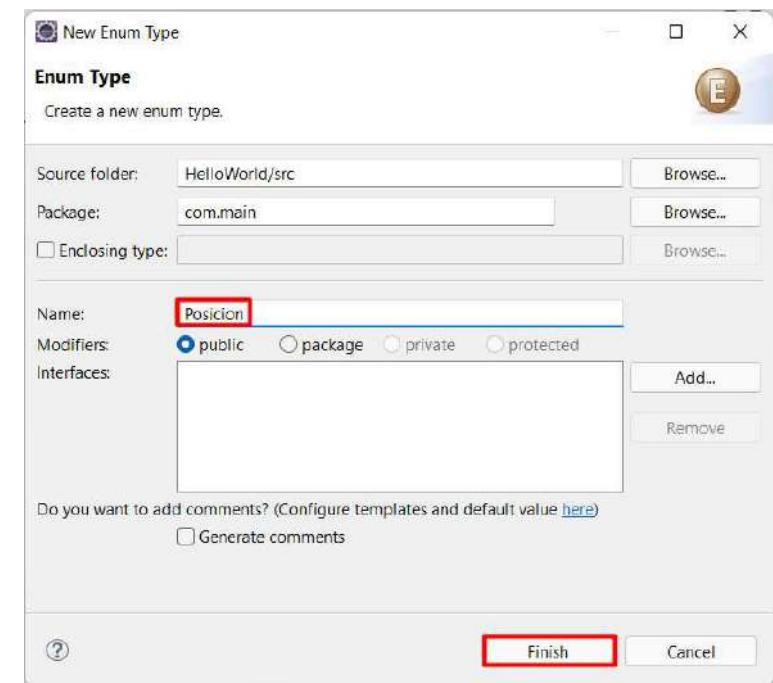
- Los ENUMS son una “clase especial” que nos permite limitar la creación de objetos a los valores que hemos especificado explícitamente en su interior, es decir, en el interior de la clase.
- Los valores almacenados en un ENUM representan un grupo/conjunto de constantes. Es decir, de variables inmutables cuyo valor no puede cambiar (como si las variables tuvieran final).
- Las constantes que formaran parte de un ENUM deben de estar separadas por una coma entre ellas. Por lo que el último valor no llevará coma.

# CREANDO UN ENUM EN ECLIPSE



# Enumeraciones ENUMs

- Para crear un Enum, sobre el package correspondiente, pulsamos botón derecho y vamos a New>Enum:



- Y, escribimos el nombre del Enum:

# Enumeraciones ENUMs

- Si nos fijamos, estas “especie de clases” (los enums) tiene una E para diferenciarlas de las clases tradicionales de Java. Vamos a verlo:



- Finalmente, definimos los valores de nuestro ENUM. En este caso, serán las posiciones de unos futbolistas. Vamos a verlo:

The screenshot shows a Java development environment with two panes. The left pane displays the project structure under the 'src' folder, showing packages 'com.main' containing files 'Main.java' and 'Posicion.java'. A red arrow points to 'Posicion.java'. The right pane shows the code for 'Posicion.java':

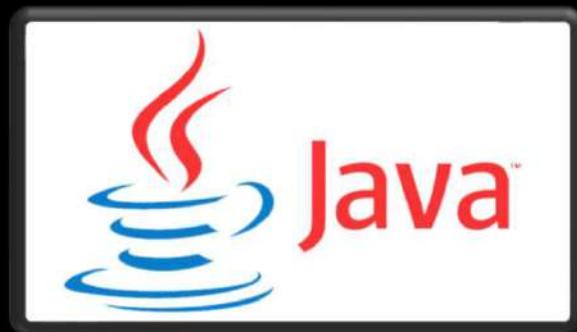
```
1 package com.main;
2
3 public enum Posicion {
4
5 }
6
```

The same IDE interface is shown again, but now the code for 'Posicion.java' includes enum constants:

```
1 package com.main;
2
3 public enum Posicion {
4 PORTERO,
5 DEFENSA,
6 CENTROCAMPISTA,
7 DELANTERO
8 }
```

# IMPRIMIENDO TODOS LOS VALORES



# Enums: imprimiendo todos los valores

- Existe la posibilidad de imprimir todos los valores que forman parte del enum mediante al método value. El cual nos devolverá un array que tenemos que recorrer. Vamos a ver un ejemplo:

The screenshot shows a Java development environment with three windows:

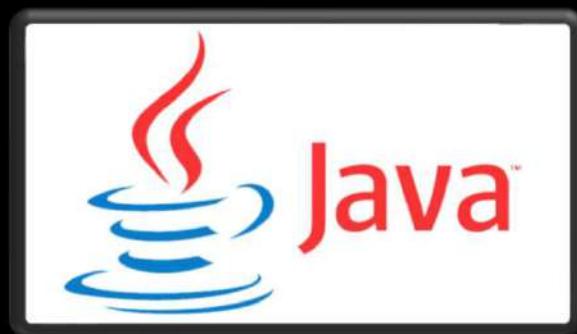
- Main.java**:

```
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 System.out.println("Posiciones:");
6 for (Posicion item : Posicion.values()) {
7 System.out.println(" - " + item);
8 }
9 }
10 }
```
- Posicion.java**:

```
1 package com.main;
2
3 public enum Posicion {
4 PORTERO,
5 DEFENSA,
6 CENTROCAMPISTA,
7 DELANTERO
8 }
9
10
```
- Console**:

```
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (4 may. 2022 15:37:59 - 15:37:59) [pid: 20172]
Posiciones:
- PORTERO
- DEFENSA
- CENTROCAMPISTA
- DELANTERO
```

# **IMPRIMIENDO UN SOLO VALOR**



**500**

# Enums: imprimiendo todos los valores

- Se utilizan mucho en IFS para comparar un valor o bien en objetos para asignar un valor.  
Vamos a ver un ejemplo:

```
Main.java ×
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 String miPosicion = "PORTERO";
6 if(miPosicion.equals(Posicion.PORTERO.toString())) {
7 System.out.println("A la portería");
8 }
9 }
10 }
```

```
Posicion.java ×
1 package com.main;
2
3 public enum Posicion {
4 PORTERO,
5 DEFENSA,
6 CENTROCAMPISTA,
7 DELANTERO
8 }
9
10
```

Console ×  
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (4 may. 2022 15:44:25 – 15:44:25) [pid: 25404]  
A la portería

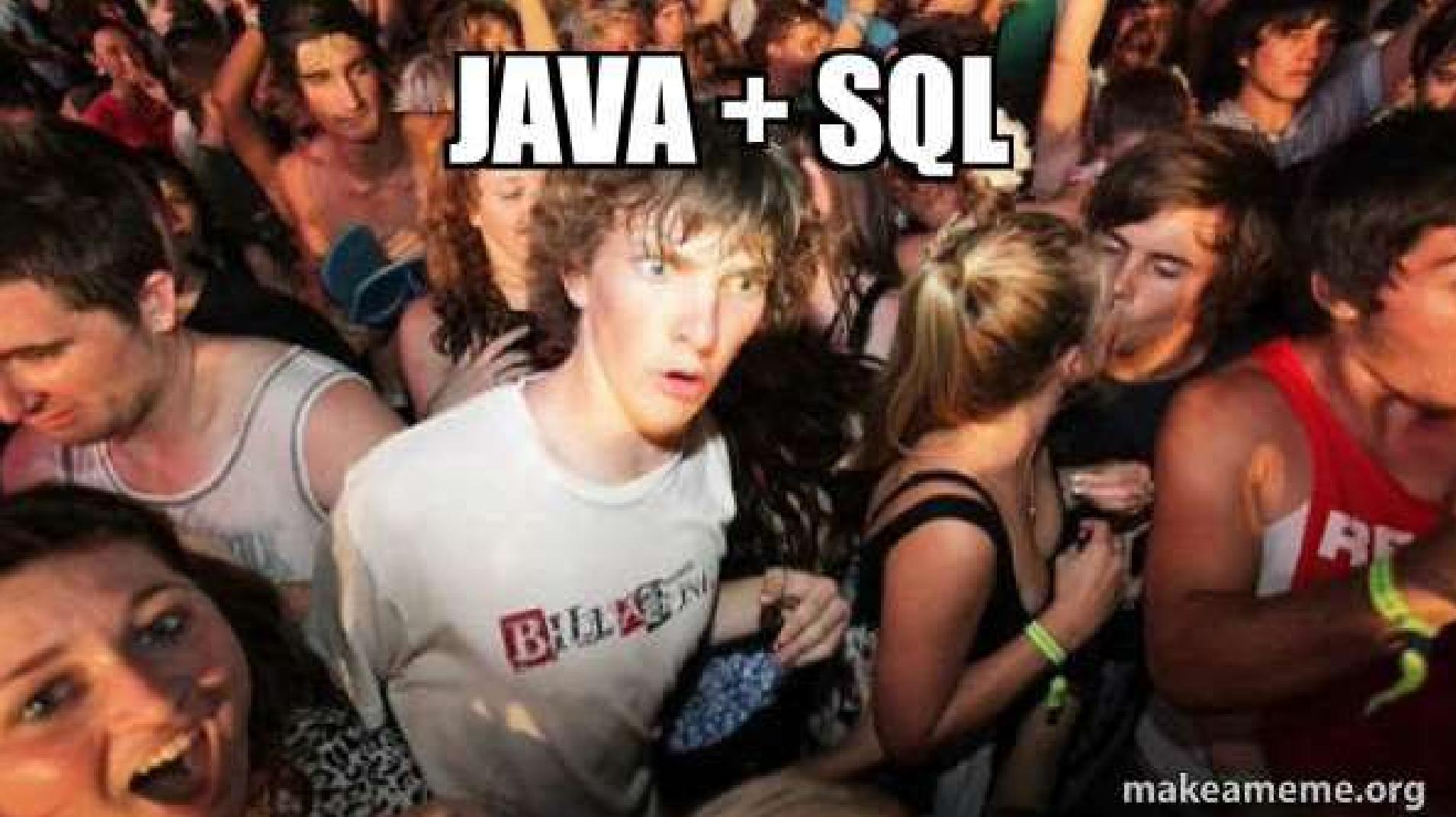
# Ejercicio de Enums

- 1. Crea un enum llamado Profesiones
- 2. Asigna las siguientes profesiones:
  - Back End Developer
  - Front End Developer
  - Full Stack Developer
- 3. Crea una clase Developer que contenga: nombre, apellidos, edad, dni y rolTecnologico encapsulada, con sus getters y setters correspondientes, etc.
- 4. Crea 3 instancias de la clase Developer (crea 3 objetos) y asigna un rol distinto a cada uno de ellos.
- 5. Comprueba los que son Back End Developers y/o Full Stack Developers y diles “Viva el Back”
- 6. Comprueba los que no son Back End Developers e imprime “JS es una  es mejor utilizar TypeScript”

# JDBC



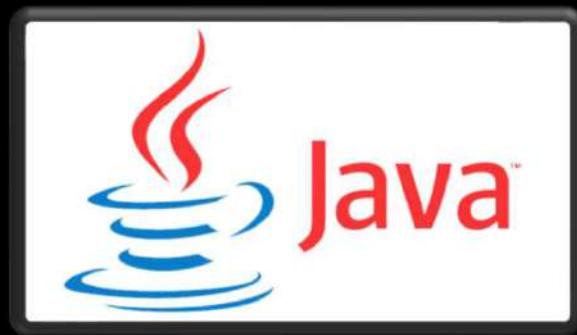
503



**JAVA + SQL**

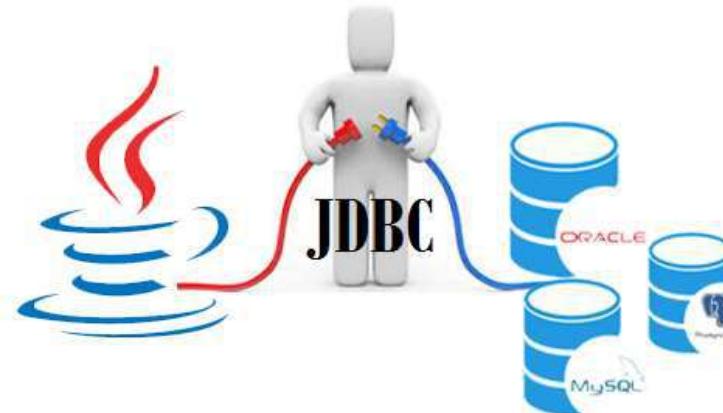
[makeameme.org](http://makeameme.org)

# ¿QUÉ ES JDBC?

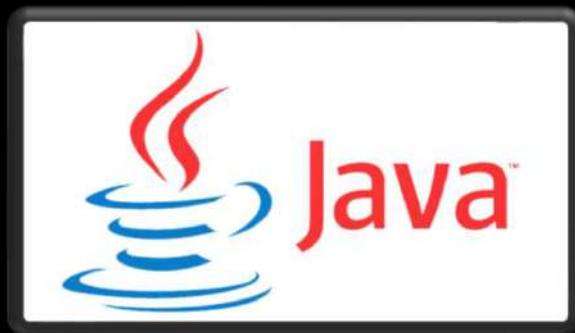


# ¿Qué es JDBC?

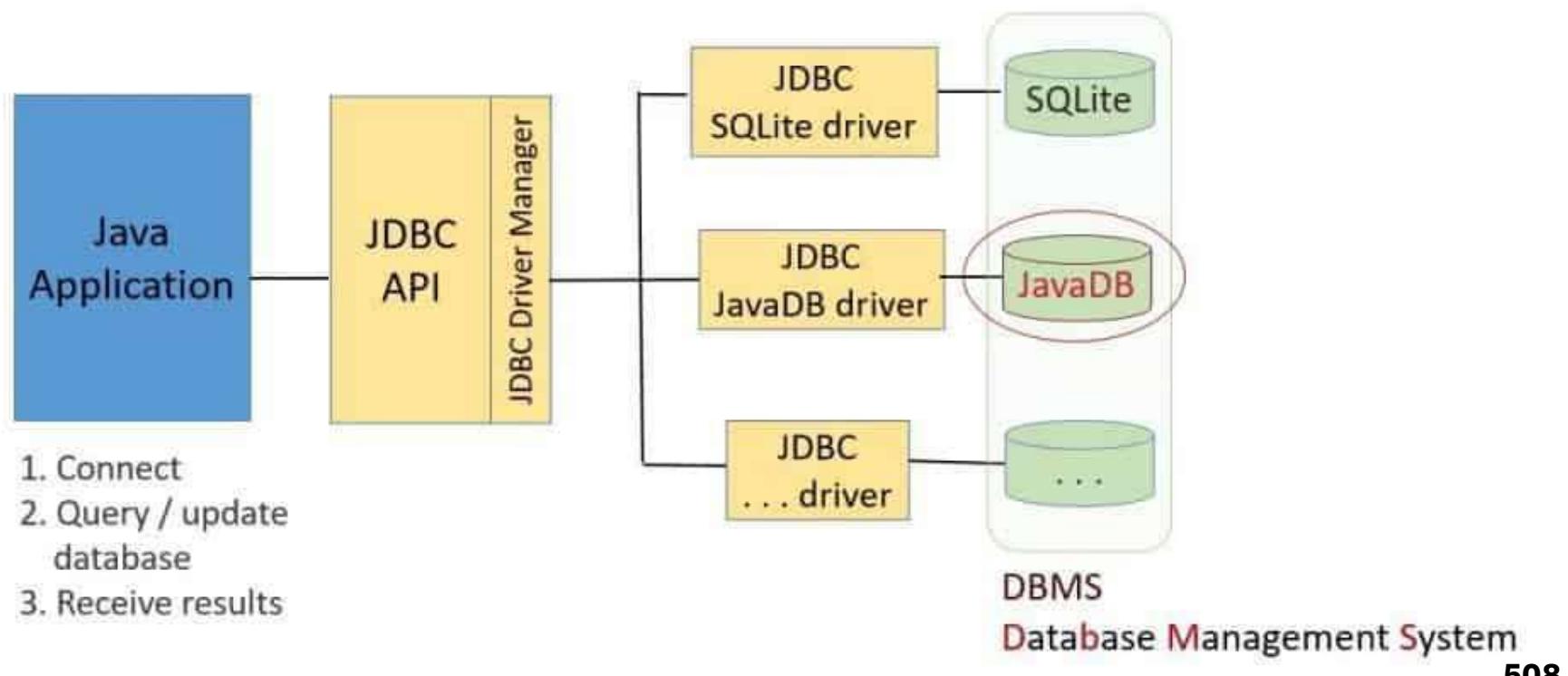
- JDBC = Java Database Connectivity = conectividad de bases de datos Java
- JDBC fue desarrollado/creado por Sun Microsystems que permite conectar aplicaciones Java con BBDD
- JDBC nos permite:
  - Conectarnos a la BBDD
  - Crear consultas SQL
  - Ejecutar consultas SQL en la BBDD
  - Mostrar los resultados de dichas consultas



# ¿CÓMO FUNCIONA JDBC?



# JDBC - Java Database Connectivity



# ¿Cómo funciona JDBC?

- El proceso para conectar una aplicación Java a una BBDD usando JDBC consta de los siguientes pasos:

- 1. Primeramente debemos descargar el driver de JDBC.
- 2. Posteriormente, realizamos la conexión a la BBDD
- 3. Una vez realiza la conexión, creamos el statement
- 4. Ejecutamos la consulta SQL
- 5. En función de la consulta, por ejemplo si es un select, podemos mostrar los resultados.
- 6. Cerramos al conexión.

## Java Database Connectivity

Register driver

Get connection

Create statement

Execute query

Close connection

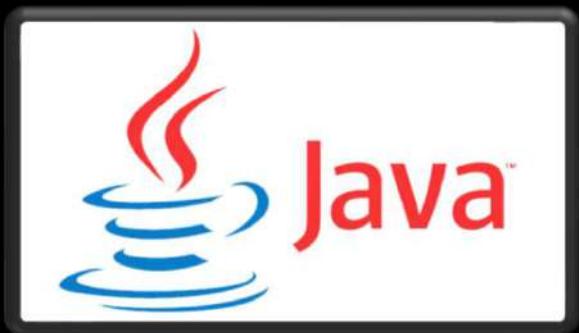


509

# EJEMPLO DE COMO USAR JDBC



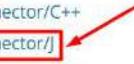
# 1. DESCARGANDO EL DRIVER

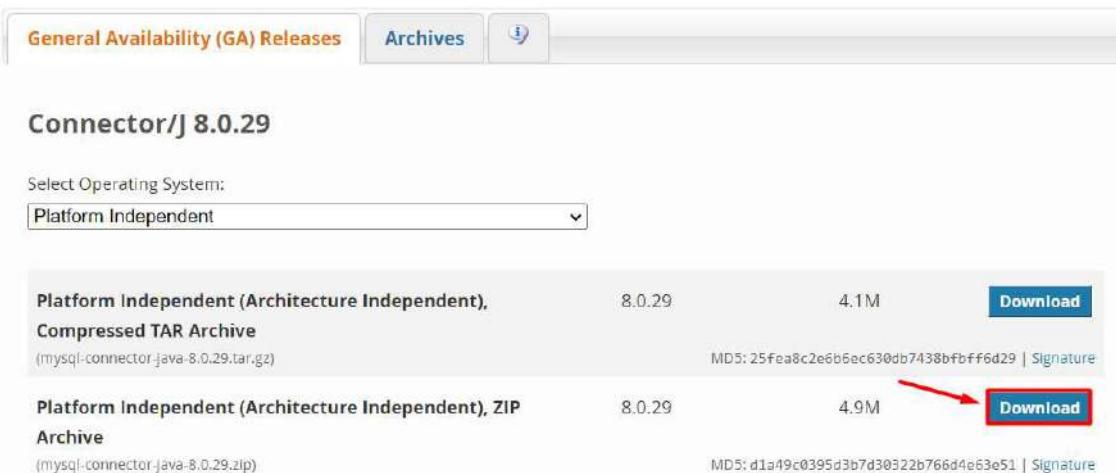


# 1. Añadiendo el driver

- Antes de nada vamos a descargar el driver (un archivo con extensión .jar) que añadiremos a nuestro proyecto de Java para poder administrar la BBDD.
- Para ello, vamos a: <https://dev.mysql.com/downloads/> y marcamos:

## ④ MySQL Community Downloads

- MySQL Yum Repository
  - MySQL APT Repository
  - MySQL SUSE Repository
  - MySQL Community Server
  - MySQL Cluster
  - MySQL Router
  - MySQL Shell
  - MySQL Workbench
  - MySQL Installer for Windows
  - MySQL for Visual Studio
- C API (libmysqlclient)
  - Connector/C++
  - **Connector/J** 
  - Connector/.NET
  - Connector/Node.js
  - Connector/ODBC
  - Connector/Python
  - MySQL Native Driver for PHP
  - MySQL Benchmark Tool
  - Time zone description tables
  - Download Archives



General Availability (GA) Releases Archives

### Connector/J 8.0.29

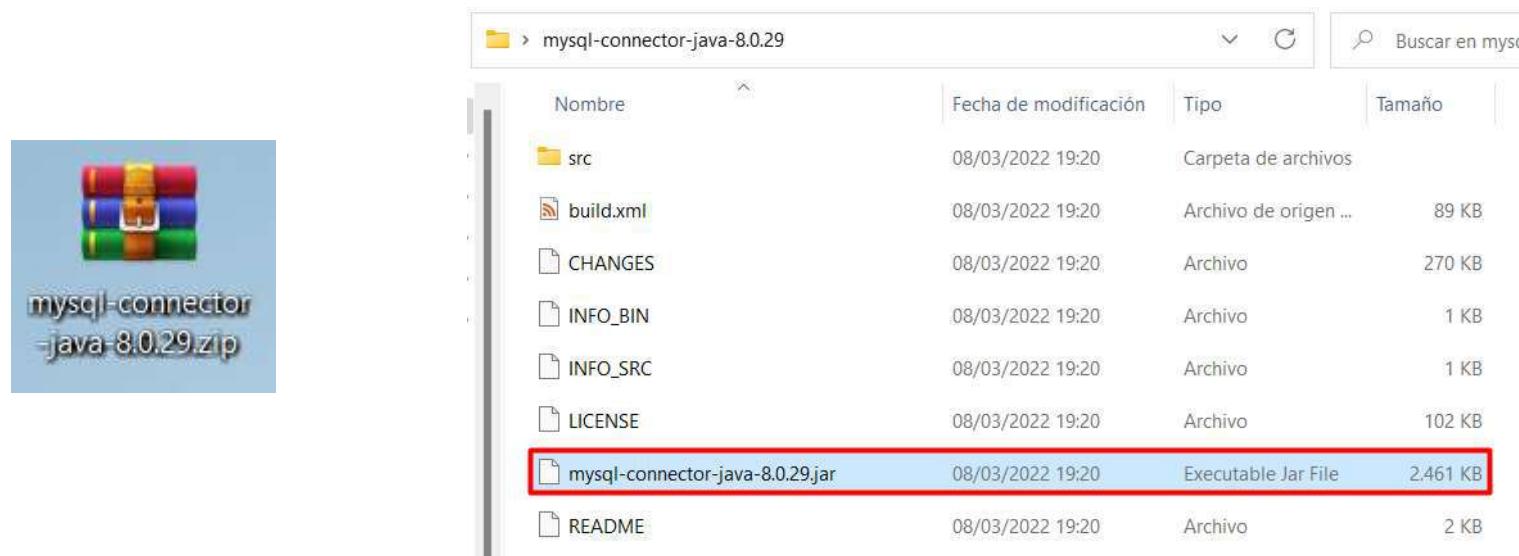
Select Operating System: Platform Independent

| Platform Independent (Architecture Independent), Compressed TAR Archive<br>(mysql-connector-java-8.0.29.tar.gz) | 8.0.29 | 4.1M | <b>Download</b> |
|-----------------------------------------------------------------------------------------------------------------|--------|------|-----------------|
| Platform Independent (Architecture Independent), ZIP Archive<br>(mysql-connector-java-8.0.29.zip)               | 8.0.29 | 4.9M | <b>Download</b> |

MD5: 25fea8c2e6b6ec630db7438bfff6d29 | Signature  
MD5: d1a49c0395d3b7d30322b766d4e63e51 | Signature

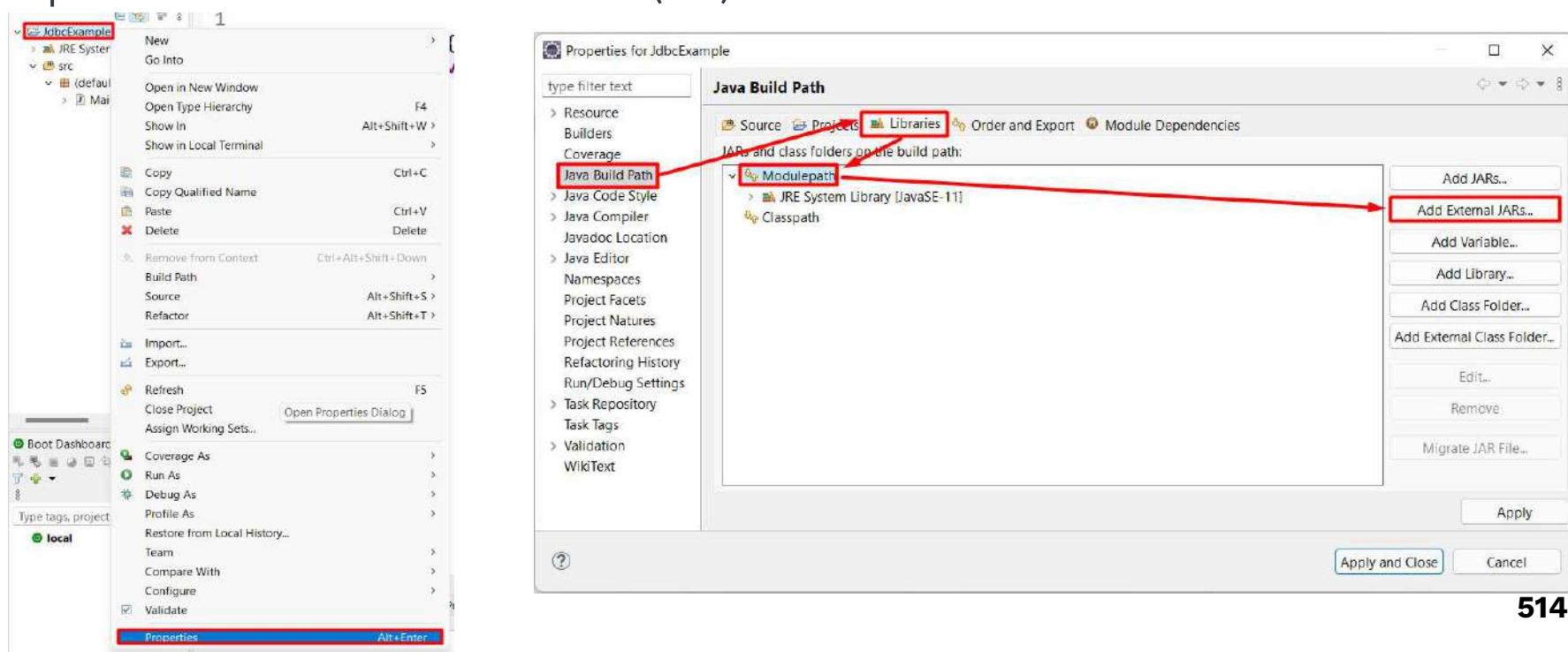
# 1. Añadiendo el driver

- Una vez descargado, si descomprimimos el ZIP, podemos ver que tenemos un fichero .jar en el interior del directorio:



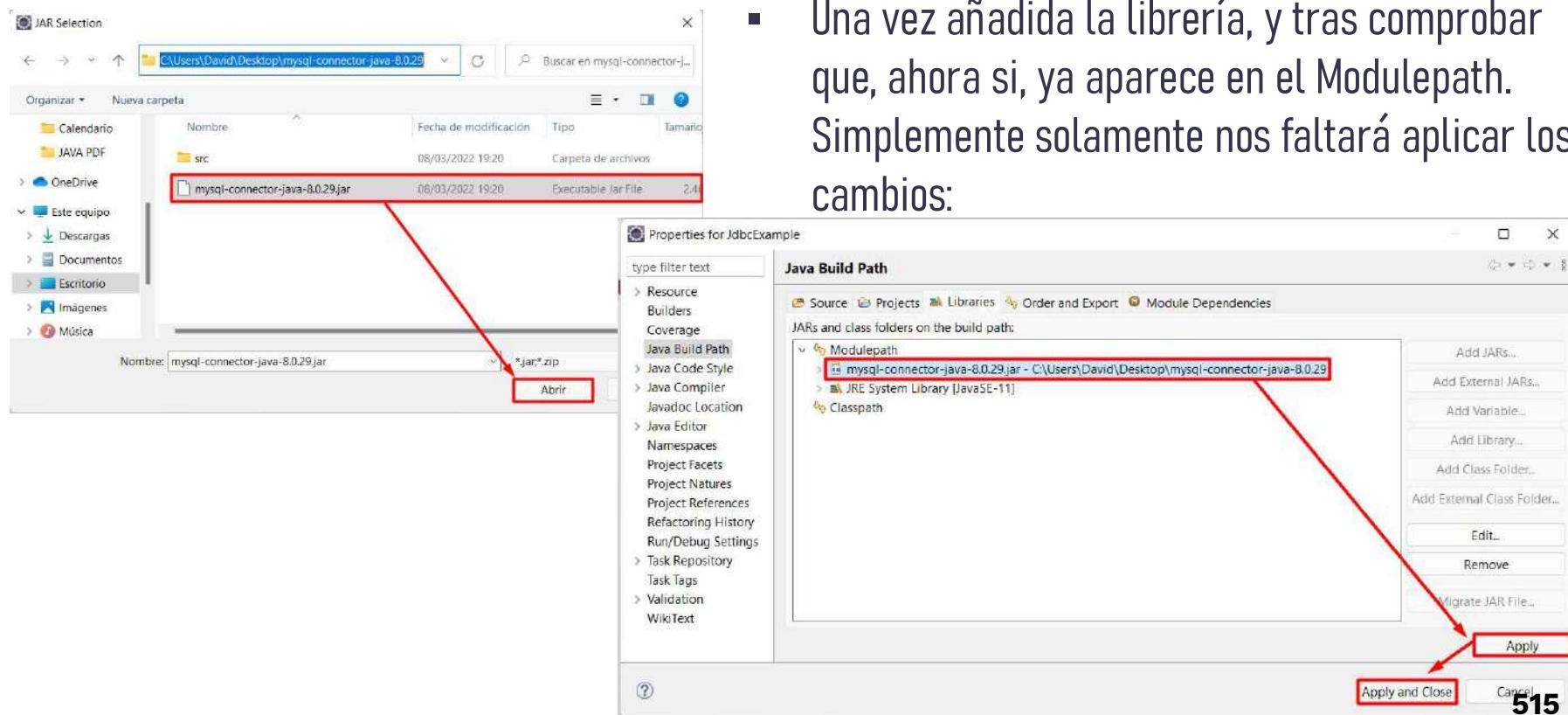
# 1. Añadiendo el driver

- Sobre el directorio del proyecto hacemos botón derecho > Properties. Y dentro de aquí, vamos a Java Build Path, Libraries, nos situamos sobre Modulepath lo que, finalmente, nos activará la opción de añadir una librería externa (JAR):



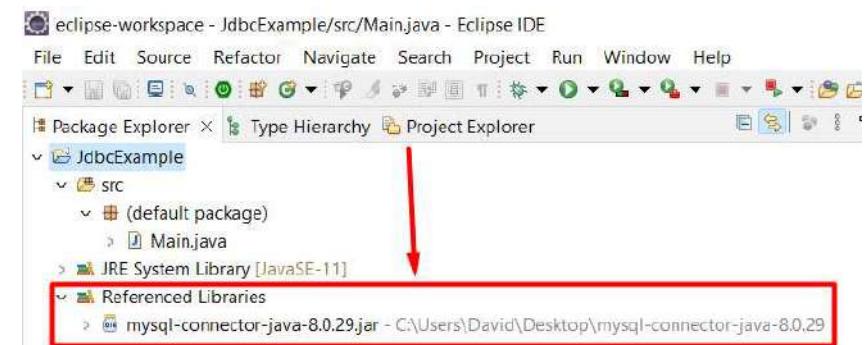
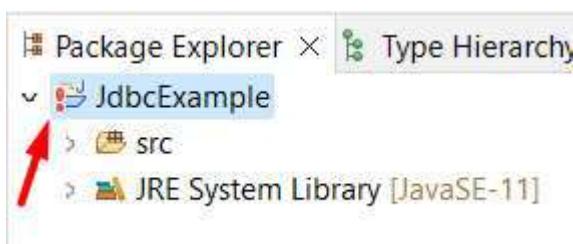
# 1. Añadiendo el driver

- Una vez añadida la librería, y tras comprobar que, ahora si, ya aparece en el Modulepath. Simplemente solamente nos faltará aplicar los cambios:

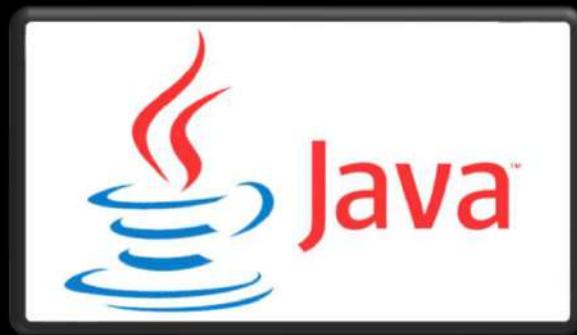


# 1. Añadiendo el driver

- Importante, es muy importante no borrar el JAR que acabamos de asociar. Ya que, si borramos la dependencia (JAR) del directorio que acabamos de asociar, es decir, en mi caos de la ruta C:\Users\David\Desktop\mysql-connector-java-8.0.29 el proyecto no podrá utilizar dicho JAE y nos aparecerá un mensaje diciendo que la librería está missing. Y un símbolo de exclamación junto al proyecto. Vamos a verlo:



## 2. CREANDO LA BBDD



## 2.Creando la BBDD

- La BBDD que vamos a utilizar es la siguiente:
  - Tenéis el código fuente en el siguiente enlace: [Gifts al código fuente](#)

```
CREATE SCHEMA IF NOT EXISTS db_test;
USE db_test;

CREATE TABLE IF NOT EXISTS db_test.usuarios(
 id_usuarios integer unsigned auto_increment primary key,
 nombre varchar(50),
 edad integer,
 nacionalidad varchar(50)
);

INSERT INTO db_test.usuarios(nombre, edad, nacionalidad) VALUES
 ('David', 28, 'España'),
 ('Matias', 33, 'Colombia'),
 ('Juan', 65, 'Francia');

SELECT * FROM db_test.usuarios;
```

## 2.Creando la BBDD

- Bastará con copilar el código del script y ejecutar la BBDD:

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the list of databases. The "db\_test" database is selected and highlighted with a red box.
- SQL File 4\***: A query editor containing the following SQL code:

```
10
11 • INSERT INTO db_test.usuarios(nombre, edad, nacionalidad) VALUES
12 ('David', 28, 'España'),
13 ('Matías', 33, 'Colombia'),
14 ('Juan', 65, 'Francia');
15
16 • SELECT * FROM db_test.usuarios;
```
- Result Grid:** A table showing the results of the SELECT query. The data is:

|   | id_usuarios | nombre | edad | nacionalidad |
|---|-------------|--------|------|--------------|
| 1 | 1           | David  | 28   | España       |
| 2 | 2           | Matías | 33   | Colombia     |
| 3 | 3           | Juan   | 65   | Francia      |
| * | NULL        | NULL   | NULL | NULL         |
- Output:** A log of the actions taken:

| # | Time     | Action                                                                                                                                       | Message                                                                                |
|---|----------|----------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| 1 | 23:57:42 | CREATE SCHEMA IF NOT EXISTS db_test                                                                                                          | 1 row(s) affected, 1 warning(s): 1007 Can't create database 'db_test'; database exists |
| 2 | 23:57:42 | USE db_test                                                                                                                                  | 0 row(s) affected                                                                      |
| 3 | 23:57:42 | CREATE TABLE IF NOT EXISTS db_test.usuarios( id_usuarios integer unsigned auto_increment primary key, ...)                                   | 0 row(s) affected                                                                      |
| 4 | 23:57:42 | INSERT INTO db_test.usuarios(nombre, edad, nacionalidad) VALUES ('David', 28, 'España'), ('Matías', 33, 'Colombia'), ('Juan', 65, 'Francia') | 3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0                                 |
| 5 | 23:57:42 | SELECT * FROM db_test.usuarios LIMIT 0, 1000                                                                                                 | 3 row(s) returned                                                                      |

### **3. REALIZANDO LA CONEXIÓN A LA BBDD**



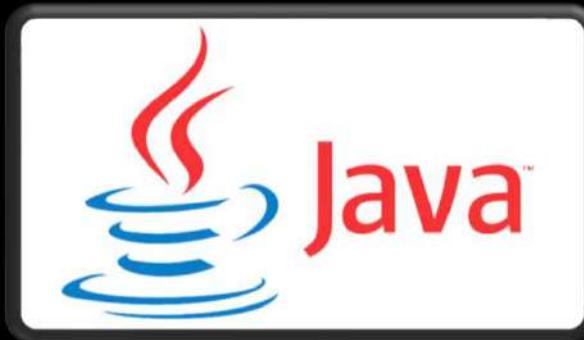
### 3. Realizando la conexión a la BBDD

- Para realizar la conexión a la BBDD realizamos lo siguiente:
  - [GIFTS al código del ejemplo](#)

The screenshot shows a Java development environment with two windows. On the left is the code editor for 'Main.java' with line numbers 1 to 29. The code imports java.sql.Connection and java.sql.DriverManager, defines a Main class with a main method, and attempts to connect to a MySQL database using JDBC. It prints 'Connection success.' if successful and 'Connection closed.' when closing the connection. On the right is the 'Console' window showing the output: '<terminated> Main (4) [Java App] Connection success. Connection closed.'.

```
1 import java.sql.Connection;
2 import java.sql.DriverManager;
3
4 public class Main {
5 public static void main(String[] args) {
6 Connection connection;
7 // Ponemos la URL de la BBDD a la que queremos conectarnos mediante a JDBC
8 final String url = "jdbc:mysql://localhost:3306/db_test";
9 // Accesos a la BBDD
10 final String user = "admin";
11 final String pass = "admin";
12
13 try {
14 // Realizando la conexión
15 connection = DriverManager.getConnection(url, user, pass);
16 // Si todo ha ido OK
17 System.out.println("Connection success.");
18 // --> Aquí continuaremos realizando el resto de las operaciones de SQL
19
20 // Si todo ha ido OK una vez finalizado el uso de la conexión la cerraremos
21 connection.close();
22 // Y en este caso, confirmamos que la conexión ha sido cerrada mediante a un
23 // mensaje
24 System.out.println("Connection closed.");
25 } catch (Exception e) {
26 System.err.println(e);
27 }
28 }
29 }
```

- 4. CREANDO EL STATEMENT**
- 5. EJECUTANDO LA QUERY**
- 6. MOSTRANDO EL RESULTADO**
- 7. CERRANDO CONEXIÓN**



## 4, 5, 6 y 7. Realizando la consulta

- Para realizar la conexión a la BBDD realizamos lo siguiente:
  - [GIFTS al código del ejemplo](#)



The screenshot shows a Java application window titled "Console X". The title bar also includes the text "<terminated> Main (4) [Java Application] C:\Program Files\Java\jdk-11.". The main area displays a table output from a database query:

| <code>id_usuarios</code> | <code>nombre</code> | <code>edad</code> |
|--------------------------|---------------------|-------------------|
| -----                    |                     |                   |
| 1                        | David               | 28                |
| 2                        | Matias              | 33                |
| 3                        | Juan                | 65                |

## 4, 5, 6 y 7. Realizando la consulta

```
1o import java.sql.Connection;
2 import java.sql.DriverManager;
3 import java.sql.ResultSet;
4 import java.sql.ResultSetMetaData;
5 import java.sql.Statement;
6
7 public class Main {
8o public static void main(String[] args) {
9 Connection connection;
10 final String url = "jdbc:mysql://localhost:3306/db_test";
11 final String user = "admin";
12 final String pass = "admin";
13
14 try {
15 connection = DriverManager.getConnection(url, user, pass);
16 // Creamos el statement
17 Statement statement = connection.createStatement();
18 // Es equivalente a realizar un SELECT * FROM usuarios; ya que ya estamos situados en la BBDD
19 ResultSet result = statement.executeQuery("SELECT * FROM db_test.usuarios");
20 // Mostrando el resultado
21 ResultSetMetaData rsmd = result.getMetaData();
22 int columnCount = rsmd.getColumnCount();
23 // Recorriendo los nombres de los encabezados de la tabla
24 for (int i = 1; i < columnCount; i++) {
25 System.out.printf("%10s", rsmd.getColumnName(i).toString());
26 }
27 int spacing = ((columnCount * 10) - 5);
28 String textoFormateado = String.format("%n%" + spacing + "s", " ", " ");
29 System.out.println(textoFormateado);
```

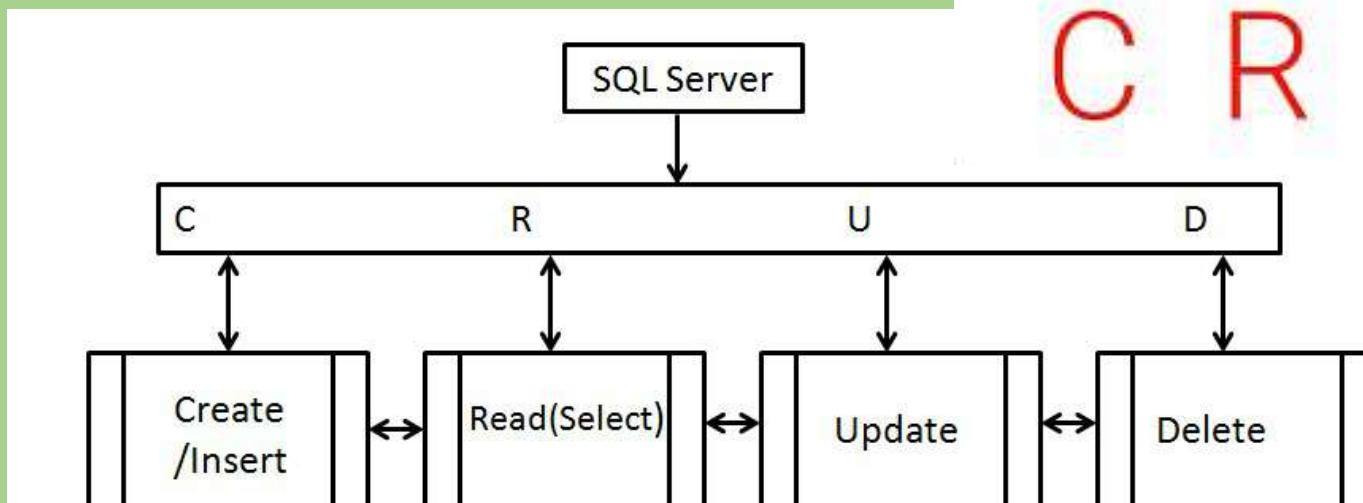
## 4, 5, 6 y 7. Realizando la consulta

```
30 // Mostrando el resultado
31 while (result.next()) {
32 int id_usuarios = result.getInt("id_usuarios");
33 String nombre = result.getString("Nombre");
34 int edad = result.getInt("Edad");
35 System.out.printf("%10d%10s%10d%n", id_usuarios, nombre, edad);
36 }
37 result.close(); // Cerrando resultset
38 statement.close(); // Cerrando statement
39 connection.close(); // Cerrando connection
40 } catch (Exception e) {
41 System.err.println(e);
42 }
43 }
44 }
```

# Ejercicio CRUD SQL + Java + Scanner

- Realiza un “CRUD” que mediante a un método Scanner nos pregunte que queremos hacer y nos permita:

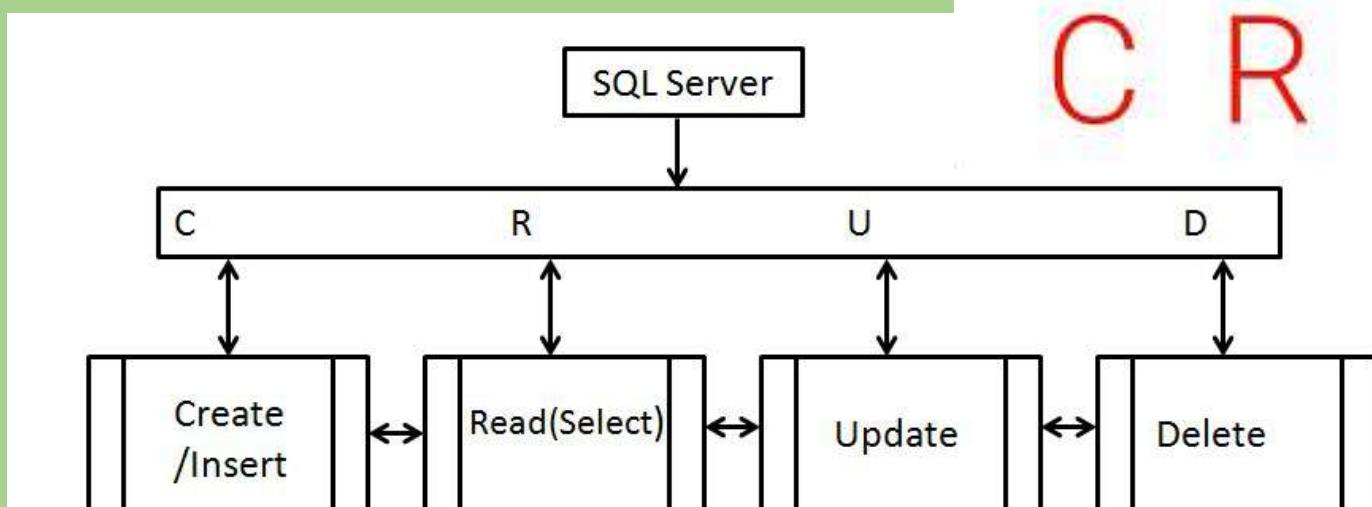
- Create (crear) mediante a un INSERT
- Read (leer) mediante a un SELECT
- Update (actualizar) mediante a UPDATE
- Delete (eliminar) mediante a DELETE



# Ejercicio CRUD SQL + Java + Scanner

- Realiza un programa que nos permita ejecutar querys SQL directamente desde nuestro terminal.  
Los métodos que podremos utilizar son:

- Create (crear) mediante a un INSERT
- Read (leer) mediante a un SELECT
- Update (actualizar) mediante a UPDATE
- Delete (eliminar) mediante a DELETE



# **PRESENTANDO A SWING GRAPHICAL USER INTERFACE (GUI)**

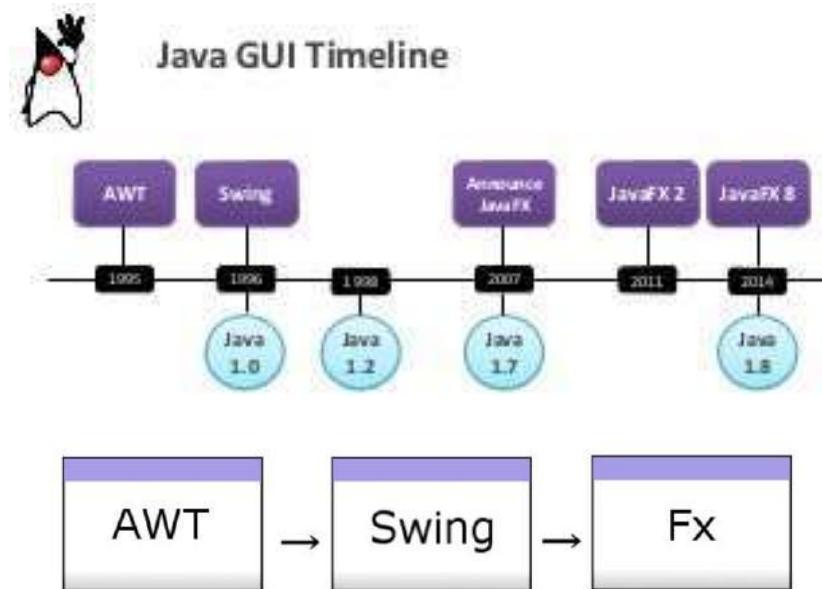


# Tipos de GUI en Java:

Para crear interfaces en Java, vamos hablar de las 3 GUIs más utilizadas/importantes. Además, las vamos a subdividir en dos grandes grupos en función de si vienen integradas o no en el JDK. Vamos a verlas:

- Vienen integradas en el JDK:
  - **AWT (Abstract Window Toolkit):** la versión más “old school” (vieja) de interfaces de Java. Dentro de sus características destaca por ser una biblioteca bastante pesada. Y, actualmente, es muy poco utilizada, por no decir, que prácticamente no es utilizada.
  - **Swing:** la versión que sucede a AWT. Es una biblioteca bastante ligera respecto a AWT.
- No vienen integradas en el JDK:
  - **Java FX:** la GUI más reciente de Java. Para trabajar con JavaFX, es necesario descargar la plataforma que nos permitirá trabajar con Java FX desde la siguiente URL: <https://openjfx.io/>

GUI = Graphical User Interface



# PRESENTANDO A SWING



# Presentando a Swing

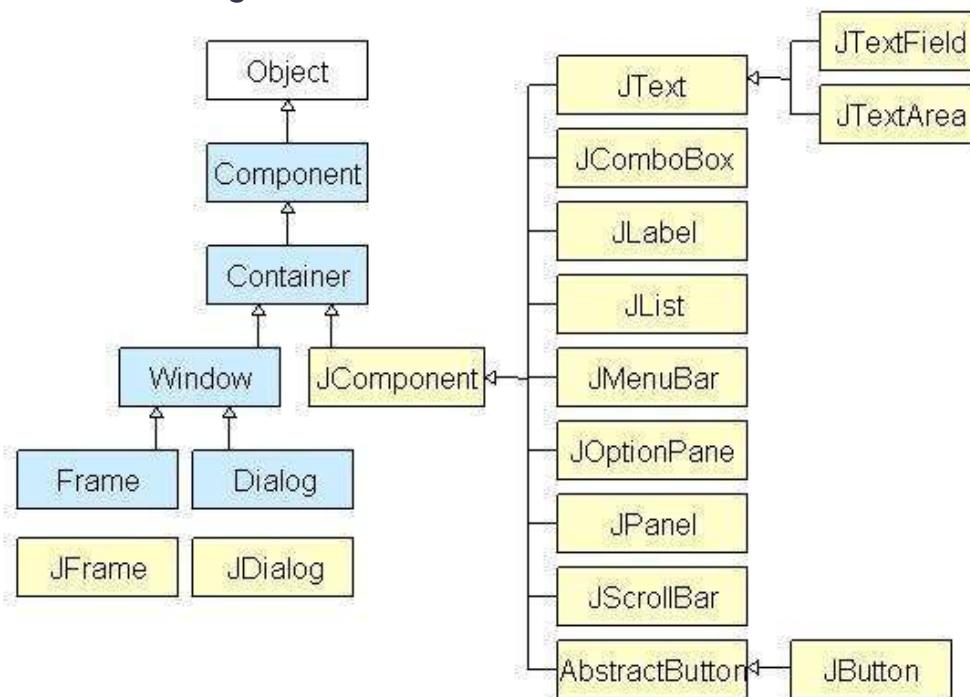


- Swing es una librería/biblioteca de Java que contiene una serie de componentes que nos permitirán construir las interfaces de nuestras aplicaciones de escritorio.
- Tal y como hemos dicho, Swing se basa en otro un sistema algo más viejo de construcción de GUIs llamado AWT (Abstract Window Toolkit)

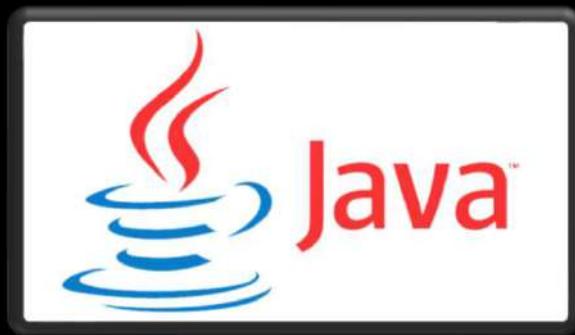
# Swing



- Swing tiene distintos componentes que podemos utilizar (o no) para realizar nuestras interfaces gráficas.



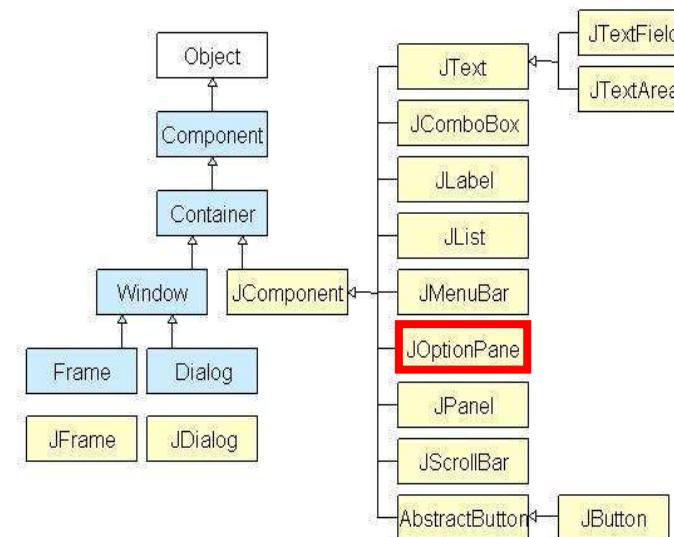
# ¿QUÉ ES JOPTIONPANE?



# ¿Qué es el JOptionPane?



- JOptionPane es un componente de Swing que nos permitirá enviar mensajes de una forma muy muy muy simple.
- Aunque el componente más importante de Swing es el contenedor, vamos a empezar con este tipo de componentes para que el empezar a trabajar con Swing sea mucho más ameno. Vamos a ver un ejemplo:



# **TIPOS DE JOPTIONPANE**



# SHOWMESSAGEBOX()



# JOptionPane.showMessageDialog()



- Dentro de la clase JOptionPane tenemos varios tipos de mensajes distintos vamos a ver el primero:
  - JOptionPane.showMessageDialog(): nos permite mostrar alertas que vendrán acompañadas de un botón de OK. Vamos a ver un ejemplo:

The screenshot shows a Java code editor with a file named Main.java. The code is as follows:

```
1 import javax.swing.JOptionPane;
2
3 public class Main {
4 public static void main(String[] args) {
5 JOptionPane.showMessageDialog(null, "Hola");
6 }
7 }
```

To the right of the code, a message dialog box titled "Message" is displayed. It contains an information icon (an exclamation mark inside a circle), the text "Hola", and an "OK" button at the bottom.

# Estructura de un showMessageDialog



- Con `JOptionPane.showMessageDialog()` es posible trabajar de varias maneras en función de la cantidad de parámetros que le pasemos. En el ejemplo anterior, hemos trabajado con la primera de ellas, la más básica.

The screenshot shows a Java code editor with the file `Main.java`. The code defines a `Main` class with a `main` method that calls `JOptionPane.showMessageDialog(null, "Hola")`. A red box highlights this call. An arrow points from this red box to the corresponding JavaDoc entry in the bottom right corner of the code editor. A message dialog titled "Message" with the text "Hola" and an "OK" button is displayed on the screen.

```
1 import javax.swing.JOptionPane;
2
3 public class Main {
4 public static void main(String[] args) {
5 JOptionPane.showMessageDialog(null, "Hola");
6 }
7 }
```

showMessageDialog(Component parentComponent, Object message) : void - JOptionPane

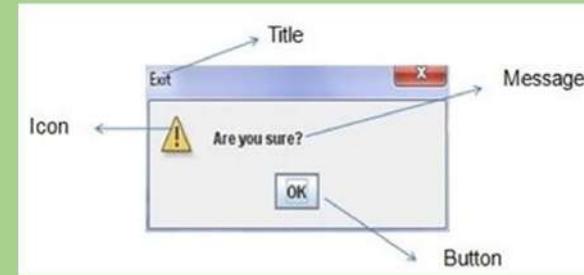
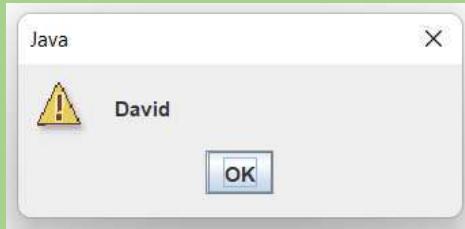
showMessageDialog(Component parentComponent, Object message, String title, int messageType) : void - JOptionPane

showMessageDialog(Component parentComponent, Object message, String title, int messageType, Icon icon) : void - JOptionPane

538

# Ejercicios parámetros showMessageDialog

- En la diapositiva anterior ya hemos visto que el método showMessageDialog, nos permite trabajar de varias maneras en función de la cantidad de parámetros que le asignemos a nuestro método showMessageDialog.
  - El ejercicio consistirá en trabajar con estos 3 métodos que hemos visto anteriormente, jugar con los iconos (en el caso de que el método lo permita), definir un título, etc.
  - En el método que tienen component, nombre, title, icon quiero que imprimáis vuestro nombre y el title Java por ejemplo:



| Icon    | Code                            | IDE Value |
|---------|---------------------------------|-----------|
| No icon | JOptionPane.PLAIN_MESSAGE       | -1        |
| ✖       | JOptionPane.ERROR_MESSAGE       | 0         |
| ⓘ       | JOptionPane.INFORMATION_MESSAGE | 1         |
| ⚠       | JOptionPane.WARNING_MESSAGE     | 2         |
| ?       | JOptionPane.QUESTION_MESSAGE    | 3         |

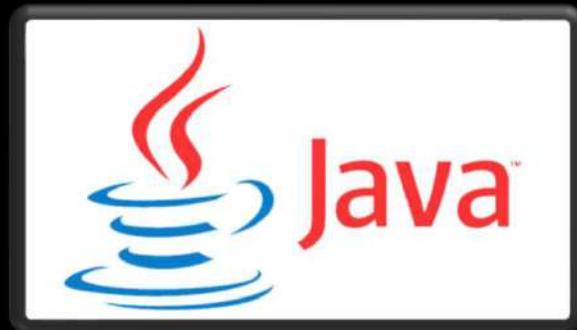
| JOptionPane Icons               |                              |
|---------------------------------|------------------------------|
| ⓘ                               | ?                            |
| JOptionPane.INFORMATION_MESSAGE | JOptionPane.QUESTION_MESSAGE |
| ⚠                               | ✖                            |
| JOptionPane.WARNING_MESSAGE     | JOptionPane.ERROR_MESSAGE    |

## Ejercicios parámetros showMessageDialog

- Haz un showMessageDialog con el siguiente formato:



# SHOWINPUTDIALOG()



# JOptionPane.showInputDialog()



- Dentro de la clase JOptionPane tenemos varios tipos de mensajes distintos vamos a ver el primero:
  - JOptionPane.showInputDialog(): nos permite realizar una especie de alert que irá acompañado de un input. Vamos a ver un ejemplo:

The image shows a Java code editor window titled "Main.java" and a running Java application window titled "Input".

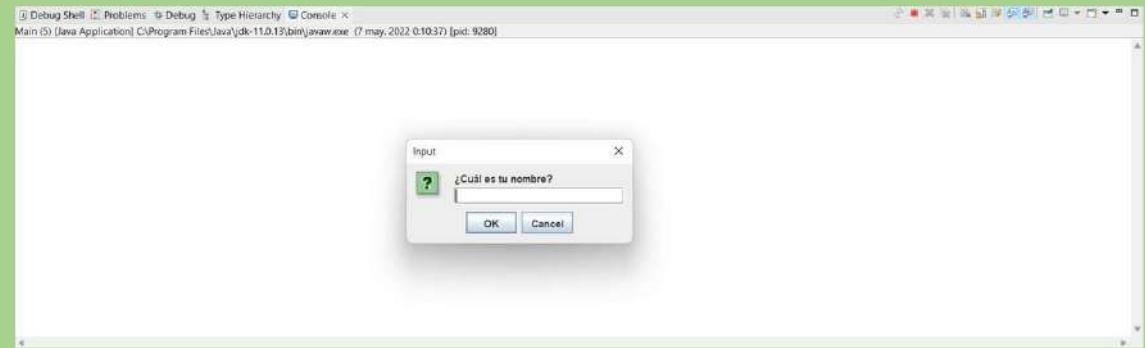
The code in Main.java is:

```
1 package com.main;
2
3 import javax.swing.JOptionPane;
4
5 public class Main {
6 public static void main(String[] args) {
7 JOptionPane.showInputDialog("¿Cuál es tu nombre? ");
8 }
9 }
```

The "Input" dialog window has a question mark icon and the text "¿Cuál es tu nombre?". It contains an input field, an "OK" button, and a "Cancel" button.

# Ejercicios parámetros showInputDialog()

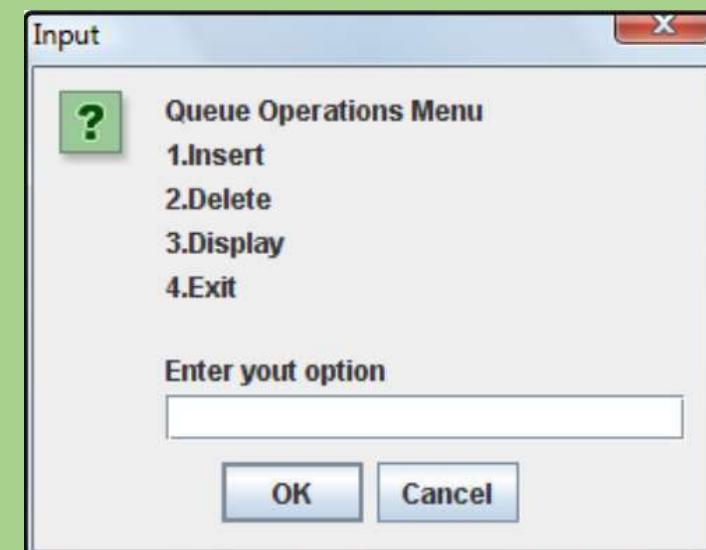
- 1. Realiza un showMessageDialog que imprima un mensaje por terminal con el texto introducido en el showInputDialog():
- 2. Realiza un ejercicio que nos imprima un showMessageDialog con el resultado introducido en un showInputDialog. Además de ello, deberemos de:
  - Comprobar si el usuario pulsa el botón de OK o de cancel. Y ante el caso de que pulse el botón de cancelar, deberemos de mostrar un showMessageDialog con el icono WARNING\_MESSAGE. Y, posteriormente, volver a enviar el showInputDialog nuevamente hasta que se rellene y se pulse OK.
  - En el caso de que el usuario pulse OK, si el texto no está vacío, mostraremos el mensaje en un showMessageDialog con el icono INFORMATION\_MESSAGE y también en la terminal. En el caso de que el input no se rellene, o bien solo tenga espacios en blanco (sin caracteres), deberemos de mostrar un showMessageDialog con el icono ERROR\_MESSAGE y posteriormente volver a mostrar un showInputDialog.



# Ejercicios parámetros showInputDialog()

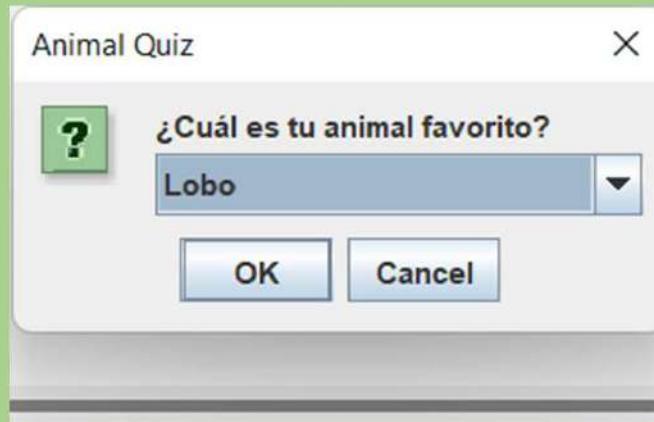
- 3. Realiza un programa similar al siguiente, que nos ofrecerá los distintos iconos para que cuando le introduzcamos uno se abra un showMessageDialog que contendrá el ícono correspondiente:

| Icon    | Code                            | IDE Value |
|---------|---------------------------------|-----------|
| No icon | JOptionPane.PLAIN_MESSAGE       | -1        |
| X       | JOptionPane.ERROR_MESSAGE       | 0         |
| i       | JOptionPane.INFORMATION_MESSAGE | 1         |
| !       | JOptionPane.WARNING_MESSAGE     | 2         |
| ?       | JOptionPane.QUESTION_MESSAGE    | 3         |



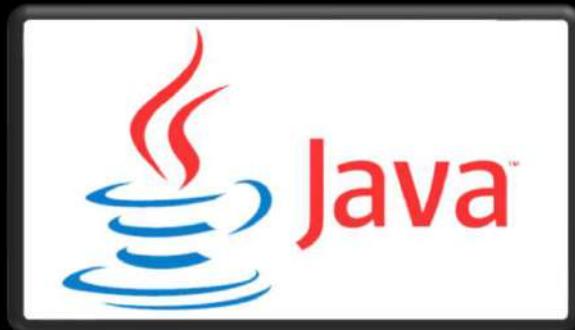
## Ejercicios parámetros showInputDialog()

- 4. Existe una función más avanzando que nos permite realizar lo siguiente en showInputDialog():



- Investiga como hacer esto y que por defecto el animal seleccionado sea el Lobo. Además, cuando el demos a OK se nos tiene que abrir un showMessageDialog() con una imagen del animal seleccionado.

# SHOWCONFIRMDIALOG()



# JOptionPane.showConfirmDialog()



- Dentro de la clase JOptionPane tenemos varios tipos de mensajes como ya hemos visto. En este caso vamos a ver:
  - **JOptionPane.showConfirmDialog()**: nos permite realizar una especie de alert que irá acompañado de 3 botones Yes No Cancel. Vamos a ver un ejemplo:

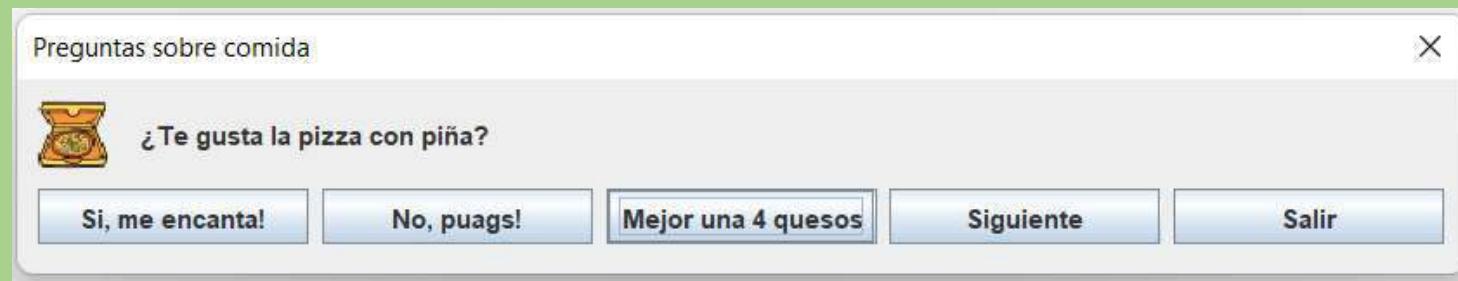
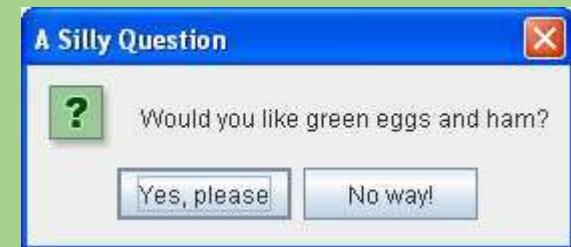
The image shows a Java code editor window titled "Main.java" and a "Select an Option" dialog box. The code in Main.java is as follows:

```
1 package com.main;
2
3 import javax.swing.JOptionPane;
4
5 public class Main {
6 public static void main(String[] args) {
7 JOptionPane.showConfirmDialog(null, "¿Te gusta Java?");
8 }
9 }
```

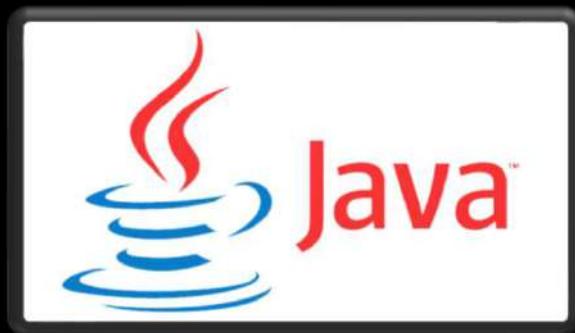
The "Select an Option" dialog box has a question mark icon and the text "¿Te gusta Java?". It contains three buttons: "Yes", "No", and "Cancel".

# Ejercicios parámetros showConfirmDialog()

- 1. Haz un showConfirmDialog() que nos muestre por consola y por pantalla el botón pulsado (Yes No Cancel) en función del botón que pulsemos.
- 2. Personalízalos botones para que sean los siguientes:
- 3. Haz un programa que muestre frases aleatorias guardadas en un arraylist de Preguntas en el que también se definen los iconos y los textos de los botones para que vayan cambiando. Por ejemplo el siguiente:



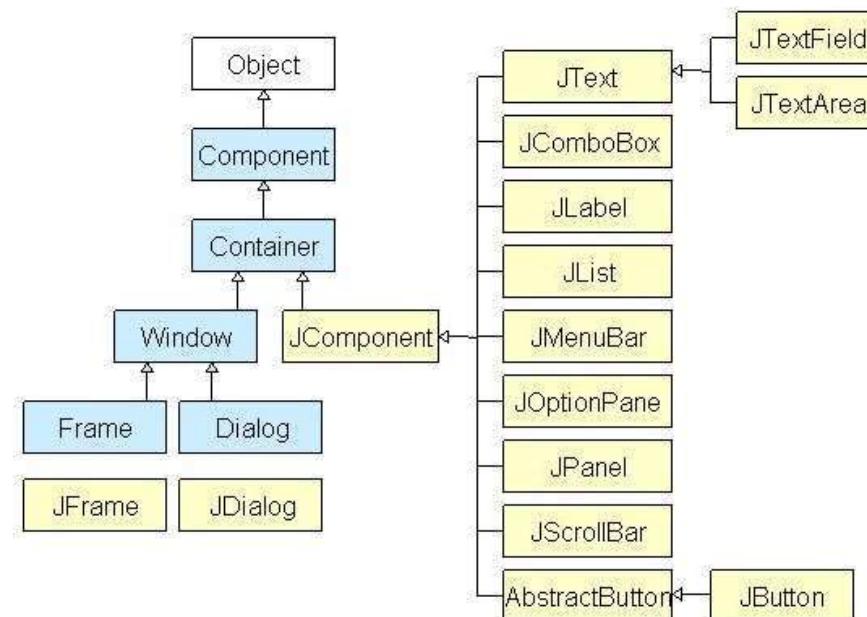
# CLASIFICACIÓN DE COMPONENTES DE SWING



# Clasificación de los distintos componentes de Swing



- Dentro de este conjunto que contiene distintos elementos, se subdivide principalmente en dos grandes grupos:
  - **Contenedores (containers)**: Son elementos que son capaces de albergar otros elementos.
  - **Componentes (components)**: Son los elementos que se albergaran dentro de los contenedores. Como por ejemplo botones, listas, textos, etc.

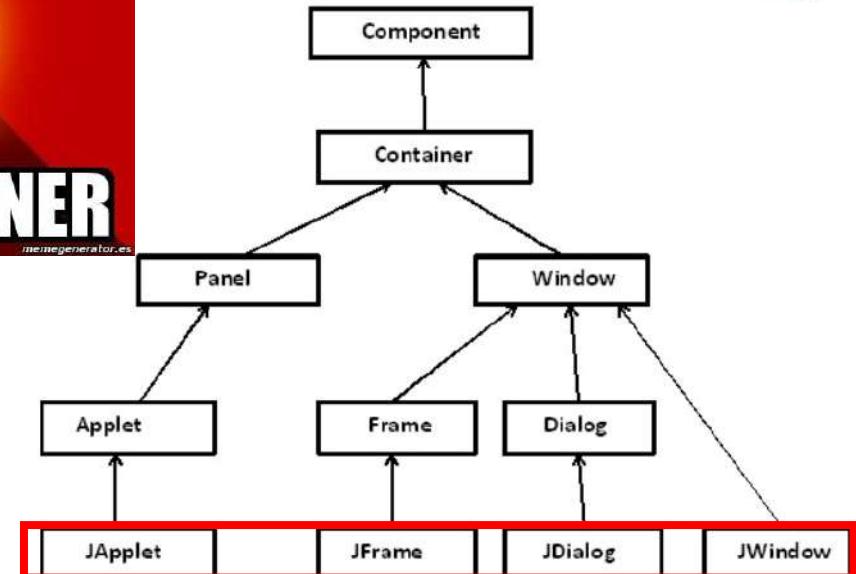


# PRESENTANDO A LOS ELEMENTOS CONTAINERS

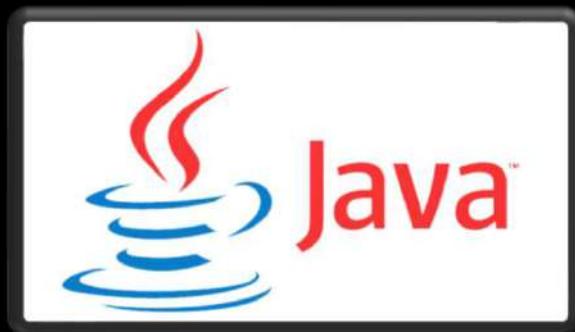


# Containers

- Dentro de los containers, tenemos las ventanas(Windows) mediante a las cuales construimos nuestra aplicación. Principalmente las dos más utilizadas son:
  - JFrame: Representa a la ventana principal de nuestra aplicación.
  - JDialog: Representa a una ventana



# JFRAME

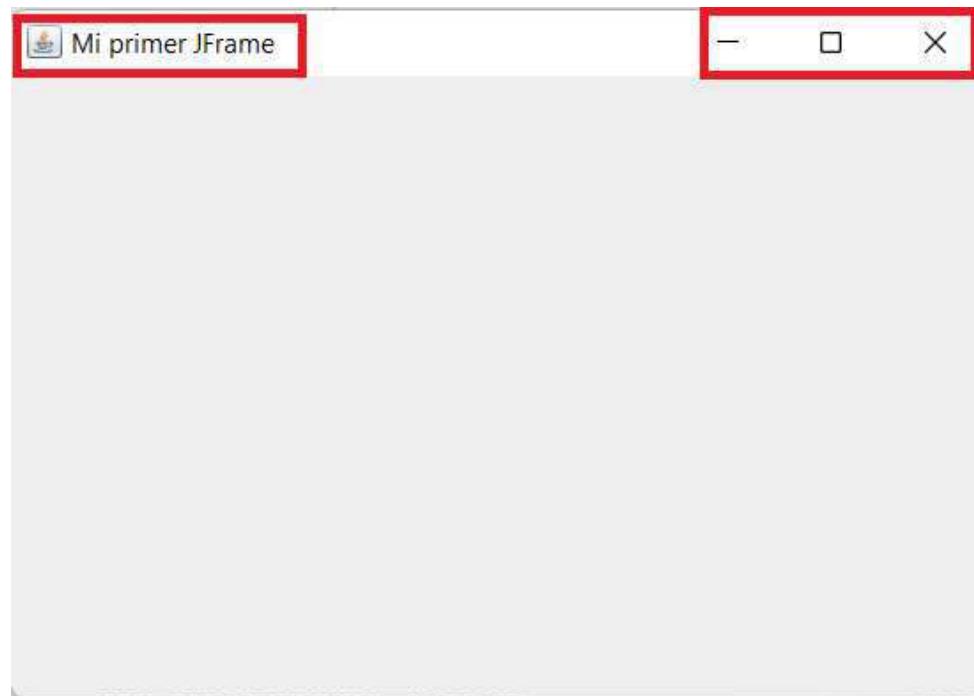


553

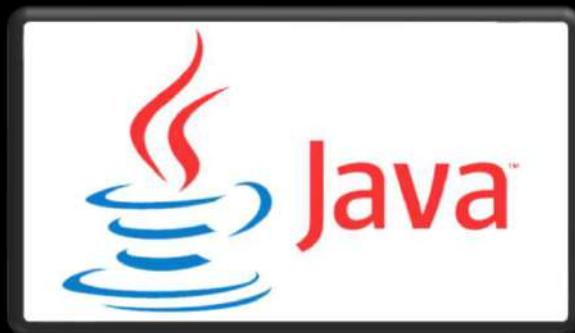
# Container JFrame



- Si nos fijamos, las características de JFrame, podemos observar que este tipo de container, tiene botones de maximizar, minimizar y cerrar. Y, también un título para dicha ventana.



# **CREANDO UN JFRAME (DESDE MAIN)**



# Container JFrame



- Para crear una JFrame hacemos lo siguiente:
- Si nos fijamos, además, vemos que estamos trabajando sobre el package javax.swing. En este caso, en concreto sobre JFrame.

```
Main.java x
1 package interfaces;
2
3 import javax.swing.JFrame;
4
5 public class Main {
6
7 public static void main(String[] args) {
8 //Nos permite crear un JFrame
9 JFrame jframe = new JFrame();
10 }
11 }
```

# ¿Por qué no se visualiza nuestro Jframe?



- **Pero si ejecutemos la aplicación, podemos ver que no se ve nada ¿Dónde está mi JFrame?**
- Algunas características que tenemos que tener en cuenta cuando construimos un JFrame son las siguientes:
  - **Por defecto, son invisibles.** Por lo que es necesario definir que sean visibles mediante el método setVisible().
  - **Nacen con un tamaño 0,0** por lo que no se visualizaran. Por lo que es necesario definir un tamaño mediante el método setSize()
  - **Es necesario definir/indicar un comportamiento cerramos nuestro programa** mediante al a la X. Como por ejemplo cierre el programa. Para ello, utilizamos el método setDefaultCloseOperation.



# Container JFrame

- Vamos a ver un ejemplo de un JFrame que si que se visualiza:

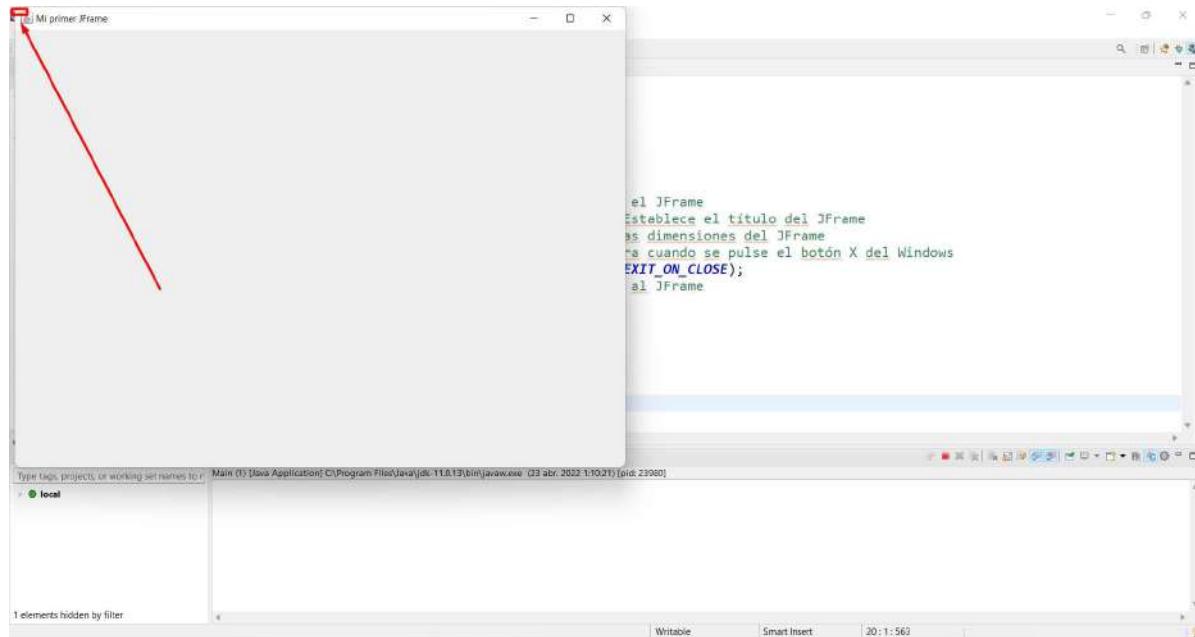


```
Main.java × Mi primer JFrame - □ ×
1 package interfaces;
2
3 import javax.swing.JFrame;
4
5 public class Main {
6
7 public static void main(String[] args) {
8 JFrame jframe = new JFrame(); //Creamos el JFrame
9 jframe.setTitle("Mi primer JFrame"); //Establece el título del JFrame
10 jframe.setSize(800, 600); //Establece las dimensiones del JFrame
11 // Permite definir un comportamiento para cuando se pulse el botón X del Windows
12 jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13 jframe.setVisible(true); //Hace visible al JFrame
14 }
15 }
16 }
```

# Container JFrame: posición por defecto del JFrame



- Si nos fijamos, el JFrame por defecto, si no le indicamos nada de como debe posicionarse, se abrirá automáticamente en el punto 0,0 de la pantalla.

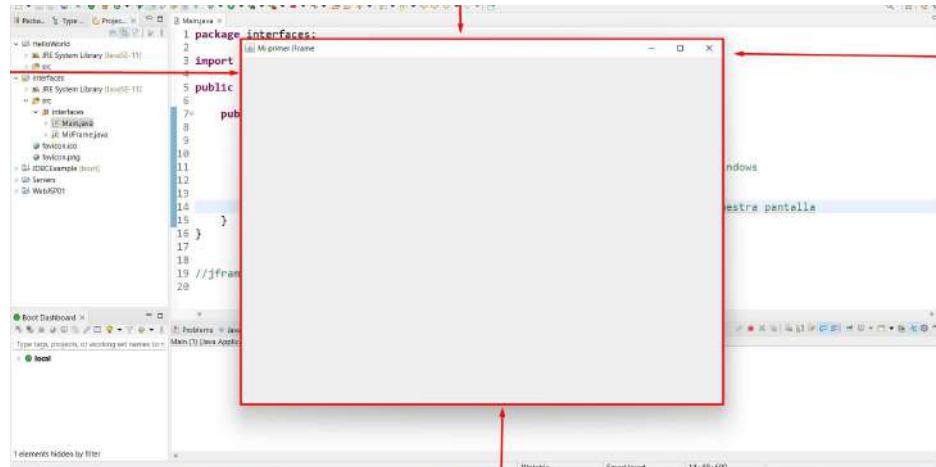


# Centrando el JFrame: setLocationRelativeTo

- Para cambiar este comportamiento por defecto, utilizamos la propiedad setLocationRelativeTo(null) sobre el JFrame. Vamos a ver un ejemplo:

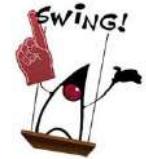


```
Main.java x
1 package interfaces;
2
3 import javax.swing.JFrame;
4
5 public class Main {
6
7 public static void main(String[] args) {
8 JFrame jframe = new JFrame(); //Creamos el JFrame
9 jframe.setTitle("Mi primer JFrame"); //Establece el título del JFrame
10 jframe.setSize(800, 600); //Establece las dimensiones del JFrame
11 // Permite definir un comportamiento para cuando se pulse el botón X del Windows
12 jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13 jframe.setVisible(true); //Hace visible al JFrame
14 jframe.setLocationRelativeTo(null); //Centrará el JFrame en el centro de nuestra pantalla
15 }
16 }
```



# Container JFrame:

## getContentPane().setBackground();

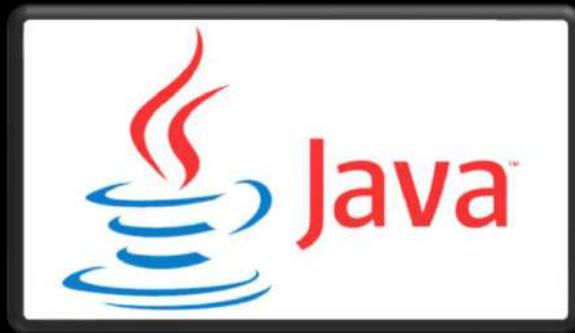


- Mediante a getContentPane().setBackground(); modificamos el color por defecto.
- Para ello, además, dentro del método setBackground(), debemos añadir un Color. Para ello la definición del color, tenemos muchas maneras de realizarlo. Vamos a ver dos de las principales:

The image shows a Java code editor with a file named MiJFrame.java. The code defines a class MiJFrame that extends JFrame. It includes imports for package, java.awt.Color, and javax.swing.JFrame. The constructor MiJFrame takes width, height, visible, and resizable parameters. Inside the constructor, it sets the size, visibility, and resizable status, then calls getContentPane().setBackground() twice: first with Color.LIGHT\_GRAY and then with a new Color object (new Color(5, 65, 90)). To the right of the code editor is a screenshot of a Java application window titled "MiJFrame". The window has a dark blue background color.

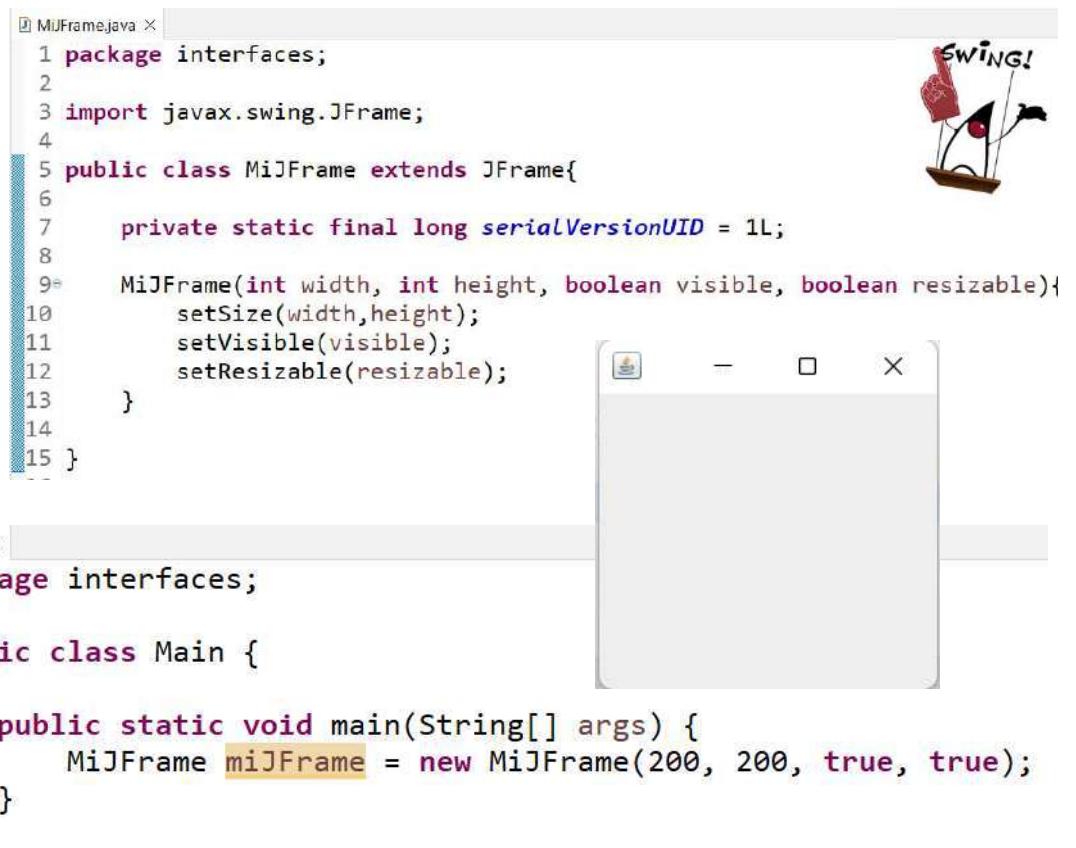
```
1 package interfaces;
2
3 import java.awt.Color;
4
5 import javax.swing.JFrame;
6
7 public class MiJFrame extends JFrame{
8
9 private static final long serialVersionUID = 1L;
10
11 MiJFrame(int width, int height, boolean visible, boolean resizable){
12 setSize(width, height);
13 setVisible(visible);
14 setResizable(resizable);
15 getContentPane().setBackground(Color.LIGHT_GRAY);
16 getContentPane().setBackground(new Color(5, 65, 90));
17 }
18
19 }
```

# **CREANDO UN JFRAME (EN UNA CLASE EXTERNA)**



# Container JFrame

- También existe la posibilidad de crear un JFrame desde una clase. Para ello, es necesario:
  - Que la clase extienda de JFrame.
  - Definir los métodos tanto desde su constructor como desde la clase sobre la que instanciamos (realizamos la creación de nuestro objeto miJFrame).
- El principal de esta forma es que sólamente podemos extender de una sola clase. Por lo que no podremos extender de otra. Es el PVP a pagar por obtener otros beneficios como la reutilización.



```
MiJFrame.java x
1 package interfaces;
2
3 import javax.swing.JFrame;
4
5 public class MiJFrame extends JFrame{
6
7 private static final long serialVersionUID = 1L;
8
9 MiJFrame(int width, int height, boolean visible, boolean resizable){
10 setSize(width,height);
11 setVisible(visible);
12 setResizable(resizable);
13 }
14
15 }
```

```
Main.java x
1 package interfaces;
2
3 public class Main {
4
5 public static void main(String[] args) {
6 MiJFrame miJFrame = new MiJFrame(200, 200, true, true);
7 }
8 }
```

# JFRAME EXTERNO CON PARAMETROS EN EL CONSTRUCTOR



# Container JFrame: asignando parámetros en el constructor

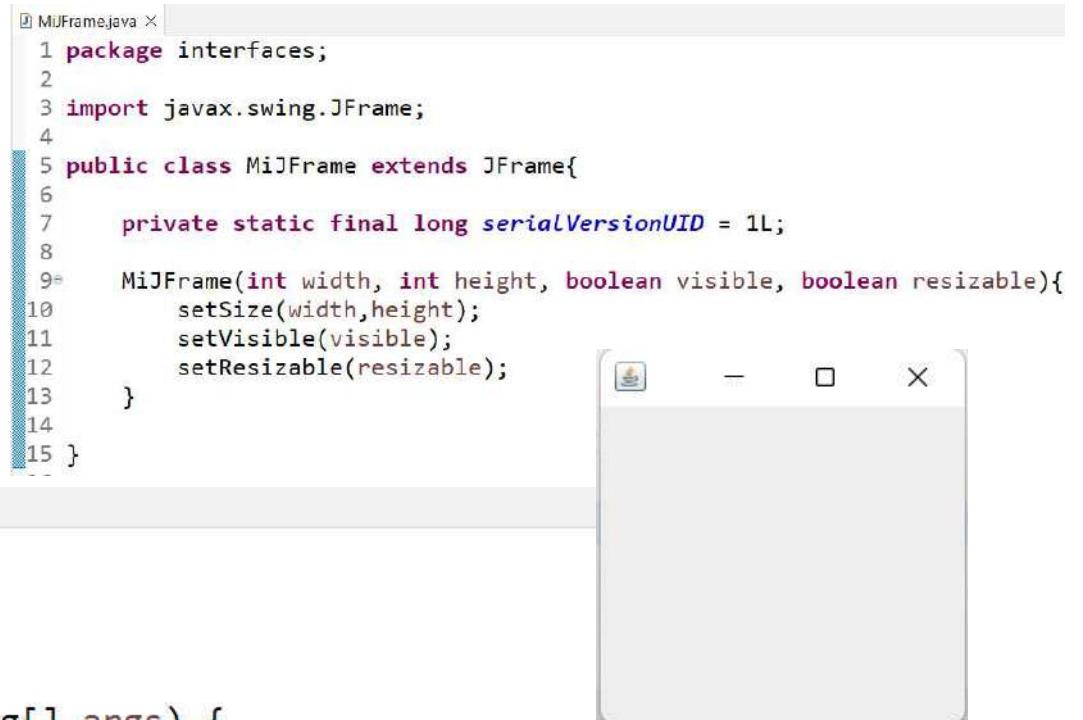


- Inclusive, podemos hacer que el constructor de la clase nos permite utilizar ciertas características:

```
MIJFrame.java x
1 package interfaces;
2
3 import javax.swing.JFrame;
4
5 public class MIJFrame extends JFrame{
6
7 private static final long serialVersionUID = 1L;
8
9 MIJFrame(int width, int height, boolean visible, boolean resizable){
10 setSize(width,height);
11 setVisible(visible);
12 setResizable(resizable);
13 }
14
15 }
```

```
Main.java x
1 package interfaces;
2
3 public class Main {
4
5 public static void main(String[] args) {
6 MIJFrame miJFrame = new MIJFrame(200, 200, true, true);
7 }
8 }
```



# JFRAME CON PROPIEDADES EN EL CONSTRUCTOR Y EN LA CLASE QUE INSTANCIA



# Container JFrame

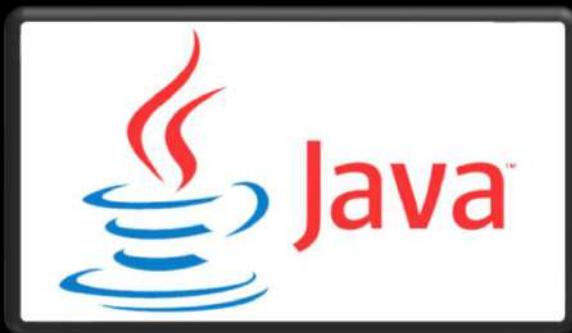


- O inclusive, trabajar partes del Jframe desde su constructor y otras partes desde la clase que instancia a dicha clase. Vamos a ver un ejemplo:

```
1 package interfaces;
2
3 import javax.swing.JFrame;
4
5 public class MiJFrame extends JFrame{
6
7 private static final long serialVersionUID = 1L;
8
9 MiJFrame(int width, int height, boolean visible, boolean resizable){
10 setSize(width, height);
11 setVisible(visible);
12 setResizable(resizable);
13 }
14 }
```

A screenshot of a Java development environment. On the left, there are two tabs: "MainJava.java" and "MiJFrame.java". The "MainJava.java" tab shows the main method of a class named Main, which creates an instance of MiJFrame and sets its title. The "MiJFrame.java" tab shows the implementation of the MiJFrame class, which extends JFrame and overrides the constructor to set size, visibility, and resizable properties. To the right of the tabs, a window titled "Mi primer JFrame" is displayed, showing a blank white frame.

# MÁS MÉTODOS DE JFRAME



# Container JFrame: setLocation



- Mediante a la propiedad `setLocation(X, Y)`, podemos definir unas ciertas coordenadas en px en las que posicionar nuestro elemento. Si aplicamos valores negativos, lo que haríamos es que no se visualizará nuestro JFrame. Vamos a verlo de `setLocation`:

The screenshot shows the Eclipse IDE interface with a Java file named `MiFrame.java` open. The code defines a `JFrame` class with a constructor that sets the frame's width to 400px and height to 200px, and its location to (400, 200). A red box highlights the line `setLocation(400, 200);`. Red arrows point from the text "400px" to the x-coordinate and "200px" to the y-coordinate in the code. The code also includes imports for `java.awt.Color` and `javax.swing.JFrame`, and a call to `getContentPane().setBackground`.

```
1 package interfaces;
2
3 import java.awt.Color;
4
5 import javax.swing.JFrame;
6
7 public class MiFrame extends JFrame{
8 private int width = 400;
9 private int height = 200;
10 private boolean visible = true;
11 private boolean resizable = true;
12
13 public MiFrame(){
14 setWidth(width);
15 setHeight(height);
16 setVisible(visible);
17 setResizable(resizable);
18 getContentPane().setBackground(Color.LIGHT_GRAY);
19 getContentPane().setBackground(new Color(5, 65, 90));
20 setLocation(400, 200);
21 }
22
23
24
25
26
27
28 }
```

# Container JFrame: setBounds



- setBounds, nos permite trabajar con setSize y con setLocation a la misma vez desde un solo método (el método setBounds). Vamos a ver un ejemplo:

The screenshot shows the Eclipse IDE interface with the file 'MiJFrame.java' open in the editor. The code defines a class 'MiJFrame' with a constructor that sets bounds, visibility, and resizable properties. The line 'setBounds(400, 50, 250, 250);' is highlighted with a red border.

```
1 package interfaces;
2
3 import java.awt.*;
4
5 import javax.swing.*;
6
7 public class MiJFrame extends JFrame {
8
9 private static final long serialVersionUID = 1L;
10
11 MiJFrame(int width, int height, boolean visible, boolean resizable){
12 //setSize(width, height);
13 //setLocation(400, 200);
14 setBounds(400, 50, 250, 250); // setBounds = setLocation(X,Y) + setSize(width, height);
15 setVisible(visible);
16 setResizable(resizable);
17 }
18}
```

# Container JFrame: setResizable();



- `setResizable();` acepta como parámetro un valor booleano que en función del valor definido nos permitirá:
  - True: es el valor por defecto, y permitirá que el elemento sea resizable (redimensionado)
  - False: no permitirá que el elemento sea resizable (redimensionado) ni tampoco maximizar el elemento.

A screenshot of the Eclipse IDE interface. The left side shows the Project Explorer with a 'HelloWorld' project containing 'src' and 'interfaces' folders, and files like 'Main.java' and 'MiJFrame.java'. The right side shows the Java Editor with the following code:

```
1 package interfaces;
2
3 import java.awt.Color;
4
5 import javax.swing.JFrame;
6
7 public class MiJFrame extends JFrame{
8
9 private static final long serialVersionUID = 1L;
10
11 MiJFrame(){
12 setBounds(400, 50, 250, 250); // setBounds = setLocation(X,Y) + setSize(width, height);
13 setVisible(true);
14 setResizable(true);
15 getContentPane().setBackground(Color.LIGHT_GRAY);
16 getContentPane().setBackground(new Color(5, 65, 90));
17 }
18
19
20 }
21
22 }
```

The code highlights the line `setResizable(true);` in blue, indicating it is selected or being edited.

# Container JFrame: setExtendedState();

- setExtentedState(); nos permite agrandar nuestro JFrame a pantalla completa, ponerlo a pantalla normal, etc. Vamos a ver un ejemplo:
  - **JFrame.MAXIMIZED\_BOTH**: nos permite maximizar el Jframe.
  - **JFrame.NORMAL**: nos permite volver a la normalidad. Es el comportamiento por defecto y sería equivalente a no tener definido el MAXIMIZED\_BOTH. Por lo que en la gran mayoría de casos, simplemente, bastará con borrar el setExtentedState.



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows a project named "interfaces" containing "HelloWorld", "src", "interfaces", and "MiFrame.java".
- MiFrame.java Content:** Displays the following Java code:

```
1 package interfaces;
2
3 import java.awt.Color;
4
5 import javax.swing.JFrame;
6
7 public class MiFrame extends JFrame{
8
9 private static final long serialVersionUID = 1L;
10
11 MiFrame(){
12 setBounds(400, 50, 250, 250); // setBounds = setLocation(X,Y) + setSize(width, height);
13 setVisible(true);
14 getContentPane().setBackground(Color.LIGHT_GRAY);
15 getContentPane().setBackground(new Color(5, 65, 90));
16 }
17
18 }
```
- Problems View:** Shows one warning: "Main.java:1: error: Main is not abstract and does not override abstract method void start() in Interface".
- Console View:** Shows the command "javac Main.java".

# Container JFrame: modificando el icono

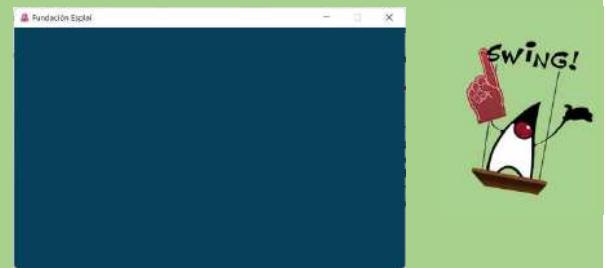


- Para modificar el icono de un JFrame hacemos lo siguiente:
  - 1. Añadimos el icono al directorio SRC. **IMPORTANTE RECOMENDADO UTILIZAN PNG**
  - 2. Utilizamos los métodos tal y como se realiza en el ejemplo

```
File Edit Source Refactor Navigate Search Project Run Window Help
File Project Type ...
MiJFrame.java
1 package interfaces;
2
3 import java.awt.Color;
4 import java.awt.Toolkit;
5
6 import javax.swing.JFrame;
7
8 public class MiJFrame extends JFrame {
9
10 private static final long serialVersionUID = 1L;
11
12 MiJFrame(){
13 setBounds(100, 100, 300, 200);
14 setVisibility(true);
15 getContentPane().setBackground(Color.LIGHT_GRAY);
16 getContentPane().setBackground(new Color(5, 65, 90));
17 setIconImage(Toolkit.getDefaultToolkit().getImage("./favicon.png"));
18 }
19}
```

# Ejercicio

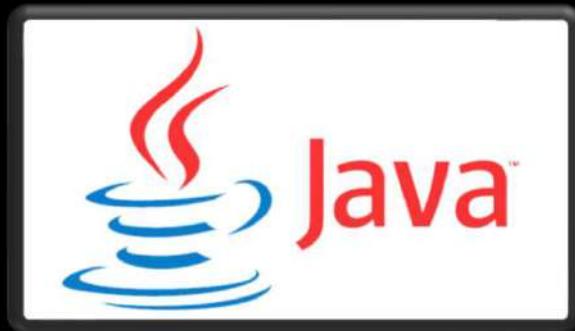
- Crea una contenedor que tenga unas características similares a las siguientes:
  - Título “Fundación Esplai”
  - Resolución: 800x600
  - Utiliza la propiedad que haga que no se pueda modificar la resolución del contenedor.
  - Cambia el color del JFrame
  - Añade un ícono con el logo de la fundación a la aplicación.  
<https://fundacionesplai.org/favicon.ico>
  - Haz que el JFrame se vea en el centro de la pantalla



# AÑADIENDO COMPONENTES EN NUESTRO JFRAME



# BOTONES Y EVENTOS



# Ejemplo de botones y eventos



- Aunque no es lo más recomendable, existe la posibilidad de añadir botones directamente sobre nuestro Jframe. Vamos a ver un ejemplo:

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows a project named "interfaces" containing "src" and "JRE System Library". Inside "src", there are "interfaces" and "Main.java". "Main.java" is currently selected and open in the editor.
- MiJFrame.java Content:**

```
1 package interfaces;
2
3 import javax.swing.JButton;
4 import javax.swing.JFrame;
5
6 public class MiJFrame extends JFrame{
7
8 private static final long serialVersionUID = 1L;
9
10 MiJFrame(){
11 JFrame jframe = new JFrame("Mi primera GUI");
12 jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13 jframe.setSize(600, 400);
14 jframe.setVisible(true);
15 // Creando y añadiendo el botón a mi frame
16 JButton btn1 = new JButton("Btn1");
17 jframe.add(btn1);
18 // Cuando hacemos click en btn1 se ejecutará dicho método
19 btn1.addActionListener(e -> {
20 System.out.println("Click en BTN1!");
21 });
22 }
23 }
```
- Console View:** Displays the message "No consoles to display at this time."
- Bottom Bar:** Shows icons for Boot Dash, Run, Stop, and others.

# Ejemplo de botones y eventos



- El principal problema de trabajar así es que si ponemos varios botones vemos que el comportamiento es bastante extraño. Y que los elementos se superponen. Por lo que el btn2 se pone encima del btn1. Haciendo que btn1 no sea visible. Vamos a ver un ejemplo:

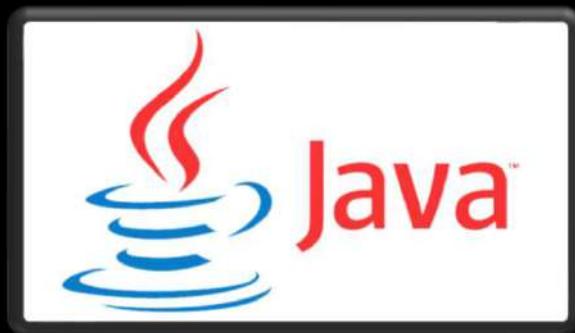
The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** eclipse-workspace - interfaces/src/interfaces/MiJFrame.java - Eclipse IDE
- Menu Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Standard Java development toolbar.
- Left Sidebar:** Shows the project structure with a tree view of files and folders under "src/interfaces".
- Central Editor:** Displays the code for `MiJFrame.java`. The code creates a JFrame and adds two JButton instances, btn1 and btn2, to it. It also adds ActionListener listeners to both buttons to print "Click en BTN1!" or "Click en BTN2!" to the console respectively.

```
private static final long serialVersionUID = 1L;
MiJFrame(){}
 JFrame jframe = new JFrame("Mi primera GUI");
 jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 jframe.setSize(600, 400);
 jframe.setVisible(true);
 // Creando y añadiendo el botón a mi frame
 JButton btn1 = new JButton("Btn1");
 jframe.add(btn1);
 // Cuando hacemos click en btn1 se ejecutará dicho método
 btn1.addActionListener(e -> {
 System.out.println("Click en BTN1!");
 });
 JButton btn2 = new JButton("Btn2");
 jframe.add(btn2);
 // Cuando hacemos click en btn2 se ejecutará dicho método
 btn2.addActionListener(e -> {
 System.out.println("Click en BTN2!");
 });
}
}
```

- Right Sidebar:** Shows the "Console" tab with the message "No consoles to display at this time."

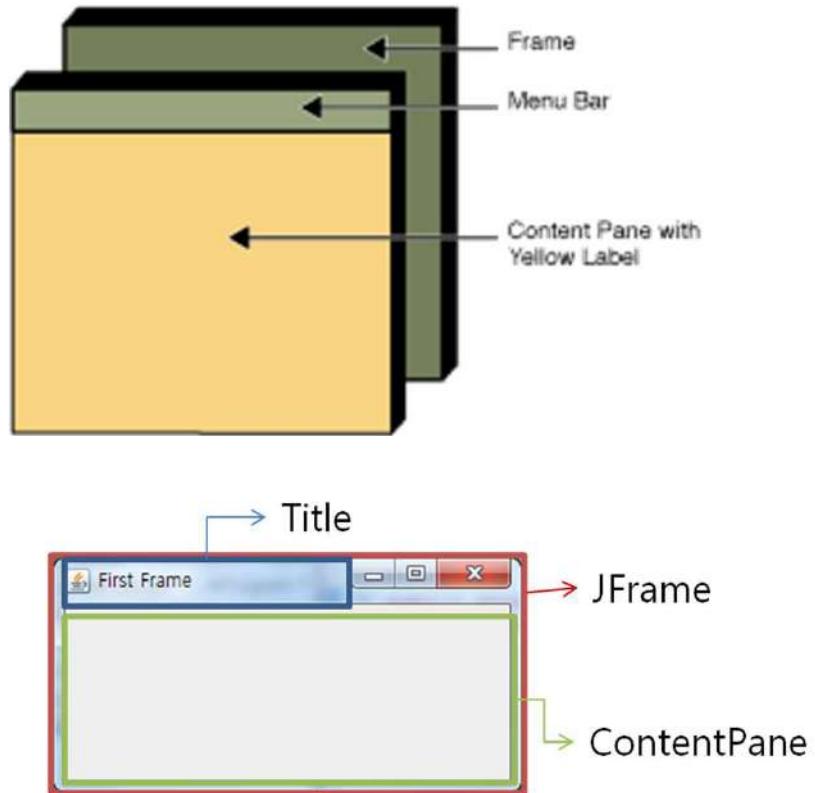
# BOTONES Y EVENTOS



# Estructura de un JFrame

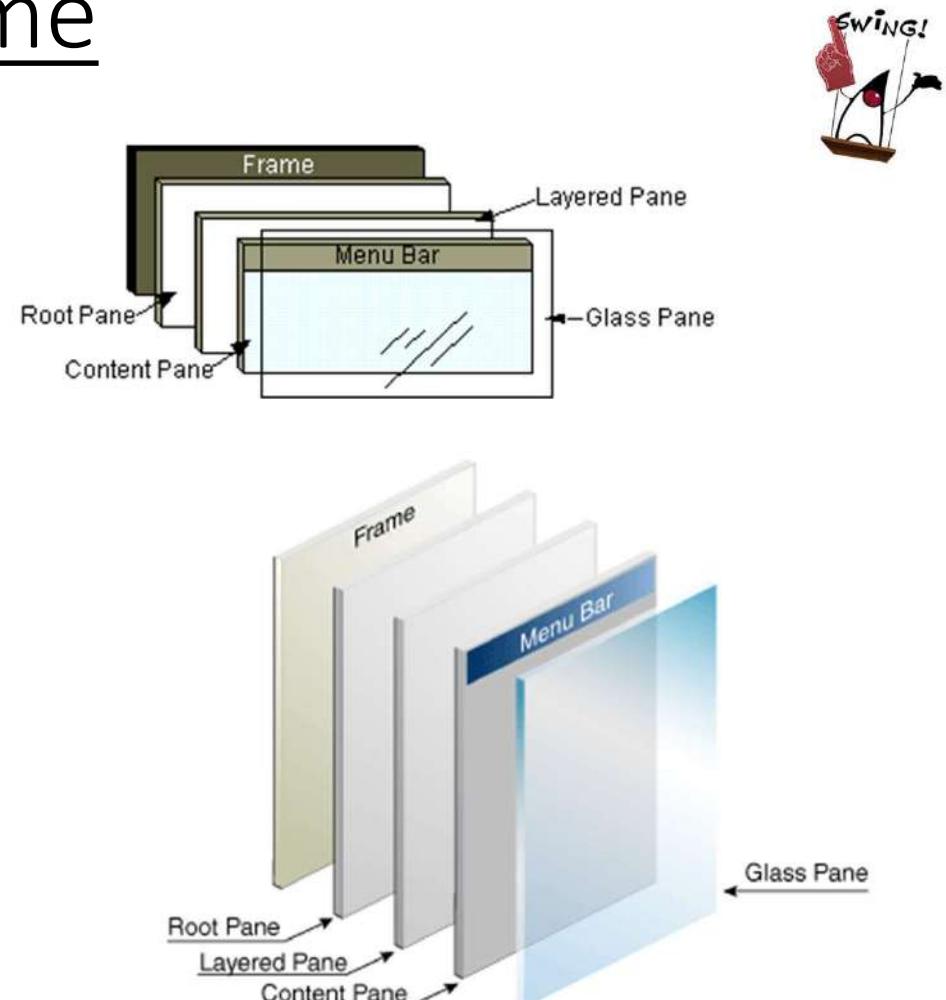


- Un JFrame, está formado principalmente por dos partes:
  - El Menu bar, que es donde se define el título del JFrame, se muestran los botones de maximizar, minimizar y cerrar.
  - El Content Pane, en el que se añadirán los componentes que añadamos a nuestro componente JFrame.
- Aunque podemos añadir componentes directamente en nuestro JFrame, no es la forma más recomendada de hacerlo.

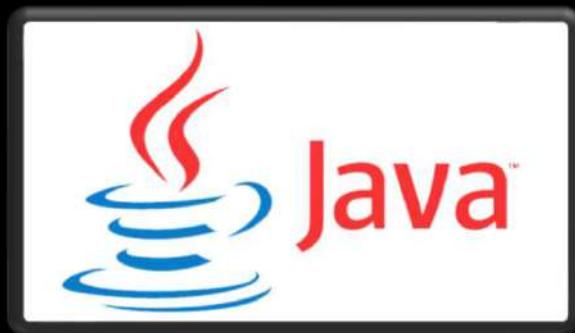


# Estructura de un JFrame

- Lo más recomendable es añadir una especie de capas/laminas, como si de un cristal se tratase.
- JPanel nos permite crear una especie de laminas transparentes



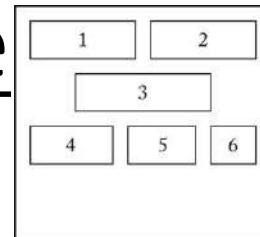
# LAYOUTS



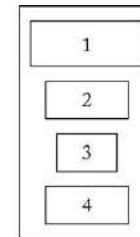
582

# Estructura de un JFrame

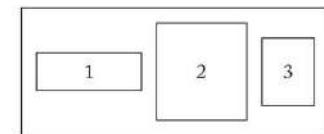
- Los layouts o disposiciones, nos permiten posicionar nuestros elementos dentro de un contenedor por ejemplo un Jframe
  - FlowLayout: es la disposición por defecto. De izquierda a derecha creando una nueva fila cada vez que ya no quepan los elementos actuales.
  - BoxLayout: de izquierda a derecha o de arriba a abajo
  - BorderLayout: divide el contenedor en 5 áreas/zonas: north, south, west y east y center
  - GridLayout: cuadrícula, en la que todos los componentes tienen el mismo tamaño
  - GridBagLayout: complejo, como una tabla HTML
  - Otra opción sería crear nuestro propia disposición de layout.



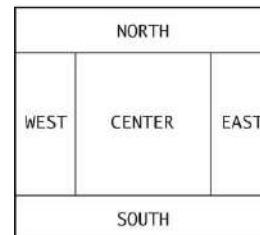
FlowLayout



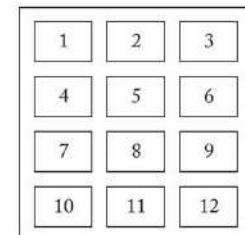
BoxLayout (vertical)



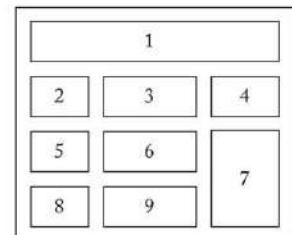
BoxLayout (horizontal)



BorderLayout

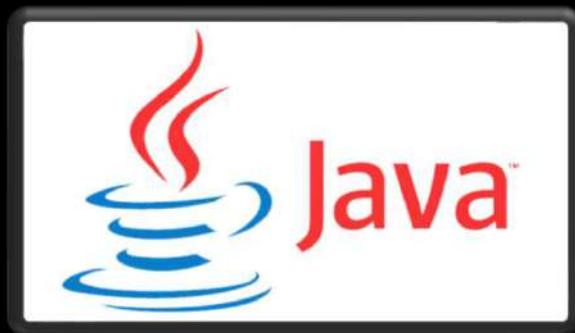


GridLayout



GridBagLayout

# BORDER LAYOUT



584

# Border Layout



- Coloca los elementos en cualquiera de los bordes del contenedor y/o en el centro.

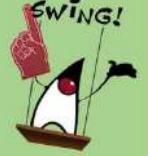
The image shows a Java code editor and a running application window side-by-side.

**Java Code (MiJFrame.java):**

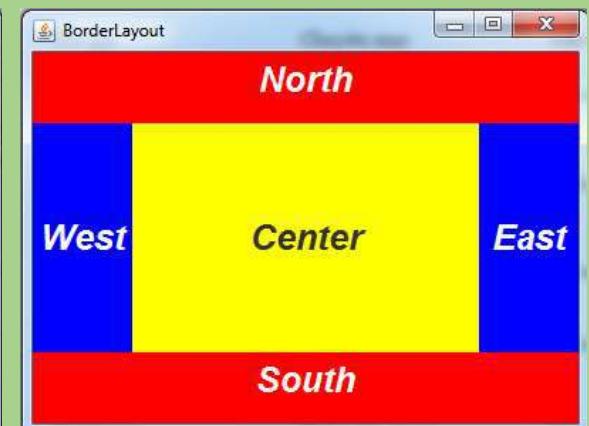
```
1 package interfaces;
2
3 import java.awt.BorderLayout;
4
5 import javax.swing.JButton;
6 import javax.swing.JFrame;
7
8 public class MiJFrame extends JFrame{
9
10 private static final long serialVersionUID = 1L;
11
12 MiJFrame(){
13 setTitle("Layout in action: BorderLayout");
14 setSize(300,300);
15 setVisible(true);
16 setLayout(new BorderLayout());
17 // Creo los elementos que vamos a añadir
18 JButton btn1 = new JButton("Btn 1");
19 JButton btn2 = new JButton("Btn 2");
20 JButton btn3 = new JButton("Btn 3");
21 JButton btn4 = new JButton("Btn 4");
22 JButton btn5 = new JButton("Btn 5");
23 //Añado los elementos a los contenedores
24 add(btn1, BorderLayout.NORTH);
25 add(btn2, BorderLayout.EAST);
26 add(btn3, BorderLayout.CENTER);
27 add(btn4, BorderLayout.WEST);
28 add(btn5, BorderLayout.SOUTH);
29 }
30 }
31 }
```

**Running Application:** A window titled "Layout in action: Bor..." displays five buttons arranged in a BorderLayout. The buttons are labeled "Btn 1" (center), "Btn 2" (east), "Btn 3" (center), "Btn 4" (west), and "Btn 5" (south).

# Ejercicio de Border Layout



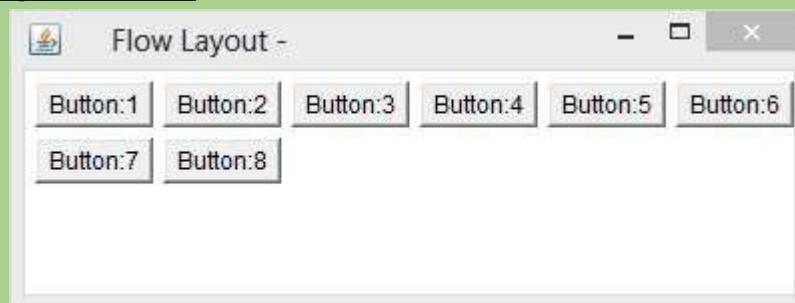
- Clona los siguientes JFrames:
  - Cuando hagas click sobre cualquiera de los elementos, deberá de aparecer un JOptionPane con una alerta mostrando Has pulsado junto a la posición del elemento North, Weest, Center, East o South.



# Ejercicio de Flog layout



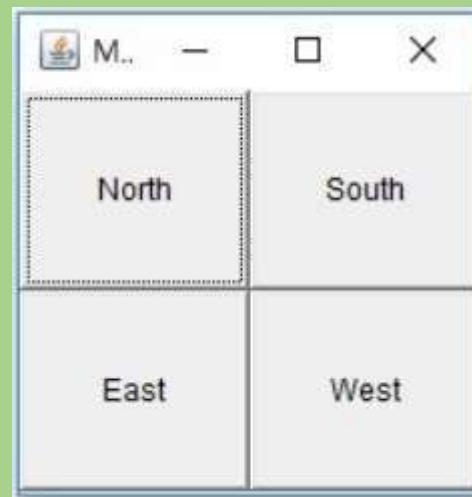
- Clona los siguientes JFrames:
  - Si os fijáis, podeis ver que los botones están alineados de distintas formas. Busca por internet como alinear los botones hacia un lado u otro en un Flog layout



# Ejercicio de Grid Layout



- Clona los siguientes JFrames:
  - Cuando hagas click sobre cualquiera de los elementos, deberá de aparecer un JOptionPane con una alerta mostrando Has pulsado junto a la posición del elemento North, Weest, Center, East o South.



# Ejercicio de Grid Layout

- Ejercicio de hundir la flota:
  - Los barcos deben situarse de forma aleatoria
  - Puedes hacer distintos niveles:
    - 50 % de las casillas (dificultad máxima)
    - 60 % de las casillas (dificultad media)
    - 75% de las casillas (dificultad mínima)



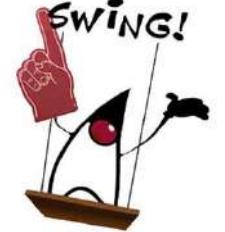
# CONTAINER FLAME



590

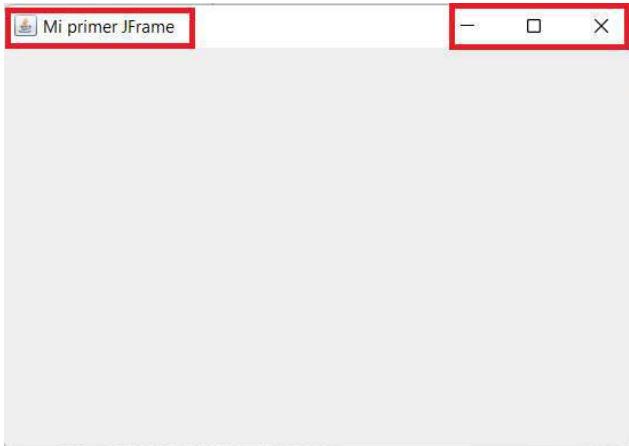
# Container JFrame

- Si nos fijamos, las características de JFrame, podemos observar que este tipo de container, tiene botones de maximizar, minimizar y cerrar. Y también un título para dicha ventana:



Main.java

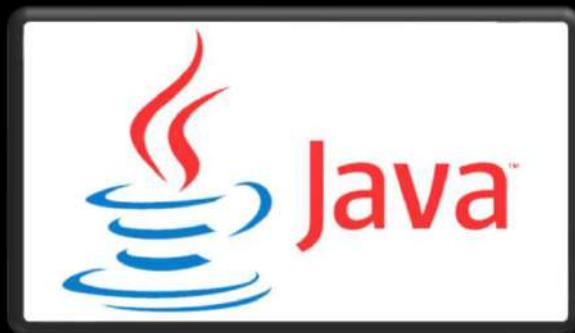
```
1 package interfaces;
2
3 import javax.swing.JFrame;
4
5 public class Main {
6
7 public static void main(String[] args) {
8 JFrame jframe = new JFrame();
9 jframe.setTitle("Mi primer JFrame");
10 jframe.setSize(800, 600);
11 jframe.setLocationRelativeTo(null);
12 jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13 jframe.setVisible(true);
14 }
15 }
```



# **APRENDIENDO A DEBUGEAR**



# ¿POR QUÉ APRENDER A DEBUGGEAR?





**He said, He has just  
"one more Bug" to Fix.**



**I'm still waiting for him**

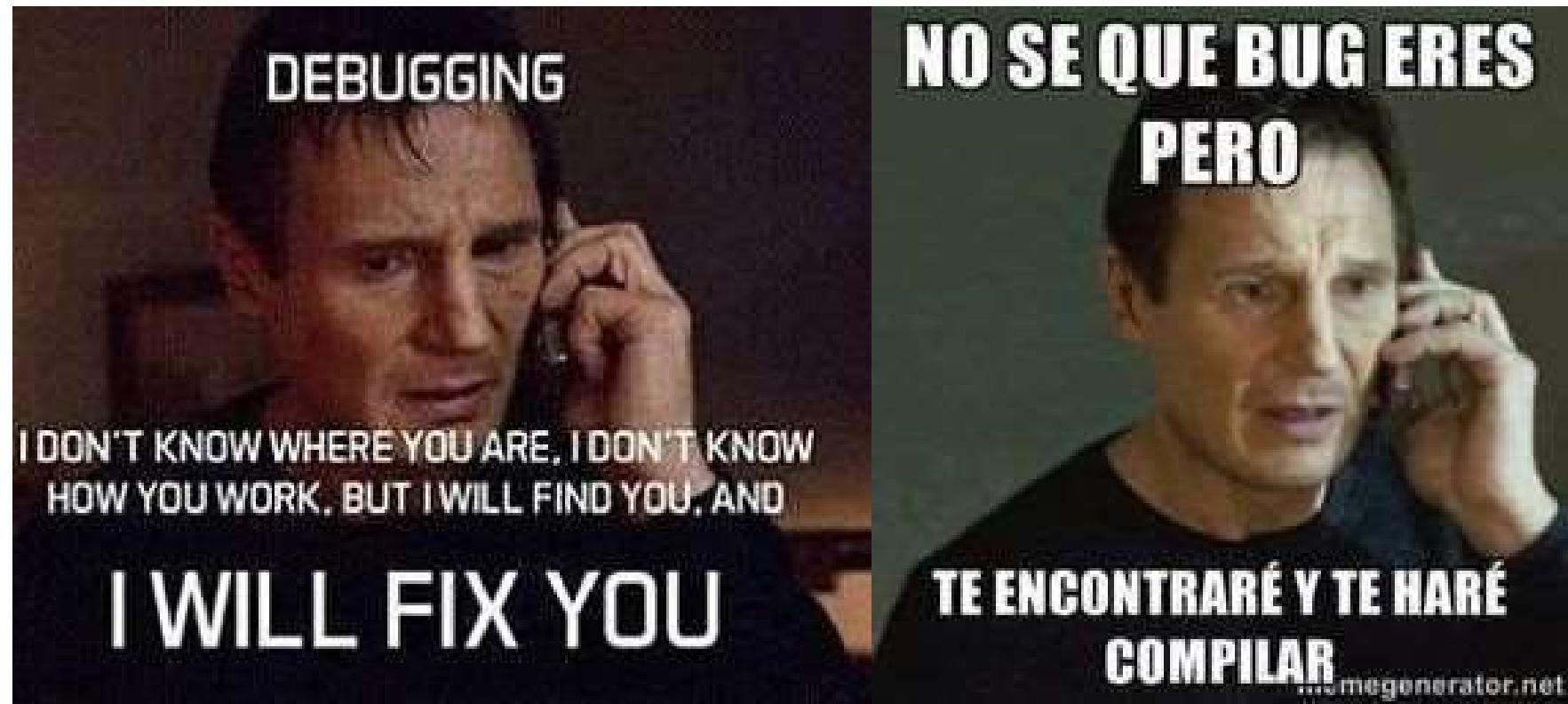
**594**



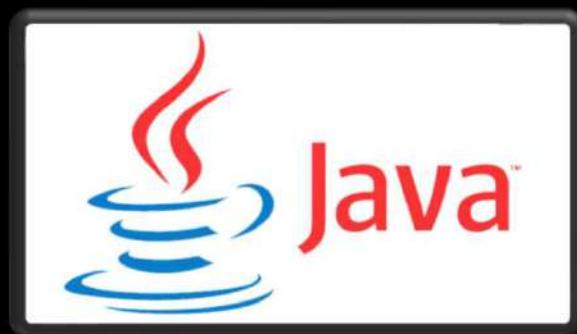
**Looking at  
programming  
memes**



**Actually  
coding**

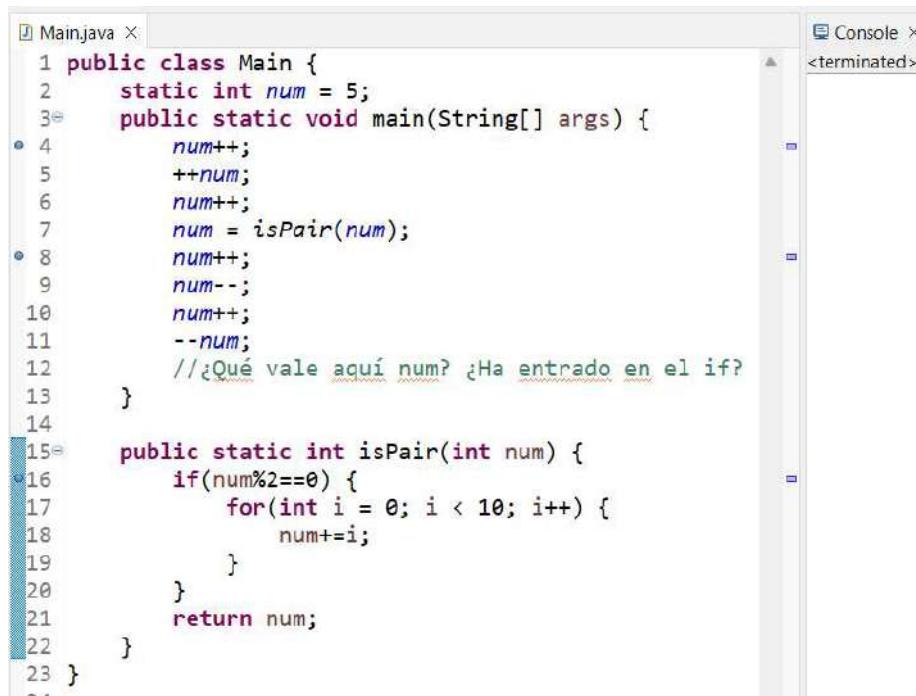


# INTRODUCCIÓN AL DEBUGGER



# Introducción al debugger

- Cuando vemos el siguiente código si queremos saber como avanza el valor de num tenemos que pensar como si fuéramos el código ¿Ha entrado dentro del if? ¿Qué vale num en una determinada instrucción?



The screenshot shows an IDE interface with two windows. On the left is the code editor for `Main.java`, which contains the following Java code:

```
1 public class Main {
2 static int num = 5;
3 public static void main(String[] args) {
4 num++;
5 ++num;
6 num++;
7 num = isPair(num);
8 num++;
9 num--;
10 num++;
11 --num;
12 //¿Qué vale aquí num? ¿Ha entrado en el if?
13 }
14
15 public static int isPair(int num) {
16 if(num%2==0) {
17 for(int i = 0; i < 10; i++) {
18 num+=i;
19 }
20 }
21 return num;
22 }
23 }
```

The code editor has line numbers on the left and syntax highlighting. A comment at line 12 is underlined with a red dotted line, indicating it is being analyzed or is a point of interest. On the right is a "Console" window titled "Console X" with the message "<terminated> N".



599

# Introducción al debugger

- En muchas ocasiones, es habitual que la persona que no sabe debuggear escriba mensajes de consola/logs constantemente en el que imprimirá por ejemplo los valores de la variable:
- El principal problema es esto es que luego habrá que remover dichos mensajes manualmente uno a uno, y posiblemente tengamos que añadirlos en muchas clases, etc.
- Además, existen determinadas acciones que son mucho más difíciles o más lentas de realizar con mensajes que con debug.

The screenshot shows an IDE interface with two main panes. On the left is the code editor for 'Main.java' containing the following Java code:

```
1 public class Main {
2 static int num = 5;
3 public static void main(String[] args) {
4 num++;
5 System.out.println(num);
6 ++num;
7 System.out.println(num);
8 num++;
9 System.out.println(num);
10 num = isPair(num);
11 System.out.println(num);
12 num++;
13 System.out.println(num);
14 num--;
15 System.out.println(num);
16 num++;
17 System.out.println(num);
18 --num;
19 //¿Qué vale aquí num? ¿Ha entrado en el if?
20 System.out.println(num);
21 }
22
23 public static int isPair(int num) {
24 if(num%2==0) {
25 for(int i = 0; i < 10; i++) {
26 num+=i;
27 System.out.println(num);
28 }
29 }
30 return num;
31 }
32 }
```

On the right is the 'Console' pane showing the output of the program:

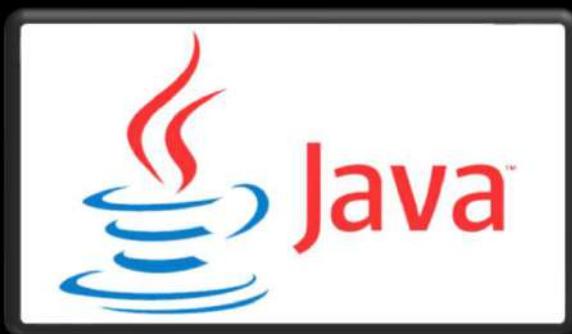
```
6
7
8
8
9
11
14
18
23
29
36
44
53
53
54
53
54
53
```

A tooltip is visible in the code editor at line 19, reading: //¿Qué vale aquí num? ¿Ha entrado en el if?



601

# ¿QUÉ ES EL FLUJO DE UN PROGRAMA?



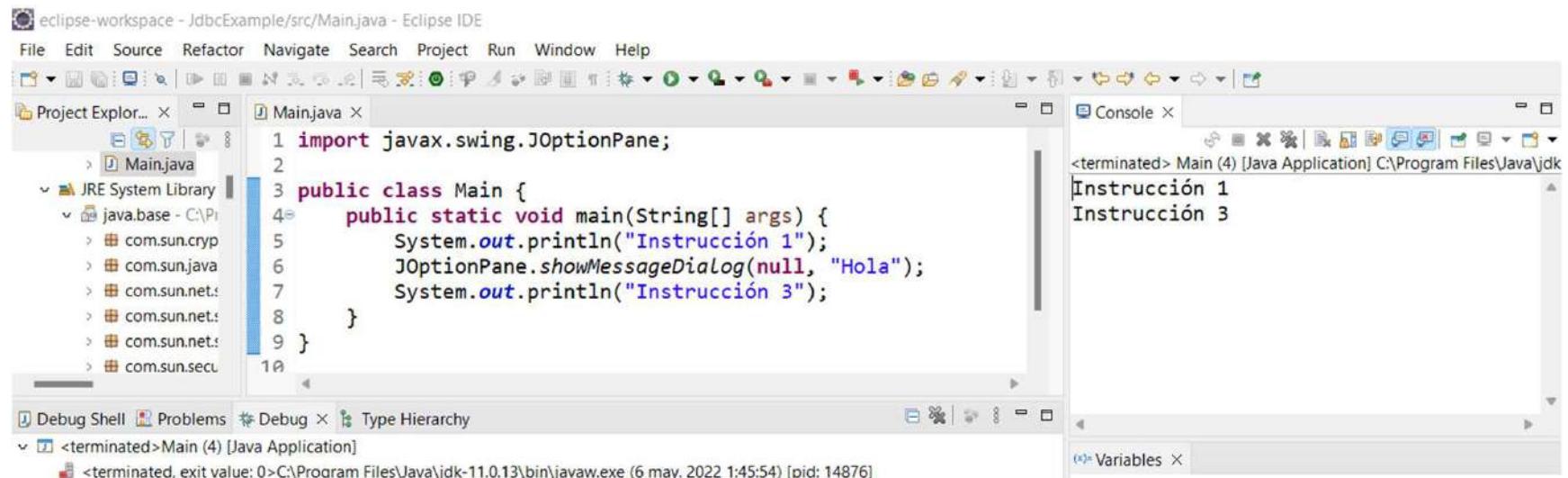
# ¿Qué es el flujo de un programa?

- El flujo de ejecución de un programa es la manera que tiene Java, concretamente, la JVM (Java Virtual Machine) de ejecutar nuestro código.
- En algunas ocasiones, existen determinadas instrucciones que detienen dicho flujo de ejecución. Vamos a ver un ejemplo:



# ¿Qué es el flujo de un programa?

- Si nos fijamos, las instrucciones que habitualmente bloquean el flujo de ejecución de un programa son las que necesitan iteración con el usuario.

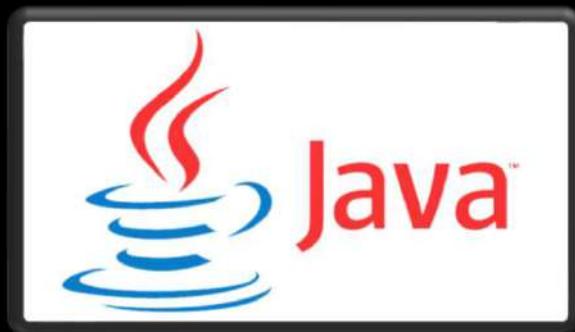


The screenshot shows the Eclipse IDE interface with a Java application running. The Project Explorer view shows a single file named Main.java. The code within Main.java is:

```
1 import javax.swing.JOptionPane;
2
3 public class Main {
4 public static void main(String[] args) {
5 System.out.println("Instrucción 1");
6 JOptionPane.showMessageDialog(null, "Hola");
7 System.out.println("Instrucción 3");
8 }
9 }
10
```

The Console view displays the output of the program: "Instrucción 1" and "Instrucción 3". The Debug Shell and Variables views are also visible at the bottom of the interface.

# ¿QUÉ ES EL DEBUGGER?

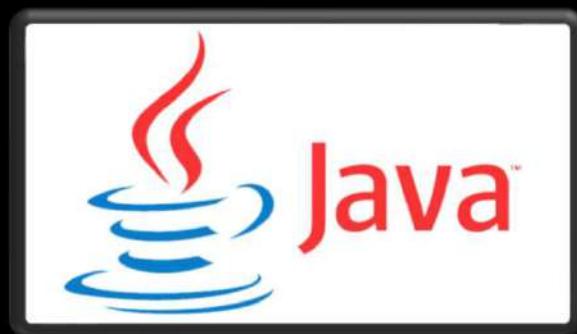


# ¿Qué es el debugger?

- El flujo de un programa empieza en la primera línea del archivo que contenga el método Main. Y, por defecto, va pasando por cada una de las instrucciones instrucción a instrucción aunque el camino se puede ver alterado si utilizamos estructuras de control condicionales, bucles, funciones, etc.
- Mediante al debug podemos parar la ejecución de nuestro programa e ir instrucción a instrucción, es decir, paso a paso por viendo que hace nuestro programa en todo momento. Que caminos coge, que valor tiene una variable, etc.

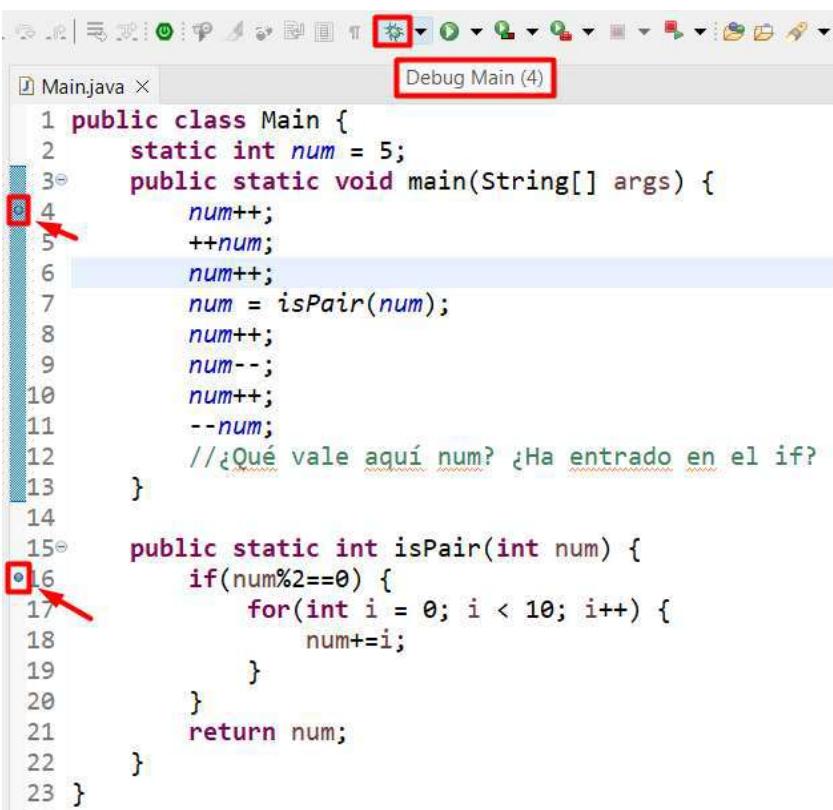
```
Main.java X
1 public class Main {
2 static int num = 5;
3 public static void main(String[] args) {
4 num++;
5 ++num;
6 num++;
7 num = isPair(num);
8 num++;
9 num--;
10 num++;
11 --num;
12 //¿Qué vale aquí num? ¿Ha entrado en el if?
13 }
14
15 public static int isPair(int num) {
16 if(num%2==0) {
17 for(int i = 0; i < 10; i++) {
18 num+=i;
19 }
20 }
21 return num;
22 }
23 }
```

# ¿QUÉ ES UN BREAKPOINT?



# ¿Qué es un breakpoint?

- Para detener el flujo de ejecución del programa, bastará con por ejemplo poner un breakpoints (puntos de ruptura) en nuestro programa.
- Para ello, simplemente hacemos doble click en el punto y finalmente, una vez tengamos el breakpoint marcado, ejecutamos el programa en modo debug (para poder el IDE entender que queremos ejecutar el programa en modo debugger)

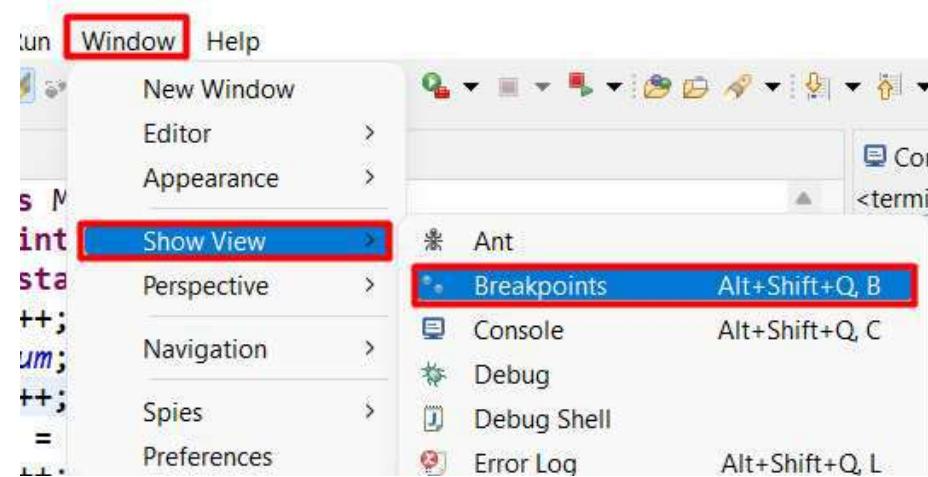


The screenshot shows a Java code editor with the file 'Main.java' open. The code defines a class 'Main' with a static variable 'num' set to 5, and a main method that increments 'num' four times, checks if it's even, and then increments it again. A comment at the end asks what 'num' is. Two breakpoints are set: one at line 4 and another at line 16. The line 16 breakpoint has a red arrow pointing to it. The status bar at the bottom of the IDE window shows 'Debug Main (4)', indicating the current state of the program.

```
1 public class Main {
2 static int num = 5;
3 public static void main(String[] args) {
4 num++;
5 ++num;
6 num++;
7 num = isPair(num);
8 num++;
9 num--;
10 num++;
11 --num;
12 //¿Qué vale aquí num? ¿Ha entrado en el if?
13 }
14
15 public static int isPair(int num) {
16 if(num%2==0) {
17 for(int i = 0; i < 10; i++) {
18 num+=i;
19 }
20 }
21 return num;
22 }
23 }
```

# ¿Qué es un breakpoint?

- Cuando trabajamos con breakpoints, es posible que tengamos que movernos por distintas clases y que tengamos muchos puntos de ruptura. Por ello, se aconseja siempre tener abierta la pestaña de breakpoints:

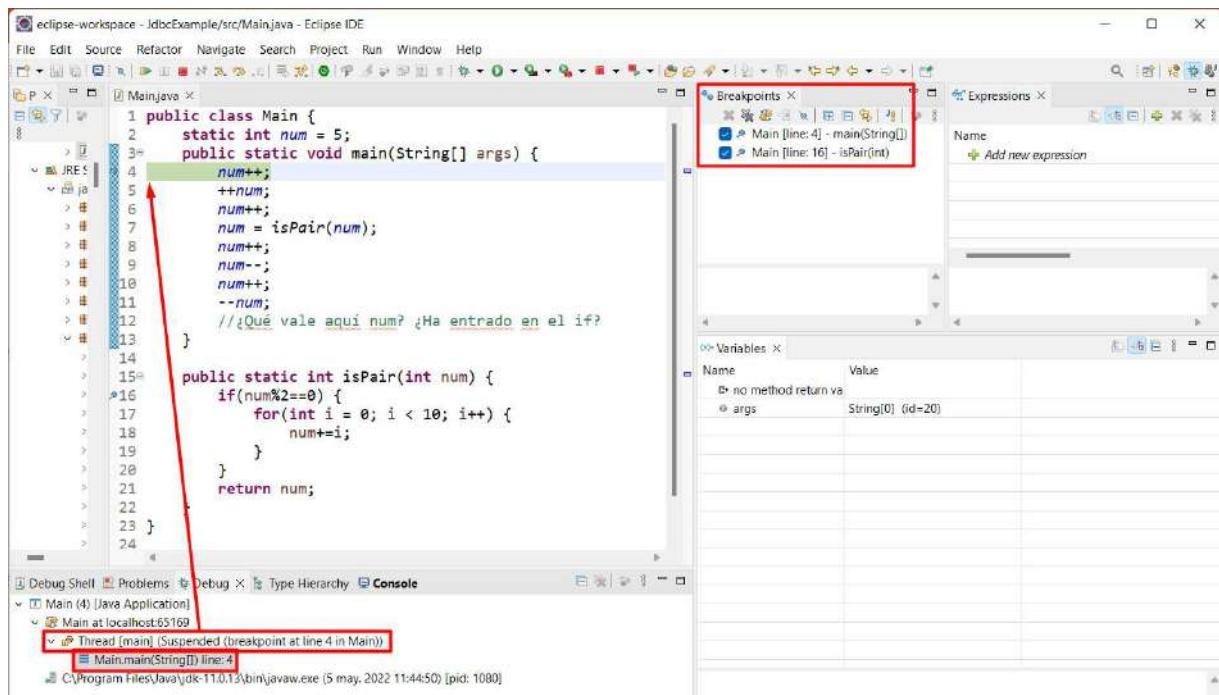


# INSTRUCCIÓN QUE VA A SER DEBUGEADA



# ¿Por qué se pone en verde una instrucción?

- Una vez arrancado nuestro programa con el modo debuger, el debug mode buscará el primer breakpoint y se detendrá en dicho punto.
- Dicha instrucción aún no ha sido ejecutada, pero será la siguiente en ser ejecutada.
- En el lateral podemos ver que, actualmente, tenemos dos breakpoints en nuestro programa:



# **RESUME (F8)**

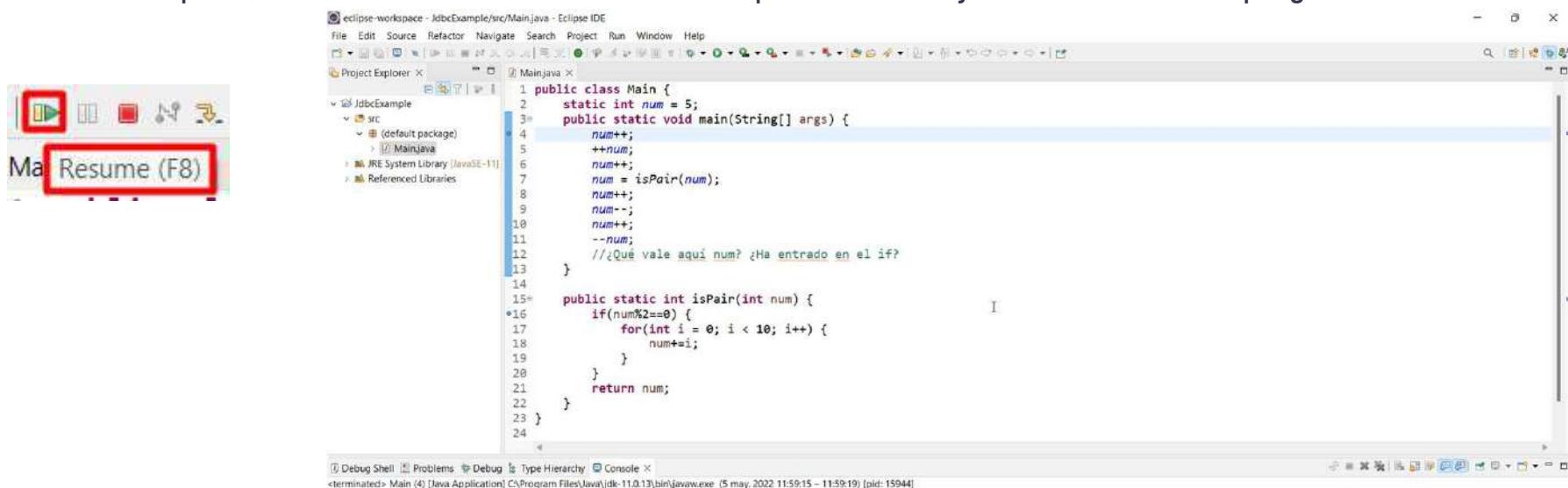


**612**

# Debugger: Resume (F8)

Para movernos entre instrucciones tenemos varias opciones. Comenzamos con la primera:

- Con F8 (resume), continuamos la ejecución “normal”, O bien hasta que nos encontremos con el siguiente breakpoint, o en el caso de no tener más breakpoints con la ejecución normal del programa.



# TERMINATE (CTRL+F2)



614

## Debugger: Terminate (Ctrl+F2)

- Con CONTROL+F2 (Terminate), terminamos la ejecución actual del programa. Por lo que dejamos de ejecutar el flujo del programa en el punto en el que estamos.
- Hay que tener cuidado con esto, ya que si suponemos que nuestro programa por ejemplo gestionase ficheros y conforme va realizando determinadas acciones en dichos ficheros, estos se van moviendo a diferentes , es posible que si detenemos el flujo no se acabe de procesar.



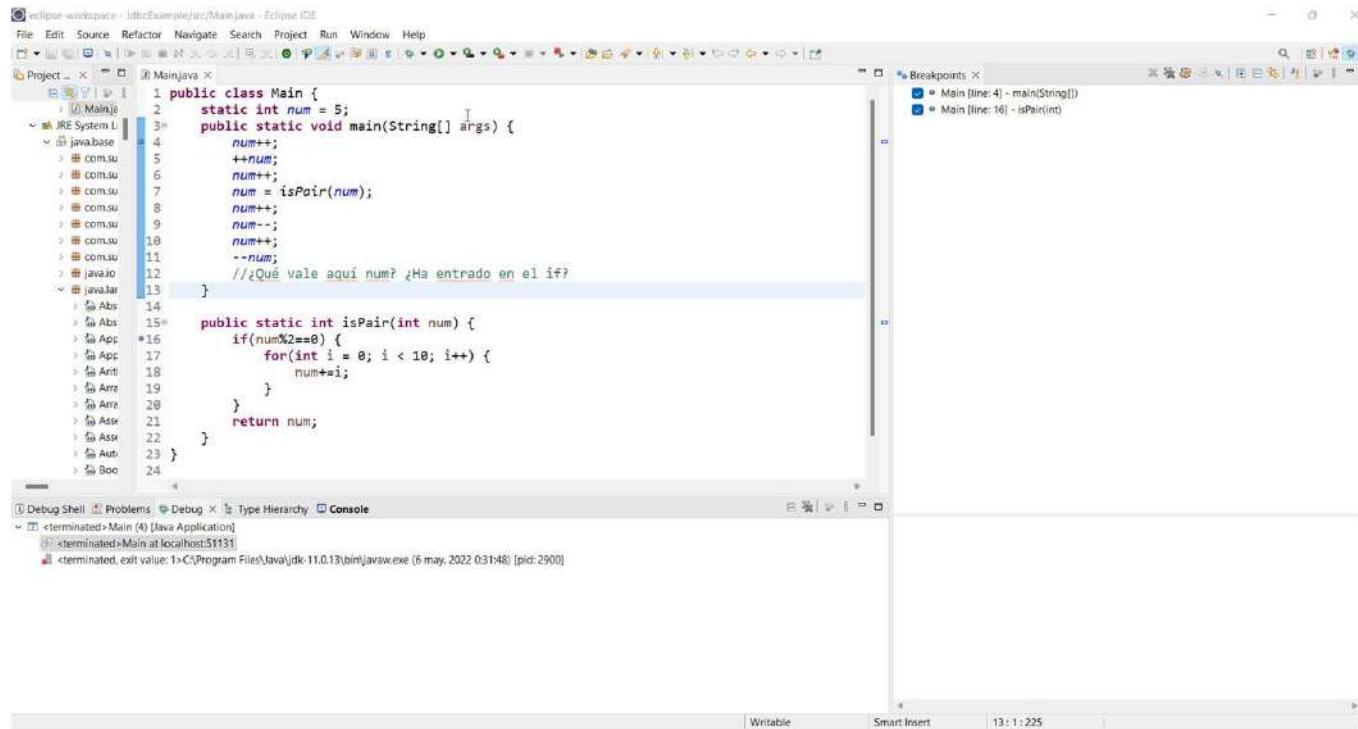
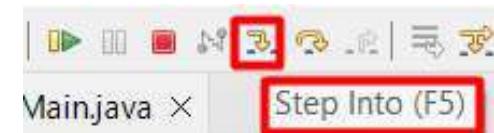
# STEP INTO (F5)



616

# Debugger: Step Into (F5)

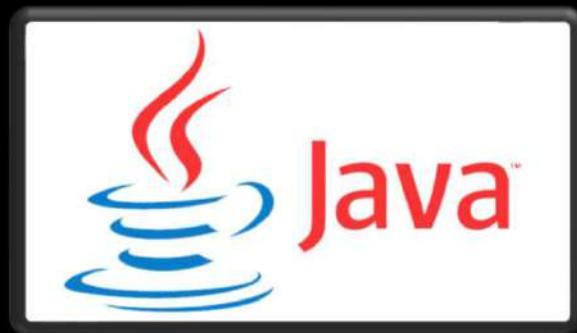
- Con F5 (Step Into), podemos entrar en las profundidades de nuestro código, lo que incluirá incluso las clases internas que utiliza Java para realizar determinadas acciones.  
Vamos a ver un ejemplo:



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with a file named Main.java selected.
- Code Editor:** Displays the Main.java code. A cursor is at line 13, which contains a comment: `//¿Qué vale aquí num? ¿Ha entrado en el if?`. Lines 4 and 16 have blue squares indicating they are breakpoints.
- Breakpoints View:** Shows two breakpoints: one at Main [line: 4] - main(String[]) and another at Main [line: 16] - isPair(int).
- Status Bar:** Shows "terminated" and "terminated, exit value: 1> C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (6 may, 2022 03:14:48) [pid: 2900]".

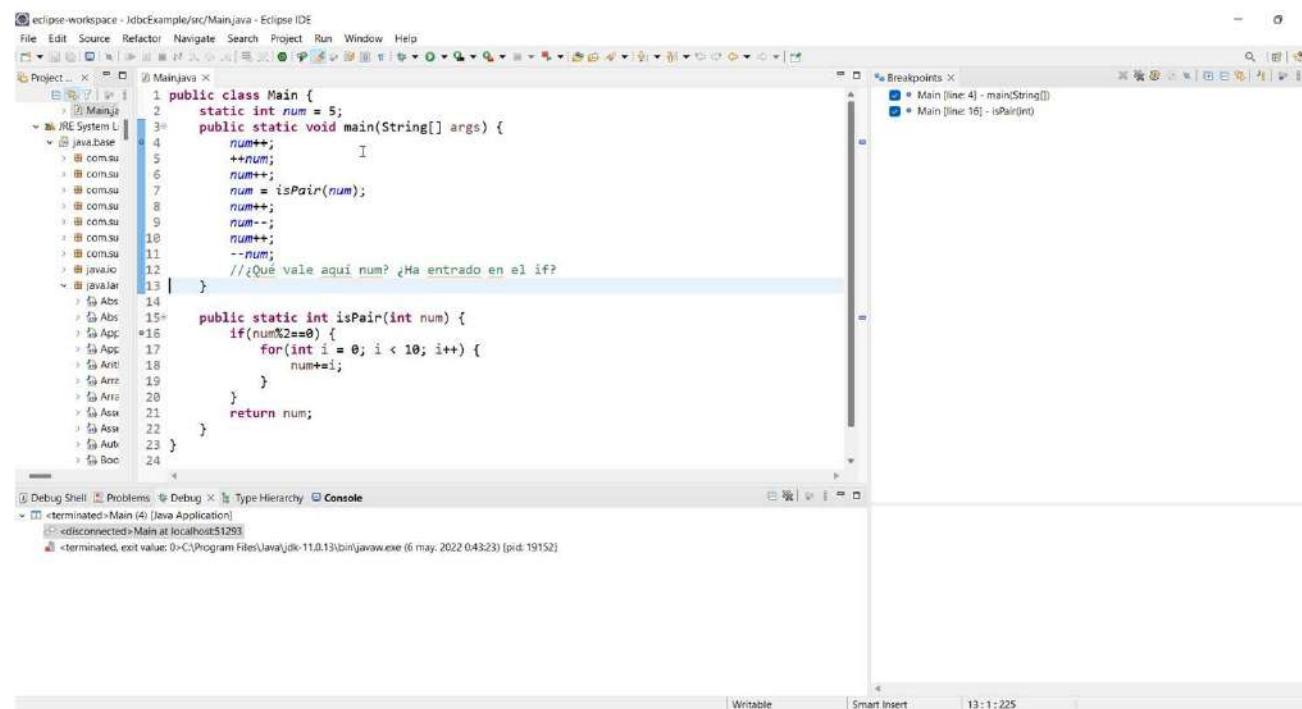
# STEP OVER (F6)



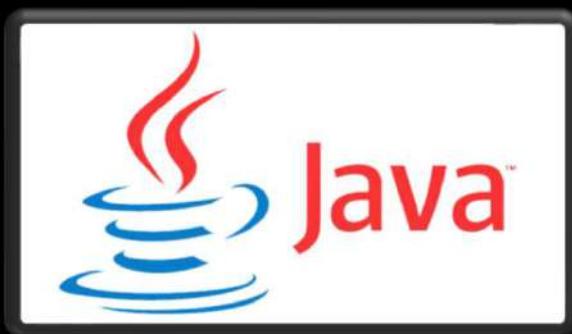
618

# Debugger: Step Over (F6)

- Con F6 (Step Over), podemos pasar por nuestras instrucciones sin pasar por las instrucciones que internamente utiliza Java para ejecutar las instrucciones de nuestro programas. Vamos a ver un ejemplo:

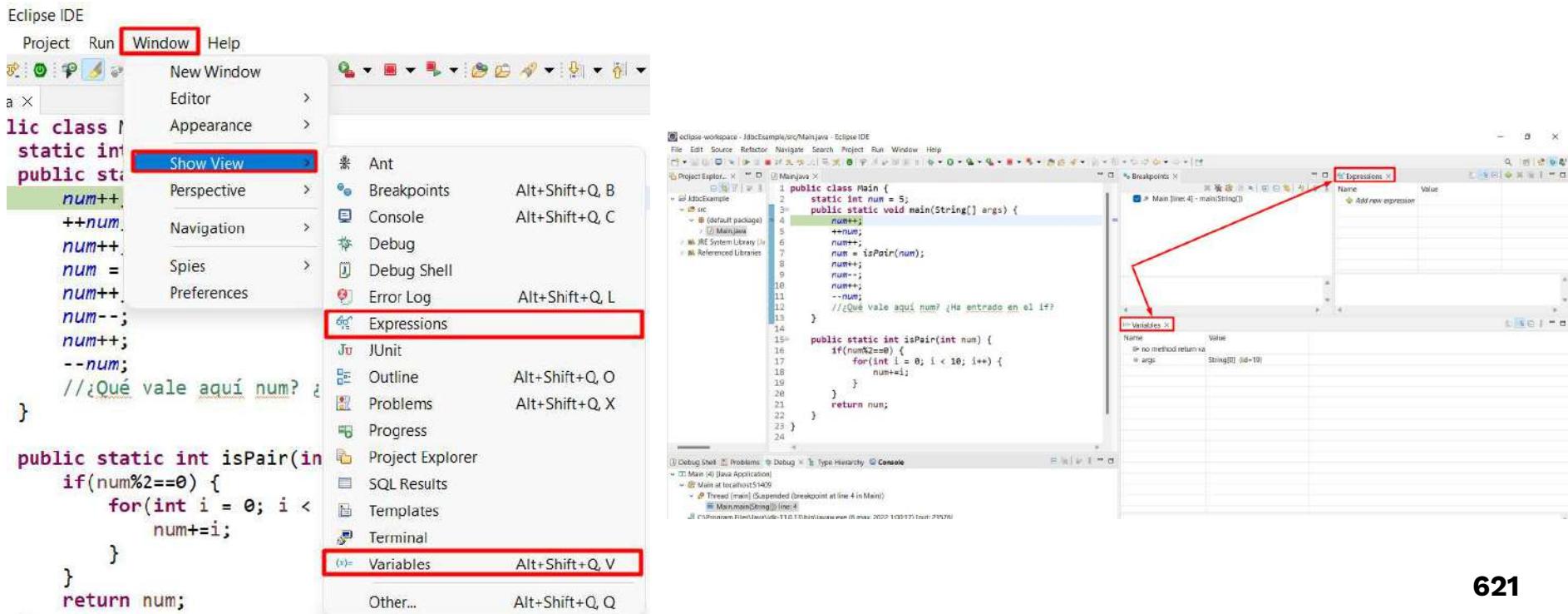


# VARIABLES Y EXPRESIONES



# Debugger: Variables y expresiones

- Bueno, ahora que ya sabemos movernos por el debugger, vamos a ver unos truquitos.  
Primeramente vamos a marcar Window>Show View y marcamos Expressions y variables

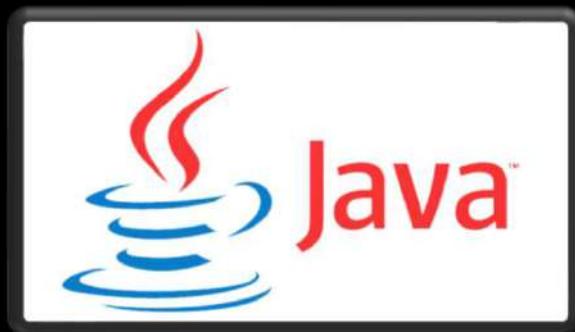


# Debugger: Variables y expresiones

- Con esto, podemos analizar expresiones y ver los valores de nuestras variables. Vamos a ver un ejemplo:

The screenshot shows the Eclipse IDE interface during a Java application debug session. The main window displays the code for a `Main.java` file:public class Main { static int num = 5; public static void main(String[] args) { num++; ++num; num++; num = isPair(num); num++; num--; num++; --num; //¿Qué vale aquí num? ¿Ha entrado en el if? } public static int isPair(int num) { if(num%2==0) { for(int i = 0; i < 10; i++) { num+=i; } } return num; }}The code editor highlights line 15, which contains the call to `isPair(num)`. The `Breakpoints` view shows a single breakpoint at line 4 of the `main` method. The `Expressions` view is open, showing a table with one row: `Main [line: 4] - main(String[])`. The `Variables` view is also visible. The `Debug Shell` view shows the application has terminated. The bottom right corner of the interface has the number **622**.

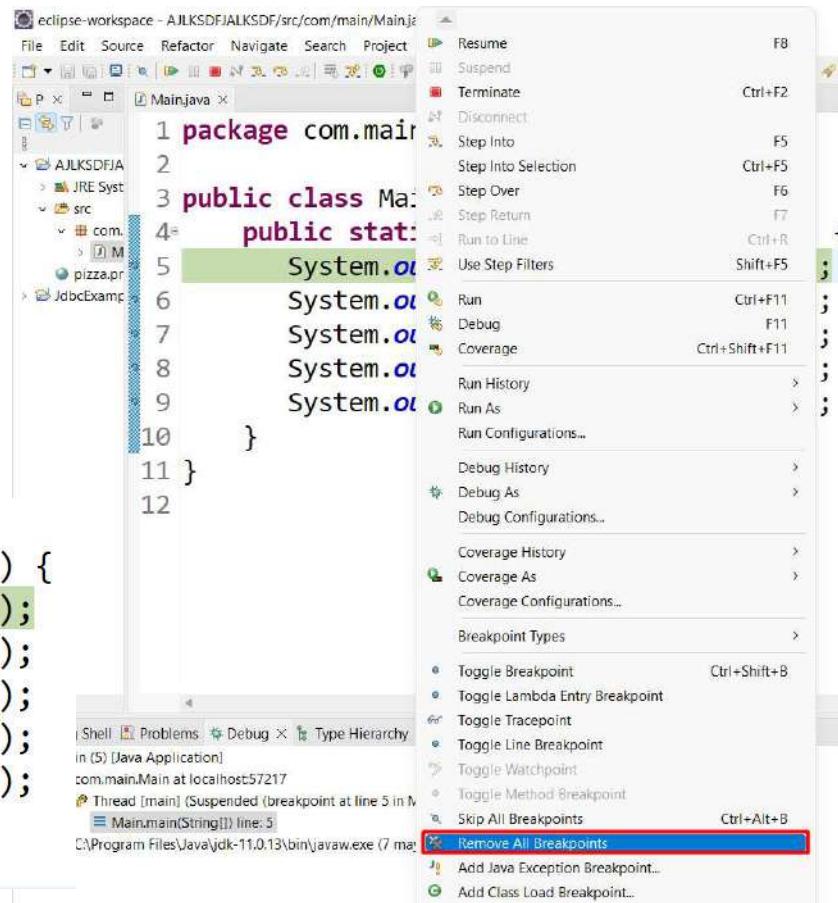
# ¿CÓMO ELIMINO LOS BREAKPOINTS?



# Debugger: Variables y expresiones

- Si tengo muchos breakpoints en muchas clases, el restirar dichos breakpoints puede ser un proceso bastante engorroso. Como Eclipse lo sabe nos ofrece la posibilidad de ir a Run>Remove all Breakpoints de la siguiente manera:

```
3 public class Main {
4 public static void main(String[] args) {
5 System.out.println("Instrucción 1");
6 System.out.println("Instrucción 2");
7 System.out.println("Instrucción 3");
8 System.out.println("Instrucción 4");
9 System.out.println("Instrucción 5");
10 }
11 }
```



# Debugger: Variables y expresiones

- Aunque también podría retirarlos uno a uno haciendo click junto al numerito o bien desde el apartado de breakpoints:

```
3 public class Main {
4 public static void main(String[] args) {
5 System.out.println("Instrucción 1");
6 System.out.println("Instrucción 2");
7 System.out.println("Instrucción 3");
8 System.out.println("Instrucción 4");
9 System.out.println("Instrucción 5");
10 }
11 }
```

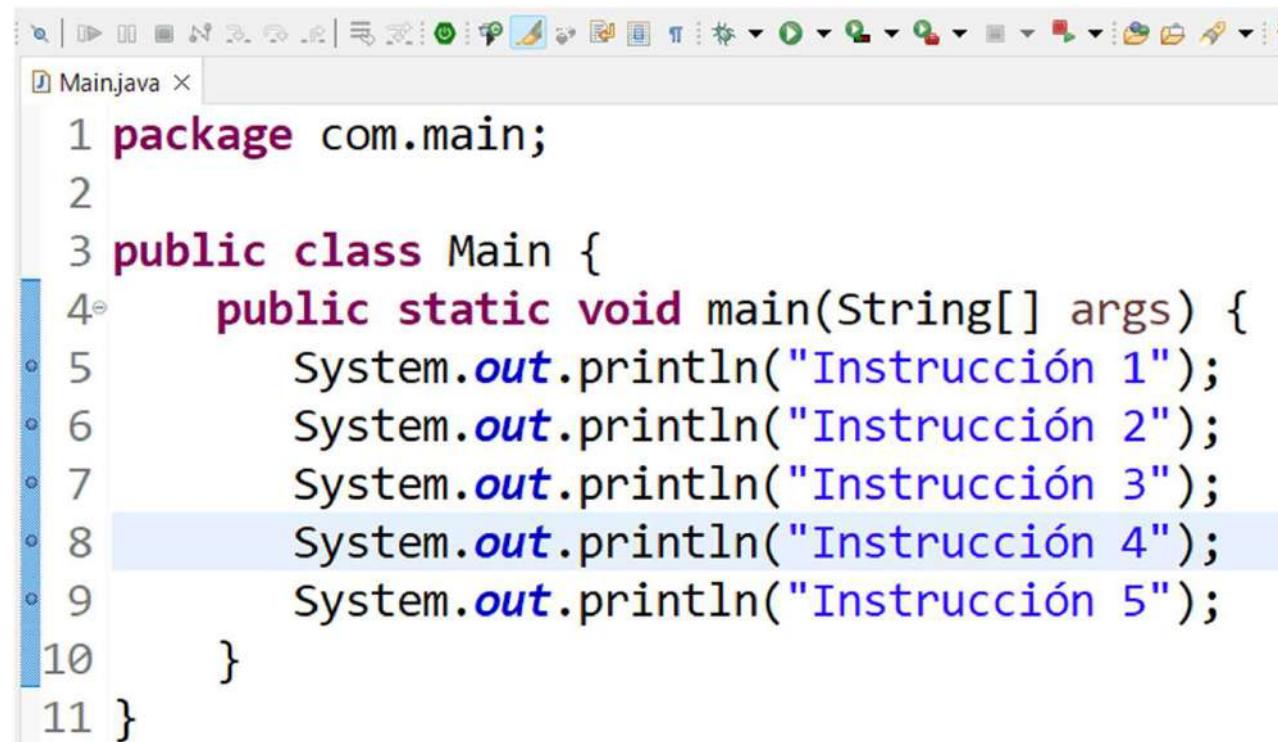


# ¿CÓMO DESABILITO UN BREAKPOINT DE FORMA TEMPORAL?



# Debugger: breakpoint disabled temporalmente

- También existe la posibilidad de deshabilitar un breakpoint temporalmente. Vamos a ver un ejemplo:



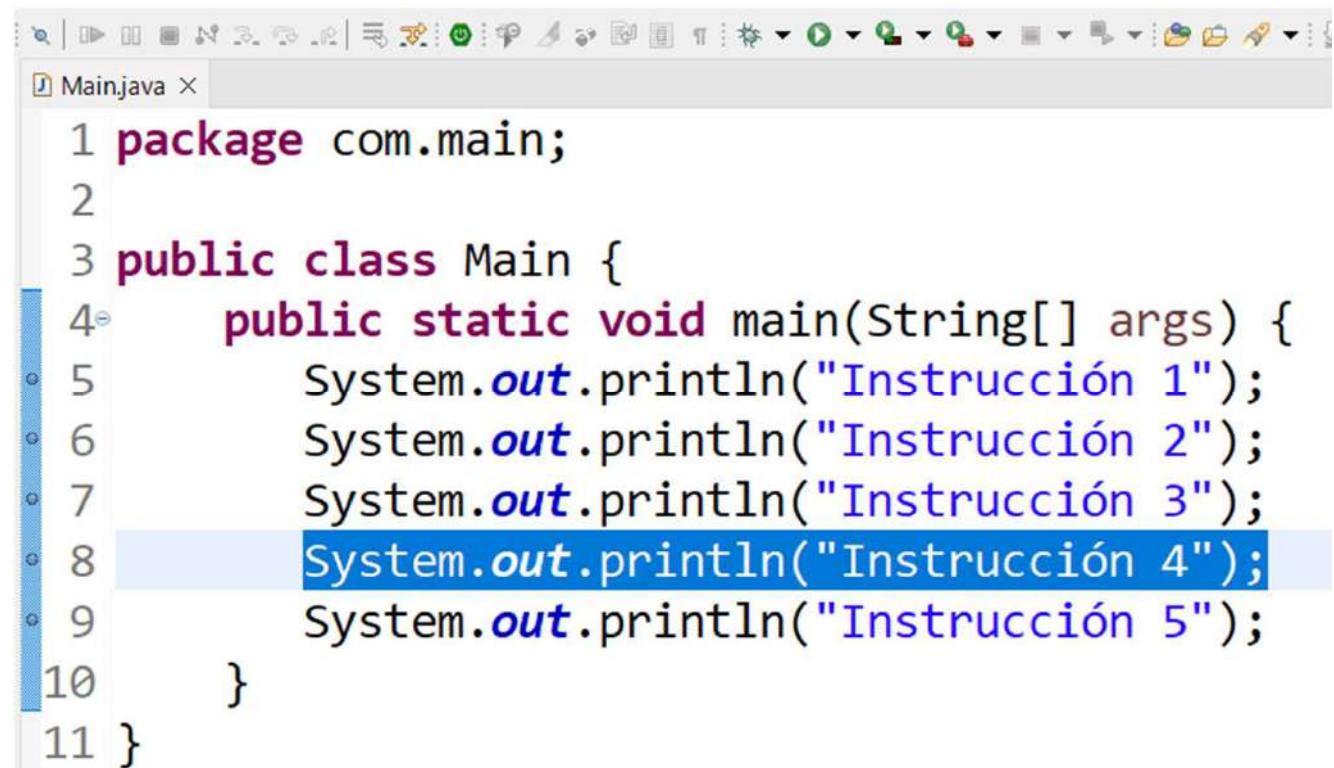
```
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 System.out.println("Instrucción 1");
6 System.out.println("Instrucción 2");
7 System.out.println("Instrucción 3");
8 System.out.println("Instrucción 4");
9 System.out.println("Instrucción 5");
10 }
11 }
```

# ¿CÓMO “SALTAR/SILENCIAR” TODOS BREAK POINTS?



# ¿Cómo modificar valores debugenado?

- Si estamos debugenado un código en un cierto punto queramos continuar con el flujo de ejecución de nuestro código sin perder los breakpoints. Para ello, podemos “silenciar/saltar” los breakpoints de la siguiente manera:



```
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 System.out.println("Instrucción 1");
6 System.out.println("Instrucción 2");
7 System.out.println("Instrucción 3");
8 System.out.println("Instrucción 4");
9 System.out.println("Instrucción 5");
10 }
11 }
```

# ¿CÓMO MODIFICAR EL VALOR DE UNA VARIABLE CUANDO ESTAMOS DEBUGENADO?



# ¿Cómo salto break points?

- Existe un pequeño “TIP” que nos proporciona la posibilidad de modificar el valor actual de una variable durante la ejecución del debugger.  
“Hardodeandolo” (es decir, introduciendo a mano el valor). Vamos a ver un ejemplo:

The screenshot shows the Eclipse IDE interface. In the top-left, there's a project tree with a single file named 'Main.java'. The code editor window displays the following Java code:

```
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 String txt = "Java desde 0";
6 if (txt.length() < 10) {
7 System.out.println(txt + " Es menor que 10");
8 } else if (txt.length() > 10) {
9 System.out.println(txt + " Es mayor o igual que 10");
10 }
11 }
12}
```

The line of code at position 7 is highlighted in blue, indicating it is the current line of execution. Below the code editor, the 'Debug Shell' view shows the output of the application's execution:

```
<terminated> Main (S) [Java Application]
<terminated> <terminated>com.main.Main at localhost:57749
<terminated, exit value: 1> C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (7 may, 2022 13:45:01) [pid: 23348]
```

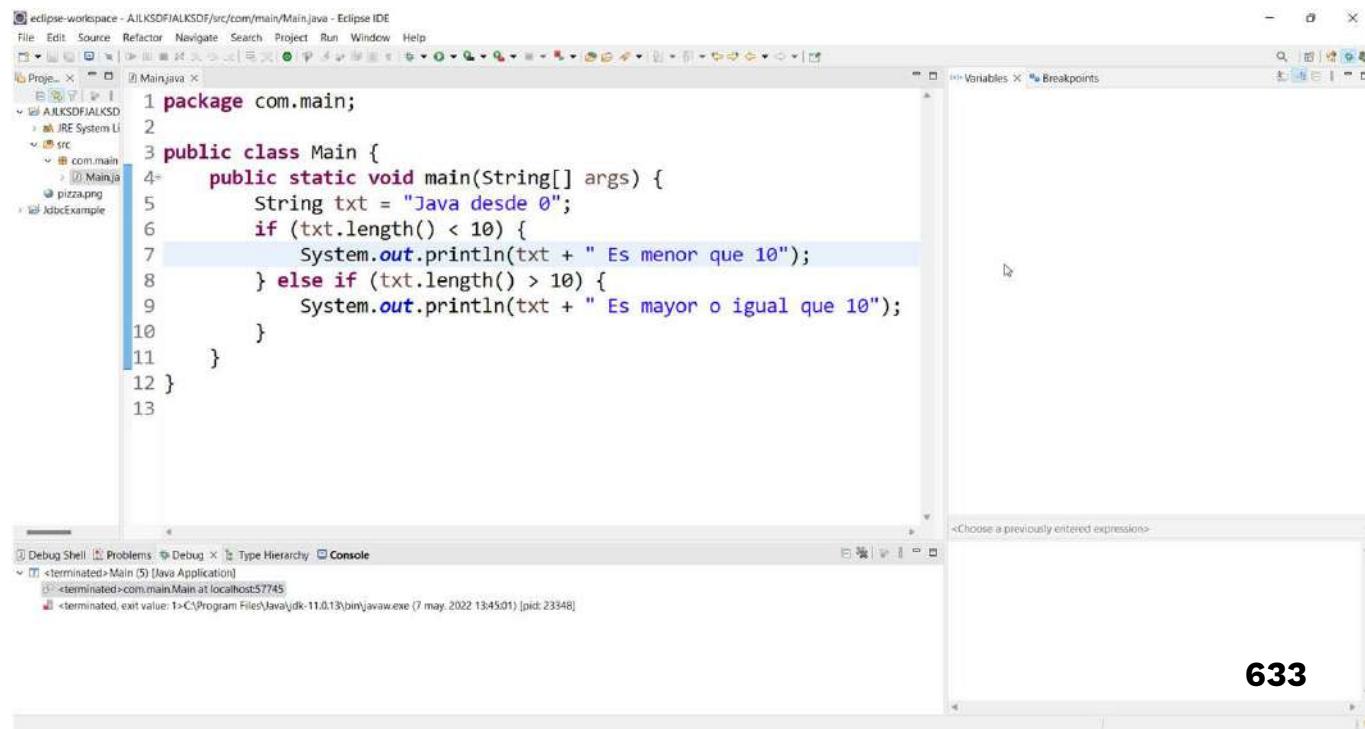
# ¿CÓMO DEJAR TRAZAS DE LAS CLASES POR LAS QUE PASAMOS CON EL DEBUG?



# Toggle trace points

- Existe un pequeño “TIP” que nos proporciona la posibilidad de modificar el valor actual de una variable durante la ejecución del debugger.  
“Hardodeandolo” (es decir, introduciendo a mano el valor). Vamos a ver un ejemplo:

El toggle trace point, nos marca si pasamos o no por dicho punto pero no nos define el flujo de ejecución:



# PATRONES DE DISEÑO

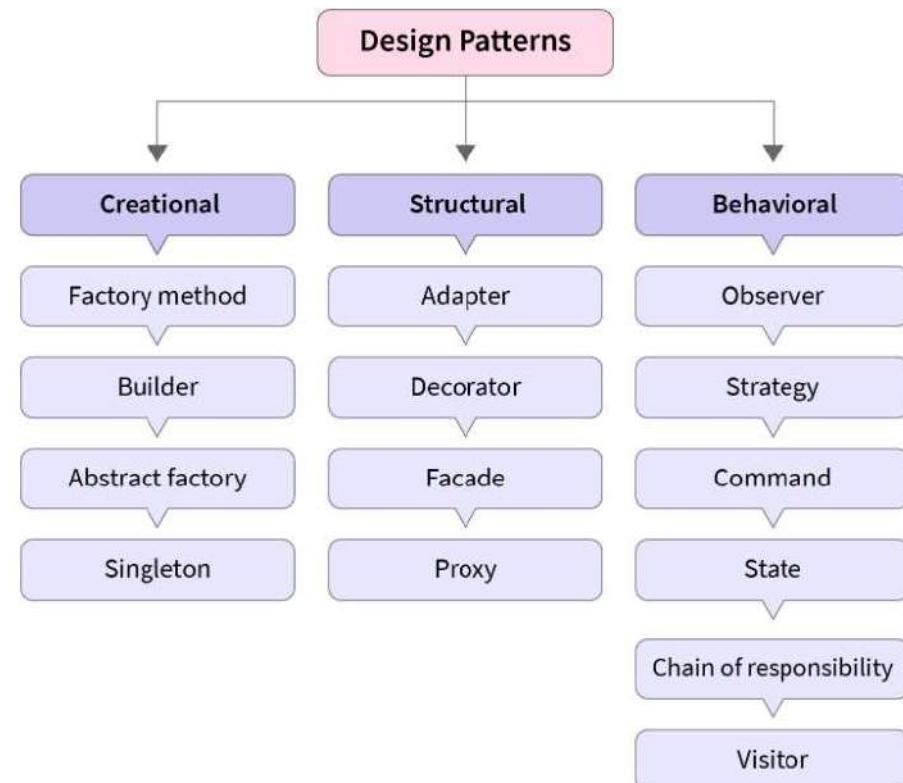


# ¿QUÉ SON LOS PATRONES DE DISEÑO?



# ¿Qué son los patrones de diseño?

- Los patrones de diseño son soluciones habituales a problemas que ocurren con frecuencia en el diseño de software.
- Los patrones de diseño se suelen subdividir en 3 categorías:
  - **Creacionales:** Su finalidad es la inicialización y configuración de objetos
  - **Estructurales:** Van a separar la interfaz de la implementación ocupándose de como se pueden agrupar las clases y los objetos para formar estructuras más grandes
  - **De comportamiento:** Van a describir a los objetos y clases que están implicados y como realizar la comunicación entre ellos.



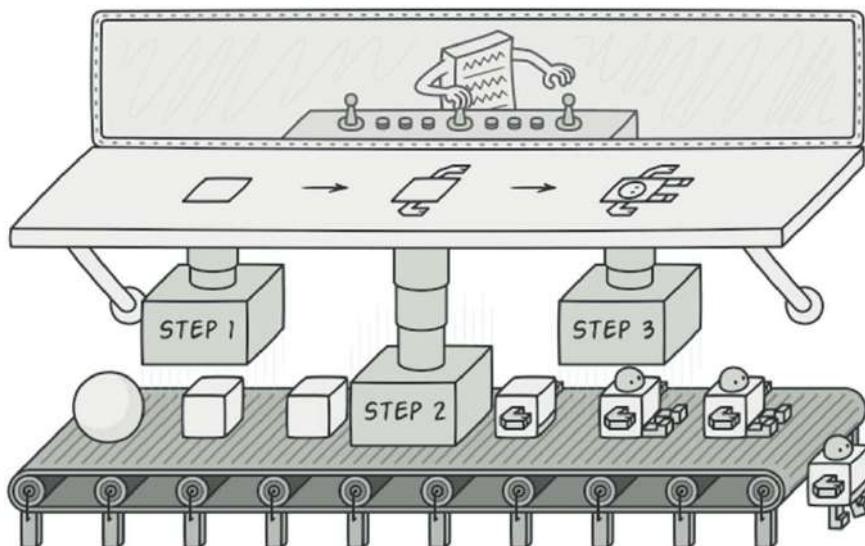
# ¿QUÉ ES BUILDER?



# Patrón de diseño creacional builder

- Builder de un patrón de diseño que nos permite crear objetos complejos paso a paso de una forma similar a como se fabrican las cosas en una cadena de producción.

- [+ Info sobre el patrón builder](#)
- [Código completo del ejemplo](#)



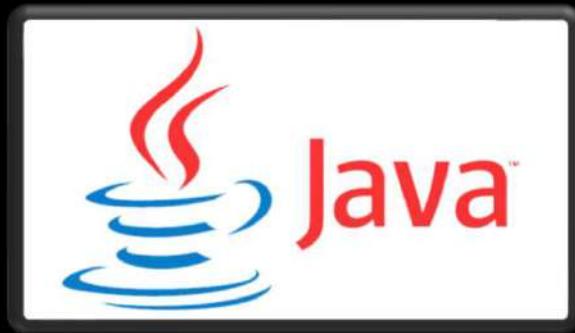
```
Main.java ×
1 package com.main;
2
3 public class Main {
4 public static void main(String[] args) {
5 Employee employee = new EmployeeBuilder()
6 .setId_user(1).setName("James")
7 .setSurname("Gosling")
8 .build();
9 System.out.println(employee.toString());
10
11 Employee employee2 = new EmployeeBuilder()
12 .setId_user(2).setName("Java")
13 .setSurname("desde0.com")
14 .build();
15 System.out.println(employee2.toString());
16 }
17 }
```

Console ×  
<terminated> Main (6) [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (8 may. 2022 14:02:  
Employee [id\_user=1, name=James, surname=Gosling]  
Employee [id\_user=2, name=Java, surname=desde0.com]

# Ejercicio sobre el design pattern Builder

- Crea una clase coche y una clase CocheBuilder mediante a la cual construiremos los objetos.
  - Los objetos deberán de tener marca, modelo, caballos y motor.
  - Construye 3 objetos uno con 1 propiedad, otro con 2 y otro con 3 mediante a Builder

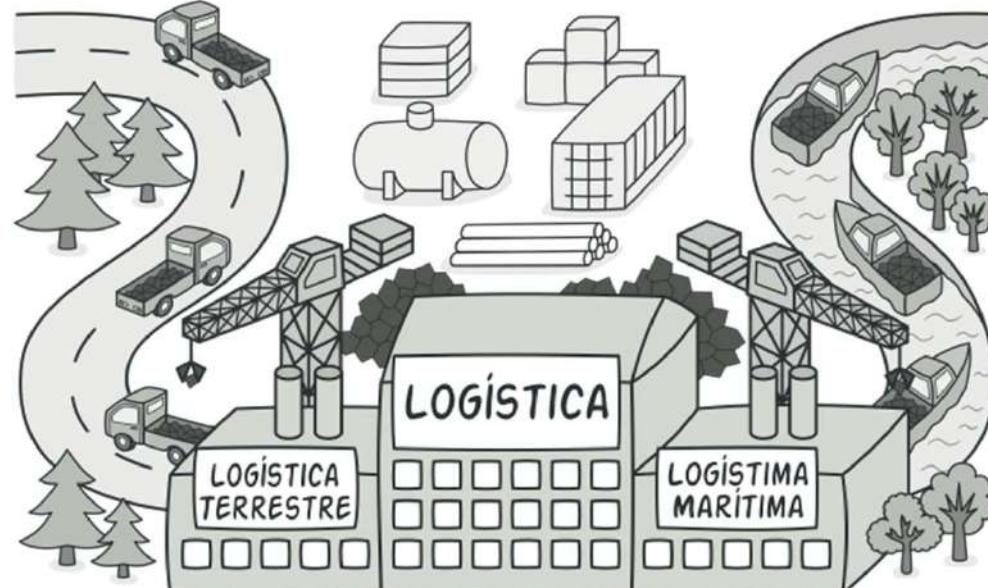
# ¿QUÉ ES FACTORY?



640

# Patrón de diseño creacional factory

- Es uno de los patrones de diseño más utilizados de Java. Se determinadas situaciones como por ejemplo cuando tenemos una interfaz y según un input necesitamos devolver una de estas subclases en concreto. Vamos a ver un ejemplo:
  - [+ Info sobre el patrón factory](#)
  - [Código completo del ejemplo](#)



## Ejercicio de investigación sobre patrones

- Hasta ahora hemos visto algunos ejemplos de cómo realizar algunos patrones de diseño. Ahora te toca investigar a ti. Por lo que debes escoger otros patrones, no vale repetir los patrones ya explicados o coger solamente patrones de un tipo. Y con apoyo de YouTube, páginas que hablen sobre ello, etc. Investigar y explica el funcionamiento de 4 patrones más.



643