

# DESARROLLO DE APLICACIONES WEB

## Unidad 2

### Características e instalación de Java

*1r DAW*

*IES La Mola de Novelda*

*Departament d'informàtica*

# ÍNDIX

1.- El lenguaje de programación Java.....	3
1.1.- Características de Java.....	3
1.2.- Java y los <i>byte-code</i> .....	4
2.- Proceso de compilación de un programa C++ y de un programa Java.....	5
3.- Preparación del entorno de ejecución.....	6
3.1.- Instalación del entorno.....	7
3.1.1.- OpenJDK.....	8
3.1.1.1.- Entorno Windows.....	8
3.1.1.2.- Entorno Linux.....	9
4.- Mi primer programa en Java.....	10
5.- Escritura de programas en Java.....	11
5.1.- Codificación del texto.....	11
5.2.- Cosas a tener en cuenta.....	11
5.3.- Javadoc.....	12
5.3.1.- Funcionamiento.....	12
5.3.2.- Colocación.....	13
5.3.3.- Ejemplo.....	13
6.- Resumen de como funciona Java.....	15

## Unidad 2: INTRODUCCIÓN A LA JAVA

### 1.- EL LENGUAJE DE PROGRAMACIÓN JAVA

Para practicar todos los conceptos de programación básica que veréis de ahora en lo sucesivo utilizaremos el lenguaje de programación Java. Una vez aprendáis a programar en un lenguaje, dominar otros lenguajes os será muy fácil, puesto que muchos de los conceptos básicos, e incluso algunos aspectos de la sintaxis, se mantienen entre diferentes lenguajes de nivel alto.

#### 1.1.-CARACTERÍSTICAS DE JAVA

Algunas de las características que presenta este lenguaje son:

- **Multiplataforma:** El código generado por el compilador Java es independiente de la arquitectura y por tanto, se puede ejecutar en cualquier plataforma sin que sea necesario volver a compilar: *Write Once, Run Anywhere* (WORA: Escríbelo una vez, ejecútalo en cualquier lugar).
- **Orientado a objetos:** La POO es un paradigma (forma de representar y manipular el conocimiento) de programación que acerca la manera como se hacen los programas al método de pensamiento humano. También puede utilizarse mediante la programación estructurada (paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora utilizando únicamente las subrutinas y tres estructuras de control básicas: la secuenciación, la selección y la iteración).
- **Distribuido:** Java proporciona herramientas para poder crear aplicaciones en red (aplicaciones TCP/IP)
- **Dispone de un amplio conjunto de bibliotecas:** La programación de aplicaciones en Java, no se basa solo con el juego de instrucciones que proporciona, sino que también tenemos al alcance un conjunto de clases que nos facilitarán muchas de las operaciones que queremos hacer.
- **Robusto:** Fue diseñado para crear programas altamente fiables. Por eso realiza muchas comprobaciones en la compilación y en el tiempo de ejecución. Por ejemplo, comprueba los tipos para evitar errores en la fase de desarrollo, utiliza la memoria para eliminar las preocupaciones por parte del programador de la liberación de memoria, comprueba los límites

de los vectores para evitar sobrescribir la memoria. Además, para asegurarse el funcionamiento de la aplicación, realiza una verificación de los *byte-code*, el cual es el resultado de la compilación de un programa Java. El *byte-code* es un código de máquina virtual (no es comprensible por el hardware) que es interpretado por el intérprete de Java y que ha pasado por torcidas las fases de compilación. Por lo tanto, Java proporciona:

1. Comprobación de punteros
  2. Comprobación de los límites de los arrays.
  3. Excepciones.
  4. Verificación de *byte-code*.
- **Fácil de aprender:** Es fácil de utilizar y su sintaxis es muy parecida a C o C++.
  - **Interpretado con *byte-code*:** Esta es una característica derivada del hecho que sea multiplataforma. Al tratarse de un lenguaje interpretado, hay que disponer del intérprete correctamente instalado y configurado a cada máquina donde queráis ejecutar vuestro programa. Esto quiere decir que hay un programa más que tenéis que configurar correctamente a vuestro sistema. Esto también hace que la ejecución de los programas en Java no siga el proceso típico de cualquier otra aplicación (por ejemplo, ejecutarlo desde línea de órdenes o hacer doble clic a la interfaz gráfica). No hay ningún fichero que se pueda identificar claramente como ejecutable.

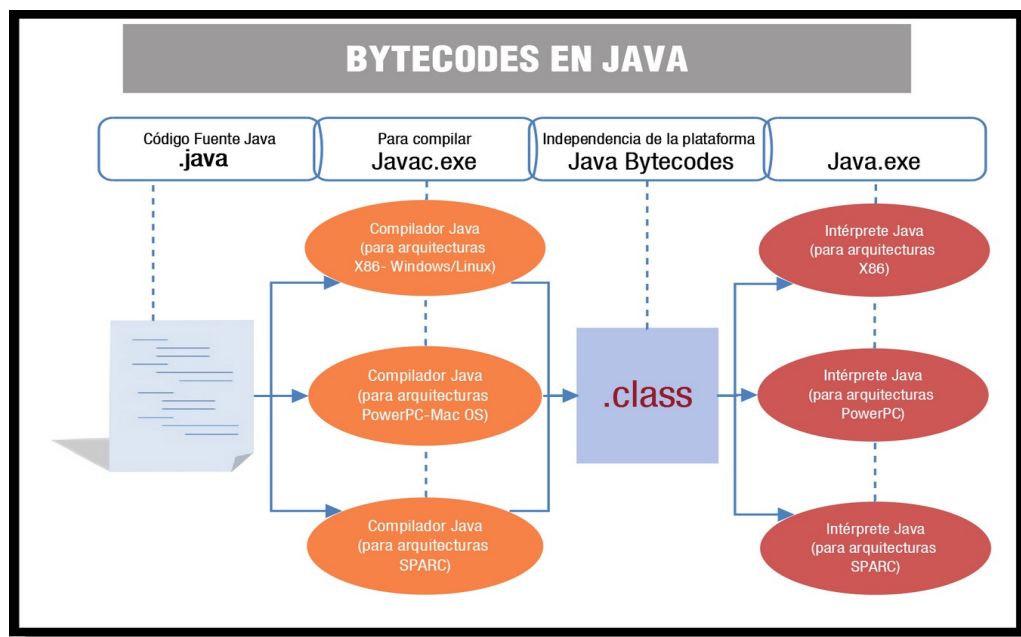
## 1.2.-JAVA Y LOS *BYTE-CODE*.

Un programa escrito en Java no es directamente ejecutable, es necesario que el código fuente sea interpretado por la Máquina Virtual Java. Una vez escrito el código fuente (archivos con extensión .java), este es precompilado generándose los códigos de bytes, *bytecodes* o Java *bytecodes* (archivos con extensión .class) que serán interpretados directamente por la Máquina Virtual Java y traducido a código nativo de la plataforma sobre la cual se esté ejecutando el programa. En este proceso, hay un verificador de bytes que se asegurará que se cumplirán las siguientes condiciones:

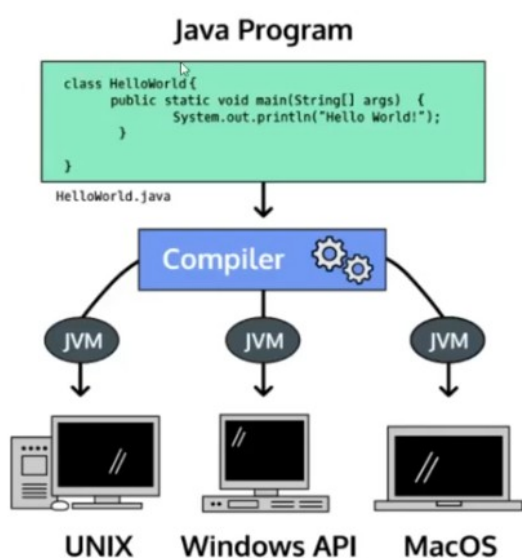
1. El código satisface las especificaciones de la Máquina Virtual Java.
2. No existe amenaza contra la integridad del sistema
3. No se producirá desbordamientos de memoria.
4. Los parámetros y tipos son adecuados

5. No existen conversiones de datos no permitidos

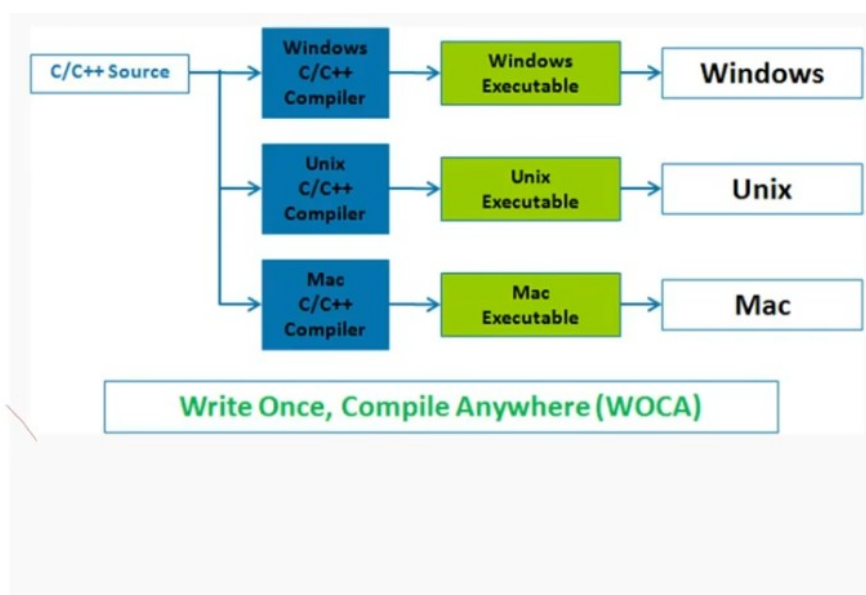
- **Byte-code:** Son instrucciones en lenguaje máquina que no son específicas de ningún procesador o sistema de cómputo. Hará falta un intérprete de código de bytes (\*bytecode) para poder ejecutarlo en una plataforma. A estos intérpretes también se los denomina Máquinas Virtuales Java e intérpretes Java en tiempo de ejecución. En general la Máquina Virtual Java es un programa de tamaño pequeño y gratuito para todos los sistemas operativos



## 2.- PROCESO DE COMPILACIÓN DE UN PROGRAMA C++ Y DE UN PROGRAMA JAVA



WORA: Write Once, Run Anywhere



### 3.- PREPARACIÓN DEL ENTORNO DE EJECUCIÓN

A partir de las características vistas en el apartado anterior, las herramientas que hacen falta para crear y ejecutar un programa en lenguaje Java son:

- Un **editor de texto** simple cualquiera. En él crearemos el archivo de código fuente en Java con la extensión **.java**. Para guardar los archivos de código fuente en Java utilizaremos la notación camello (es una recomendación), la cual consiste al utilizar solo letras consecutivas sin acentos (ni espacios, subrayados o números), y en que la inicial de cada palabra usada sea siempre en mayúscula. Ejemplos:

MiArchivo.java, Prueba.java, HolaDonPepito.java

- Un **compilador** del lenguaje **Java**, para generar *byte-code*.
- Un **intérprete** de **Java**, para poder ejecutar los programas ejecutables de *byte-code* de Java.

Todos los sistemas operativos incorporan un editor de texto simple (Notepad++ en Windows 10 y el kwrite en Lliurex21, como ejemplos). Por lo tanto, el compilador y el intérprete de Java son dos programas que tendremos que instalar a nuestro ordenador, puesto que normalmente no están preinstalados por defecto.

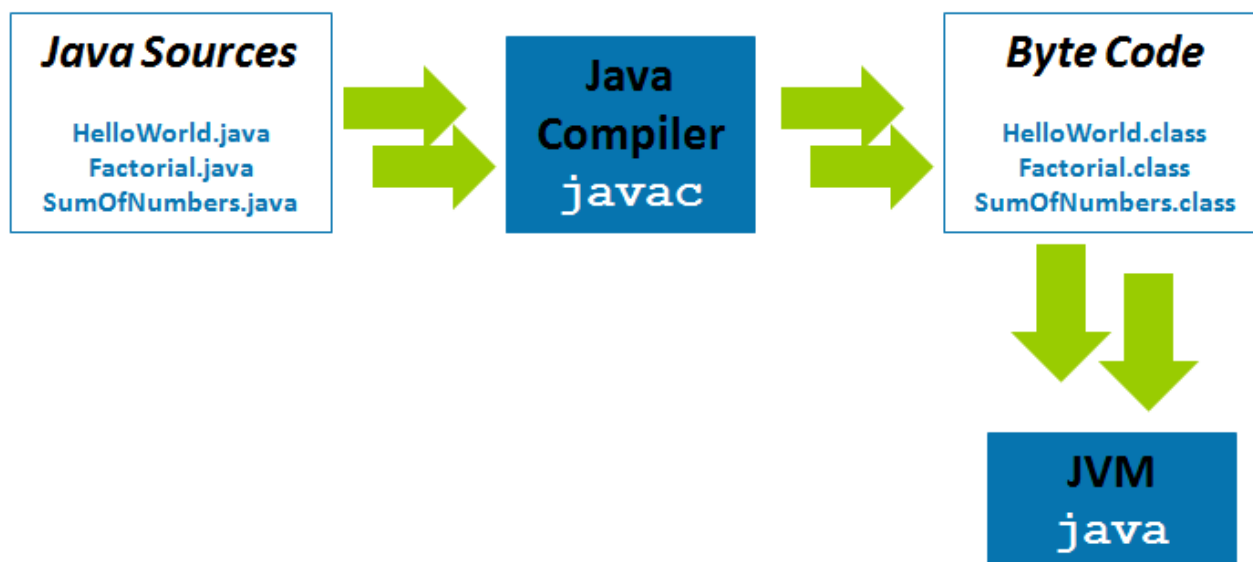


Si queremos desarrollar código de Java y compilarlo para genera el Java *byte-code* necesitaremos el componente **JDK** (*Java Development Kit* / Equipo de desarrollo de Java). Dentro de él está incluido el **compilador** de Java (**javac**) que transforma el código fuente escrito en Java (.java) a *bytecode* (.class). Pero además, también tiene otras utilidades como por ejemplo un depurador y un archivador.

Una parte del JDK es el **JRE** (*Java Runtime Environment* / Entorno de ejecución de Java). Este ayuda a ejecutar los programas. Contiene la **JVM** (*Java Virtual Machine* / Máquina Virtual de Java), las clases/paquetes java y las bibliotecas en tiempos de ejecución. Si la única cosa que queremos es ejecutar programas escritos por otros, tan solo necesitaremos el JRE. La JVM es una parte importante del JRE que en realidad es quien ejecuta los programas (ficheros .class), utiliza las bibliotecas de clases de Java y las bibliotecas en tiempos de ejecución para ejecutar los programas. Cada Sistema Operativo o plataforma tendrá una JVM diferente. La **aplicación que ejecuta** los ficheros .class es **java**. El que hace se llamar internamente a la JVM para ejecutar los programas.

JIT (Just in time Compiler) es un módulo dentro de la **JVM** que ayuda a **compilar** determinadas partes del *bytecode* al código de la máquina (código binario) para obtener un mayor rendimiento. Algunas partes del *bytecode* se compilarán al código máquina, las otras partes normalmente se interpretan y se ejecutan.

A la siguiente figura podemos ver que la aplicación **javac** se encarga de compilar el programa y la aplicación **java** de ejecutarlo.



En definitiva, el JDK se utiliza para construir y compilar código fuente Java, y el JRE junto con la máquina virtual JVM se utilizan para ejecutar *byte-code* de Java. Todo junto se llama el **JSE** (Java *Standard Edition* / Edición Estándar de Java).

### 3.1.-INSTALACIÓN DEL ENTORNO

Si queremos utilizar el JDK oficial, tendremos que ir a la página oficial de [Oracle](#) y bajarnos el paquete según nuestro sistema operativo e instalarlo (es posible que se os pida que os registréis antes de descargar el software). Después de seguir estos pasos, es posible que las órdenes JDK no funcionen todavía, especialmente en Windows y Linux hasta que configuremos algunas variables de entorno, como veremos más adelante.

En nuestro caso, vamos a utilizar una alternativa libre al JDK original de la empresa Oracle. Esta empresa es quien administra Java y si queremos tener una aplicación con fines lucrativas en producción tendremos que pagarle (Oracle prioriza la versión comercial para las empresas que lo pueden pagar). Es por eso que hay alternativas libres mantenidas por la comunidad de desarrolladores que nos sirven para un curso de programación. Una de ellas es **OpenJDK** y será la que vamos a utilizar en este módulo.

### 3.1.1.- OpenJDK

- Vayamos a la página web [OpenJDK](https://openjdk.org), la cual es administrada por la empresa Oracle pero que está publicada como software libre. Aquí podremos descargar una versión de Java gratuita, sin ningún tipo de limitación ni restricción. No siempre tenemos que estar con la última versión, por lo tanto podemos usar una versión anterior a la más actual. Además, la 17 es LTS (*Long Time Support*), es decir, aunque sea una versión anterior todavía tiene un soporte más largo del normal.
- Vamos al apartado *Ready for use*.
- Para descargar una versión anterior iremos al apartado *Archive* y a la versión 17.02.
- A la sección de *build* buscamos la plataforma y sistema operativo que vayamos a utilizar. Hacemos un clic al *zip*.

#### 3.1.1.1.- Entorno Windows

- Extraemos la carpeta en un lugar donde no se cambiará nunca puesto que tenemos que configurar unas variables del sistema que apuntarán en esta carpeta. Por ejemplo a «Archivos de programa» que es el lugar donde se instalan normalmente los programas cuando se utiliza un instalador. En ella crearemos la carpeta Java y dentro de ella extraeremos el *zip*.
- Ahora comprobamos que dentro de la carpeta *jdk-versión* hay la carpeta *bin*. En ella se encuentra el archivo ejecutable *javac*.
- Creación de la variable de entorno que será como un parámetro que le dirá al S.O como se tiene que comportar:
  - Menu Inicio → Escribimos al buscador «Variables» → Elegimos la opción «Editar las variables de entorno del sistema».
  - A la ventana «Propiedades del sistema» vamos al botón «Variables de entorno» y se abrirá un editor para configurar estas variables.
  - En el apartado «Variables del sistema» vamos a crear una nueva. Hacemos clic en «Nueva» y le ponemos el nombre de JAVA\_HOME. Para darle un valor vamos al botón «Examinar directorio ...» y buscamos la carpeta donde está el JDK, que es la carpeta raíz que contiene el JDK. Aceptamos para guardar los cambios. Veremos esta variable con la ruta especificada.



- A continuación configuraremos la variable de entorno `PATH`, la cual tiene un muchas carpetas configuradas. La seleccionamos y hacemos un clic en «Editar ...». Pondremos la ruta de la carpeta `bin` que tenemos en la carpeta `jdk-versión`. Cuidado en no borrar nada. Hacemos un clic en una parte que esté blanca o un clic en «Nuevo»
- A continuación hacemos un clic en «Examinar ...» y seleccionamos la carpeta `bin` de nuestro `jdk`.
- Seguidamente, tenemos que comprobar que la instalación se ha hecho correctamente. Abrimos un símbolo de sistema escribiendo en el buscador `cmd` o bien un `powershell` (la diferencia más grande es que con el `PowerShell` podemos administrar el registro del sistema operativo Windows, cosa que con el símbolo de sistema no).
- Al administrador de ordenes escribimos: **javac -versió** para ver la versión del **compilador** que tenemos y **java -version** para la versión del **OpenJDK**.

### 3.1.1.2.- Entorno Linux

- A pesar de que hay muchas formas diferentes de hacer la instalación, la opción más recomendada es instalar OpenJDK usando el repositorio predeterminado de Ubuntu. En primer lugar tendremos que ver el que hay disponible al repositorio escribiendo en el terminal la siguiente orden.

**apt-cache search openjdk | grep openjdk-17**

La orden `apt-cache` se utiliza para obtener información de programas o paquetes que pueda haber a los repositorios o que se tenga instalado al ordenador. Con el parámetro `search` le decimos que busco el paquete `openjdk`.

La orden `grep` sirve para buscar cadenas de texto y encontrar coincidencias con estas. Con el carácter `|` (pipe: yuberia) indicamos que el resultado del comando `apt-cache` será la entrada del orden `grep`. De esa manera filtramos la información para la cadena de texto `openjdk-17`.

Si el resultado de esa orden no da ningún resultado, tendremos que activar los repositorios de Ubuntu. Hagamos clic al menú de inicio y al apartado busca escribimos *Repoman*. En el apartado Repositorios por defecto activamos Ubuntu y aplicamos

```
administrador@PRL3-LLIUREX:~$ apt-cache search openjdk | grep openjdk-17
administrador@PRL3-LLIUREX:~$
```



Volvemos al terminal y ya podemos ver la disponibilidad del OpenJDK-17

```
administrador@PRL3-LLIUREX:~$ apt-cache search openjdk | grep openjdk-17
openjdk-17-dbg - Java runtime based on OpenJDK (debugging symbols)
openjdk-17-demo - Java runtime based on OpenJDK (demos and examples)
openjdk-17-doc - OpenJDK Development Kit (JDK) documentation
openjdk-17-jdk - OpenJDK Development Kit (JDK)
openjdk-17-jdk-headless - OpenJDK Development Kit (JDK) (headless)
openjdk-17-jre - OpenJDK Java runtime, using Hotspot JIT
openjdk-17-jre-headless - OpenJDK Java runtime, using Hotspot JIT (headless)
openjdk-17-jre-zero - Alternative JVM for OpenJDK, using Zero
openjdk-17-source - OpenJDK Development Kit (JDK) source files
administrador@PRL3-LLIUREX:~$
```

- Empezamos la instalación con la orden:

```
sudo apt-get install openjdk-17-jre
```

```
sudo apt-get install openjdk-17-jdk
```

- Confirmamos la instalación con las órdenes:

```
java --version
```

```
javac --version
```

- Si ya no los queremos utilizar, los podemos eliminar con las órdenes:

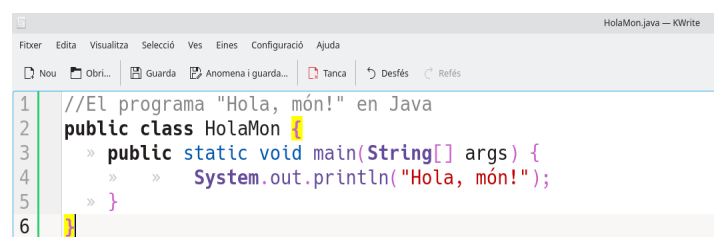
```
sudo apt-get remove openjdk-17-jre openjdk-17-jdk -purge
```

Tenéis que tener en cuenta que esto eliminará las dependencias sobrantes no correspondidas y borrará completamente la instalación y los datos tanto como sea posible de vuestro sistema.

## 4.- MI PRIMER PROGRAMA EN JAVA

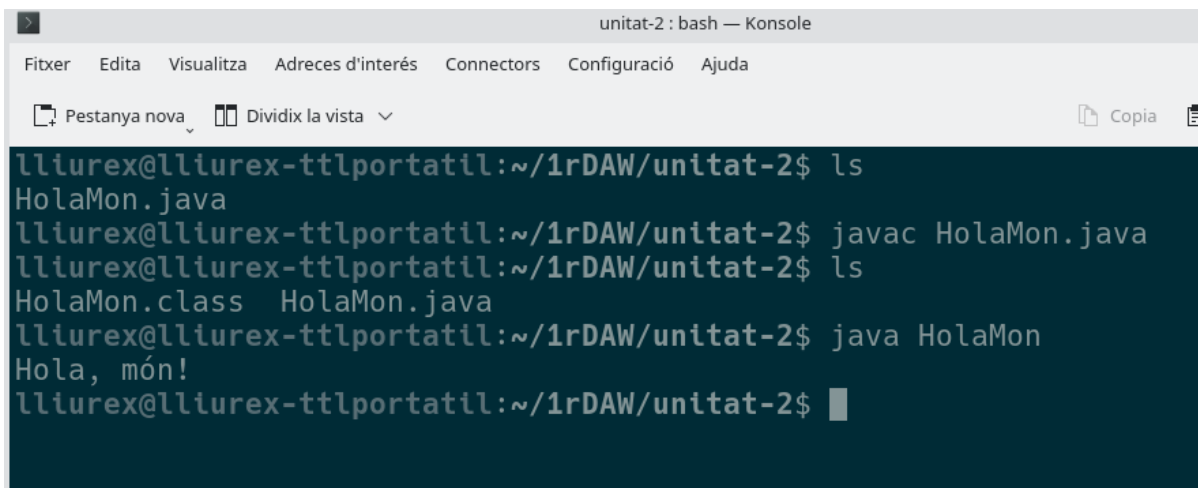
Vamos a comprobar que el entorno de trabajo se encuentra correctamente instalado y configurado. Dependiendo del sistema operativo en que te encuentras, abre un editor de texto simple y copia el siguiente código escrito en lenguaje Java:

```
//El programa "Hola, món!" en Java
public class HolaMon {
    public static void main(String[] args) {
        System.out.println("Hola, món!");
    }
}
```



Grabad el archivo en vuestra carpeta de trabajo con el nombre **HolaMon.java**. Seguidamente abrid un intérprete de comandos en la misma carpeta donde habéis guardado el archivo. Por último

compiláis y ejecutar el archivo de la siguiente manera:



```
unitat-2 : bash — Konsole
Fitxer  Edita  Visualitza  Adreces d'interés  Connectors  Configuració  Ajuda
Pestanya nova  Dividix la vista  Copia

lliurex@lliurex-ttlportatil:~/1rDAW/unitat-2$ ls
HolaMon.java
lliurex@lliurex-ttlportatil:~/1rDAW/unitat-2$ javac HolaMon.java
lliurex@lliurex-ttlportatil:~/1rDAW/unitat-2$ ls
HolaMon.class  HolaMon.java
lliurex@lliurex-ttlportatil:~/1rDAW/unitat-2$ java HolaMon
Hola, món!
lliurex@lliurex-ttlportatil:~/1rDAW/unitat-2$
```

## 5.- ESCRITURA DE PROGRAMAS EN JAVA

### 5.1.-CODIFICACIÓN DEL TEXTO

Todo el código fuente se escribe en un documento de texto con extensión **.java**. Como Java es un lenguaje para Internet, la codificación del texto tiene que permitir a todos los programadores de cualquier lengua poder escribir ese código. Por lo tanto, Java es un lenguaje compatible con la codificación **Unicode**.

[Unicode](#) proporciona un único y extenso conjunto de caracteres que pretende incluir todos los caracteres necesarios para cualquier sistema de escritura del mundo, incluyendo sistemas ancestrales (como el cuneiforme, el gótico y los jeroglíficos egipcios). Hoy resulta fundamental para la arquitectura de la Web y de los sistemas operativos, y las principales aplicaciones y navegadores web incluyen soporte para este elemento.

### 5.2.-COSAS A TENER EN CUENTA

Cuando programamos en Java tenemos que tener en cuenta lo siguiente:

- En Java (igual que en C) hay diferencia entre las mayúsculas y las minúsculas.
- Cada una de las líneas de código tienen que finalizar con uno ;
- Comentarios del código:
  - `//Comentario de una línea`
  - `/* Comentario de más de una línea*/`
  - También podemos incluir comentarios **javadoc**.

- {  
..... código interno del bloque  
}  
código fuera del bloque

## 5.3.-JAVADOC

**Javadoc** es una herramienta muy interesante del kit de desarrollo de Java para generar automáticamente documentación Java. genera documentación para paquetes completos o para archivos java. Su sintaxis básica es:

**javadoc** archivo.java o paquete

### 5.3.1.- Funcionamiento

Los comentarios javadoc comienzan con el símbolo **/\*\*** y terminan con **\*/** . Cada línea javadoc se inicia con un símbolo de asterisco. Dentro se puede incluir cualquier texto. Incluso se pueden utilizar códigos HTML para que al generar la documentación se tenga en cuenta el código HTML indicado.

En el código javadoc se pueden usar etiquetas especiales, las cuales comienzan con el símbolo **@**. Pueden ser:

- **@author** → Tras esa palabra se indica el autor del documento.
- **@version** → Tras lo cual sigue el número de versión de la aplicación
- **@see.** → Tras esta palabra se indica una referencia a otro código Java relacionado con éste.
- **@since.** → Indica desde cuándo esta disponible este código
- **@deprecated.** → Palabra a la que no sigue ningún otro texto en la línea y que indica que esta clase o método esta obsoleta u obsoleto.
- **@throws.** → Indica las excepciones que pueden lanzarse en ese código.
- **@param.** → Palabra a la que le sigue texto qué describe a los parámetros que requiere el código para su utilización (el código en este caso es un método de clase). Cada parámetro se coloca en una etiqueta **@param** distinta, por lo que puede haber varios **@param** para el mismo método.
- **@return.** → Tras esta palabra se describe los valores que devuelve el código (el código en

este caso es un método de clase).

### 5.3.2.- Colocación

El código javadoc hay que colocarlo en tres sitios distintos dentro del código java de la aplicación:

- **Al principio del código de la clase** (antes de cualquier código Java). En esta zona se colocan comentarios generales sobre la clase o interfaz que se crea mediante el código Java. Dentro de estos comentarios se pueden utilizar las etiquetas: @author, @version, @see, @since y @deprecated
- **Delante de cada método.** Los métodos describen las cosas que puede realizar una clase. Delante de cada método los comentarios javadoc se usan para describir al método en concreto. Además de los comentarios, en esta zona se pueden incluir las etiquetas: @see, @param, @exception, @return, @since y @deprecated
- **Delante de cada atributo.** Se describe para qué sirve cada atributo en cada clase. Puede tener las etiquetas: @since y @deprecated

### 5.3.3.- Ejemplo

```
/** Esto es un comentario para probar el javadoc
 * este texto aparecerá en el archivo HTML generado.
 * <strong>Realizado en septiembre de 2022</strong>
 *
 * @author Pepito de los palotes
 * @version 1.0
 */
public class prueba1 {
    //Este comentario no aparecerá en el javadoc
    /** Este método contiene el código ejecutable de la clase
     *
     * @param args Lista de argumentos de la línea de comandos
     * @return void
     */
    public static void main(String args[]){
        System.out.println("¡Mi segundo programa! ");
    }
}
```

```
lliurex@lliurex-ttlportatil:~/Música$ ls
Prova.java
lliurex@lliurex-ttlportatil:~/Música$ javadoc Prova.java
Loading source file Prova.java...
Constructing Javadoc information...
Building index for all the packages and classes...
Standard Doclet version 17.0.4+8-Ubuntu-120.04
Building tree for all the packages and classes...
Generating ./Prova.html...
Prova.java:13: error: invalid use of @return
    * @return void
      ^
Generating ./package-summary.html...
Generating ./package-tree.html...
Generating ./overview-tree.html...
Building index for all classes...
Generating ./allclasses-index.html...
Generating ./allpackages-index.html...
Generating ./index-all.html...
Generating ./index.html...
Generating ./help-doc.html...
1 error
lliurex@lliurex-ttlportatil:~/Música$
```

Prova x +

Fitxer | /home/lliurex/Música/Prova.html

PACKAGE CLASS TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

### Class Prova

java.lang.Object<sup>u</sup>  
Prova

---

public class **Prova**  
extends Object<sup>u</sup>

Esto es un comentario para probar el javadoc este texto aparecerá en el archivo HTML generado. **Realizado en septiembre de 2022**

#### Constructor Summary

Constructors	
Constructor	Description
Prova()	

#### Method Summary

All Methods	Static Methods	Concrete Methods
Modifier and Type	Method	Description
static void	main(String <sup>u</sup> [] args)	Este método contiene el código ejecutable de la clase

Methods Inherited from class java.lang.Object<sup>u</sup>

clone<sup>u</sup>, equals<sup>u</sup>, finalize<sup>u</sup>, getClass<sup>u</sup>, hashCode<sup>u</sup>, notify<sup>u</sup>, notifyAll<sup>u</sup>, toString<sup>u</sup>, wait<sup>u</sup>, wait<sup>u</sup>, wait<sup>u</sup>

#### Constructor Details

**Prova**

public Prova()

#### Method Details

**main**

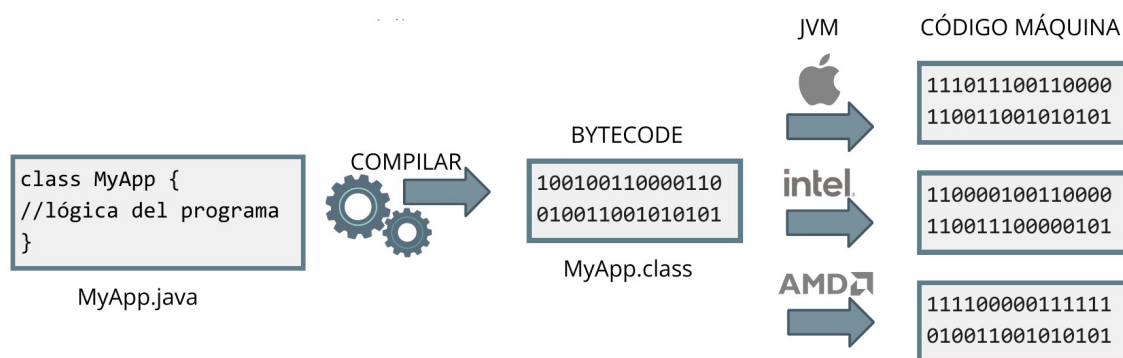
public static void main(String<sup>u</sup>[] args)

Este método contiene el código ejecutable de la clase

**Parameters:**

args - Lista de argumentos de la línea de comandos

## 6.- RESUMEN DE COMO FUNCIONA JAVA



## Compilación y ejecución en JAVA

