



Escuela de Ingeniería Informática



UNIVERSIDAD DE OVIEDO

INTRODUCCIÓN A ECLIPSE

Metodología de la programación
Curso 2021-2022

¿Qué es Eclipse?

- **Eclipse** es una comunidad de código abierto que se centra en la construcción de una *plataforma compuesta por marcos extensibles (Frameworks)* y herramientas para la construcción, despliegue y gestión del software.
- La **Fundación Eclipse**, es una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios y servicios.
- **Proyectos eclipse**, existen multitud de proyectos:
 - El IDE (Entorno de Desarrollo Integrado) de java (JDK), aplicaciones de modelado, software para dispositivos, herramientas para la generación de informes, etc.

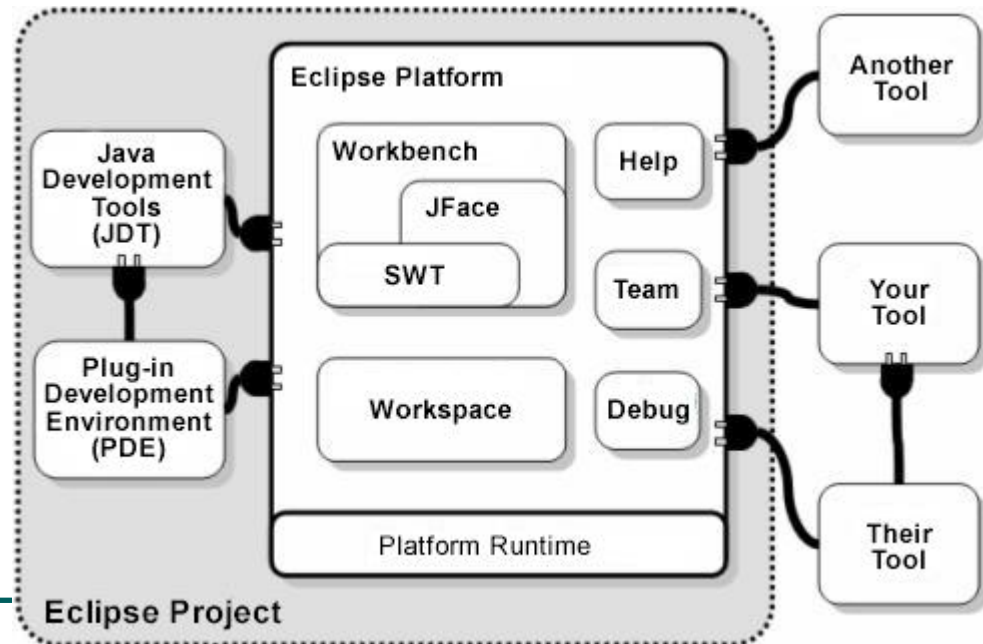
Arquitectura

- Esta basada sobre el concepto de **plug-in**
 - Código que extiende la funcionalidad del IDE.
 - Existen plug-ins para GUIs, pruebas, modelado ...

La **plataforma** está **desarrollada en java**

Soporta diferentes lenguajes:

Java, C/C++, Cobol, PHP, AspectJ, JavaScript ...



Distribución

- Descarga gratuita en la dirección:

<https://www.eclipse.org/downloads/packages/release/2021-03>

Versiones para Windows, Linux y Mac OS X.

- Cada versión (release) se libera anualmente y se identifica con un nombre.

- **2021-12**

- **2021-03**

- **2020-12**

- **2019-12**

- Photon año 2018

- Oxigen año 2017

- Neon año 2016

- Mars año 2015

- Versión de Java Estándar Edition

- **JSE-15**

The screenshot shows the Eclipse IDE 2021-03 R Packages download page. The main heading is "Eclipse IDE for Java Developers". Below it, there is a download button with a download icon. To the right of the download button, the supported operating systems are listed: Windows x86_64, macOS x86_64, and Linux x86_64 | AArch64. The page also displays the file size (328 MB) and the number of downloads (1,526,123). Below the main heading, there is a section titled "MORE DOWNLOADS" which lists various Eclipse builds and versions, including Eclipse 2021-12 (4.22), Eclipse 2021-09 (4.21), Eclipse 2021-06 (4.20), Eclipse 2021-03 (4.19), Eclipse 2020-12 (4.18), Eclipse 2020-09 (4.17), Eclipse 2020-06 (4.16), and Older Versions.

Entorno de trabajo- Workbench

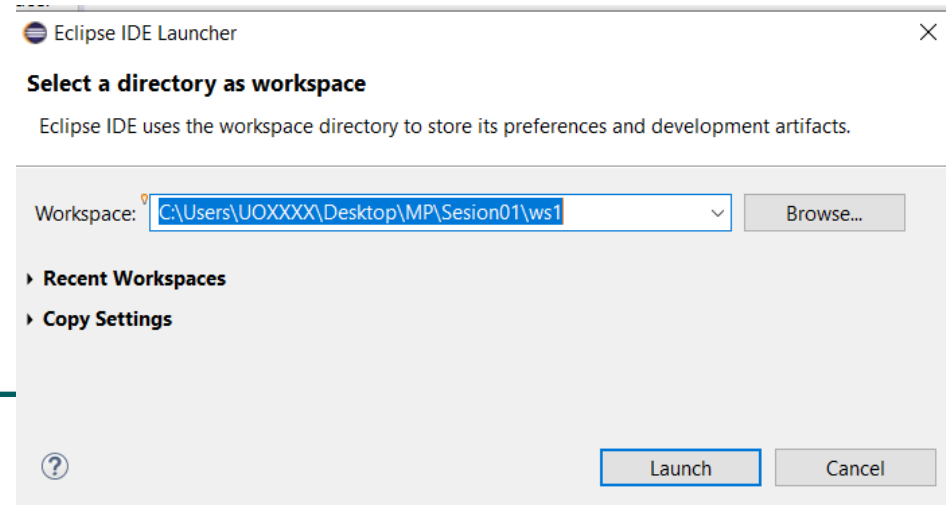
- El IDE (Integrated Development Environment) incluye:
 - Explorador de ficheros.
 - Editor.
 - Compilador.
 - Depurador.
- También incluye herramientas de ayuda al programador:
 - Refactorización (Refactoring)
 - Generación de código (Code generation)
 - Pruebas (Testing)

Workspace (Espacio de trabajo)

- Un **workspace** es una carpeta donde se almacena un conjunto de proyectos
- Guarda el estado de cada proyecto .
- Se pueden mantener tantos espacios de trabajo como se quiera.
- **Usaremos un workspace para cada sesión de laboratorio**
- Llamaremos al espacio de trabajo ws1 para la sesión 1.
- Después de cada clase de laboratorio se deben copiar los proyectos del workspace en un “lapiz” USB o mejor subirlos a OneDrive, Dropbox...

Arrancar el entorno

- Crea la **carpeta Sesión 01** dentro de **carpeta MP** y guarda aquí el material de esta sesión bajado del campus.
- Crea el espacio de trabajo (carpeta ws1) dentro de Sesión 01
`c:\Users\uoXXXXXX\desktop\MP\Sesión 01\ws1`
- Ejecuta Eclipse.
 - eclipse.exe (icono eclipse del escritorio)
- Selecciona el espacio de trabajo.



Entorno

The screenshot displays the Eclipse IDE interface with the following components:

- Package Explorer:** Located on the left, it shows the project structure. The project is named "apellido1_apellido2_nombre_session01_helloworld". It contains a "src" folder with a package "uo.mp.s1.helloworld" which includes the "HelloWorld" class. The class has a method "main(String[]): void".
- Editor:** The central area shows the source code of "HelloWorld.java". The code includes a Javadoc comment for the "main" method, which describes the session and the purpose of the code. The code is as follows:

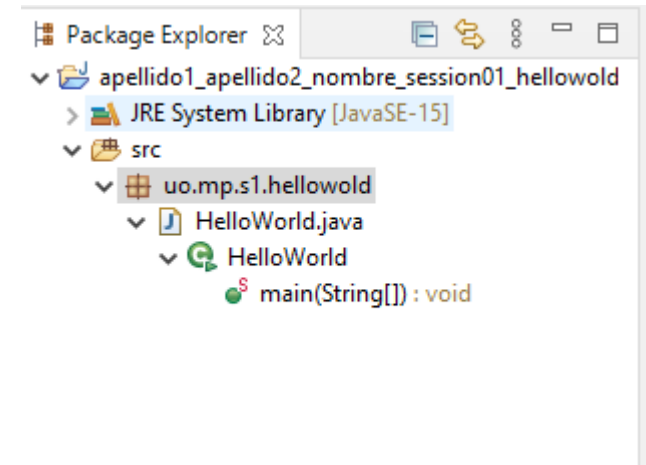
```
2 // **
3 *
4 * @author mp22
5 *
6 */
7 public class HelloWorld {
8
9     /**
10      * Sesión 1. Imprime por pantalla Hola Mundo
11      *
12      * @param args
13      * Los parámetros de entrada no se usan en este caso
14      */
15
16     public static void main(String[] args) {
17         System.out.println("Hola Mundo!");
18     }
19
20 }
```
- Outline:** Located on the right, it shows the class hierarchy. It lists the package "uo.mp.s1.helloworld" and the class "HelloWorld". The "main" method is listed as "main(String[]): void".
- Console:** At the bottom, it shows the output of the program, which is "Hola Mundo!".

Four blue boxes with arrows point to these components, labeled as follows:

- Explorador de paquetes:** Points to the Package Explorer.
- Editor:** Points to the central code editor.
- Perspectivas:** Points to the Outline view.
- Vistas:** Points to the Console view.

Package Explorer

- Muestra el contenido del workspace (espacio de trabajo) actual.
 - Contiene un conjunto de proyectos
 - Cada proyecto contiene diversos paquetes y librerías.
 - Los paquetes contienen clases que tienen relación entre si.
- Está sincronizado automáticamente con el sistema de ficheros.
- **Presionar F5** para actualizar la sincronización.

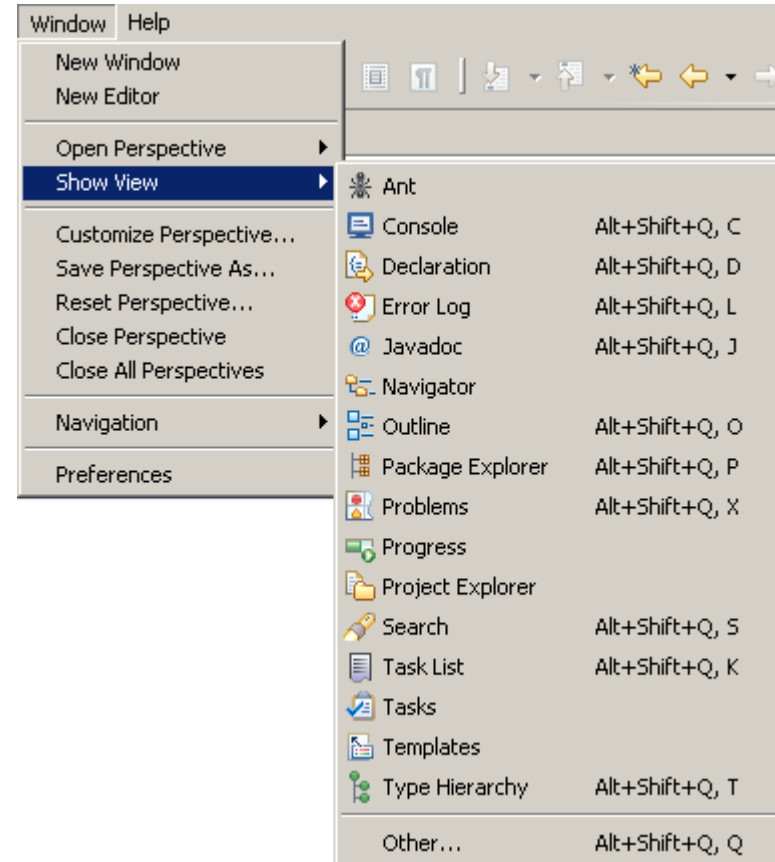


Paquete

- Un **paquete** puede ser definido como un **contenedor de tipos relacionados** (clases, interfaces, enumeraciones y anotaciones).
- Los componentes del mismo paquete están relacionados entre sí; por ejemplo, están enfocados a una función común (interfaz con el usuario, lógica de negocio, interfaz con bases de datos....)
- Los paquetes también **proporcionan protección de acceso** y gestión del espacio de nombres.

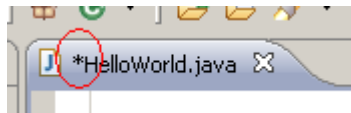
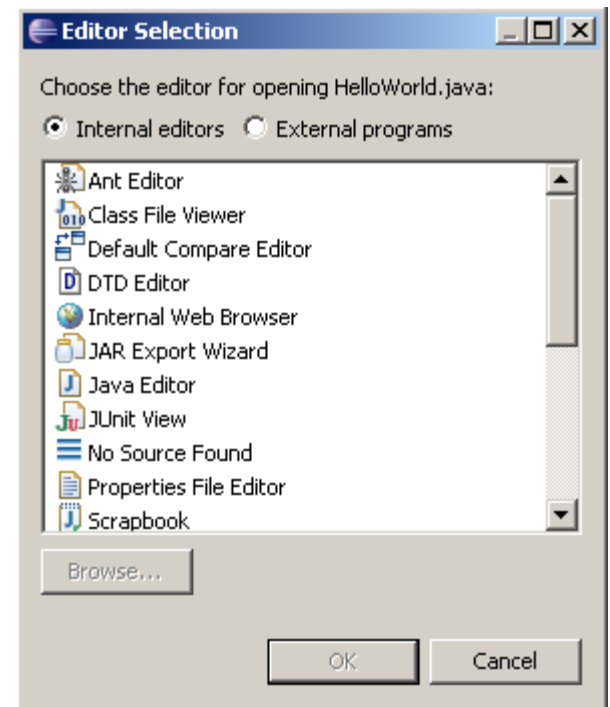
Vistas

- Es una ventana que muestra algo concreto del proyecto.
 - Consola de Java.
 - Errores de compilación.
 - etc...
- Se puede abrir cualquier vista desde el menú.
- Aparecen agrupadas y solo se puede ver una.
- No permite modificar.



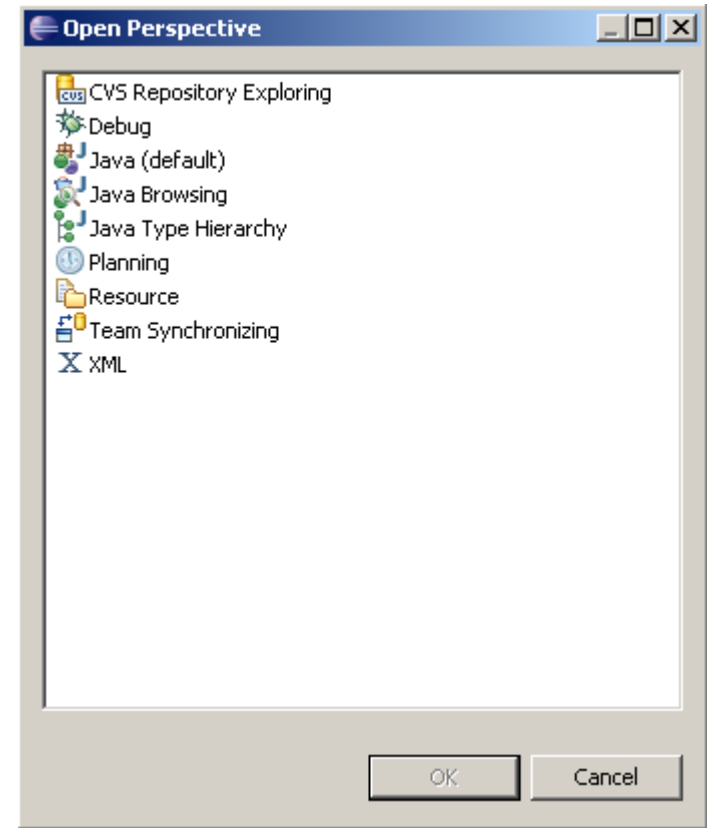
Editores

- Similares a las vistas.
- Múltiples editores, gráficos o de texto.
- Asociados por la extensión.
 - Configurable desde las preferencias.
- Es posible forzar el uso del editor que queramos.
- Un “*” al lado del nombre del archivo => sin salvar.



Perspectivas

- Adaptan el entorno para una tarea de alto nivel.
- Seleccionan un conjunto de vistas, editores y barras de herramientas adecuados a la tarea.
 - **Java:** Desarrollo en java.
 - **Debug:** Depuración.
 - **CVS:** Control de versiones.
 - etc...
- En cualquier momento se puede cambiar la perspectiva.



CÓMO SE CREA UN NUEVO PROYECTO

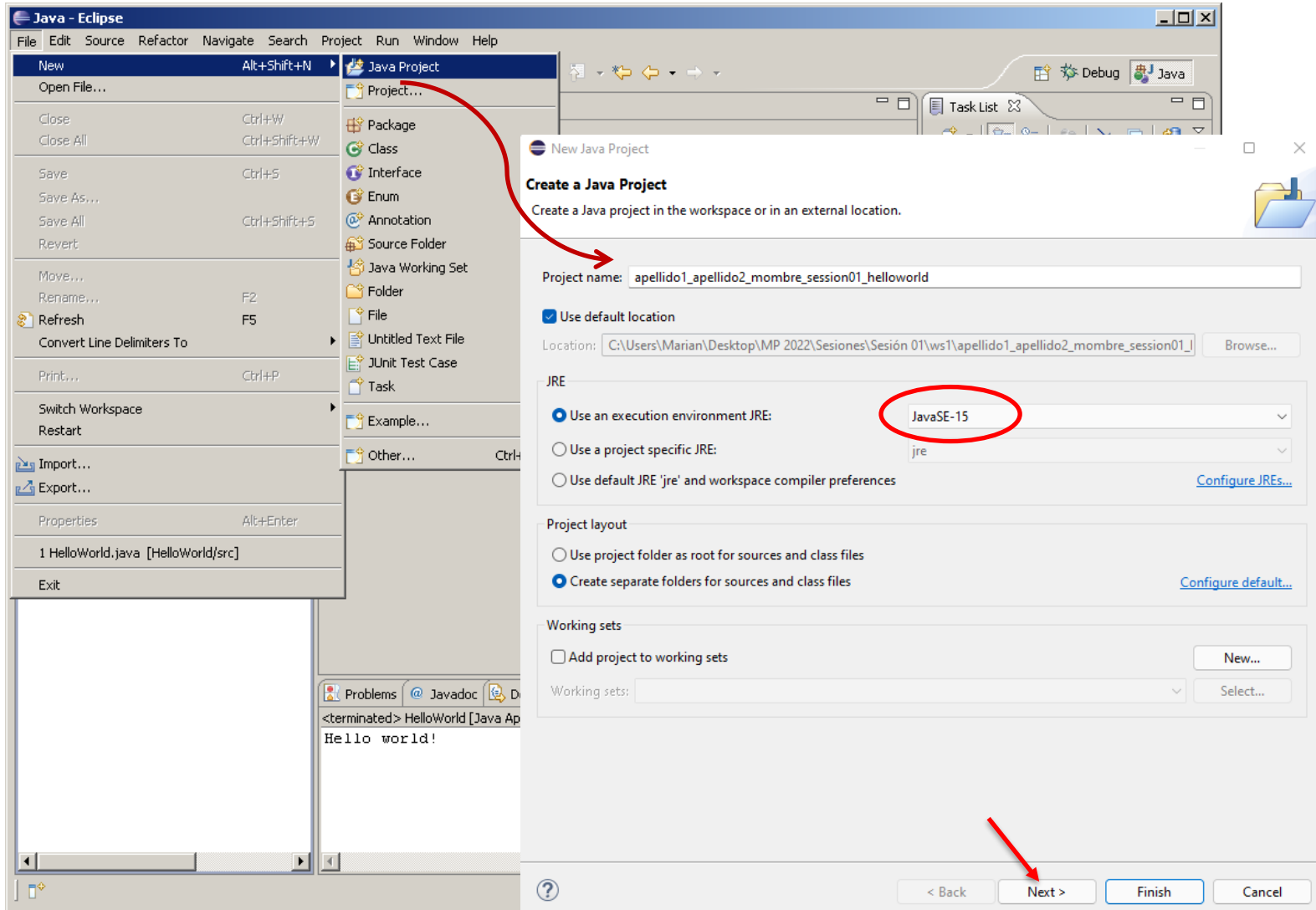
Organización y nomenclatura

- **Carpeta para la asignatura: MP**
- **Carpeta para cada sesión del curso (semanal): Sesión 01**
Sesión 01, Sesión02...Sesión 10, (para que queden ordenadas)
 - *Guarda toda la documentación que se baje del campus y los proyectos que se hagan.*
- **Espacio de Trabajo para la sesión: ws1**
 - *Carpeta que guarda todos los proyectos que se realicen*
- **Nombre de los proyectos**
 - *apellido1_apellido2_nombre_session01_helloworld*
*Todo en **minúsculas** y en **inglés***
 - *apellido1_apellido2_nombre_session01_task_game*
*Las tareas (a entregar) llevan la palabra **task***
- **Nombre del paquete**
 - *uo.mp.s1.game.model (siempre uo.mp. Además sesión, proyecto, paquete)*
 - *Su último nombre depende del contenido. Se suelen incluir paquetes como **model** , **service**, **etc**. Según la función*
 - *Ejemplo: uo.mp.s1.game.model*

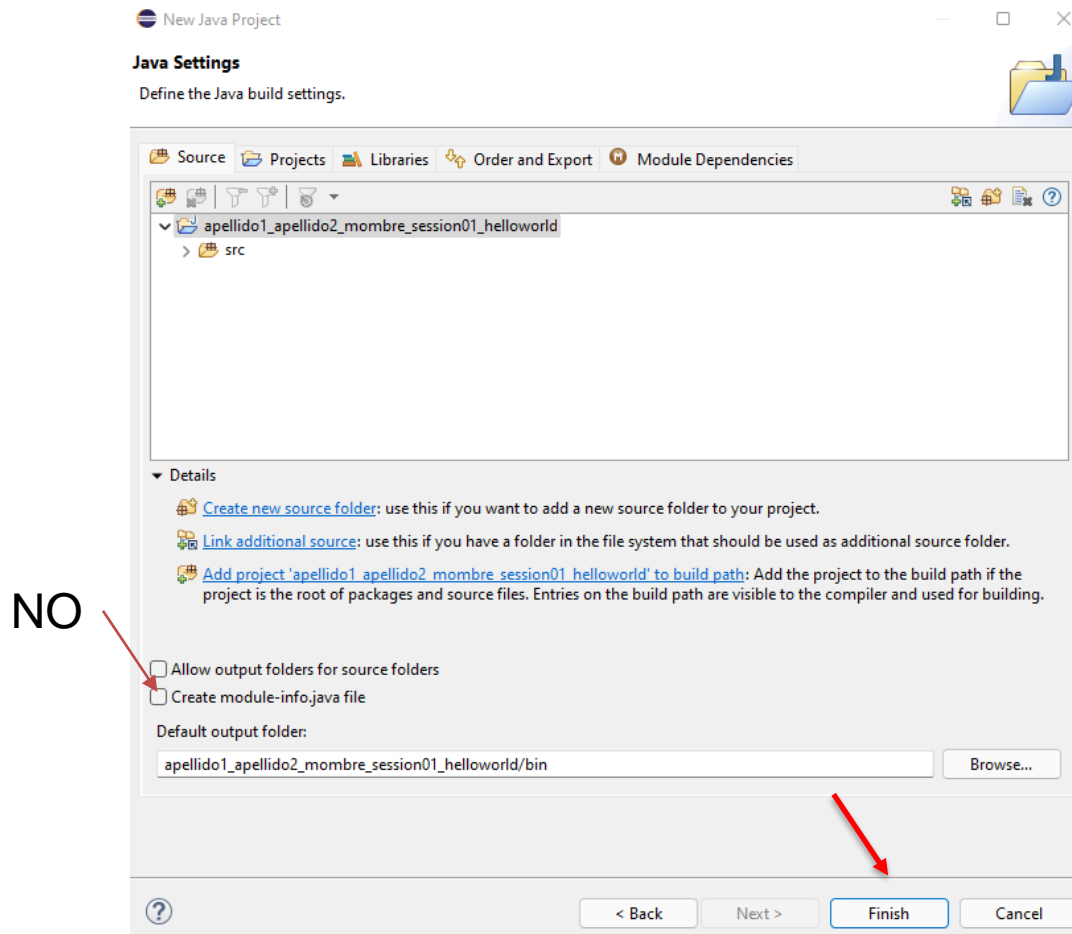
Ejercicio

- Crear un proyecto llamado helloworld que al ejecutarlo muestre por pantalla “Hola Mundo”
 - Se crea un proyecto:
 - apellido1_apellido2_nombre_session01_helloworld
 - Se crea un paquete en la carpeta src
 - uo.mp.s1.helloworld
 - Como el proyecto es muy simple, el nombre del paquete coincide con el nombre del proyecto
 - Se crea una **clase con el método estático Main**
 - HelloWorld (Sigue norma de denominación de clases)

Crear un nuevo proyecto



Crear un nuevo proyecto



Crear un paquete

New Java Package

Java Package
Create a new Java package.

Creates folders corresponding to packages.

Source folder: Browse...

Name:

☐ Create package-info.java

☐ Generate comments (configure templates and default value [here](#))

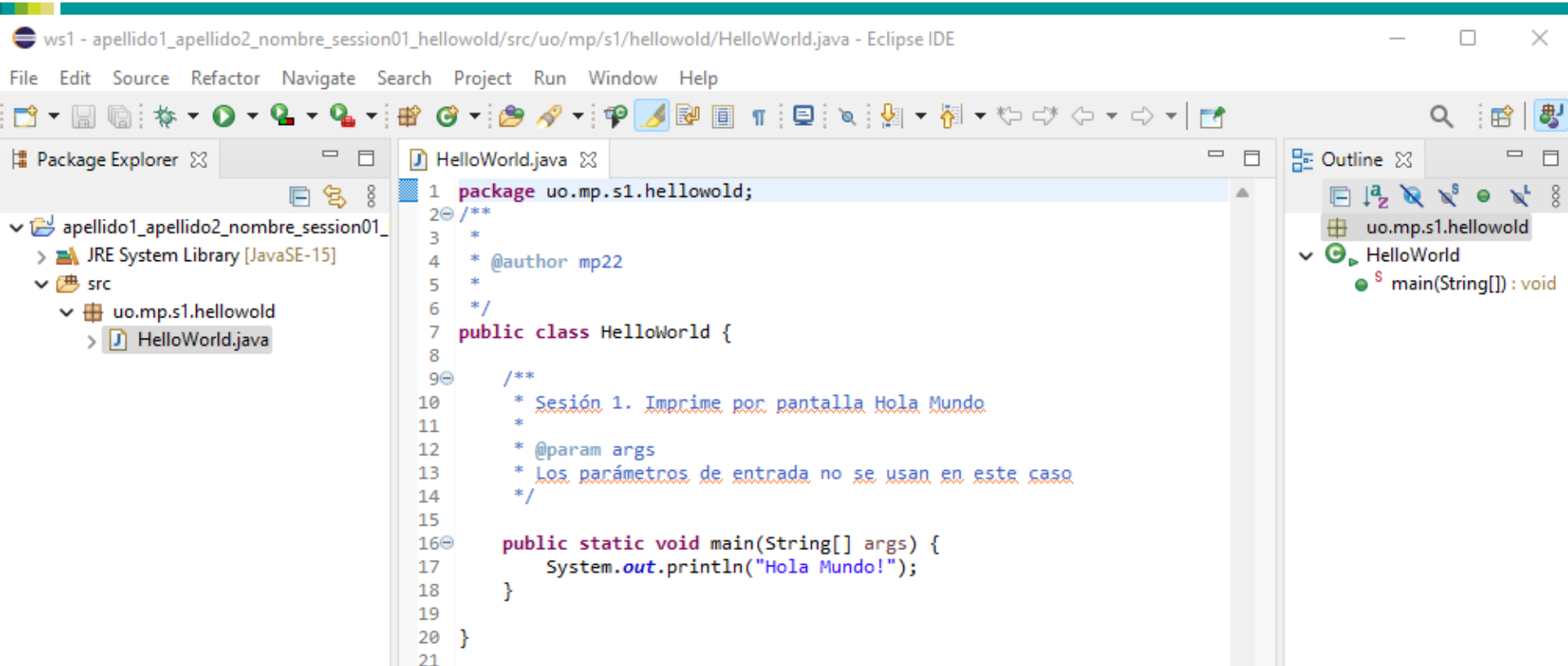
Crear una clase

The image shows the Eclipse IDE interface. On the left, the 'File' menu is open, and the 'Class' option is highlighted. A red arrow points from this option to the 'New Java Class' dialog box on the right.

New Java Class Dialog:

- Title:** New Java Class
- Source folder:** apellido1_apellido2_nombre_session01_helloworld/src
- Package:** uo.mp.s1.helloworld
- Enclosing type:** (empty)
- Name:** HelloWorld
- Modifiers:** ☒ public, ☐ package, ☐ private, ☐ protected, ☐ abstract, ☐ final, ☐ static
- Superclass:** java.lang.Object
- Interfaces:** (empty)
- Which method stubs would you like to create?:** ☒ public static void main(String[] args), ☐ Constructors from superclass, ☒ Inherited abstract methods
- Do you want to add comments?:** (Configure templates and default value [here](#)), ☐ Generate comments

Editar para añadir el código



Utilizar parámetros

The screenshot shows the Eclipse IDE interface. At the top, the title bar indicates the file path: `ws1 - apellido1_apellido2_nombre_session01_hellowold/src/uo/mp/s1/hellowold/HelloWorld.java - Eclipse IDE`. The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations and development tools. The Package Explorer on the left shows the project structure, including a package named `uo.mp.s1.hellowold`. The main editor displays the `HelloWorld.java` file with the following code:

```
1 package uo.mp.s1.hellowold;  
2  
3
```

Below the editor, the Run Configurations dialog is open, showing the configuration for the `hellowWord` application. The dialog has tabs for Main, Arguments, JRE, Classpath, Source, Environment, and Common. The Main tab is selected, and the Program arguments field contains the text `Pedro`. The VM arguments field is empty. The Working directory is set to `Default` with the value `${workspace_loc:pedro_suarez_alonso_session1_hellowWorld}`. The dialog also includes buttons for Variables..., Workspace..., File System..., and Variables... at the bottom. The Run button is highlighted.

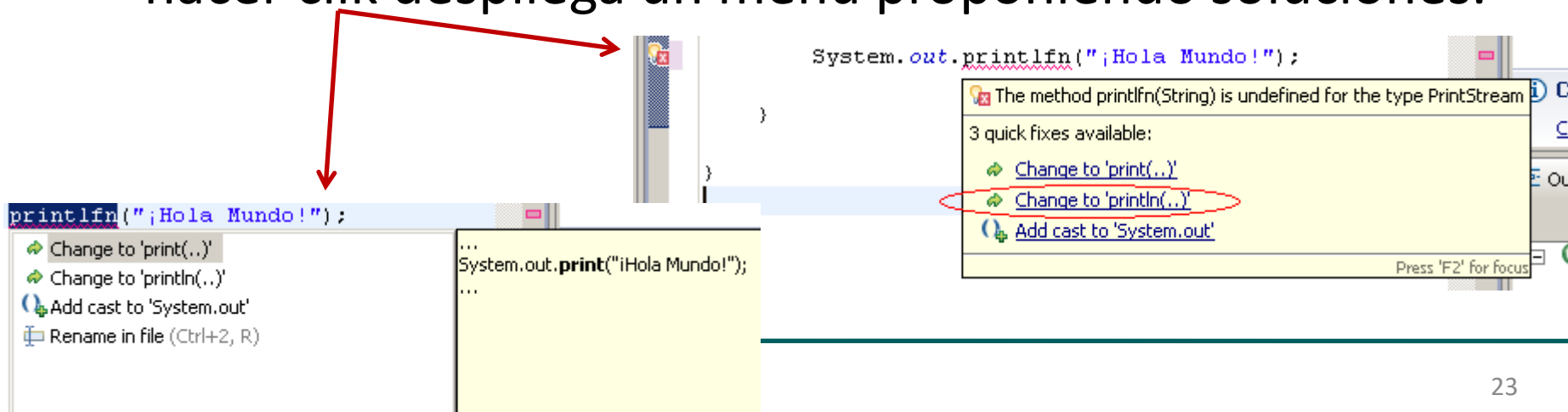
To the right of the Run Configurations dialog, the `hellowWorld` class is shown with the following code:

```
hellowWorld {  
  
    public void main(String[] args) {  
        out.println("Hola Mundo!, me llamo args[0]");  
    }  
}
```

The `main` method signature and its body are circled in red, highlighting the use of parameters.

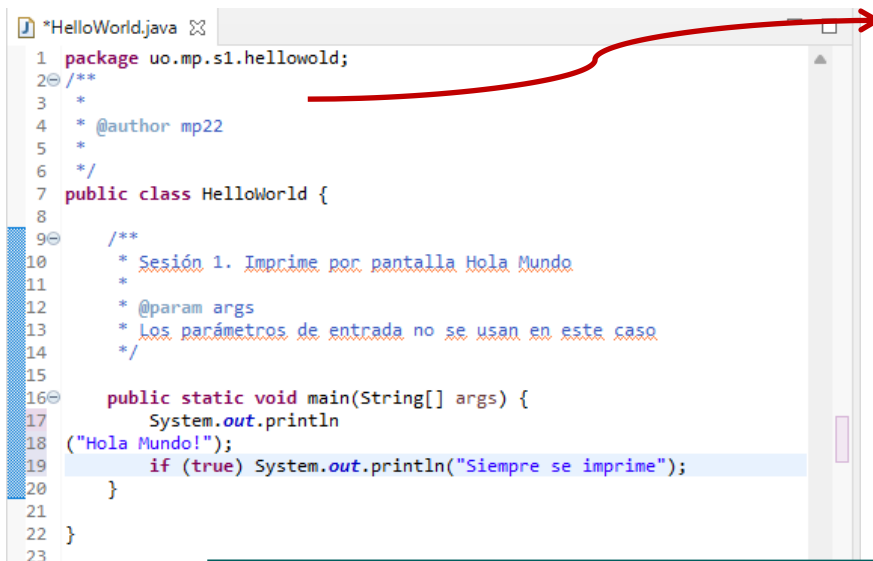
Detección y corrección de errores

- Similar a un corrector ortográfico.
- Subraya el error con una línea roja ondulada (amarilla si es un warning).
- Situando el ratón sobre la línea muestra una descripción del error y puede proponer soluciones.
- A la izquierda muestra un icono de error en el que al hacer clic despliega un menú proponiendo soluciones.

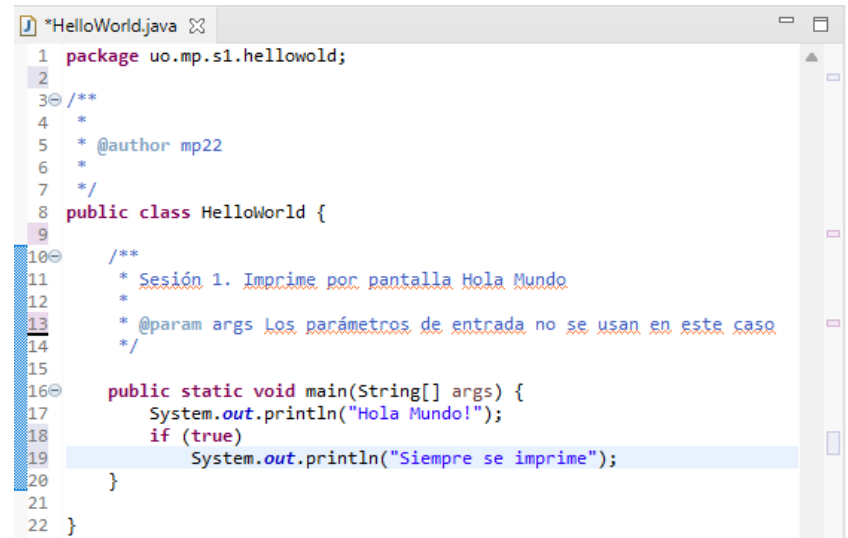


Manipulación del código fuente

- Opciones agrupadas en el menú “Source”
- Source → Format : Formatea e indenta el código fuente.
- Shortcut: Control + Shift + F



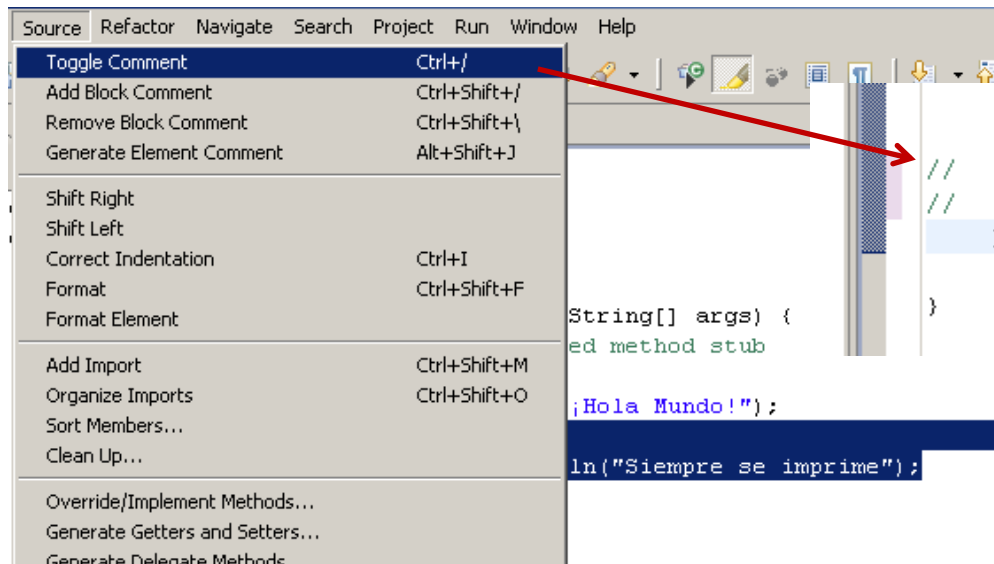
```
1 package uo.mp.s1.helloworld;
2 /**
3  *
4  * @author mp22
5  */
6
7 public class HelloWorld {
8
9     /**
10      * Sesión 1. Imprime por pantalla Hola Mundo
11      *
12      * @param args
13      * Los parámetros de entrada no se usan en este caso
14      */
15
16     public static void main(String[] args) {
17         System.out.println
18         ("Hola Mundo!");
19         if (true) System.out.println("Siempre se imprime");
20     }
21
22 }
23
```



```
1 package uo.mp.s1.helloworld;
2
3 /**
4  *
5  * @author mp22
6  */
7
8 public class HelloWorld {
9
10     /**
11      * Sesión 1. Imprime por pantalla Hola Mundo
12      *
13      * @param args Los parámetros de entrada no se usan en este caso
14      */
15
16     public static void main(String[] args) {
17         System.out.println("Hola Mundo!");
18         if (true)
19             System.out.println("Siempre se imprime");
20     }
21
22 }
```


Código fuente: Comentar bloques

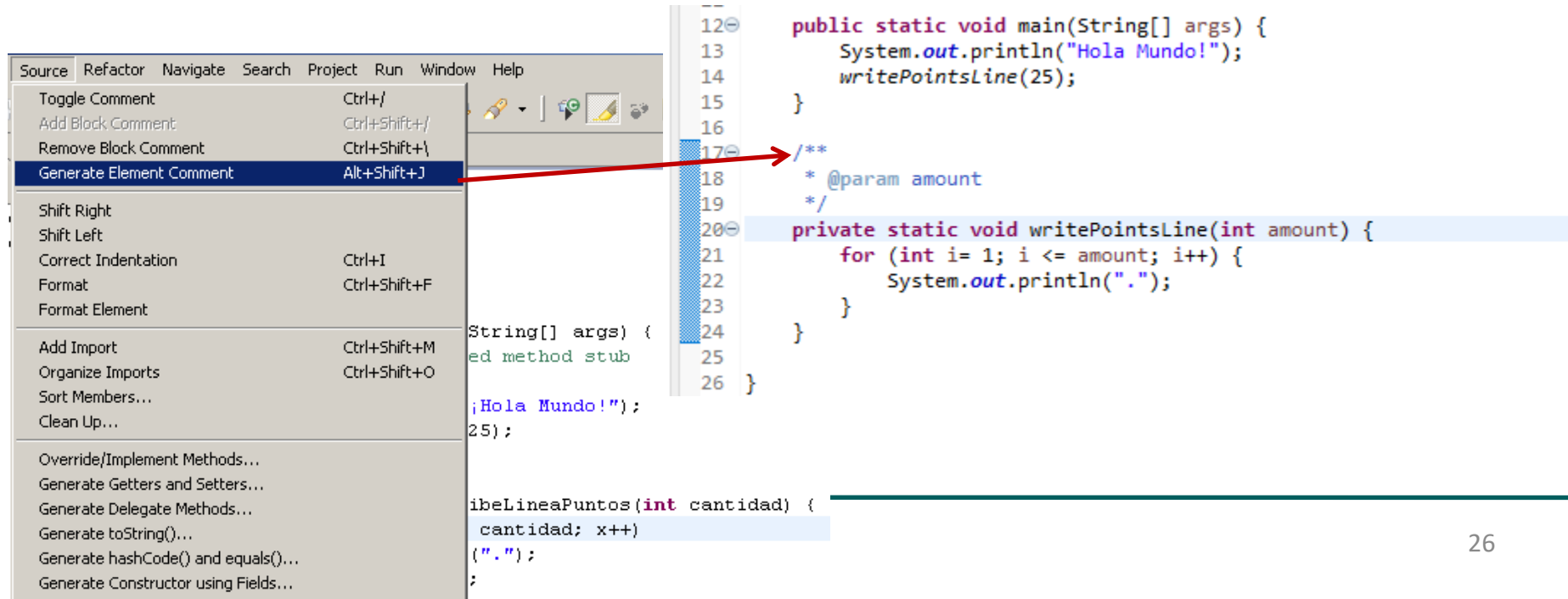
- Comentar y (des) comentar bloques de código previamente seleccionados.
- Source → Toggle Comment : Utiliza `//`
Shortcut: **Control + /**
- Source → Add Block Comment: Utiliza `/* ... */`



```
// System.out.println(";Hola Mundo!");  
// if (true)  
//     System.out.println("Siempre se imprime");  
}  
  
String[] args) {  
    ed method stub  
  
    ;Hola Mundo!);  
    ln("Siempre se imprime");  
}
```

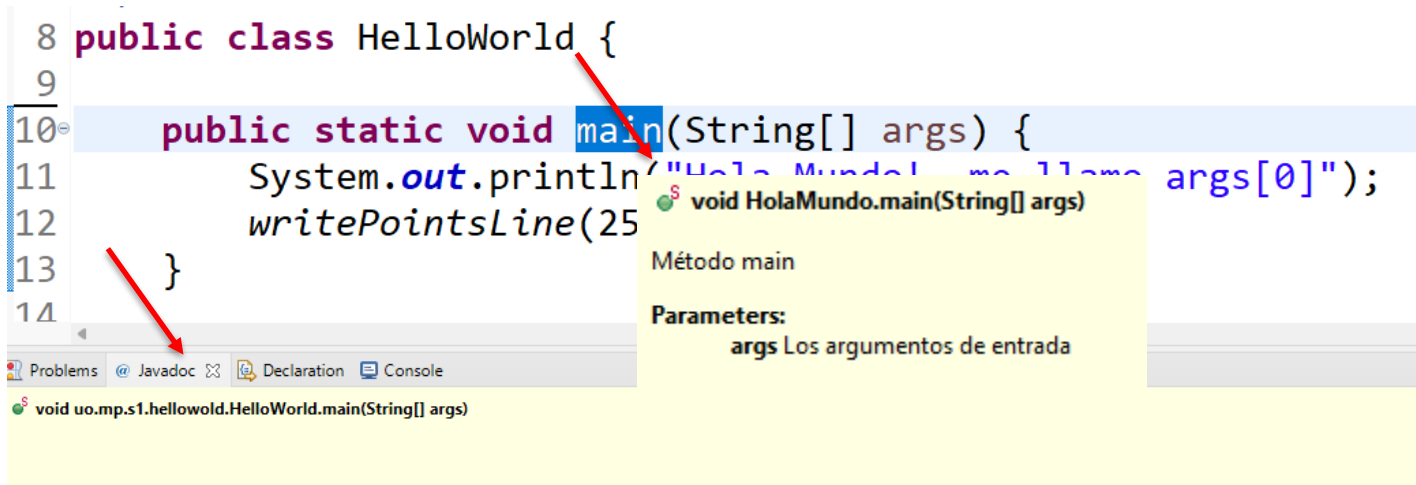
Código fuente: Comentarios Javadoc

- Añadir bloques apropiados para generar Javadoc a un elemento (método o clase).
 - Source → Generate Element Comment
 - o bien `/**` y presionando enter



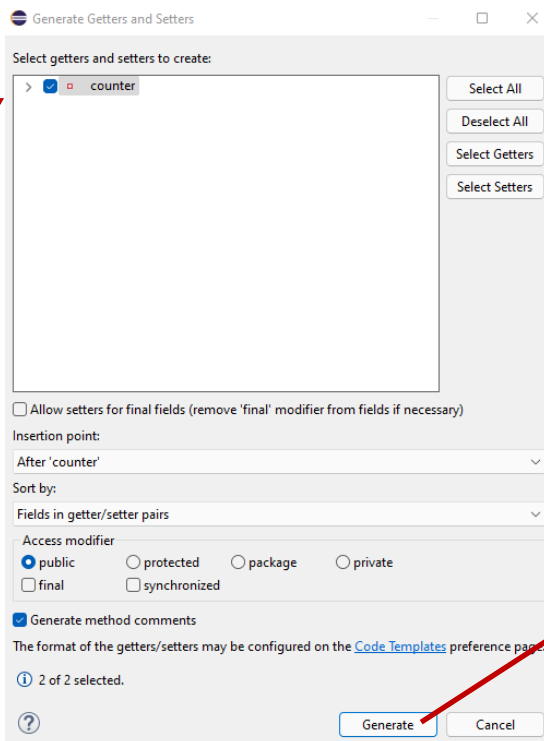
Comentarios Javadoc

- La información de Javadoc es usada:
 - En la **vista Javadoc** cuando se selecciona un elemento
 - En la **ventana emergente** cuando se coloca el ratón sobre un elemento.



Código fuente: Generar código

- Utiliza plantillas para añadir código. Ver `Source`
- Por ejemplo para generar métodos get y set:
 - `Source` → `Generate Getters and Setters ...`



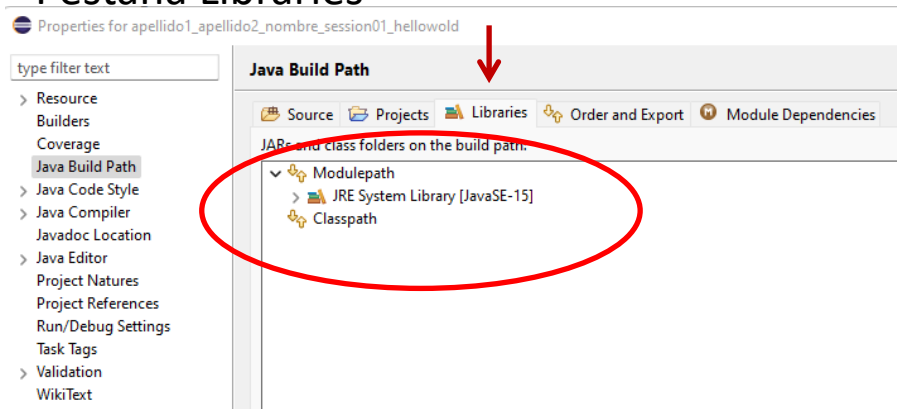
```
private int counter;

/**
 * @return the counter
 */
public int getCounter() {
    return counter;
}

/**
 * @param counter the counter to set
 */
public void setCounter(int counter) {
    this.counter = counter;
}
```

Configuración del proyecto

- Asegurarse que tiene la versión de Java JSE-15
- Sobre el proyecto botón derecho
 - build Path /configure build path
- Pestaña Libraries

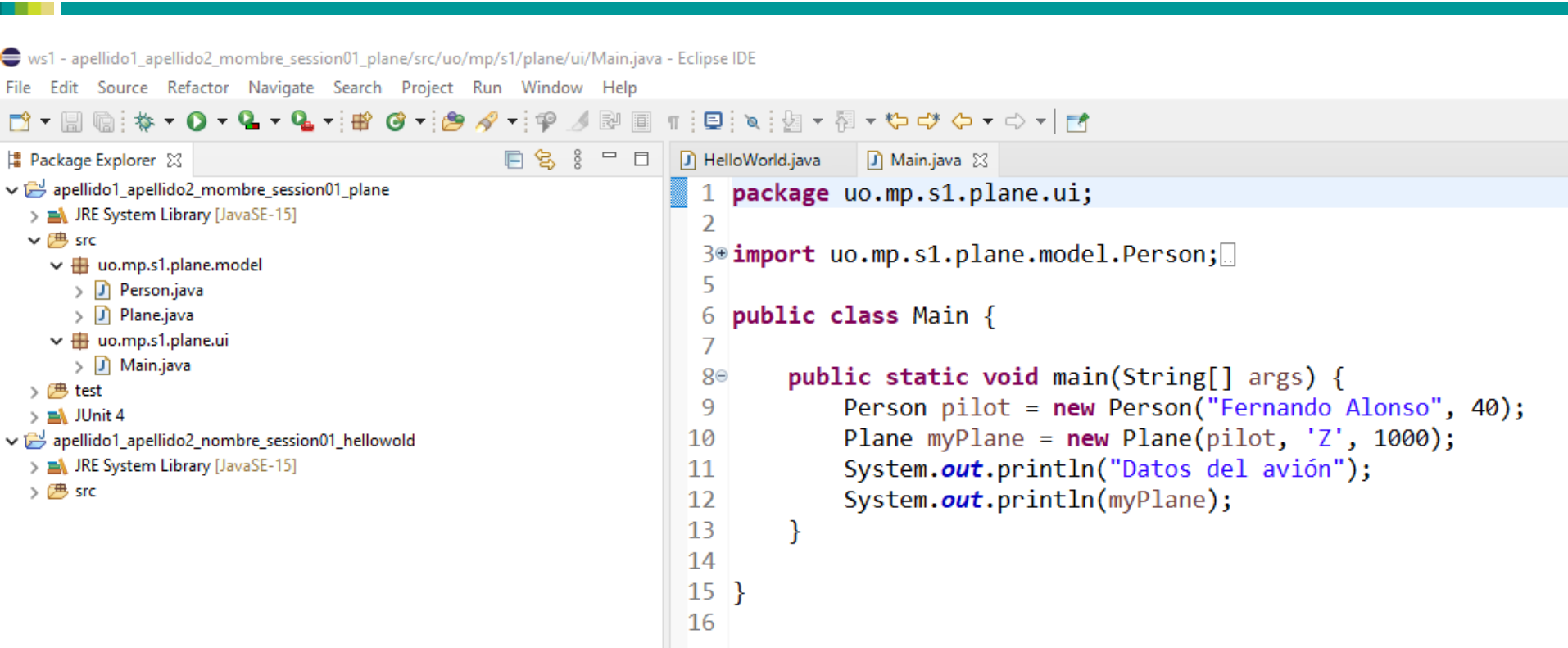


Si no fuera esa la que está incluida, se borra y se añade la 15

Proyecto plane

- Creamos un nuevo proyecto para cargar las clases Plane y Person desarrolladas en IP y usarlas.
 - Creamos nuevo proyecto
apellido1_apellido2_nombre_session01_plane
 - Creamos paquete `uo.mp.s1.plane.model`
 - Copiamos en él las clases Plane y Person
 - Creamos paquete `uo.mp.s1.plane.ui`
 - Creamos en él la clase Main con el método estático main
 - Dentro del método main
 - Creamos un piloto
 - Creamos un avión pasando el piloto, un identificador y combustible
 - Imprimimos por consola los datos del avión

Proyecto plane



ws1 - apellido1_apellido2_mombre_session01_plane/src/uo/mp/s1/plane/ui/Main.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

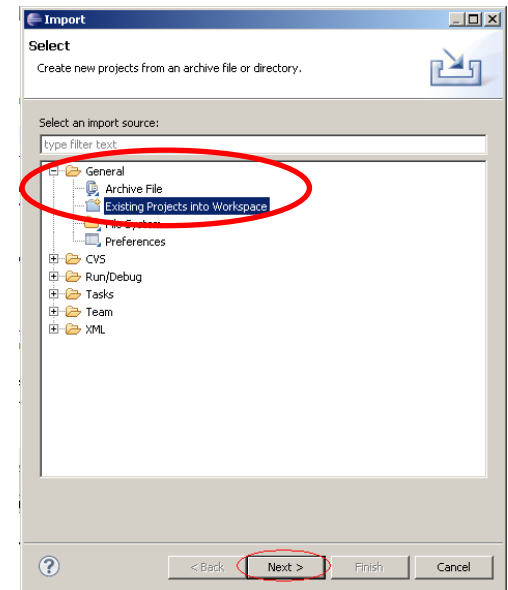
- apellido1_apellido2_mombre_session01_plane
 - JRE System Library [JavaSE-15]
 - src
 - uo.mp.s1.plane.model
 - Person.java
 - Plane.java
 - uo.mp.s1.plane.ui
 - Main.java
 - test
 - JUnit 4
- apellido1_apellido2_nombre_session01_hellowold
 - JRE System Library [JavaSE-15]
 - src

```
1 package uo.mp.s1.plane.ui;
2
3 import uo.mp.s1.plane.model.Person;
4
5
6 public class Main {
7
8     public static void main(String[] args) {
9         Person pilot = new Person("Fernando Alonso", 40);
10        Plane myPlane = new Plane(pilot, 'Z', 1000);
11        System.out.println("Datos del avión");
12        System.out.println(myPlane);
13    }
14
15 }
16
```

CÓMO SE ABRE UN PROYECTO YA CREADO

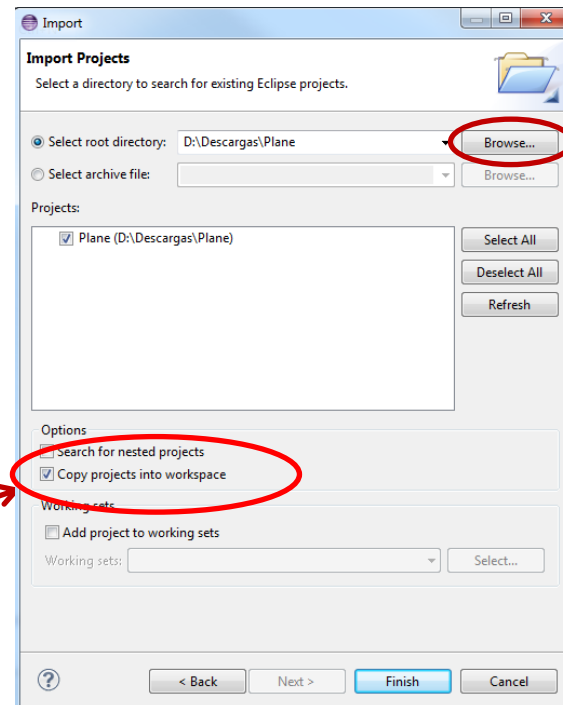
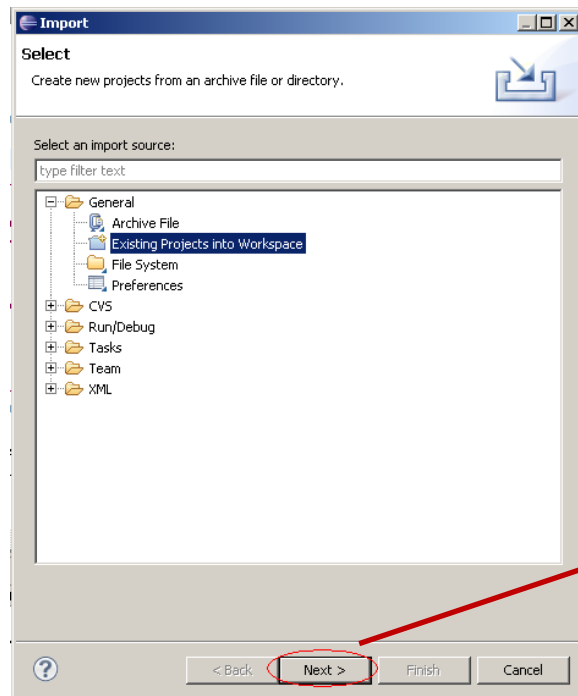
Importar un proyecto

- Menú File → Import
/General/Existing Projects into Workspace...
 - Dos posibilidades
 1. Seleccionar directorio
La carpeta con el proyecto
 2. Seleccionar fichero
El proyecto comprimido
- > Más cómodo usar segunda opción



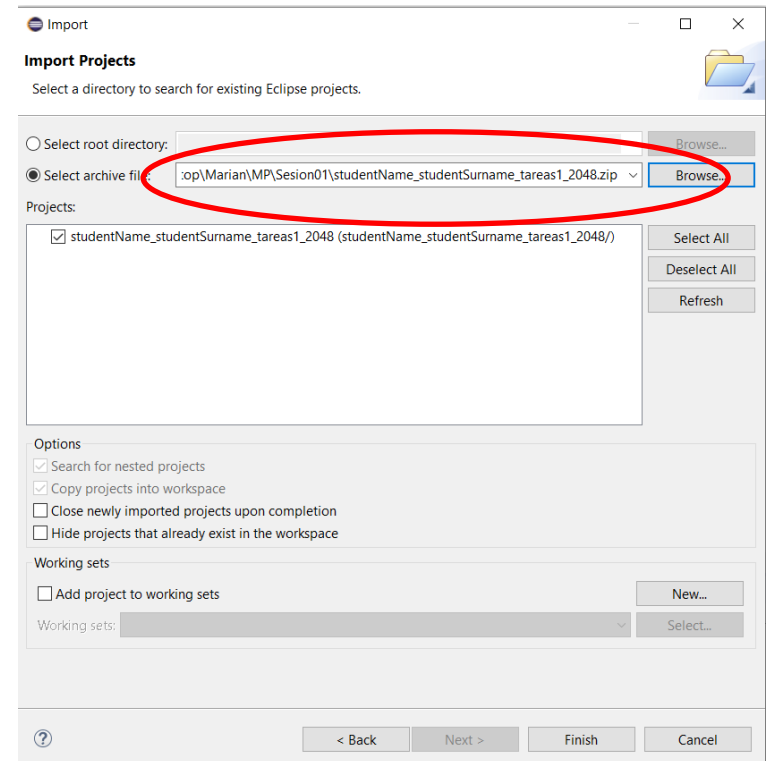
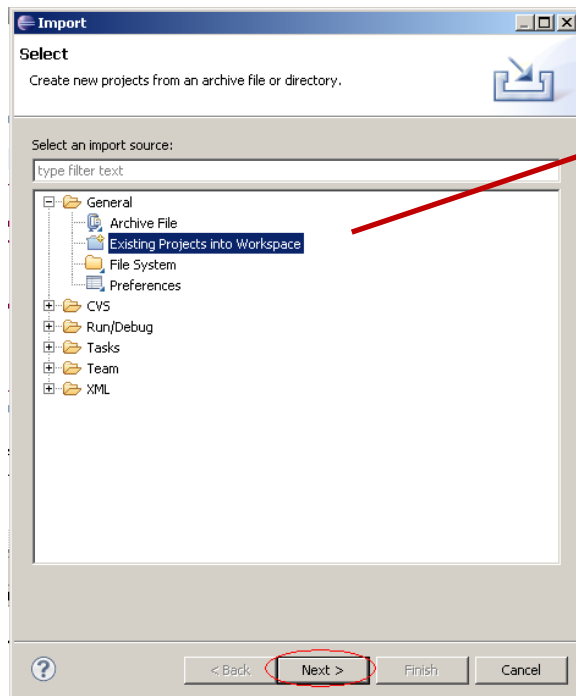
Importar un proyecto

- Si se importa proyecto sin comprimir es importante hacer una copia en el espacio de trabajo
- Si se importa proyecto comprimido ya la hace el sistema



Importar un proyecto

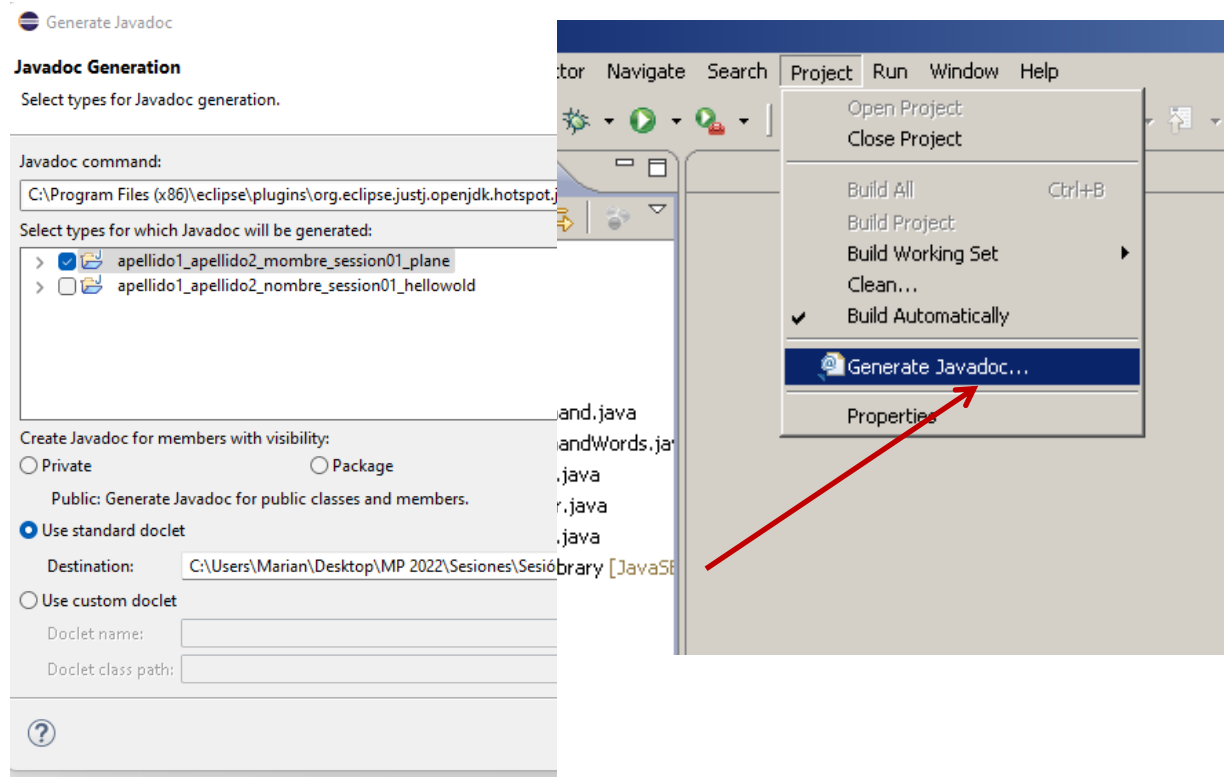
- Importaremos el proyecto para la tarea student_session01_game.



CÓMO SE GENERA LA DOCUMENTACIÓN CON JavaDoc

Generar Javadoc

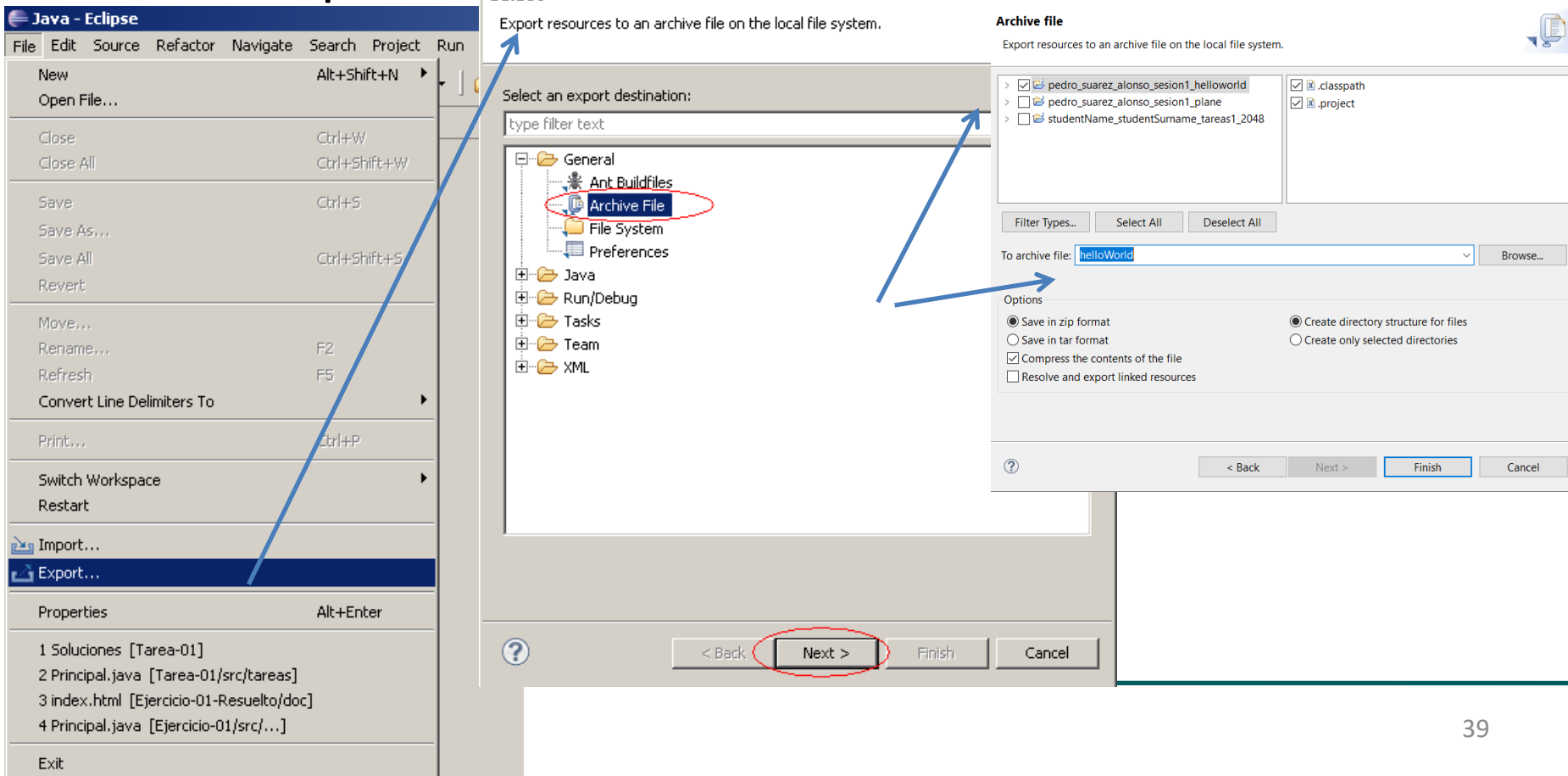
- Me permite generar el Javadoc a partir de los comentarios apropiados del código fuente.
 - Project → Generate Javadoc ...



CÓMO SE EXPORTA UN PROYECTO Y SE CREA UN FICHERO COMPRIMIDO

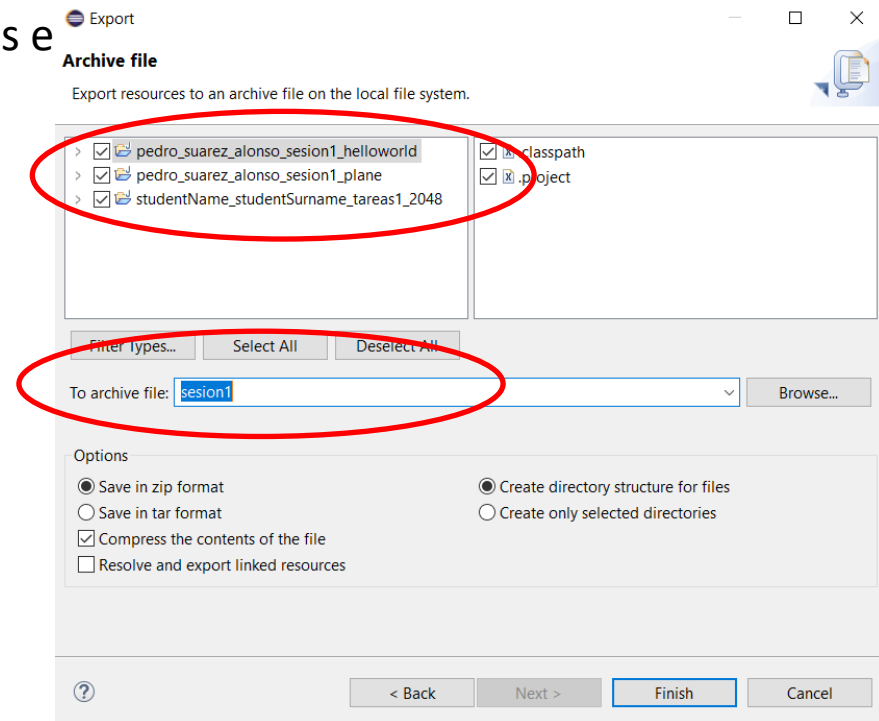
Exportar proyecto

- Permite copiar el proyecto a un archivo comprimido.
- File → Export



Exportar todos

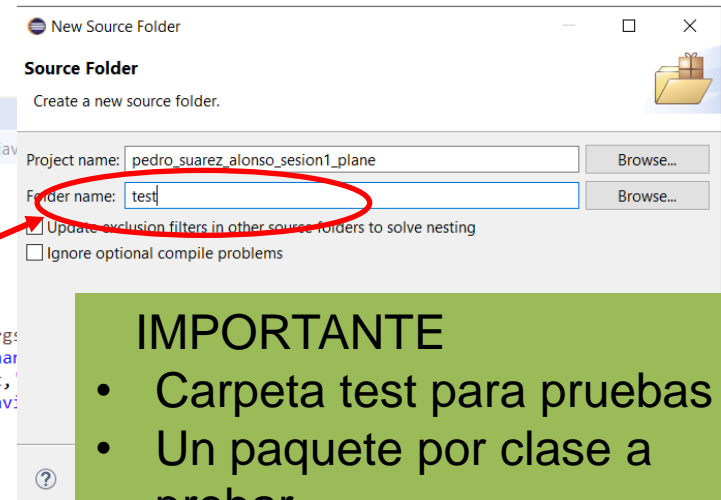
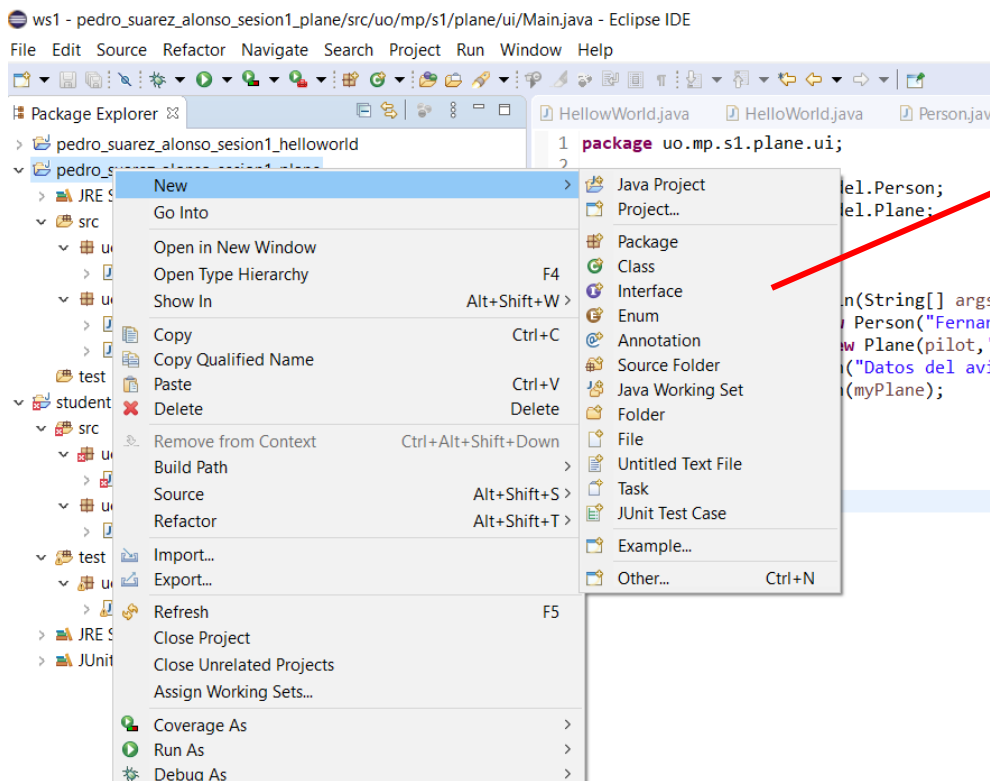
- Cuando se finalice la sesión
 - Se exportan TODOS los proyectos juntos e
 - **Se guarda el comprimido en la nube**
- En casa
 - Se crea carpeta MP
 - Se crea carpeta session01 en MP
 - Se crea carpeta ws1 en sesion01
 - Se arranca eclipse y se le asigna ws1
 - Se importan el fichero comprimido



CÓMO SE CREAN PRUEBAS

Pruebas con Junit. Carpeta test

- Se debe crear una carpeta (**source folder**) llamada test
- Dentro de esta carpeta se crea **un paquete por cada clase a probar** con el mismo nombre que el paquete donde está la clase a probar más el nombre de la clase.

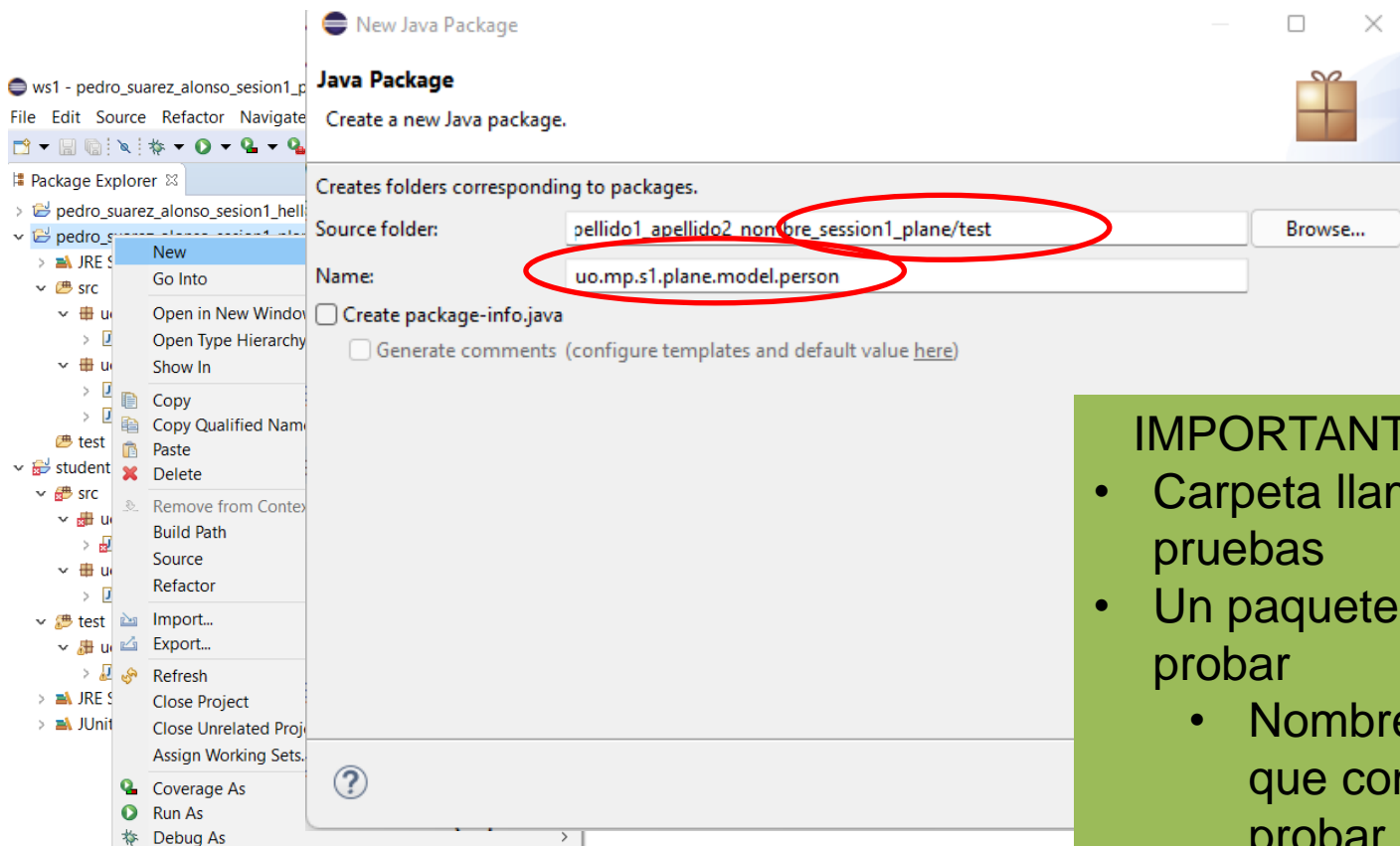


IMPORTANTE

- Carpeta test para pruebas
- Un paquete por clase a probar
- Una clase JUnit por método a probar
- Un método por caso a probar

Pruebas con Junit. Paquetes en test

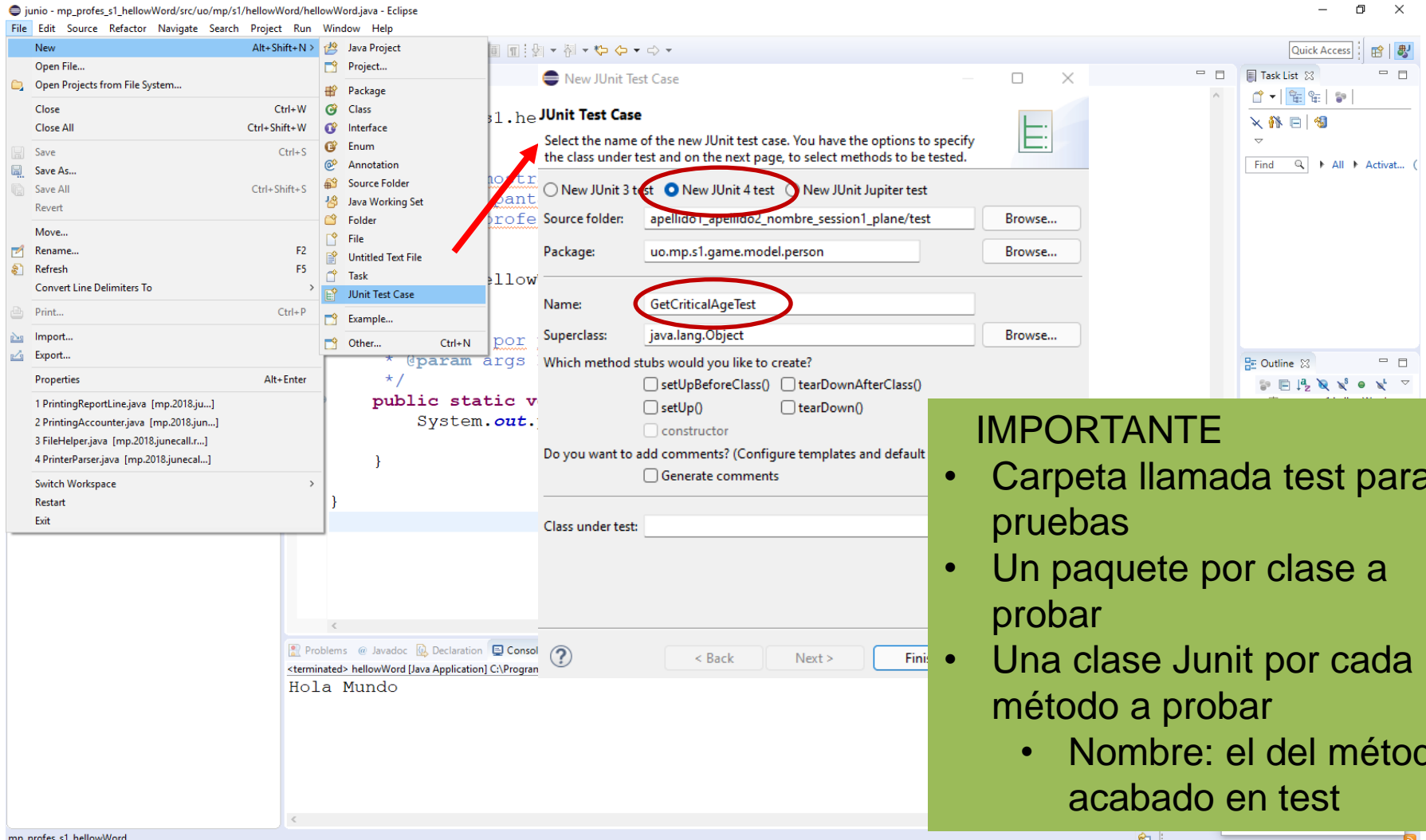
- Dentro de esta carpeta se crea **un paquete por cada clase a probar** con el mismo nombre que el paquete donde está la clase a probar más el nombre de la clase.



IMPORTANTE

- Carpeta llamada test para pruebas
- Un paquete por clase a probar
 - Nombre: el del paquete que contiene la clase a probar + .nombreClase

Pruebas con Junit. Clase JUnit



The screenshot shows the Eclipse IDE interface. On the left, the 'File' menu is open, and 'JUnit Test Case' is selected. In the center, the 'New JUnit Test Case' dialog is displayed. A red arrow points to the 'New JUnit 4 test' radio button, which is also circled in red. The 'Name' field contains 'GetCriticalAgeTest', also circled in red. The 'Package' field contains 'uo.mp.s1.game.model.person'. The 'Source folder' field contains 'apellido1_apellido2_nombre_session1_plane/test'. The 'Superclass' field contains 'java.lang.Object'. The 'Which method stubs would you like to create?' section has checkboxes for 'setUpBeforeClass()', 'tearDownAfterClass()', 'setUp()', 'tearDown()', and 'constructor', all of which are unchecked. The 'Do you want to add comments?' section has a checkbox for 'Generate comments', which is unchecked. The 'Class under test' field is empty. At the bottom, a console window shows the output 'Hola Mundo'.

IMPORTANTE

- Carpeta llamada test para pruebas
- Un paquete por clase a probar
- Una clase Junit por cada método a probar
 - Nombre: el del método acabado en test

Pruebas con Junit. Método

```
/**
 * Pruebas para el método getCriticalAge
 *
 * Casos de uso
 * 1- Edad menor 18
 * 2- Edad igual a 18
 * 3- Edad entre 18 y 65
 * 4- Edad igual a 65
 * 5- Edad mayor de 65
 *
 * @author mp_profes
 */
public class GetCriticalAgeTest {

    /**
     * Caso 1. Prueba getCriticalAge cuando 1
     * Debe devolver lo que le falta para lle
     */
    @Test
    public void testLessThanAdultHoodAge() {
        Person person1 = new Person("Pedro", Person.ADULTHOOD_AGE - 1);
        assertEquals(1, person1.getCriticalAge());
    }
}
```

IMPORTANTE

- Carpeta llamada test para pruebas
 - Tipo carpeta: source code
 - Nombre carpeta: test
- Un paquete por clase a probar
 - Nombre: el del paquete que contiene la clase a probar + .nombreClase
- Una clase Junit por cada método a probar
 - Nombre: el del método acabado en test
- Un método por cada caso de uso
 - Nombre: indicativo del caso que trata comenzando por palabra test

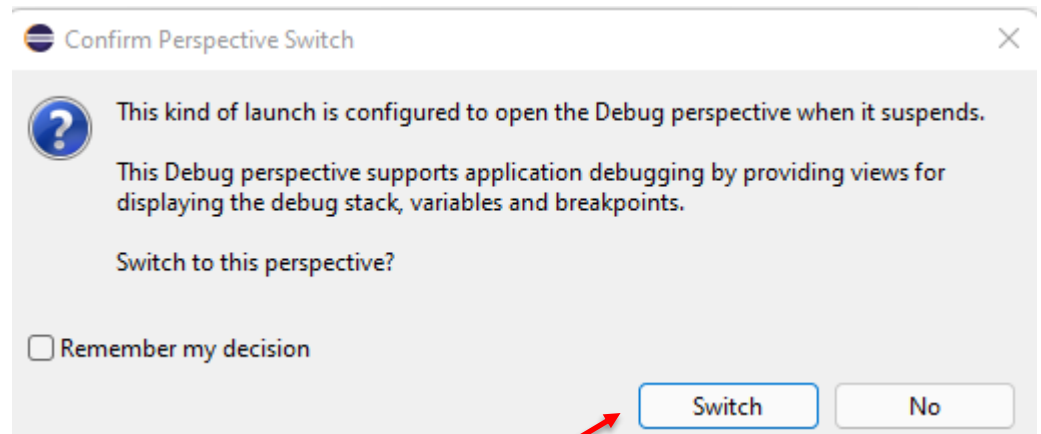
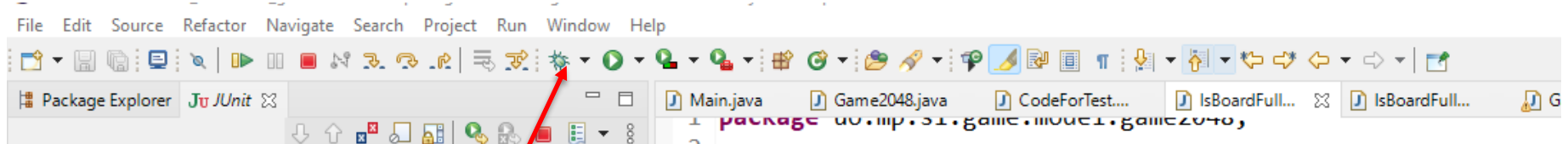
Depuración

Depuración

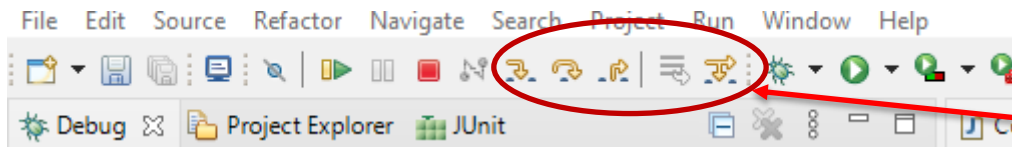
Punto de ruptura en el código. Igual que en BlueJ

Ejecución en modo depuración. Salta a la perspectiva de depuración

Ejecución paso a paso

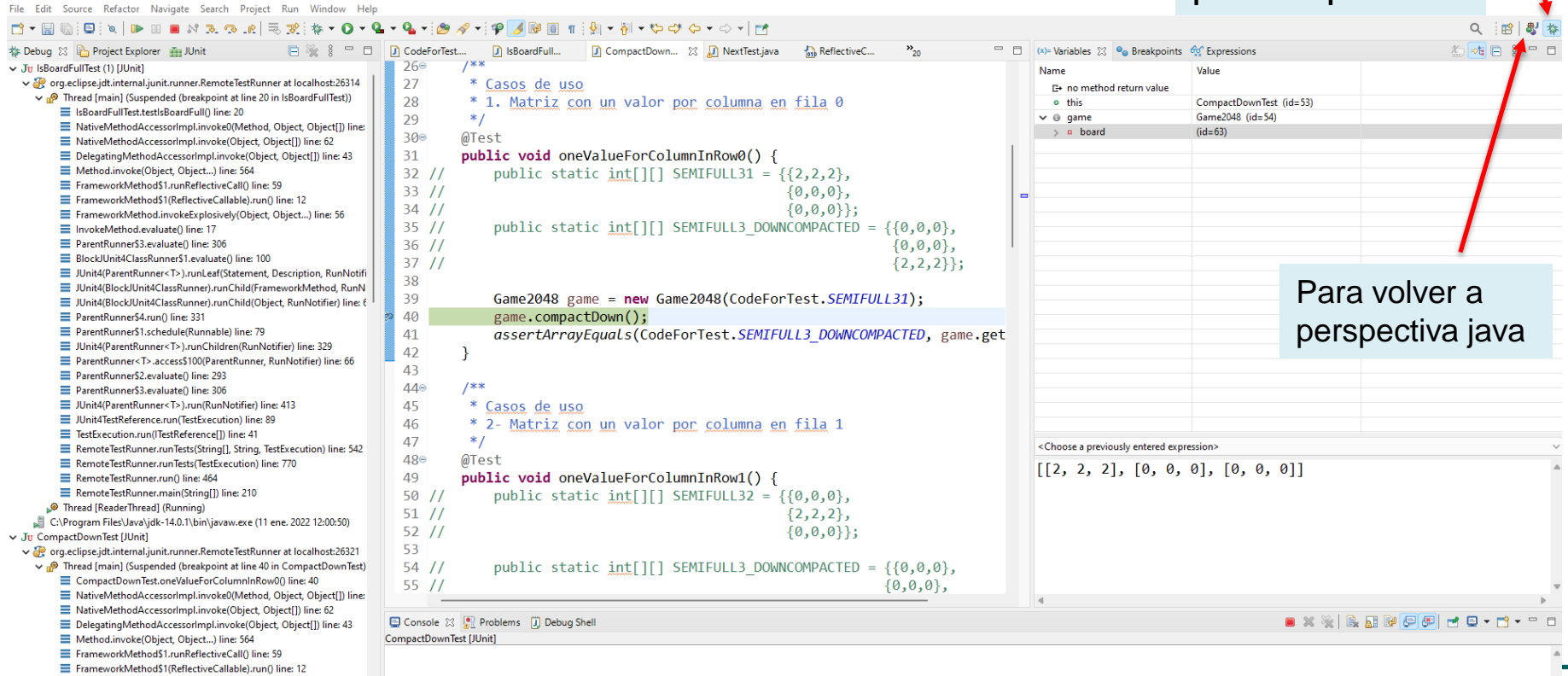


Depuración



Perspectiva depuración

Para ejecutar
paso a paso



Para volver a
perspectiva java

Tarea a entregar

- Proyecto original **game completado con las operaciones que se piden en el enunciado** y con el siguiente nombre
apellido1_apellido2_nombre_session01_task_game

Para renombrar un proyecto, paquete o tarea. Sobre él se pulsa botón derecho y opción: Refactor/rename

- **Revisar el proyecto mediante la checkList** para asegurarse de que está bien. Si algún aspecto de la checkList no se consigue, identificarlo en la lista mediante una cruz.
- **Entregar en el campus virtual**, en la tarea al respecto, **a lo sumo 24 horas antes de la siguiente tarea**, el **proyecto y la checkList** juntos en un fichero comprimido denominado: apellido1 apellido2 nombre

Normas generales

- Las tareas deben estar acabadas **24 horas antes** de la siguiente clase de laboratorio.
- **Podrán ser usadas en la siguiente clase.**
- **Podrán ser usadas en los exámenes de laboratorio.**
- **Todas las tareas de trabajo autónomo** (no presencial) que se pidan, **deberán subirse al campus virtual.**
- Más de 2 tareas inválidas o no entregadas supone **la pérdida de la evaluación continua**