Gobierno de Canarias
Servicio Canario de Empleo

Centro GASTRONÓMICO del Atlántico

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL
SEPE

Módulo 1: Programación en Java 2023

Tema 2 Estructura de un programa en Java

Tema 2

Contenido

1. Estructura de un programa en Java

1.1. Objetivos

- Describir la estructura del código fuente de una aplicación Java.
- Presentar los conceptos de comentario y de identificador dentro del código fuente de un programa.

Esta unidad tiene como objetivo mostrar dos elementos típicos del código fuente: los comentarios y los identificadores. La estructura de un programa de Java es similar a la de un programa de C/C++. Por su diseño, permite a los programadores de cualquier otro lenguaje leer código en Java sin mucha dificultad. Java emplea siempre la Programación Orientada a Objetos por lo que todo el código se incluye dentro de las clases. Aunque ya se explicarán detenidamente más adelante, las clases son combinaciones de datos (constantes y variables) y rutinas (métodos).

1.2. La Clase Principal y el Método main()

Un programa puede construirse empleando varias clases. En el caso más simple se utilizará una única clase. Esta clase contiene el programa, rutina o método principal: main() y en éste se incluyen las sentencias del programa principal. Estas sentencias se separan entre sí por caracteres de punto y coma.

La estructura de un programa simple en Java es la siguiente:

```
public class ClasePrincipal {
    public static void main(String[ ] args) {
    sentencia_1;
    sentencia_2;
   // comentario_1
   ...
   sentencia_N;
  }
}
```

Ejemplo de un programa en Java llamado que muestra un mensaje por la pantalla del ordenador llamado **HolaMundo.java**:

```
/**
* La clase HolaMundo construye un programa que
* muestra un mensaje en pantalla
*/
public class HolaMundo {
    public static void main(String[ ] args) {
   System.out.println("Hello World!");
   }
}
```

Como se ha indicado anteriormente, en un programa de Java todo se organiza dentro de clases. En el ejemplo anterior, **HolaMundo** es el nombre de la clase principal y del archivo que contiene el código fuente. Todos los programas o aplicaciones independientes escritas en Java tienen un método **main( )** o principal que, a su vez, contiene un conjunto de sentencias.

¿Qué Significa cada Término de la Clase Principal?

La primera parte del programa corresponde a la definición de la clase:

- **public**: es un modificador de acceso que determina quién puede acceder a las clases o propiedades y métodos de una clase.
- **class**: palabra reservada que se utiliza para definir una clase. Contiene un conjunto de propiedades y métodos que luego sirven de modelo o plantilla para crear objetos o instancias de esa clase.
- **HolaMundo**: es un identificador de clase, no una palabra reservada, que identifica a la clase pública que se ha creado.

Resumiendo, esta línea define una clase pública identificada como *HolaMundo*.

```
public class HolaMundo {
...
}
```

La segunda parte del programa corresponde a la definición del método main:

- **public**: cada método en una clase puede ser público, protegido o privado dependiendo del nivel de acceso que el programador quiere darle. *public* significa que el método puede ser accedido desde cualquier otro método que tenga una instancia de esta clase.
- **static**: un método puede ser de instancia o de clase. Un método no estático es un método de instancia (que necesita una instancia de la clase donde se declara para ser invocado) y un método estático es un método de clase (no necesita una instancia de la clase donde se declara para ser invocado).

- **void**: Los métodos pueden devolver algo, por ejemplo, un método que suma dos números, devuelve el resultado de la suma; pero hay métodos que no devuelven nada y que sólo ejecutan acciones. Dichos métodos se declaran con la palabra reservada void (vacío) como tipo de retorno.
- **main()**: Es el nombre del método que la máquina virtual de Java busca para comenzar a ejecutar un programa. Siempre ha de llamarse main.
- **String**: Es una clase que define cadenas de caracteres. Gracias a la clase String, Java puede manipular textos.
- **String args[ ]**: Cualquier clase que se declara con corchetes "[ ]", quiere decir que lo que tienes no es una instancia de esa clase, sino un array de objetos de dicha clase. Por ejemplo, String[ ] meses contendría los nombres de los meses del año. El método main() de Java necesita un String[ ] porque el usuario de nuestro programa puede pasar parámetros extra desde la línea de comando a nuestro programa. Esos parámetros, el programador los recibe a través de ese array.

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

La tercera parte del código corresponde a las sentencias:

- **System.out.println()**[1]: es un método contenido en la API de Java SE que imprime por pantalla el texto que se le pasa entre paréntesis, en este caso imprimirá el mensaje *Hello World!*:

```
System.out.println("Hello World!");
```

Documentación Oracle sobre cómo crear la aplicación Hello World! en Java:

https://docs.oracle.com/javase/tutorial/getStarted/application/

1.3. Elementos del Lenguaje Java

1.3.1. Sentencias

Una sentencia es una orden que se le da al programa para realizar una tarea específica, esta puede ser: mostrar un mensaje en la pantalla, declarar una variable (para reservar espacio en memoria), inicializarla, llamar a una función, etc. Las sentencias acaban con el carácter

[1] System.out.println():
https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/System.html

";". Este carácter separa una sentencia de la siguiente. Normalmente, las sentencias se ponen unas debajo de otras (una en cada línea), aunque sentencias cortas pueden colocarse en una misma línea. Algunos ejemplos de sentencias:

```
int i=1;
import java.awt.*;
System.out.println("Mi primer programa");
rect.mover(10, 20);
```

Nota: En el lenguaje Java, los caracteres espacio en blanco se pueden emplear libremente. Como podremos ver en los siguientes ejemplos, es muy importante para la legibilidad de un programa la colocación de unas líneas debajo de otras empleando tabuladores. El editor del IDE nos ayudará a realizar esta tarea de manera automática sin apenas percibirlo.

1.3.2. Bloques de código

Un **bloque de código** no es más que una agrupación de sentencias que están agrupados entre llaves "{ ... }"..

Ejemplo de bloque de código:

```
int edad = 15;
if (edad < 18 > {   <-----   Comienzo del bloque de código
    String mensaje = "Usuario menor de 18 años";
    System.out.println(mensaje);
}   <-----   Fin del bloque de código
```

En el ejemplo anterior, hay una sentencia condicional donde se pregunta si el valor de la variable edad es menor de 18, en caso de que se cumpla la condición, se imprime el mensaje por pantalla "Usuario menor de 18 año"el conjunto de sentencias se reduce a tres sentencias, que llaman a los métodos predefinidos en Java print y println que permiten visualizar texto por el dispositivo de salida de datos (por defecto la pantalla)

1.3.3. Expresiones

Una **expresión** es todo aquello que se puede poner a la derecha del operador de asignación "=". Por ejemplo:

```
x = 123;
y = (x + 100) / 4;
area = circulo.calcularArea(2.5);
Rectangulo r = new Rectangulo(10, 10, 200, 300);
```

La primera expresión asigna un valor a la variable *x*.

- La segunda, realiza una operación aritmética y el resultado se asigna a la variable y.
- La tercera, es una llamada a una función miembro *calcularArea* desde un objeto círculo de una clase determinada y el resultado se asigna a la variable *area*.
- La cuarta, reserva espacio en memoria para un objeto de la clase *Rectangulo* mediante la llamada a una función especial que tienen todas las clases denominada **constructor**.

1.3.4. Comentarios

Un **comentario** es un texto adicional que se añade al código para explicar su funcionalidad, bien a otras personas que lean el programa, o al propio autor como recordatorio. Los comentarios son una parte importante de la documentación de un programa. Los comentarios son ignorados por el compilador, por lo que no incrementan el tamaño del archivo ejecutable; se puede, por tanto, añadir libremente al código para que pueda entenderse mejor.

El primer tipo de comentarios son los **comentarios de una sola línea**. En ellos se utiliza la secuencia de caracteres //. El compilador ignora todo lo que se incluya entre la secuencia de caracteres // y el final de la línea.

Por ejemplo:

```
// Este es un comentario estilo C++, llega al final de la linea
sentencia_1;
```

Nota: La pareja de caracteres **//** hay que escribirla sin dejar ningún espacio en blanco entre ellos.

El segundo tipo de comentarios son los **comentarios de múltiples líneas**. En ellos se utilizan las secuencias de caracteres **/*** para abrir el comentario y ***/** para cerrar el comentario.

Por ejemplo:

```
/* En este otro comentario estilo C, el final
lo indica la marca */
sentencia_1;
```

Nota: Los comentarios pueden incluir cualquier carácter válido en Unicode y no pueden anidarse.

Estos dos formatos de comentario se emplean para los denominados comentarios de implementación. Los comentarios de implementación se utilizan en el programa fuente para resaltar código o para aclarar un desarrollo en particular.

Además, en Java existe un tercer tipo de comentario llamados **comentarios de documentación** (doc comments) que facilita la generación de documentación en formato HTML al emplear algunas herramientas de documentación (por ejemplo, **Javadoc** incluida en el Kit de Desarrollo de Java). Los comentarios para la documentación se emplean para describir la especificación del código, desde una perspectiva independiente cómo se ha implementado y ser leído por desarrolladores que no tengan necesariamente el código fuente a la vista. Ejemplos de este tipo de comentarios:

```
/** Comentario estilo javadoc, se incluye
automáticamente en documentacion HTML */
/**
* Muchos programadores
* suelen incluir un asterisco
* al principio de cada linea del
* comentario para facilitar su lectura
*/
```

Nota: Los comentarios deberían contener sólo información relevante para la lectura y comprensión del programa. Todos los programas deben empezar por un comentario que describa el propósito general del programa.

Artículo de Oracle sobre cómo escribir comentarios de documentación para la herramienta Javadoc:

https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html

Convención para los comentarios al inicio de los programas

Todos los archivos fuente deberían comenzar con un comentario liste el nombre de la clase, información de la versión, fecha y el aviso de copyright:

```
/**
* The HelloWorld program implements an application that
* simply displays "Hello World!" to the standard output.
*
* @author Roger González
* @version 1.0
* @since 21-10-2020
*/
```

```
public class HelloWorld {
public static void main(String[ ] args) {
    // Imprime el texto Hello, World! en la salida estándar.
    System.out.println("Hello World!");
    }
}
```

A continuación, se muestran algunas **etiquetas** (tags) que reconoce la herramienta *Javadoc*:

Etiqueta	Descripción	Sintaxis
@author	Añade el autor de una clase	@autor nombre
@version	Añade el número de la versión de la clase	@version versión
@since	Añade información de la creación de la clase	@since fecha
@deprecated	Indica que el método o clase es obsoleto (propio de versiones anteriores) y que no se recomienda su uso.	@deprecated descripción

1.3.5. Identificadores

Los **identificadores** son nombres que se les asignan a *variables*, *métodos*, *clases*, ..., en el código fuente de un programa. Los *identificadores* sólo existen en el código del programa fuente y no en el programa objeto (resultado de la compilación del programa fuente). Todo nuevo identificador que se emplee en un programa Java debe definirse previamente a su utilización. Las *normas para la construcción de un identificador* empleando el lenguaje de programación Java son las siguientes:

Un identificador comienza por una letra, un carácter de subrayado "_" o un carácter de dólar "$". Aunque no se recomienda emplear el carácter $, ya que el compilador suele utilizarlos de forma interna para crear identificadores propios.

- Los siguientes caracteres pueden ser también dígitos, pero no pueden emplearse espacios en blanco u otros caracteres como el signo de interrogación (?) o el signo del tanto por ciento (%).
- No hay límite máximo de caracteres.
- En los identificadores del código fuente de un programa en Java se distinguen las mayúsculas de las minúsculas. Por ejemplo, las palabras *casa*, *CASA* y *Casa* son tres identificadores diferentes (El lenguaje Java es sensible a las mayúsculas).
- Pueden incluir caracteres Unicode, con lo que se pueden emplear secuencias de escape /uxxxx para representar estos caracteres.
- No puede emplearse el identificador de una variable o cualquier otro elemento del código fuente del programa para otro ya existente en el mismo bloque. Excepción: *variable miembro* y *local* con el mismo identificador.
- Existe una serie de palabras reservadas que no pueden emplearse como identificadores por el programador en el código fuente para otros usos. Por ejemplo, la palabra **double** se utiliza para definir un tipo de dato real y la palabra **for** se emplea para construir un tipo determinado de bucle.

En la siguiente tabla se muestran las palabras reservadas en Java.

A-D	D-I	I-P	P-T	T-Z
abstract	do	implements	protected	throw
boolean	double	import	public	throws
break	else	instanceof	rest	transient
byte	extends	int	return	true
case	false	interface	short	try
catch	final	long	static	void
char	finally	native	strictfp	volatile
class	float	new	super	while
const*	for	null	switch	
continue	goto*	package	synchronized	
default	if	privative	this	

Algunos de estos identificadores reservados no tienen todavía uso en la implementación actual del lenguaje Java. En concreto, los identificadores marcados con un asterisco * no se utilizan actualmente.

Las palabras reservadas se pueden clasificar en las siguientes categorías:

- **Tipos de datos**: *boolean*, *float*, *double*, *int*, *char*.
- **Sentencias condicionales**: *if*, *else*, *switch*.
- **Sentencias iterativas**: *for*, *do*, *while*, *continue*.
- **Tratamiento de las excepciones**: *try*, *catch*, *finally*, *throw*.
- **Estructura de datos**: *class*, *interface*, *implements*, *extends*.
- **Modificadores y control de acceso**: *public*, *private*, *protected*, *transient*.
- **Otras**: *super*, *null*, *this*.

A continuación, se muestran los nombres de métodos reservados en Java cuyo significado y utilización se explicará más adelante cuando se mencione la clase predefinida **Object**[2]:

A-E	F-G	H-N	N-T	W-Z
clone	finalize	hashCode	notifyAll	wait
equals	getClass	notify	toString	

[2] Object: https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/Object.html

1.3.5.1. Recomendaciones a la hora de escribir identificadores

Aunque la forma de escribir los identificadores en Java no está normalizada, a continuación, se dan algunas recomendaciones para la elección de estos identificadores:

- Todos los identificadores han de comenzar con una letra, el carácter subrayado "_" o el carácter dollar "$".
- Puede incluir, pero no comenzar por un número.
- No puede incluir el carácter espacio en blanco.
- Distingue entre letras mayúsculas y minúsculas.
- No se pueden utilizar las palabras reservadas como identificadores.

Además de estas restricciones, hay ciertas convenciones que hacen que el programa sea más legible, pero que no afectan a la ejecución del programa. La primera y fundamental es la de encontrar un nombre que sea significativo, de modo que el programa sea lo más legible posible. El tiempo que se pretende ahorrar eligiendo nombres cortos y poco significativos se pierde con creces cuando se revisa el programa después de cierto tiempo:

Tipo de Identificador	Convención	Ejemplo
Nombre de una clase	Comienza por letra mayúscula	Rectángulo, CinematicaApplet
Nombre de método	Comienza con letra minúscula	calcularArea, getValue, setColor
Nombre de constante	En letras mayúsculas	PI, MAX_ANCHO

Estilos de escritura

CamelCase[3] es un estilo de escritura que se aplica a frases o palabras compuestas. El nombre se debe a que las mayúsculas a lo largo de una palabra en *CamelCase* se asemejan a las jorobas de un camello. El término *CamelCase* se podría traducir como literalmente como *Mayúsculas/Minúsculas Camello*.

Existen dos tipos de estilos de escritura *CamelCase*:

- **UpperCamelCase**: cuando la primera letra de cada una de las palabras es mayúscula.
- **lowerCamelCase**: igual que la anterior con la excepción de que la primera letra es minúscula.

Ejemplo de **UpperCamelCase**: *EjemploDeUpperCamelCase*

Ejemplo de **lowerCamelCase**: *ejemploDeLowerCamelCase*

[3] CamelCase: https://es.wikipedia.org/wiki/CamelCase

Convenciones para la Legibilidad del Programa Fuente

identación[4] consiste en aplicar una sangría al código, mejorando así su legibilidad. Aunque la definición exacta de la *indentación* (espacios vs tabuladores) no se especifica en el lenguaje Java, se recomienda emplear una secuencia de **cuatro espacios en blanco** o **tabulador** como unidad de *indentación*.

Por otro lado, se recomienda evitar las líneas de más de 80 caracteres en el código fuente ya que no se manejan correctamente por muchos terminales y herramientas de edición. En el caso de que una sentencia sea muy grande se recomienda especificarla en dos líneas:

Ejemplo de una sentencia definida en dos líneas:

```
longName1 = longName2 * (longName3 + longName4 - longName5)
+ 4 * longname6; // PREFER
```

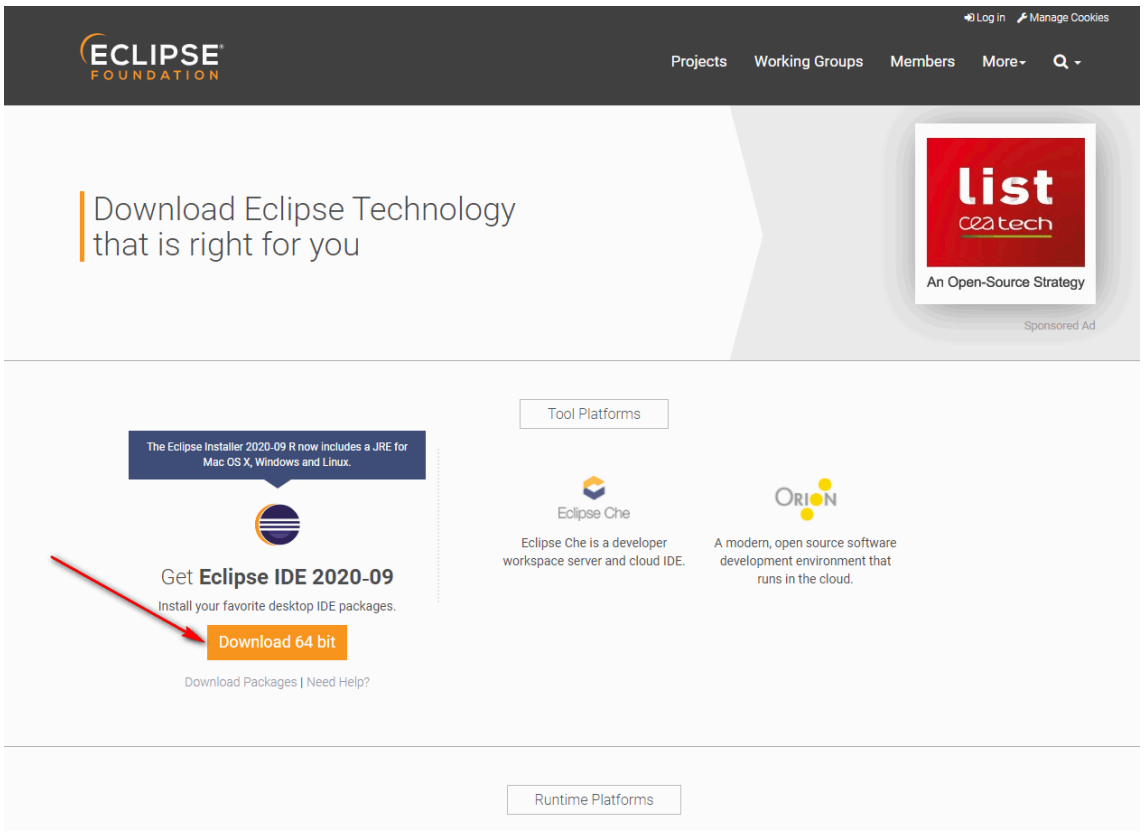
Ejemplo de la declaración de un método en dos líneas:

```
private static synchronized horkingLongMethodName(int anArg,
Object anotherArg, String yetAnotherArg,
Object andStillAnother) {
...
}
```

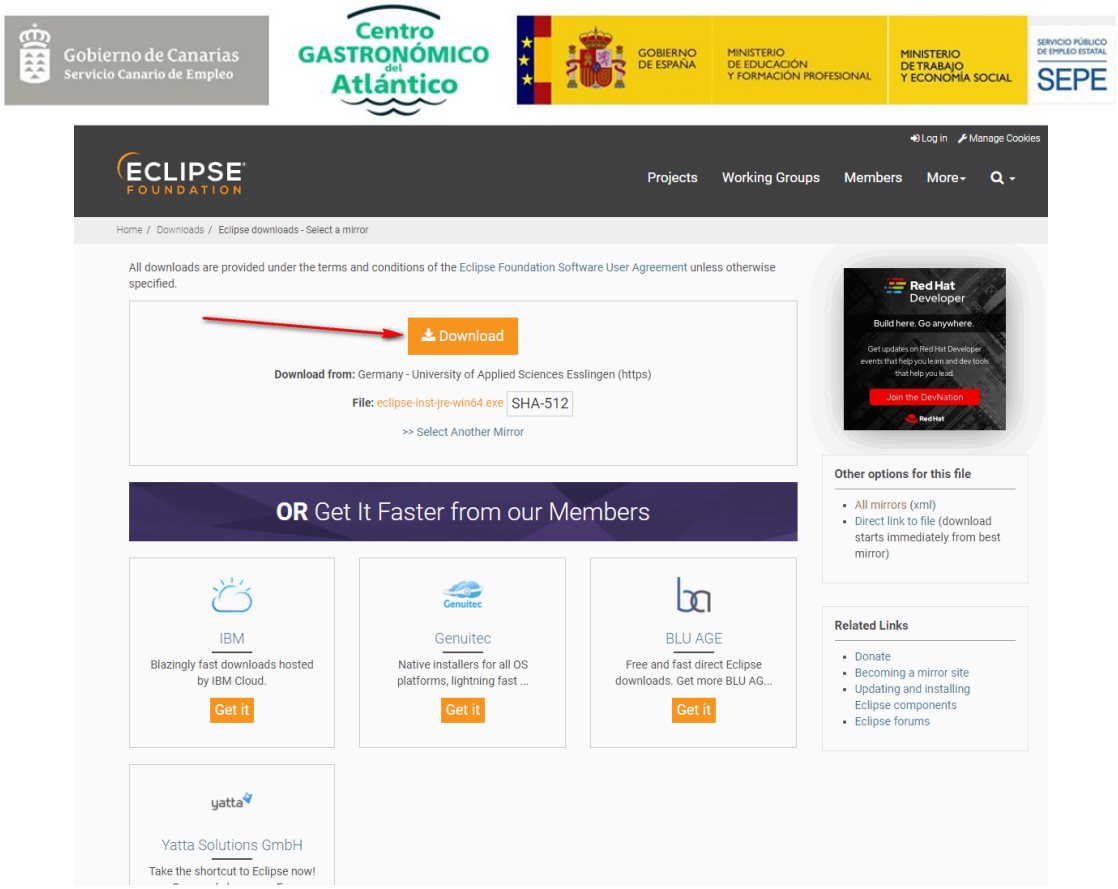
[4] Identación: https://es.wikipedia.org/wiki/Indentaci%C3%B3n

1.4. ANEXO I. INSTALACIÓN DE ECLIPSE

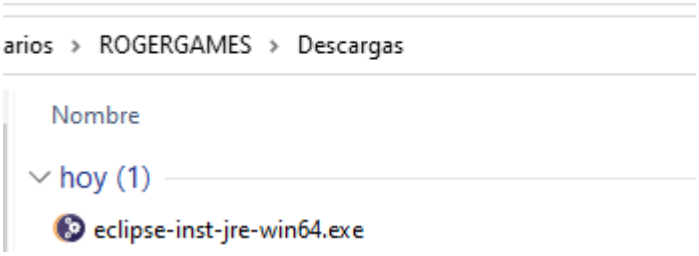
Descargar Eclipse de la URL https://www.eclipse.org/downloads/



Pulsamos en el bótón de Download:

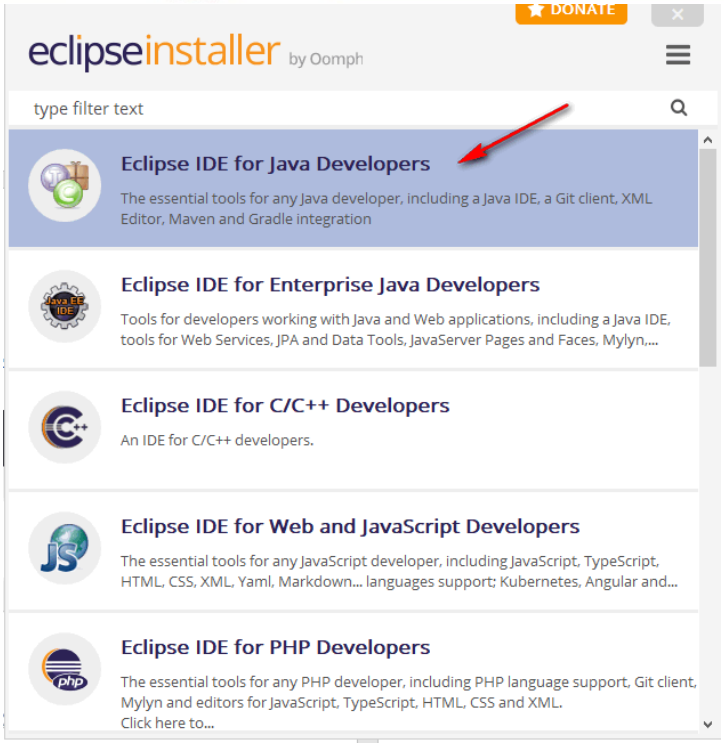


Una vez descargado, abrimos un Explorador de Windows y localizamos el fichero descargado llamado **eclipse-inst-jre-win64.exe**. Una vez localizado, para instalarlo pulsamos doble click sobre el fichero para abrir el asistente de instalación:

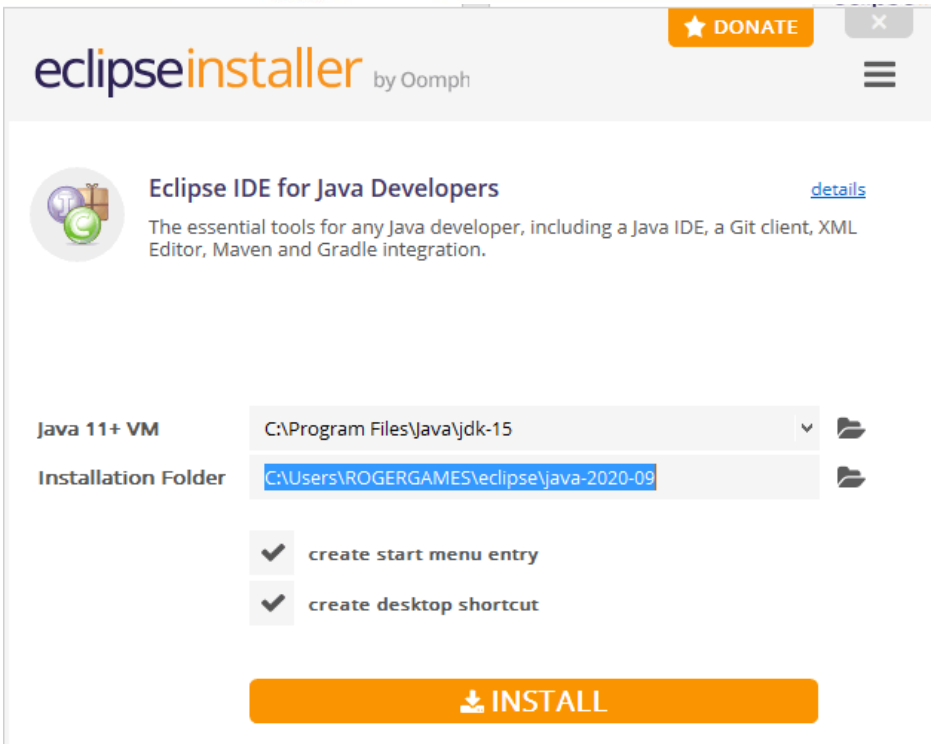


Pulsamos sobre la opción **Eclipse IDE for Java Developers** que trae las herramientas necesarias para desarrollar aplicaciones Java.

Nota: Si quisiéramos desarrollar Aplicaciones Web, necesitaríamos descargar la versión *Eclipse IDE for Java EE Developers*:

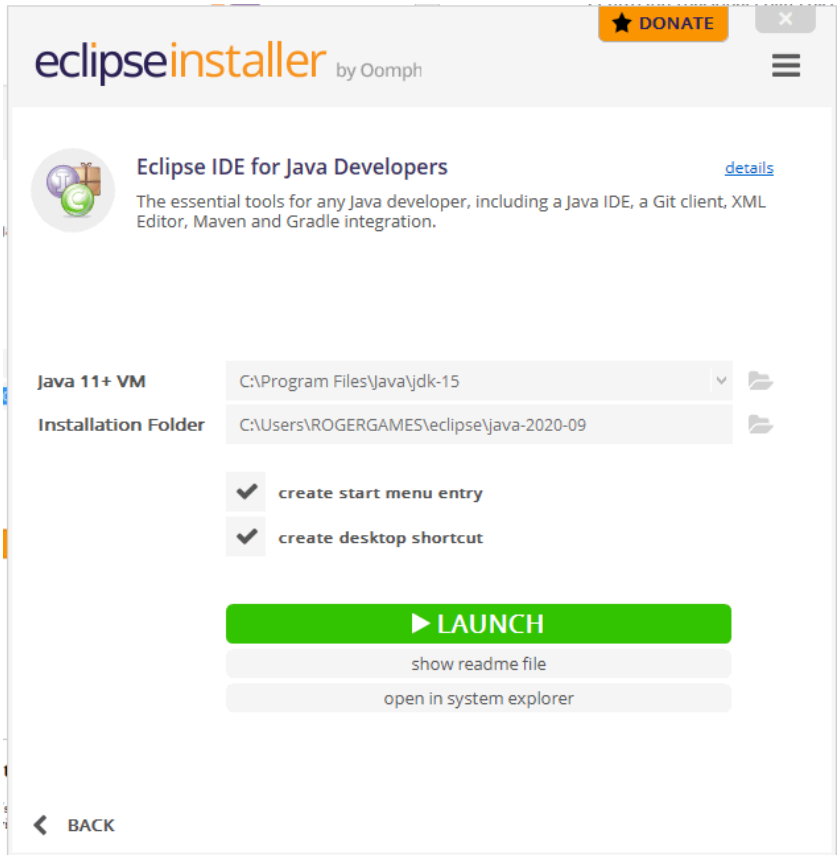


Elegimos la ubicación donde vamos a instalar Eclipse. En este caso no vamos a dejar la ubicación que viene por defecto, sino que lo instalaremos en la ubicación *C:\Users\<usuario>\eclipse\java-oxygen*, donde usuario corresponde al usuario del sistema con el que hemos iniciado sesión en nuestro equipo. Posteriormente pulsamos en el botón *INSTALL*:

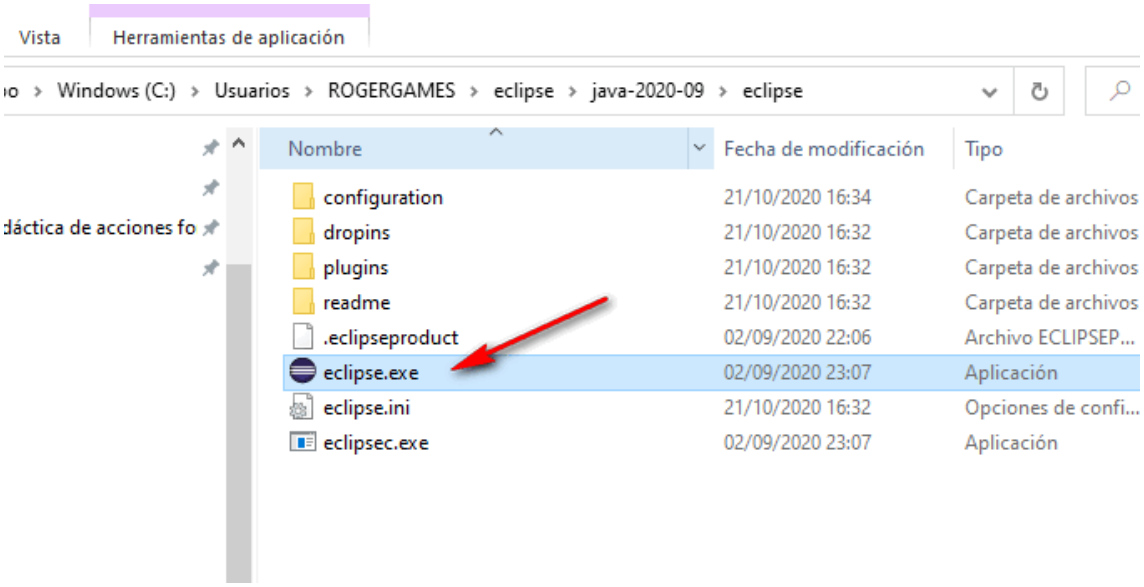


Aceptamos los términos:

Una vez finalizado el proceso de instalación debería verse la siguiente pantalla. Pulsamos el botón X situado en la esquina superior derecha. Debería haberse creado un icono en el escritorio que nos permita ejecutar la aplicación o darle click en el botón Launch.



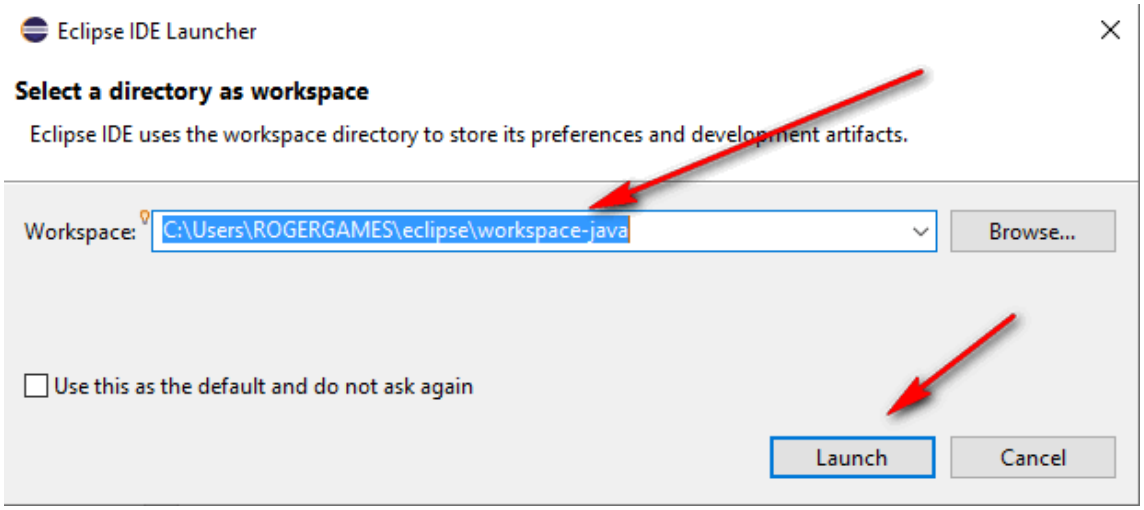
Abrimos un Explorador de archivos y localizamos la carpeta de instalación C:\Usuarios\<usuario>\eclipse\java-2020-09\eclipse. Aquí tenemos el ejecutable:



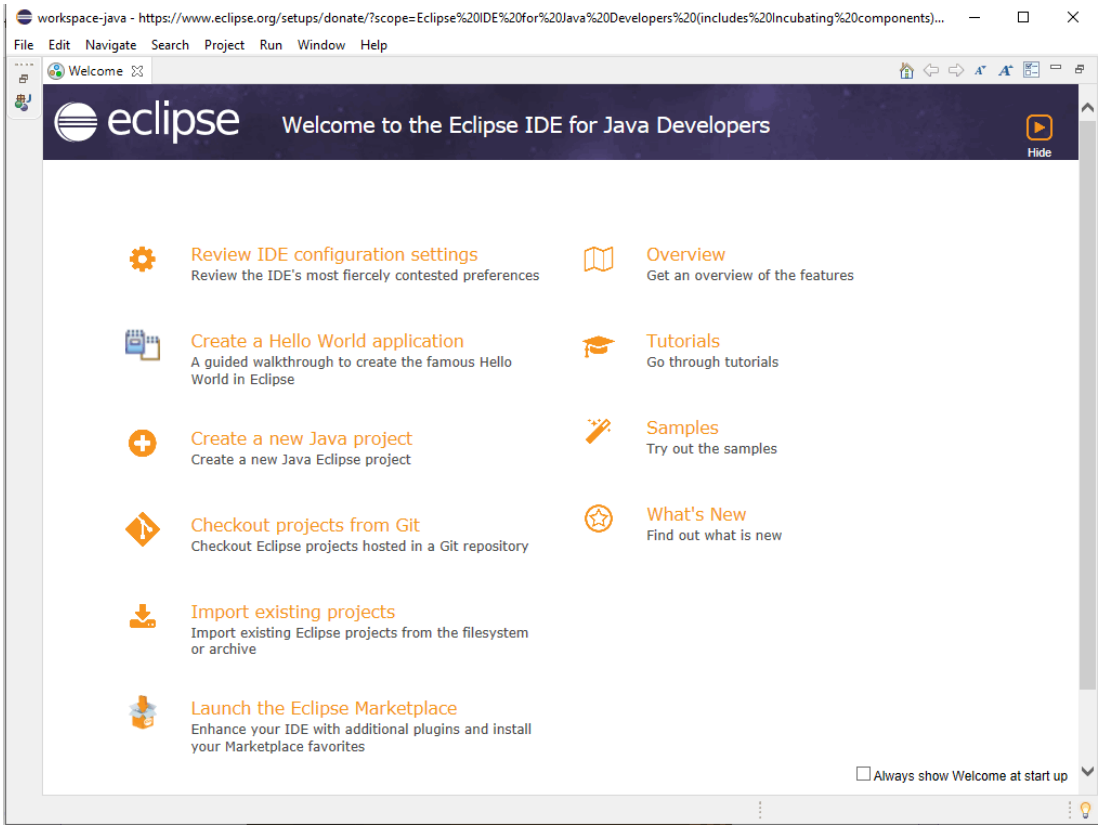
1.4.1. Configurar Java en Eclipse

Abrimos la aplicación Eclipse descargada en el apartado anterior (Deberíamos tener en el escritorio un enlace a la aplicación).

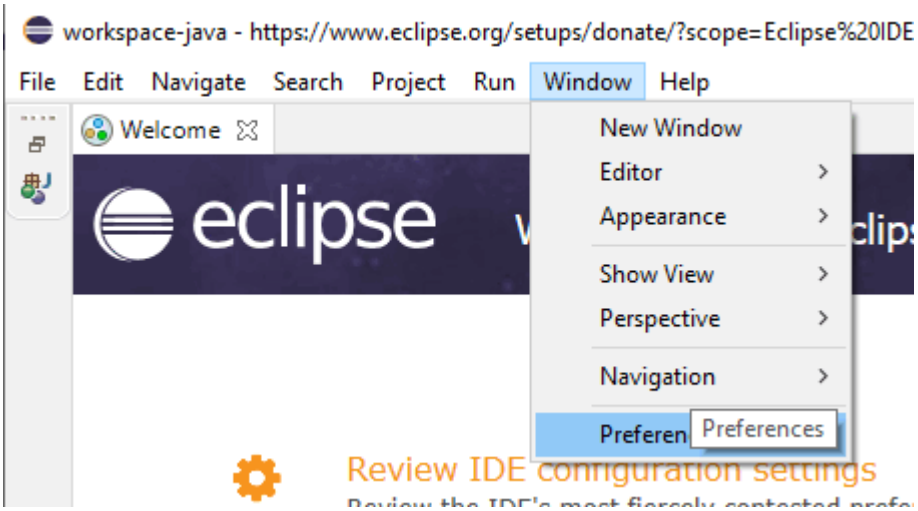
Modificamos la ubicación para el Workspace que viene por defecto, en este caso C:\Users\<usuario>\eclipse\workspace-java.



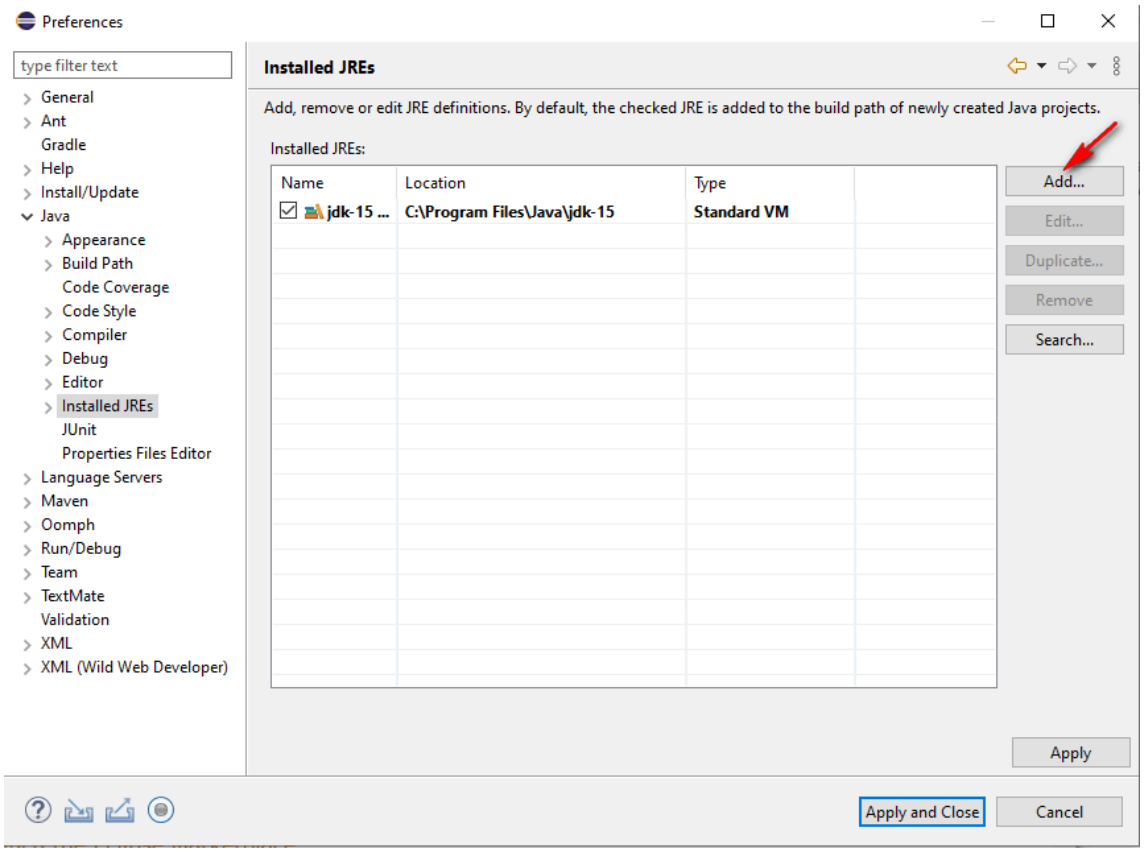
Esta es la pantalla principal del IDE Eclipse.



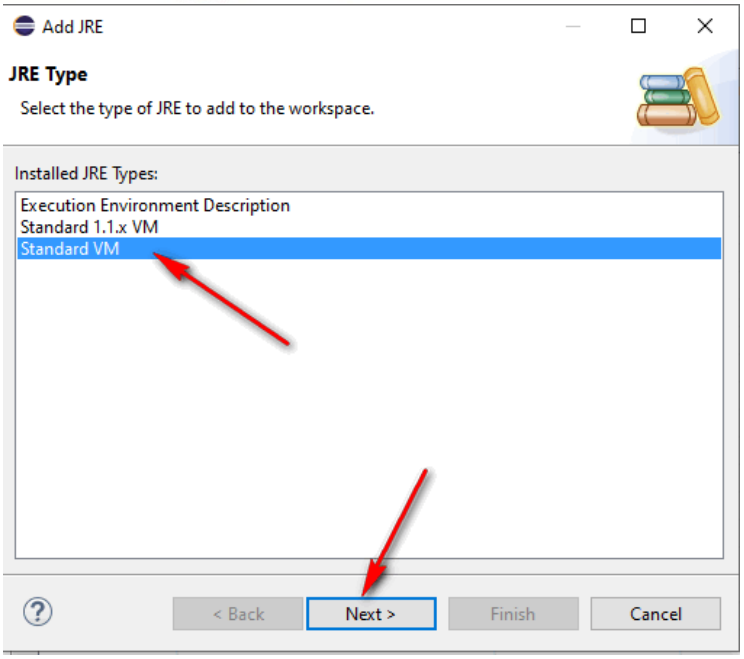
Configurar el JDK instalado en el paso anterior, para ello abrimos el menú Window - Preferences y pulsamos Enter:



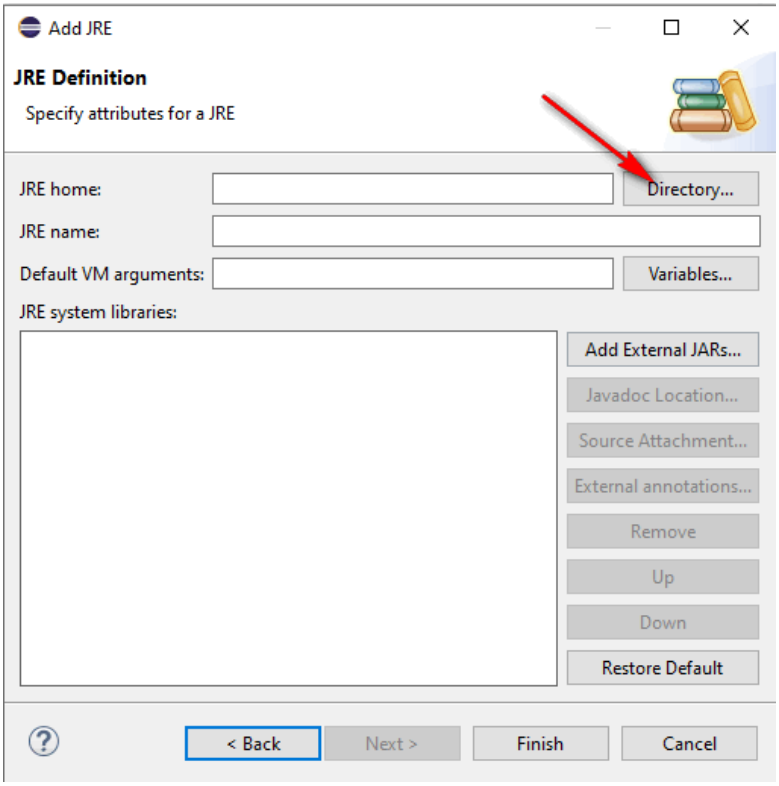
Pulsamos el botón Add... para añadir una nueva instalación de las JDK:



En Installed JRE Types seleccionamos Standard VM y pulsamos el botón Next:

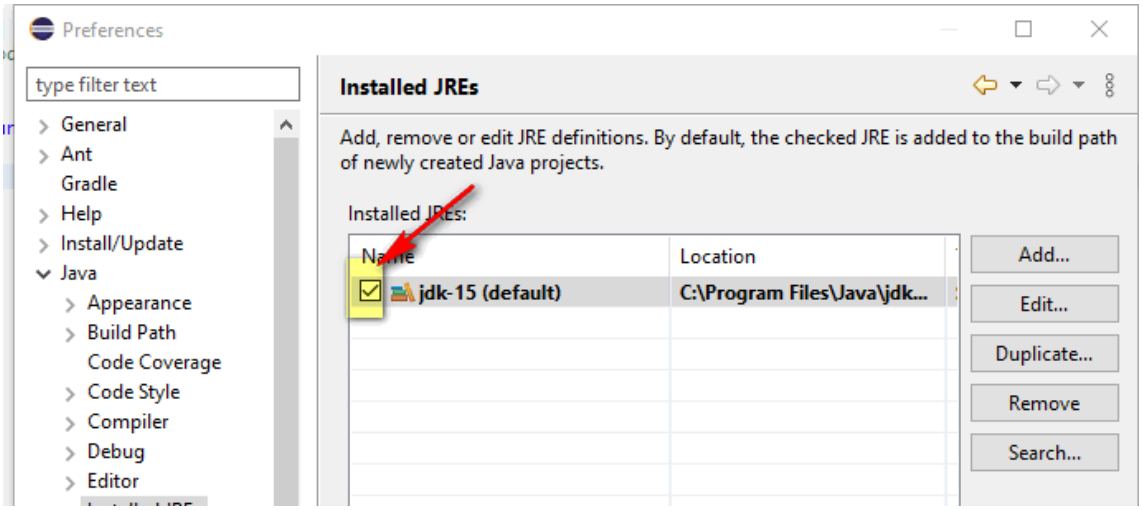


Pulsamos en el botón Directory y en la ventana del Explorador de archivos que se abre buscamos la ruta donde se encuentra instalada las JDK, en este caso C:\Program Files\Java\jdk-15

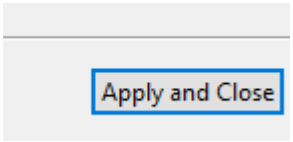


Vemos que se han completado los campos **JRE home** con *C:\Program Files\Java\jdk-15* y **JRE name** con *jdk-15*. Pulsamos el botón *Finish*:

Marcamos la opción jdk-15, para que sea la versión JDK por defecto:



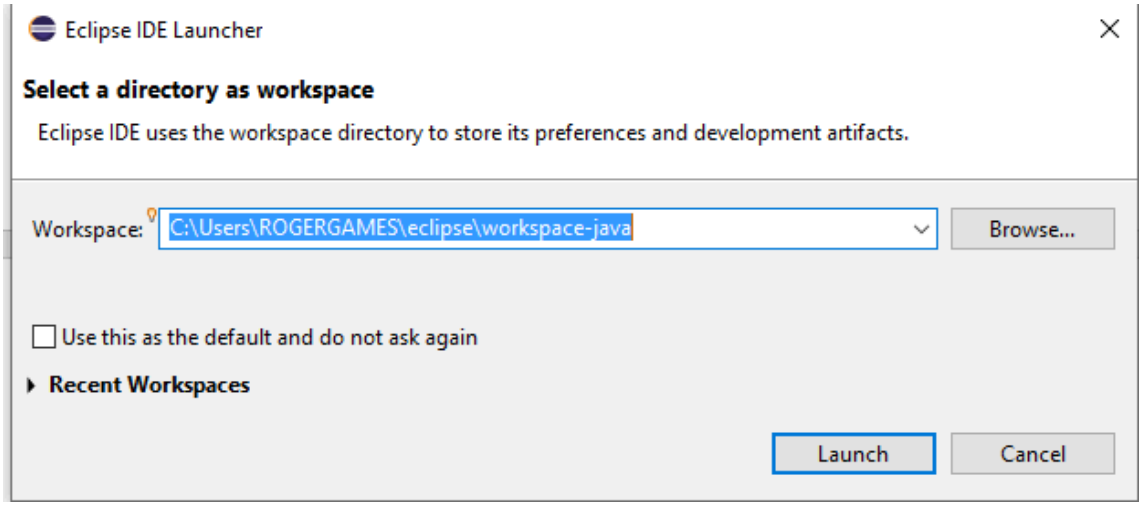
Una vez marcada la opción jdk-15 debe aparecer en negrita y pulsamos el botón Apply and Close:



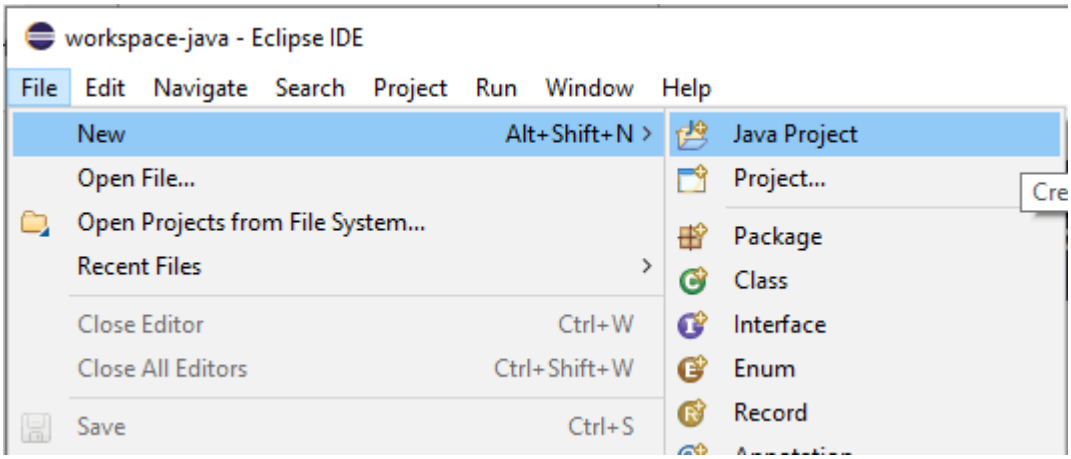
¡Listo! Ya hemos instalado el IDE Eclipse.

1.5. ANEXO II. CREAR UNA APLICACIÓN EN JAVA UTILIZANDO ECLIPSE

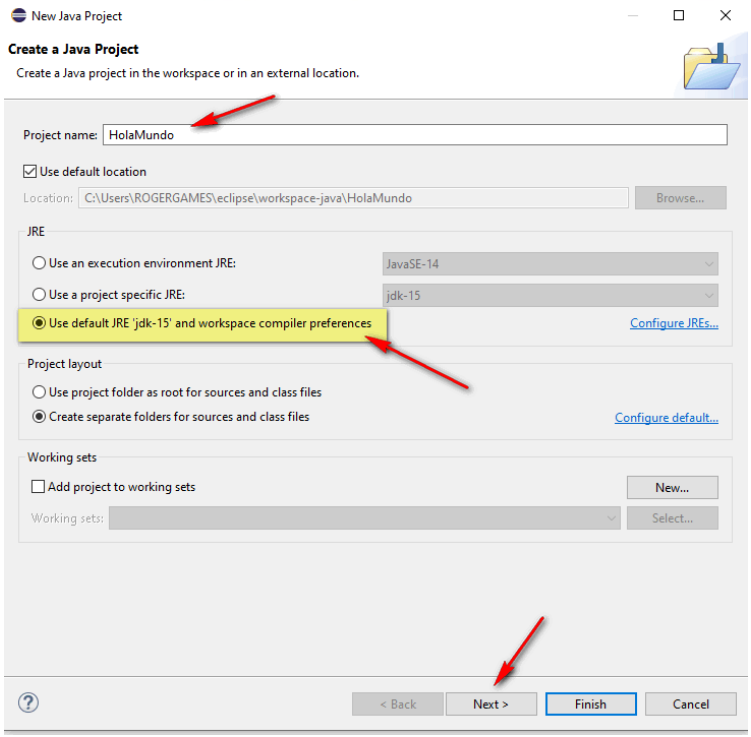
Ejecutamos el *Eclipse* y seleccionamos el *Workspace* creado anteriormente **java-workspace** y pulsamos el botón *Launch*:



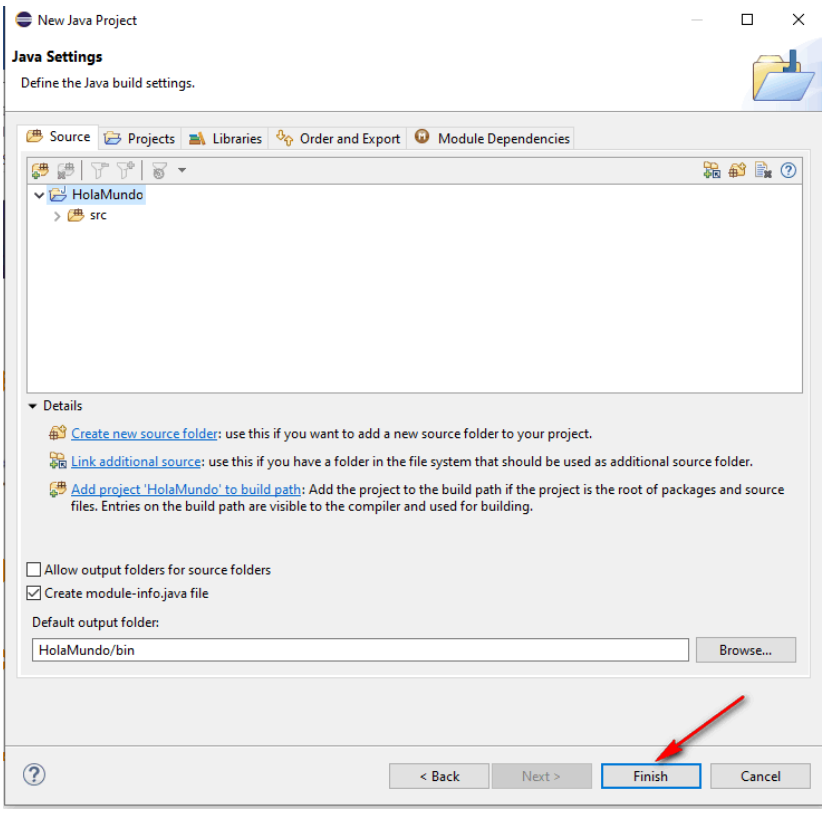
Para crear un nuevo proyecto, vamos al menú File-New-Java Project:



En el campo *Project name* definimos el nombre del proyecto, en este caso **HolaMundo**. Posteriormente seleccionamos las JDK que instalamos anteriormente y pulsamos el botón *Next*:

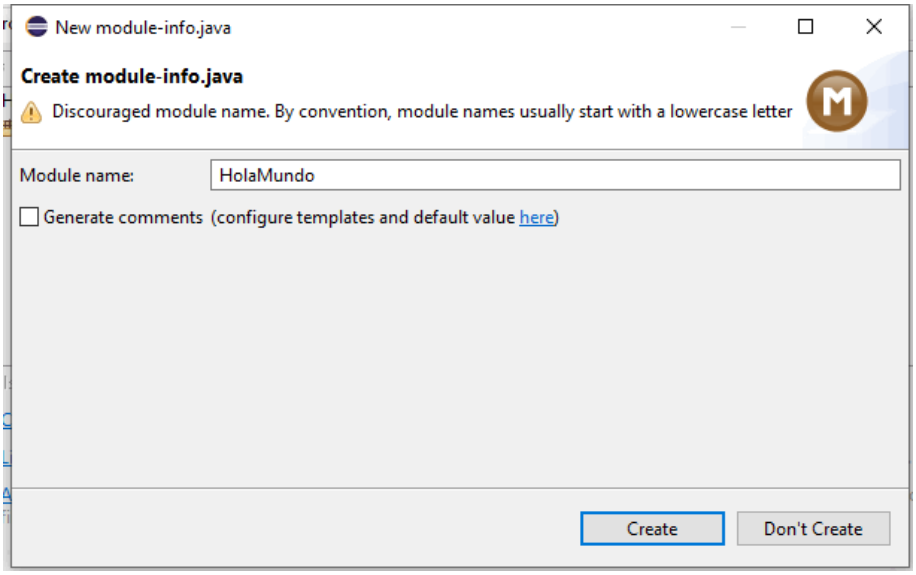


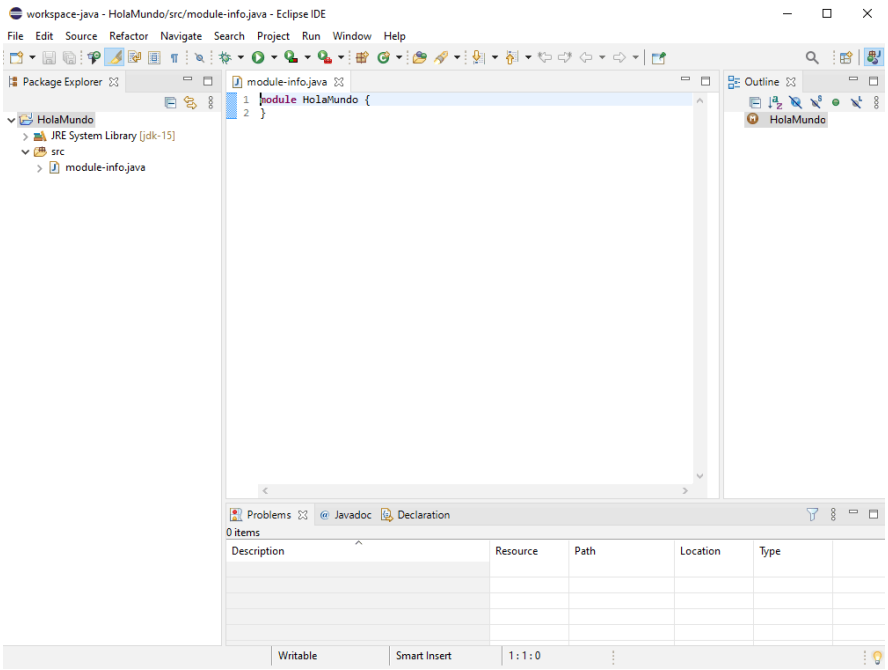
En esta ventana del asistente se pueden añadir a nuestro proyecto librerías externas, pero como no es el caso pulsamos el botón *Finish*:

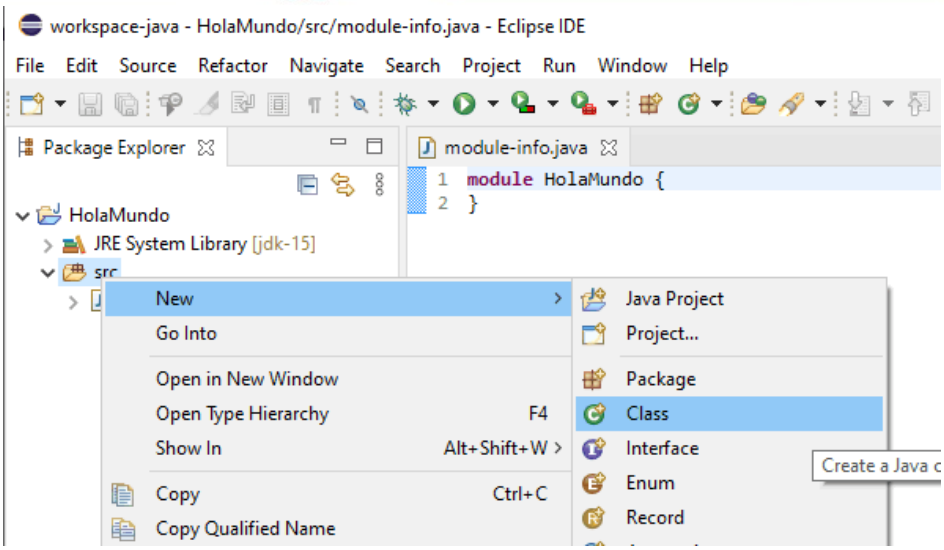


En el panel izquierdo, en la pestaña *Package Explorer* podemos ver el nuevo proyecto creado. Por defecto crea una carpeta llamada *src* donde va a ir el código de nuestra aplicación.

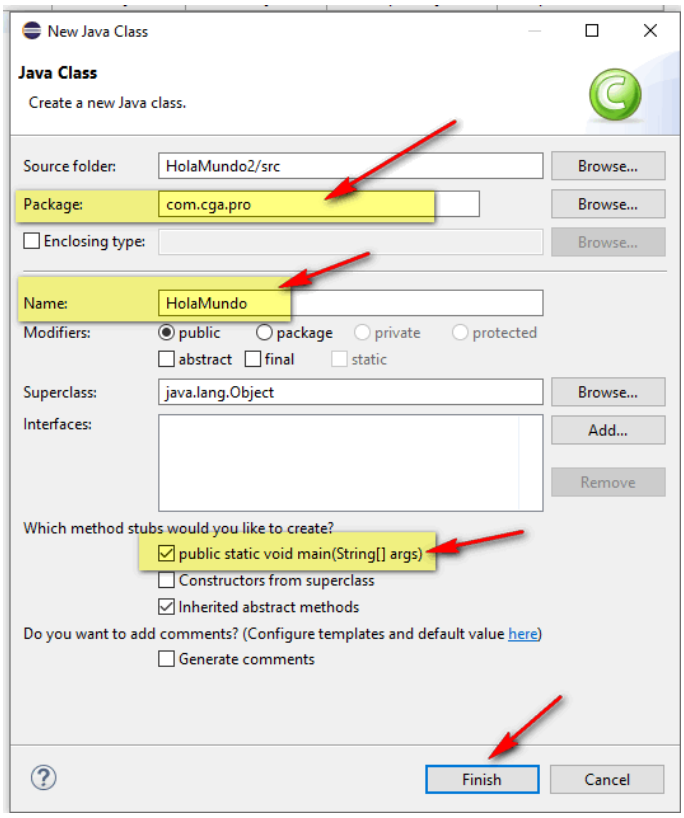
Para crear la clase principal, pulsamos botón derecho sobre la carpeta *src* y luego seleccionamos la opción del menú contextual *New-Class*:





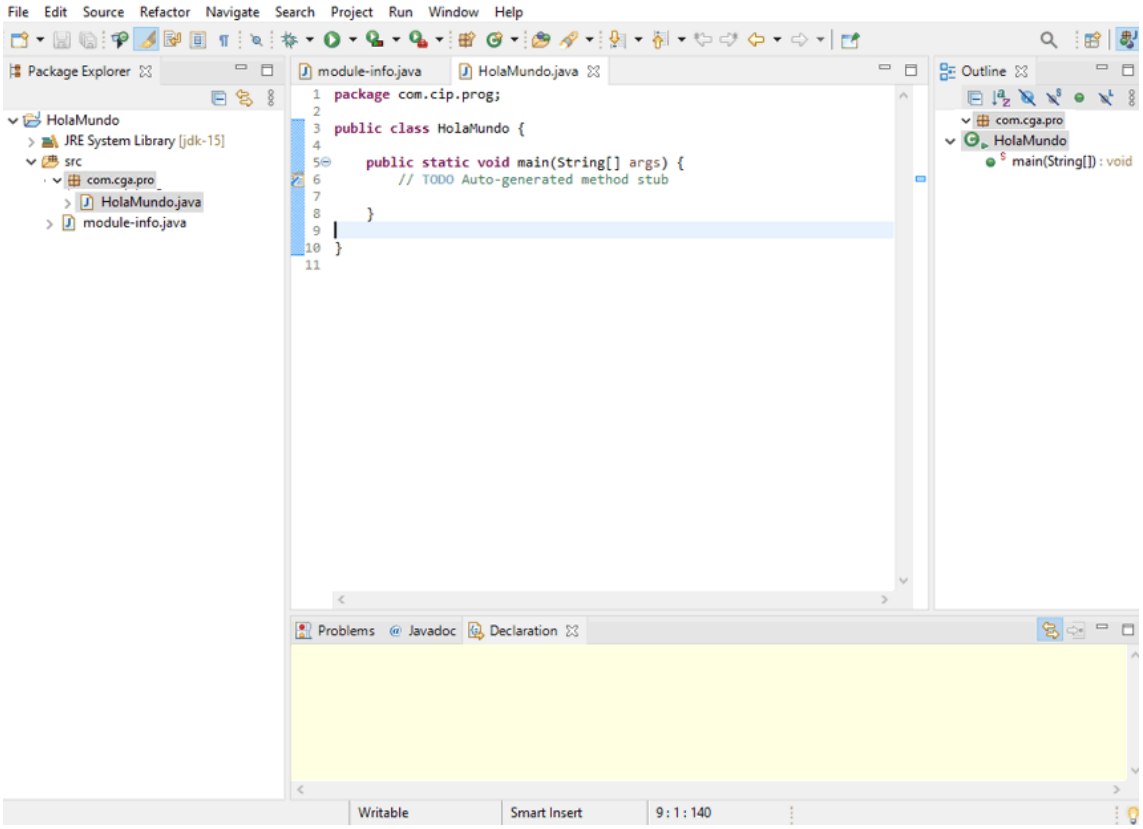


En esta ventana del asistente, en el campo Package debemos definir el paquete donde se creará la clase, para este caso hemos **com.cga.prog**. En el campo *Name* definimos el nombre de la clase principal, en este caso **HolaMundo**. Marcamos la opción *public static void main(String[ ] args)* para que dentro de la clase cree el método *main()*. Pulsamos en el botón *Finish*:



En el panel izquierdo, en la pestaña *Package Explorer*, vemos cómo se ha creado la clase *HolaMundo* dentro del paquete *com.cga.prog* en la carpeta *src* del proyecto.

En la ventana principal podemos ver el código fuente que ha generado Eclipse. Vemos cómo ha generado el método *main()* vacío sin ninguna sentencia:



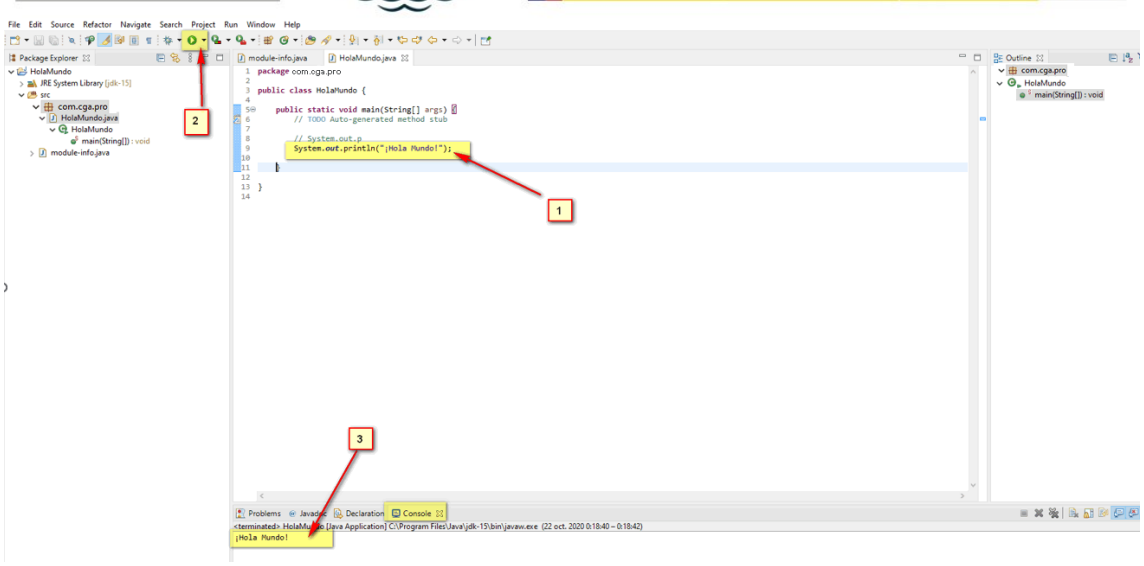
Dentro del método *main()* añadimos la siguiente sentencia:

System.out.println("¡Hola Mundo!");

Esta línea imprimirá por consola el siguiente mensaje:

texto ¡HolaMundo!.

Guardamos el fichero y pulsamos en el botón *Run* para ejecutar la aplicación:



Si todo va bien podremos ver en el panel inferior, en la pestaña Console el mensaje ¡Hola Mundo!.

2. Bibliografía

- Material suministrado por el CGA
- https://desarrolloweb.com/home/java
- https://www.arquitecturajava.com/
- https://www.monografias.com/docs114/estructura-programa-java/estructura-programa-java.shtml
- https://javaparajavatos.wordpress.com/2015/09/28/presentacion/