Guía rápida: Test unitarios en Java (Junit con Eclipse)

Introducción

La prueba de programas es una aproximación dinámica a la verificación de los mismos, en la cual se ejecuta el software con unos datos o casos de prueba para analizar el buen funcionamiento de los requisitos esperados del programa. Podemos decir que la prueba es un tipo de verificación que se basa en la ejecución del código.

La prueba de unidad es la prueba de un módulo concreto dentro de un software que incluirá muchos otros módulos. Un módulo es una unidad de código que sirve como bloque de construcción para la estructura física de un sistema. El concepto de "módulo" se debe interpretar en términos del lenguaje de programación que estemos utilizando. Por ejemplo, podemos considerar que una clase Java o C++ es un módulo, pero también lo es un módulo del lenguaje Modula-2 o una unidad en TurboPascal.

Una prueba de unidad es un programa que prueba a otro programa. En términos de Java, es una clase con el objetivo de probar a otra clase. Es decir, es un trozo de código que escribimos para determinar si otro trozo de código se comporta como esperábamos o no.

¿Cómo se hace esto?

Para esto usamos aserciones. Una aserción, *assertion*, es un simple método que verifica si algo es verdadero. Por ejemplo, el método *assertTrue*, comprueba si una condición booleana es verdadera y falla en caso contrario. Su implementación podría ser:

```
public void assertTrue(boolean condition) {
  if (!condition) {
    abort();
  }
}
```

En los últimos años, ha aparecido una metodología de desarrollo de programas denominada UExtreme Programming (XP) U(Beck 1999), que hace un énfasis especial en las prácticas y técnicas de prueba unitaria. Como resultado de ese énfasis, se han creado una serie de frameworks para ayudar a realizar pruebas unitarias en diferentes lenguajes. Al conjunto de esos frameworks se les denomina xUnit.

Nos centramos en JUNIT, que es "el XUNIT para Java", los conceptos son muy similares a los que nos encontramos en los frameworks XUNIT para otros lenguajes (incluyendo a C++, Delphi, Smalltalk y Visual Basic).

Assert junit.framework

Object
└Assert

protected	**Assert** ()
public static . . . . . void	**assertEquals** (String message, Object expected, Object actual)
public static . . . . . void	**assertEquals** (Object expected, Object actual)
public static . . . . . void	**assertEquals** (String message, *[double|float]* expected, *[double|float]* actual, *[double|float]* delta)
public static . . . . . void	**assertEquals** (*[double|float]* expected, *[double|float]* actual, *[double|float]* delta)
public static . . . . . void	**assertEquals** (String message, *[boolean|byte|char|int|long|short]* expected, *[boolean|byte|char|int|long|short]* actual)
public static . . . . . void	**assertEquals** (*[boolean|byte|char|int|long|short]* expected, *[boolean|byte|char|int|long|short]* actual)
public static . . . . . void	**assertNotNull** (String message, Object object)
public static . . . . . void	**assertNotNull** (Object object)
public static . . . . . void	**assertNull** (String message, Object object)
public static . . . . . void	**assertNull** (Object object)
public static . . . . . void	**assertSame** (String message, Object expected, Object actual)
public static . . . . . void	**assertSame** (Object expected, Object actual)
public static . . . . . void	**assertTrue** (String message, boolean condition)
public static . . . . . void	**assertTrue** (boolean condition)
public static . . . . . void	**fail** (String message)
public static . . . . . void	**fail** ()

Método	Qué hace
`assertTrue(boolean condicion)`	Falla si la condición es false; pasa en otro caso.
`assertEquals(Object esperado, Object actual)`	Falla si esperado y actual no son iguales, según el método equals(); pasa en otro caso.
`assertEquals(int esperado, int actual)`	Falla si esperado y actual no son iguales, según el operador ==; pasa en otro caso. Este método está sobrecargado para cada tipo primitivo: int, float, double, char, byte, long, short, y boolean.
`assertSame(Object esperado, Object actual)`	Falla si esperado y actual se refieren a diferentes objetos en memoria; pasa si se refieren al mismo objeto en memoria.
`assertNull(Object object)`	Pasa si es null; En otro caso falla.

Además están los métodos assertFalse(), assertNotSame() y assertNotNull().

Para aritmética con punto flotante :

assertEquals([String message], expected, actual**, tolerance**)
Ejemplo: assertEquals (expectedDouble, actualDouble, 0.0001d).

También resulta de mucha utilidad:

fail

fail([String message])

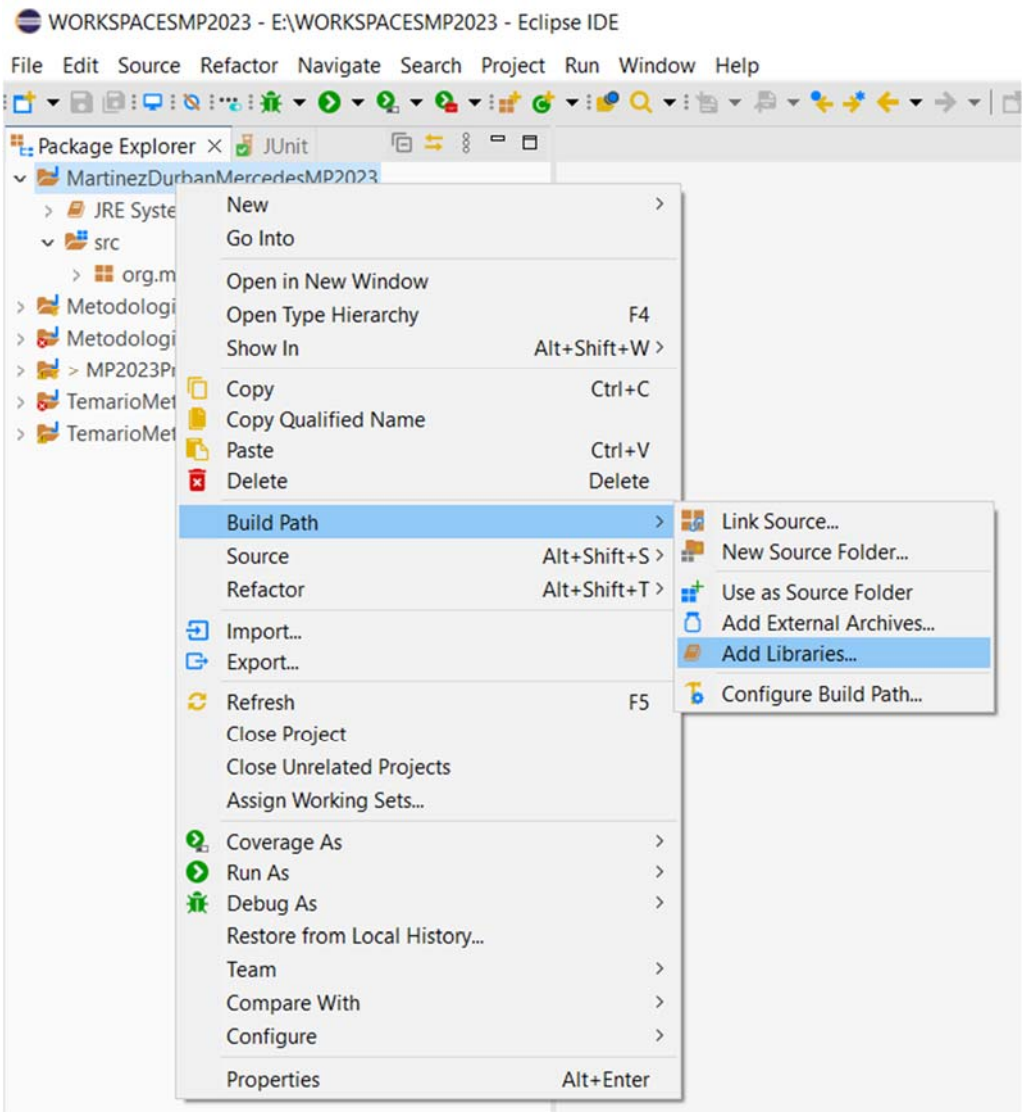
Hace que el test falle inmediatamente con el mensaje que se indique que es opcional. Normalmente se usa para probar código que tiene excepciones y estás no han sido tratadas.

Pruebas JUnit en Eclipse

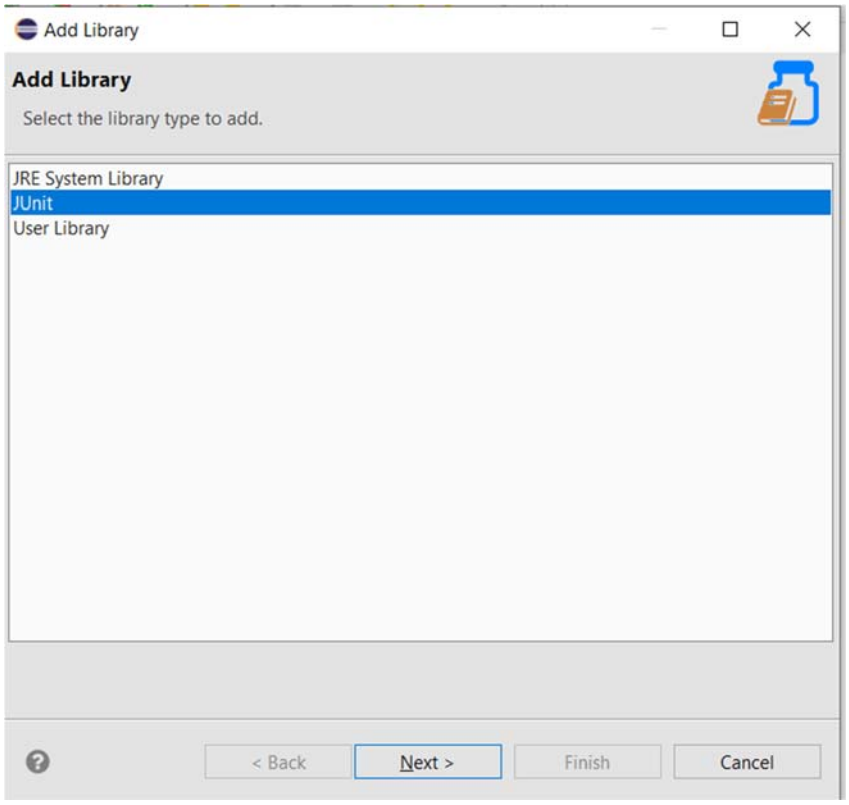
Eclipse facilita la tarea de crear y ejecutar pruebas unitarias utilizando JUnit. **El objetivo** de este curso es **ejecutar los test unitarios que el profesor proporcione a los alumnos**. Es decir, el alumno no tiene que crear ningún test, esta tarea se verá en otras asignaturas.

Para ejecutar en nuestro proyecto los test unitarios que se vayan proporcionando en las distintas sesiones se llevarán a cabo los siguientes pasos:

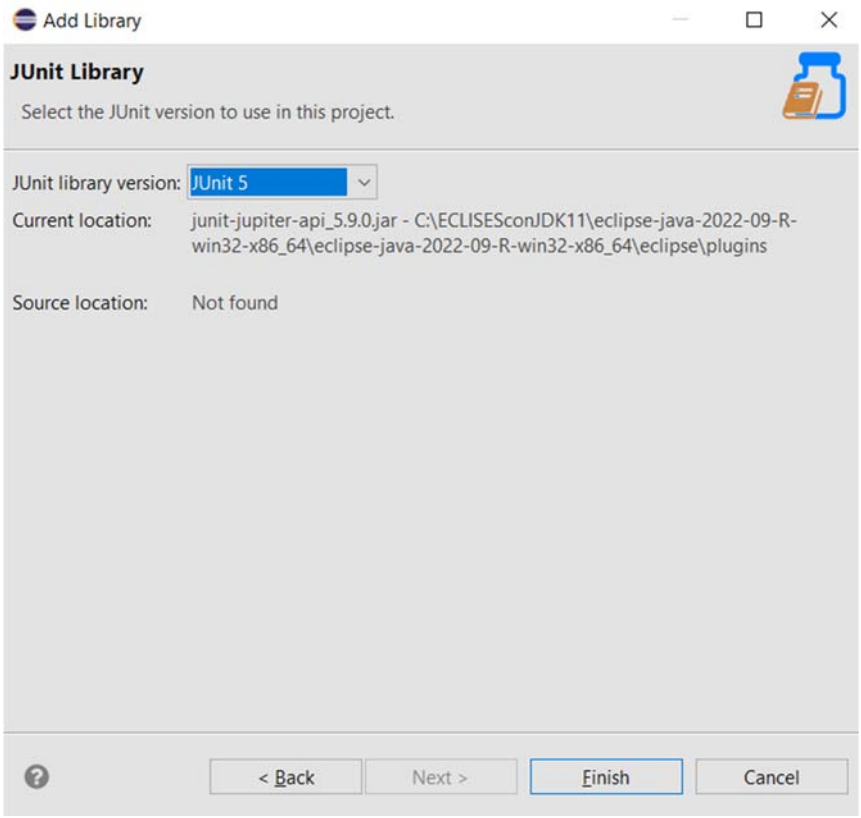
1. Añadir la librería que permita reconocer y ejecutar los test. Para ello, sobre nuestro proyecto, hacemos clic con el botón derecho y seleccionamos **Build Path > Add Libraries**.



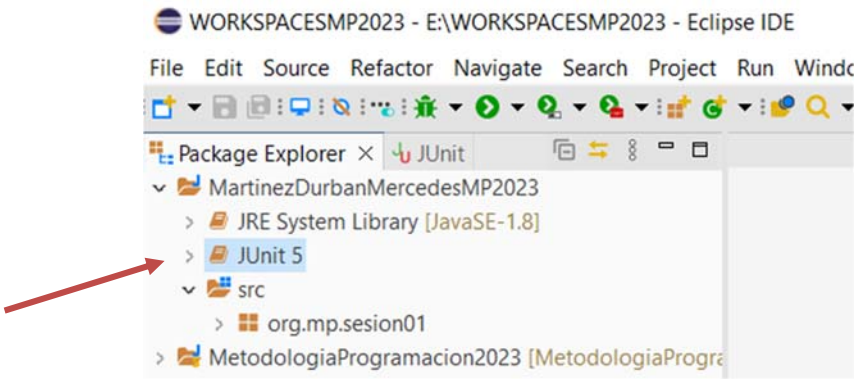
Aparecerá la siguiente pantalla



Seleccionamos **JUnit** y pulsamos **Next**.



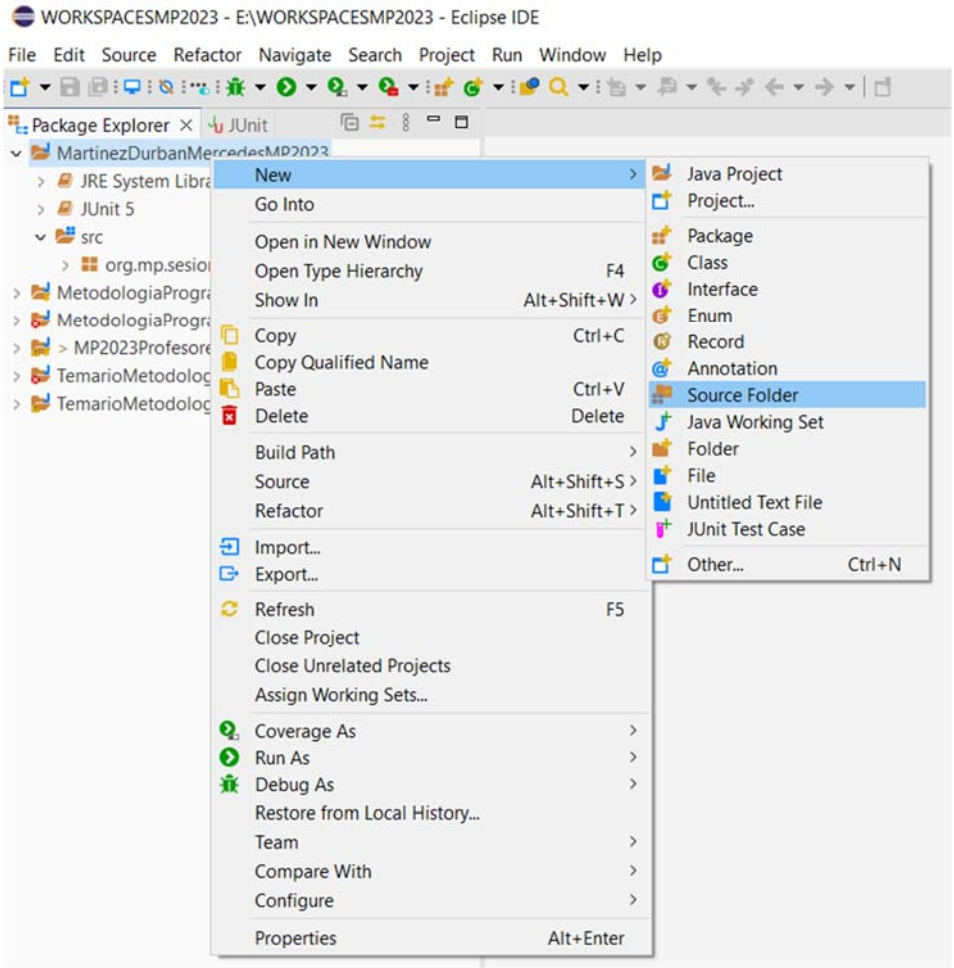
Pulsamos **Finish** y tendremos incluida la librería **JUnit5** en nuestro proyecto tal y como se muestra a continuación.



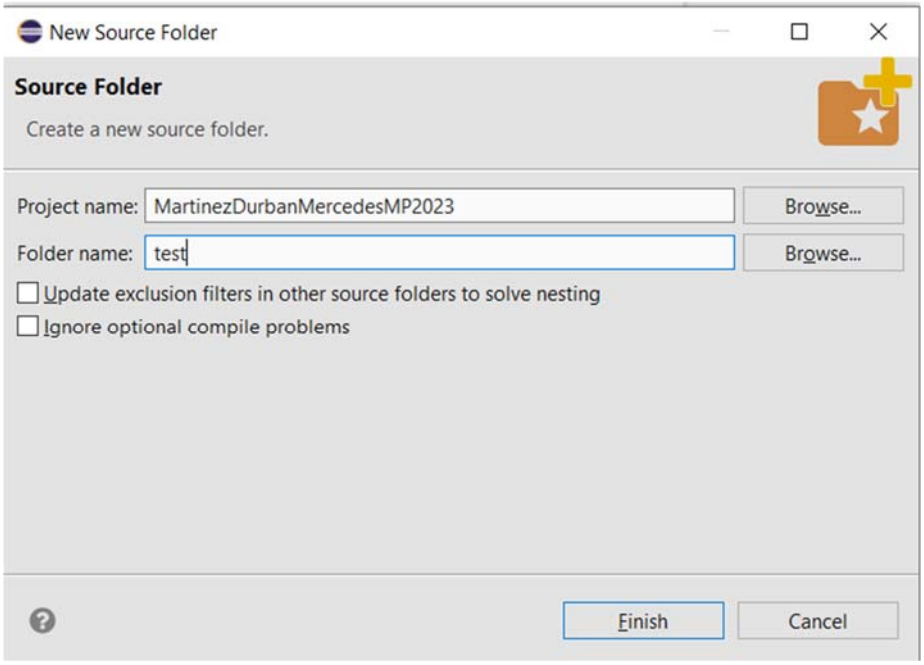
Ahora podemos ejecutar en nuestro proyecto los test unitarios.

2. Crear una carpeta **test** para situar los distintos test.

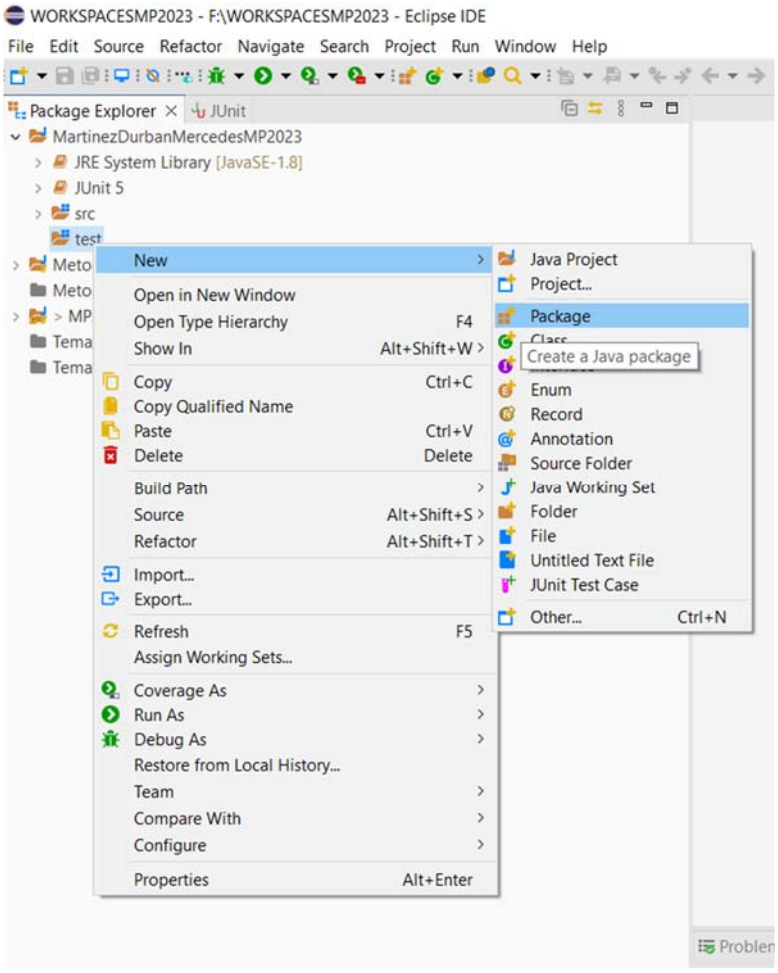
Para ello, hacemos clic con el botón derecho del ratón sobre nuestro proyecto y después seleccionamos **New > Source Folder** tal y como se muestra a continuación.

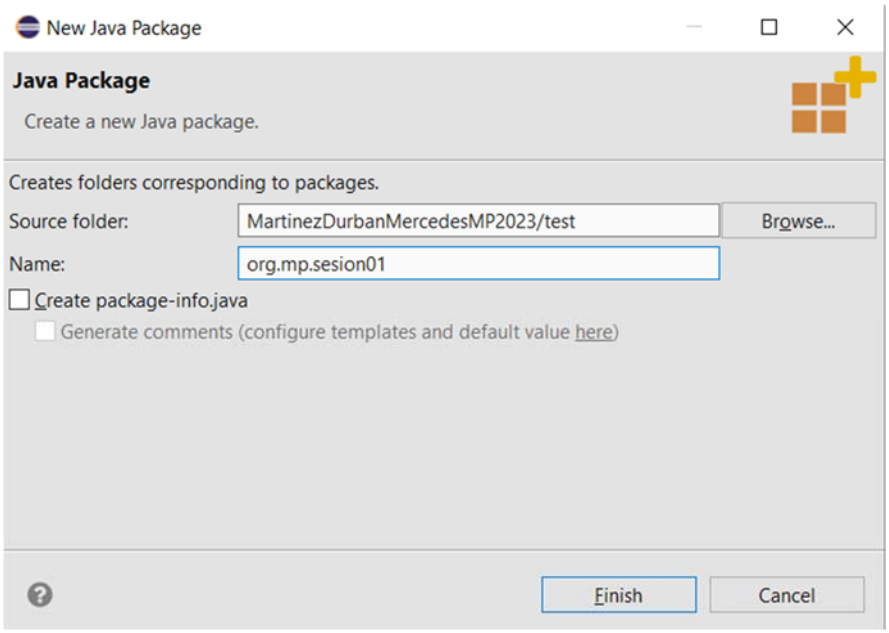


En la siguiente pantalla indicaremos el nombre de la carpeta, **test** y **Finish.**

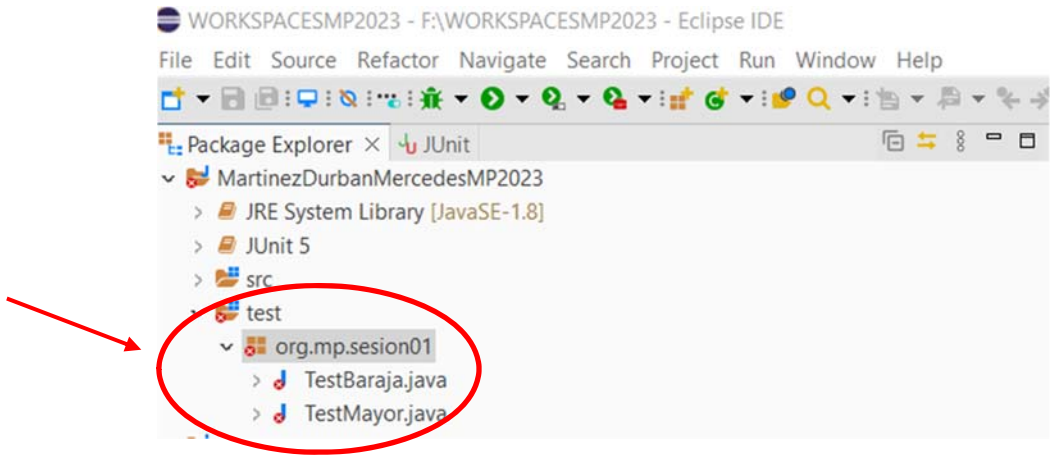


En nuestro proyecto habremos creado una carpeta de código llamada **test** donde guardaremos los test unitarios de las distintas sesiones. En esta carpeta crearemos paquetes con los mismos nombres que tenemos en la carpeta **src**. Por ejemplo, el primer paquete será **org.mp.sesion01** y en él copiaremos los test correspondientes a la primera sesión que se proporcionan en el proyecto de la asignatura. En las siguientes sesiones crearemos el resto de paquetes tanto en la carpeta **src** como en la carpeta **test.**



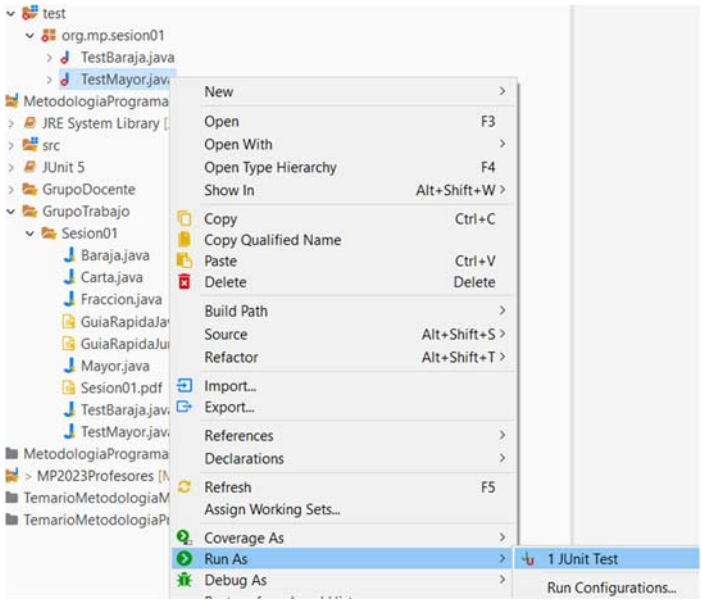


El resultado después de copiar los test unitarios es el que se muestra.

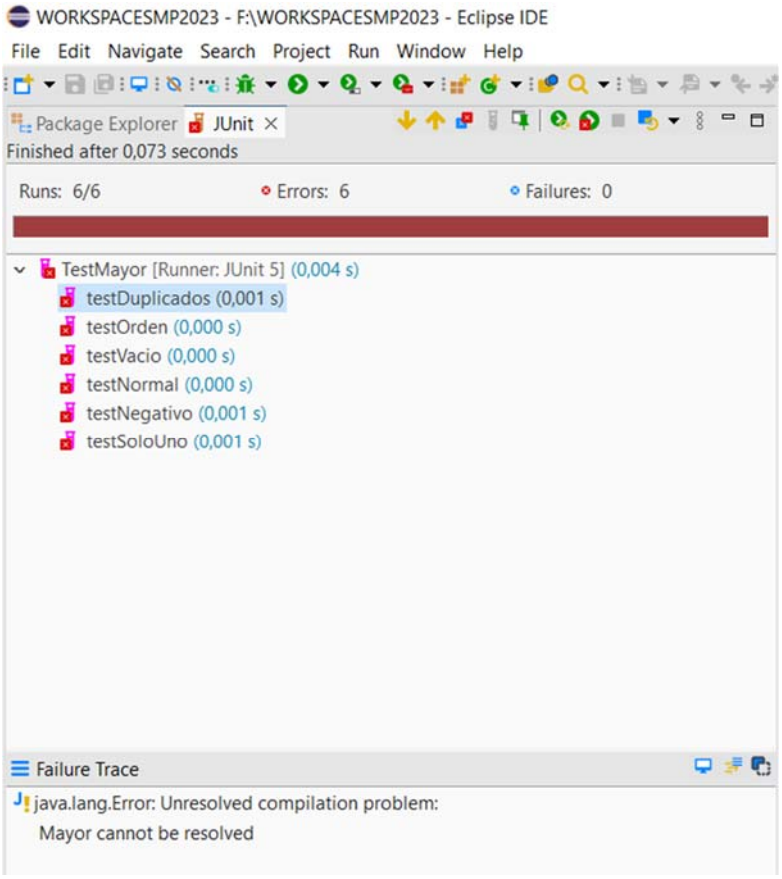


Los errores que aparecen en la carpeta, en el paquete y en los archivos se deben a que aún no se han implementado las clases y los métodos que hay en los test.

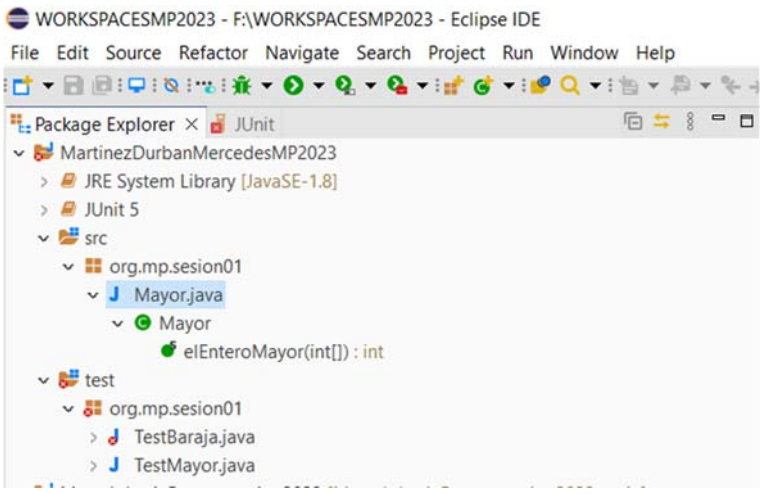
Para **ejecutar un test**, por ejemplo, **TestMayor**, nos situaremos sobre el archivo, botón derecho y **Run As > JUnit Test**. Obviamente aparecerá la temida **barra roja** en una nueva vista, **JUnit**.



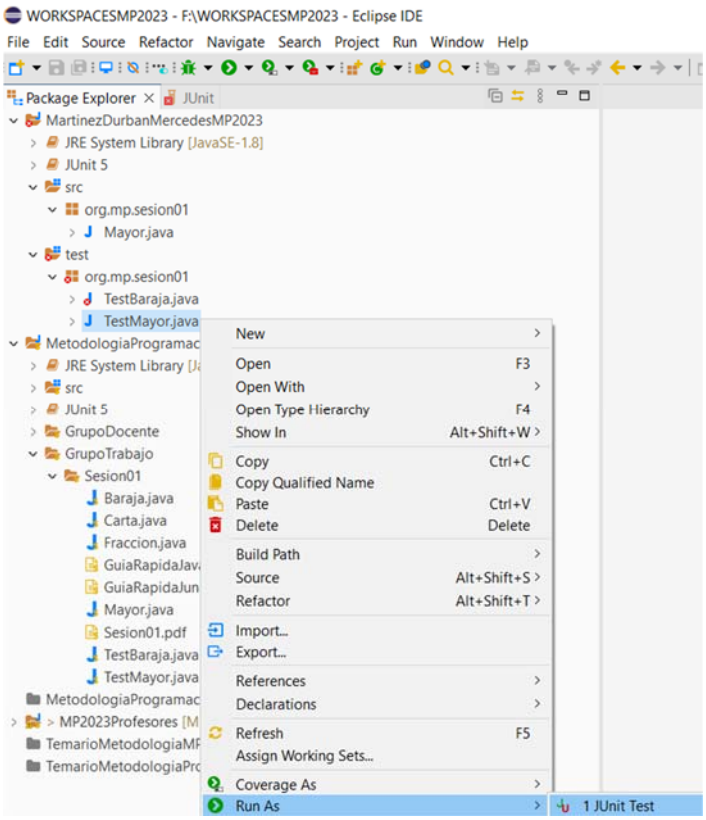
Resultado de la ejecución del test, **TestMayor**.

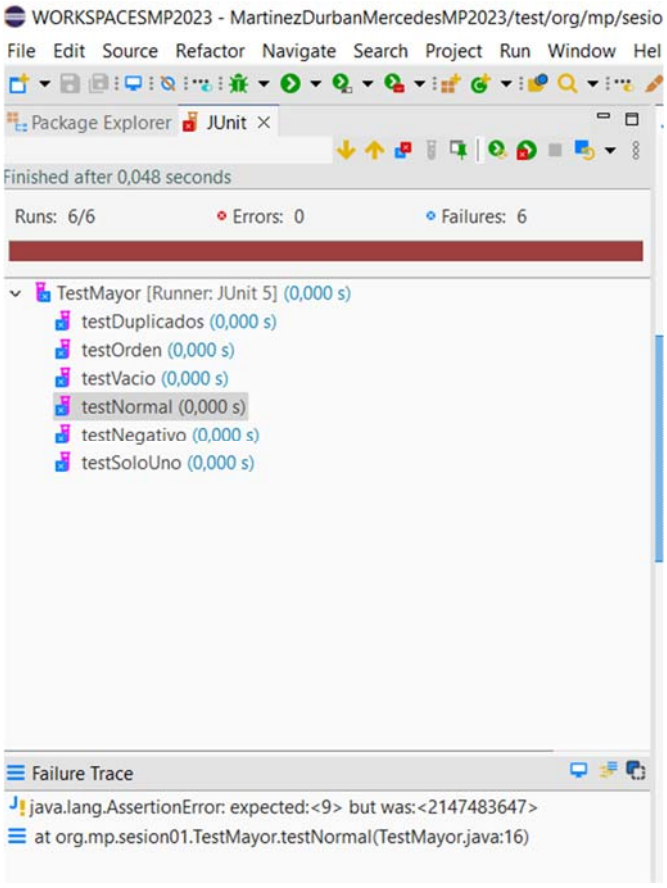


Podemos copiar en la carpeta **src** la clase **Mayor** que se proporciona en la sesión. Observamos que el test, **TestMayor** no tiene errores debido a que en la carpeta **src** ya existe la clase y en este caso un único método que tiene dicha clase.

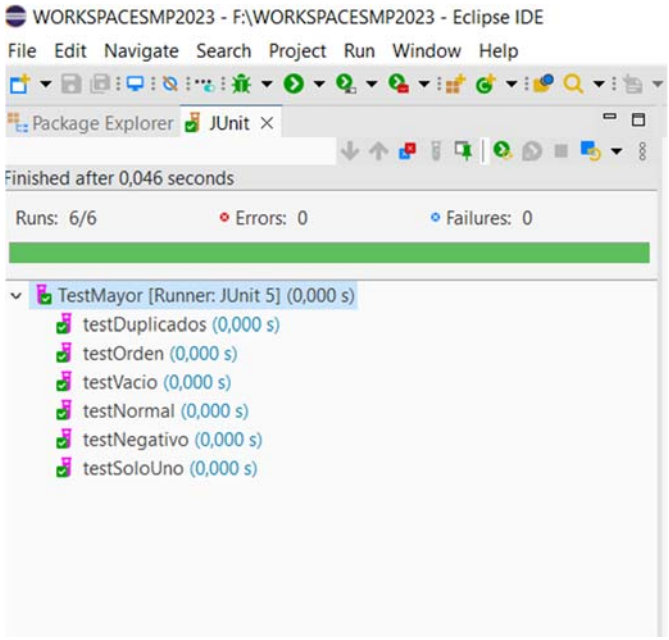


El resultado de volver a ejecutar el test se muestra a continuación.

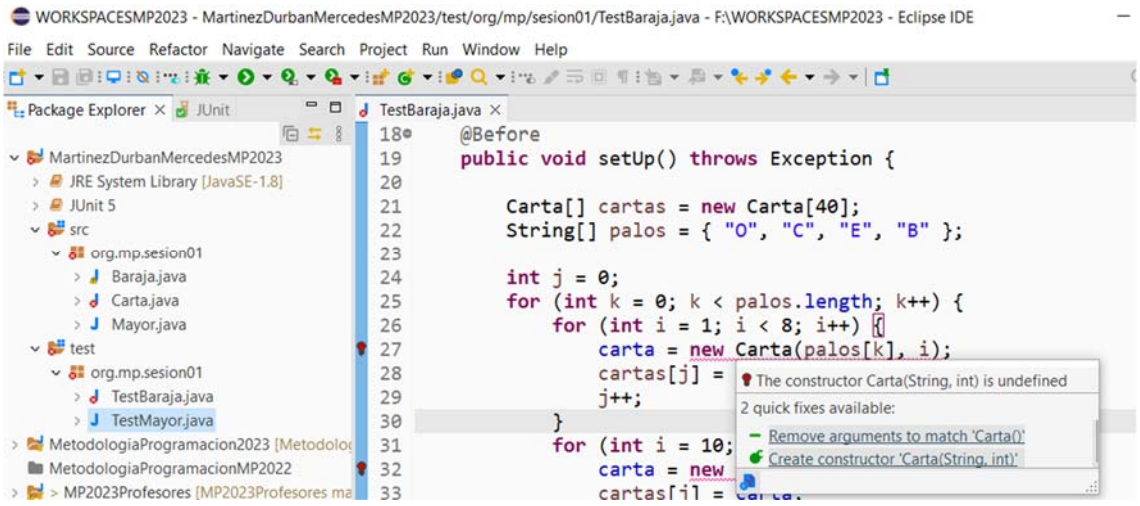




La clase **Mayor** deberá modificarse para que se verifiquen las distintas aserciones de la clase **TestMayor** y consigamos la **barra verde**.

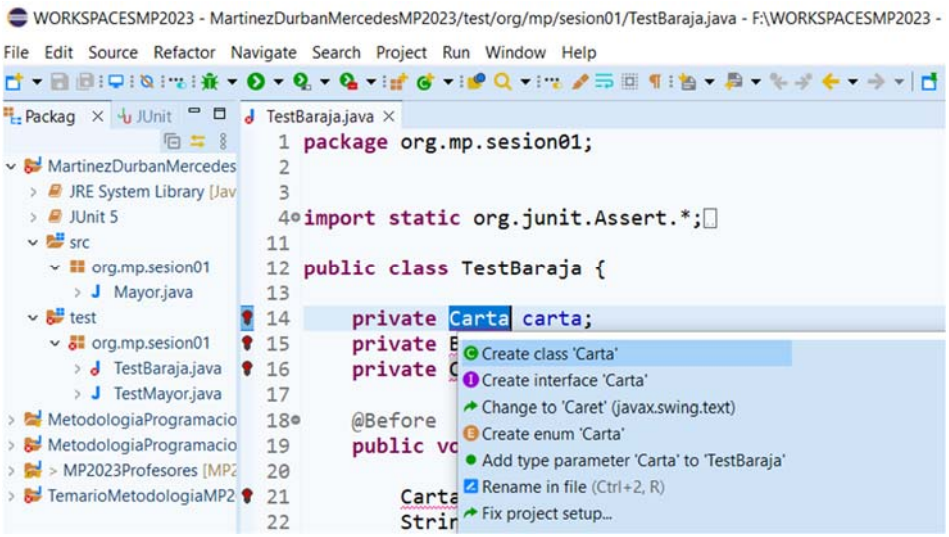


3. Eclipse facilita la labor de implementación a partir de los test. Para mostrar un ejemplo, nos fijaremos en las clases **Carta** y **Baraja** cuya implementación hemos de terminar. Primero copiaremos esas clases en la carpeta **src**. A continuación, abriremos el archivo **TestBaraja** y nos situaremos en el error que aparece en una determinada la línea de código, en este caso, en la línea 27. La propuesta que nos hace es la de crear un constructor para la clase puesto que no está definido.

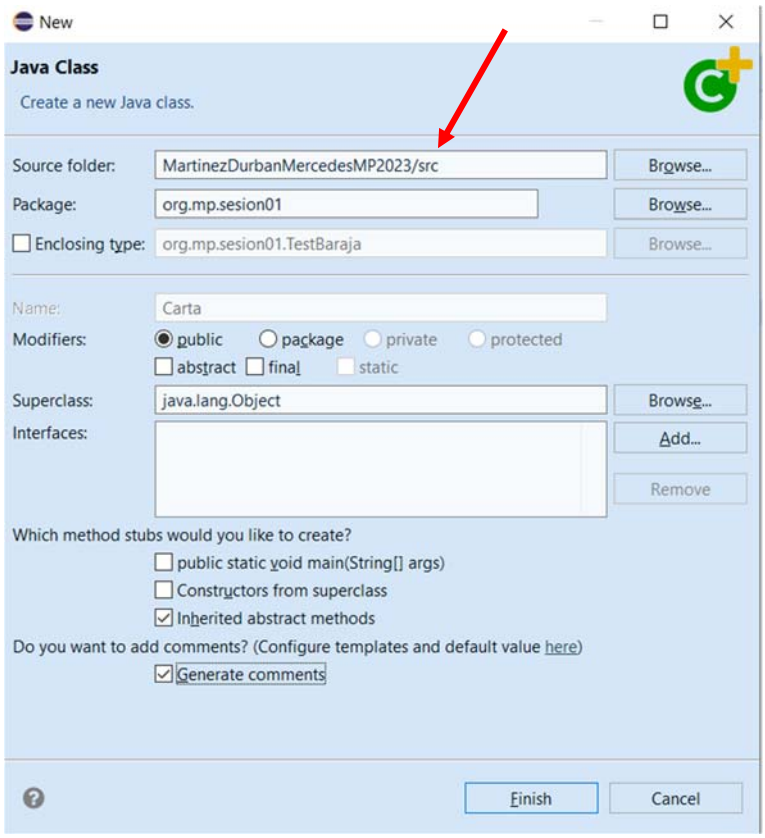


De esta manera podemos ir creando métodos e incluso clases a partir de los errores de las distintas líneas de código de los test. Si el error lo tenemos sobre una clase, nos posicionamos sobre ella y con **Create class…** aparecerá una ventana para la creación de dicha clase. Hemos de tener en cuenta que en **Source folder** debemos cambiar la carpeta **test** por la carpeta **src** para que la nueva clase la cree en el paquete y en la carpeta adecuados. Aunque la clase **Carta** se proporciona con la

sesión 1 no la copiaremos en principio en el paquete correspondiente para mostrar la creación de esta clase como ejemplo.



Cambiamos la carpeta **test** por **src**



4. Por último, deberemos implementar todas las clases y métodos para que pasen los test correctamente (**barra verde**)