SESIÓN 8: Arrays unidimensionales

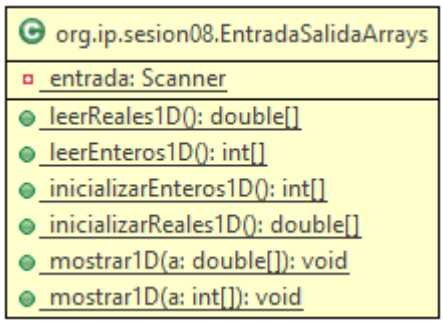
Objetivos

- Saber declarar, crear e inicializar arrays.
- Saber acceder a componentes individuales de un array.
- Saber realizar operaciones comunes con arrays (mostrar un array, calcular la media, etc.).
- Usar métodos de la clase *Arrays* del paquete java.util.
- Crear clases que contengan un array como atributo e implementar métodos asociados.
- Introducir el uso de los test unitarios con JUnit en Eclipse.

Nota importante: Siga el esquema de nombrado de paquetes que se indicó en la sesión 01 es decir: **org.ip.sesion08**. En ese paquete se crearán todos los programas que se proponen en la sesión dándoles un nombre al programa y que se indica en cada ejercicio entre **paréntesis y en negrita**.
Al final de la sesión, el alumno deberá cargar el trabajo realizado a su repositorio personal indicando la clave correspondiente a la sesión.

Ejercicios propuestos

1. Implementa la clase **(EntradaSalidaArrays)** que contiene los métodos de clase (estáticos) que se detallan en el diagrama de clases UML que sigue:



leerReales1D(): double[] ⇒ Leerá de teclado valores reales que guardará en un array unidimensional de reales.
leerEnteros1D(): int[] ⇒ Leerá de teclado valores enteros que guardará en un array unidimensional de enteros.
inicializarEnteros1D(): int[] ⇒ Guardará en un array unidimensional valores enteros generados aleatoriamente.
inicializarReales1D(): double[] ⇒ Guardará en un array unidimensional valores reales generados aleatoriamente.
mostrar1D(a: double[]):void ⇒ Muestra por pantalla los componentes de un array unidimensional de reales.
mostrar1D(a: int[]):void ⇒ Muestra por pantalla los componentes de un array unidimensional de enteros.

Por ejemplo:

```
public static int [] leerEnteros1D() {
	entrada = new Scanner(System.in);
	System.out.println("Introduce el número de componentes del array de
	enteros");
	int dimension = entrada.nextInt();
	int [] a = new int [dimension];
	System.out.println("Introduce valores enteros en el array ");
	for (int i = 0; i < a.length; i++) {
		System.out.print("Introduce valor " + (i + 1) + "=> ");
		a[i] = entrada.nextInt();
	}
	return a;
}
```

Trabajo autónomo

2. Implementa una clase **(EstadisticasArrays)** en el paquete **src.org.ip.sesion08** que contenga la los métodos de clase (estáticos) que se detallan en el diagrama de clase UML que se muestra a continuación:

Como aclaración de las operaciones más importantes, se proporcionan las siguientes descripciones, algunas de ellas con sus respectivas fórmulas

max(a:double[]): double ⇒ valor máximo del array a.
max(a:double[], inferior:int, superior:int): double ⇒ valor máximo del subarray[inferior..superior]
min (a:double[]): double ⇒ valor mínimo del array o subarray
media(a:double[]): double ⇒ media de los valores del array o subarray:

$$media = \frac{\sum_{i=1}^{n} a_i}{n} = \frac{a_1 + a_2 + \cdots + a_n}{n}$$

var(a:double[]): double ⇒ varianza de los valores del array o subarray:

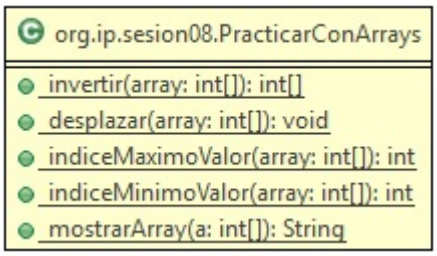
$$varianza = \frac{\sum_{i=1}^{n} (a_i - media)^2}{n-1}$$

stdDev (a:double[]): double ⇒ desviación estándar del array o subarray:

$$stdDev = \sqrt{var}$$

Conforme se vaya implementado dicha clase se deberá ir pasando el test (**EstadisticasArraysTestJUnit4**) que permita comprobar su correcto funcionamiento. Dicho test se proporcionará con el material de esta sesión y se deberá almacenar en el paquete **test.org.ip.sesion08**. Recordemos que hay que crear un nuevo paquete en la carpeta de fuentes **test**, éste será **org.ip.sesion08**. A continuación, debemos copiar el archivo de test **EstadisticasArraysTestJUnit4.java** y a partir de este momento hay que implementar todas las operaciones en el archivo **EstadisticasArrays.java** que deberá estar ubicado en la carpeta de fuentes **src** en el paquete **org.ip.sesion08**.

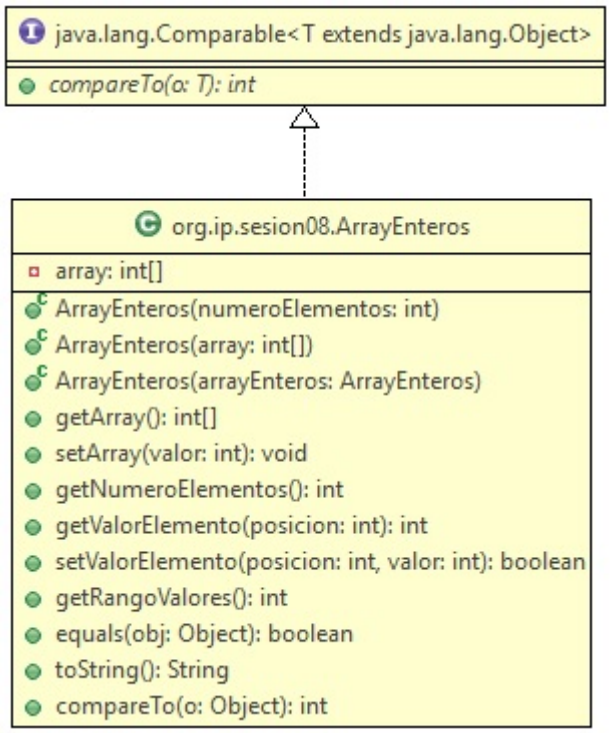
3. Implementa una clase **(PracticarConArrays)** en el paquete **src.org.ip.sesion08** que contiene los métodos de clase (estáticos) que se detallan en el diagrama de clase UML que se muestra a continuación:



Los nombres de los métodos son suficientemente descriptivos para saber cuál debe ser su funcionalidad, destacando que algunos de ellos se han visto en clase de teoría.

Conforme se vaya implementado dicha clase se deberá pasando el test (**PracticarConArraysTestJUnit4**) que permita comprobar su correcto funcionamiento. Dicho test se proporcionará con el material de esta sesión y se deberá almacenar en el paquete **test.org.ip.sesion08**. Recordemos que hay que crear un nuevo paquete en la carpeta de fuentes **test**, éste será **org.ip.sesion08**. A continuación, debemos copiar el archivo de test **PracticarConArraysTestJUnit4.java** y a partir de este momento hay que implementar todas las operaciones en el archivo **PracticarConArrays.java** que deberá estar ubicado en la carpeta de fuentes **src** en el paquete **org.ip.sesion08**.

4. Implementa la clase **(ArrayEnteros)** en el paquete **src.org.ip.sesion08** que contiene todo lo que se detalla en el diagrama de clase UML que se muestra a continuación:



Aclaraciones sobre la implementación a realizar:
1. Se implementarán 3 constructores diferentes según los parámetros que se le pasen. Destacar que para el constructor `public ArrayExamen(int numeroElementos)` se creará el array y se le asignará a cada elemento el valor de su posición en el array (array[i] = i).

2. El método `public void setArray(int valor)` le asignará a todos los elementos del array el mismo valor, y éste será el que se le pasará como parámetro a través de la variable *valor*.

3. El método `public int getValorElemento(int posicion)` devolverá el valor que tiene el array en la posición *(posicion)*, sabiendo que $0 \leq posicion \leq numeroElementos - 1$.

4. El método `public boolean setValorElemento(int posicion, int valor)` devolverá `true` si ha podido realizar correctamente la asignación del *valor* en la posición *(posicion)* del array, `false` en caso contrario. Todo en el rango, $0 \leq fila \leq numeroElementos - 1$.

5. El método `public boolean equals(Object obj)` devuelve `true` si los dos arrays son exactamente iguales en contenido, `false` en caso contrario.

6. La salida del método `public String toString()` debe verificar la salida establecida en el *test*.

7. Se utilizará el valor del rango de valores (`public int getRangoValores()`) de todos los elementos que hay almacenados en el array como criterio para la comparación de arrays (`public int compareTo(Object o)`). El rango de valores (`getRangoValores()`) se obtiene restando al mayor valor del array el valor del menor valor (*mayor – menor*).

ANEXO: Test unitarios en Java (JUnit con Eclipse)

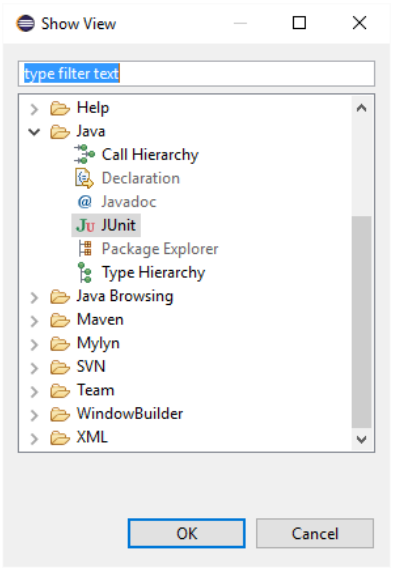
Una **prueba unitaria** o **test unitario** según *Wikipedia* es *una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado*. Para algunos, este tipo de pruebas es una pérdida de tiempo, aunque en muchas empresas esto es una práctica habitual. Hay que decir que crear las pruebas supone un tiempo adicional al que podríamos dedicar a programar la aplicación en sí, pero en grandes proyectos esta práctica está perfectamente amortizada. Debemos valorar si merece la pena o no el desarrollo dirigido por test (**TDD**, Test Driven Development), según las dimensiones y la complejidad del proyecto. Lo cierto es que si las hacemos, dependiendo del caso, podemos ahorrarnos mucho tiempo, ya que habremos automatizado las pruebas e instantáneamente sabremos si nuestra aplicación cumple con los requisitos correctos. Con el uso de esta metodología de desarrollo (TDD) mejoraremos la calidad del software, asegurándonos de que, aunque hagamos cambios en nuestro código, seguirá cumpliendo con los requisitos esperados. En un futuro podremos modificar, ampliar o eliminar código y los test unitarios seguirán sirviéndonos para realizar las comprobaciones pertinentes.

En las sesiones 08, 09 y 10 utilizaremos test unitarios para que nos guíen en la comprobación del correcto funcionamiento de determinados ejercicios. Hablando de forma coloquial, utilizaremos los test unitarios como si fuesen 'test main' de los que debemos de verificar su correcto funcionamiento. En esta asignatura, los test unitarios serán una herramienta que se proporcionará a los estudiantes, no siendo un objetivo de ella que ellos aprendan a diseñarlos, esta tarea se verá en asignaturas propias de la intensificación de Ingeniería del Software. Además, será habitual el uso de test unitarios en sucesivas asignaturas de programación como: Metodología de la Programación, Estructuras de Datos I, etc.

La estructura de un test unitario (una clase java que se crea en un paquete como … **New > JUnit Test Case**) la veremos en clases de laboratorio y para empezar ésta tendrá una estructura muy sencilla, que se complicará según los requerimientos del test y, por supuesto, del proyecto.

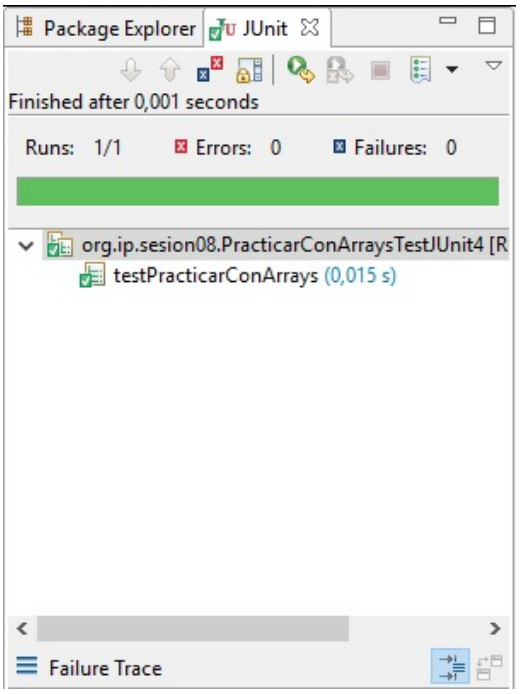
Para que nuestro proyecto personal de IP pueda ejecutar los test unitarios que nos proporcione el profesor en las respectivas sesiones, se deberán de realizar los siguientes pasos.

1. Activamos la vista **JUnit** en Eclipse. Para ello hay que pulsar en: **Window > Show View > Other... > Java > JUnit** y luego pulsar **OK.**

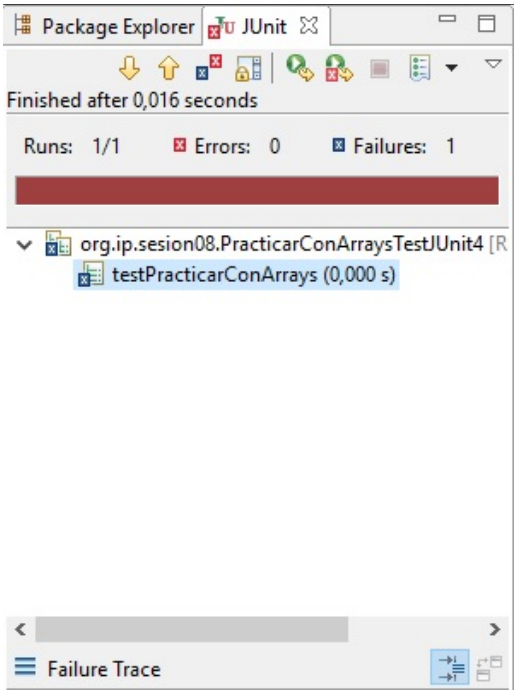


Y nos aparecerá una nueva vista, **JUnit**, al lado del **Package Explorer**, tal y como se indica en la siguiente figura

Al ejecutar un test unitario que funcione correctamente nos saldrá la **tan deseada** barra **verde**.

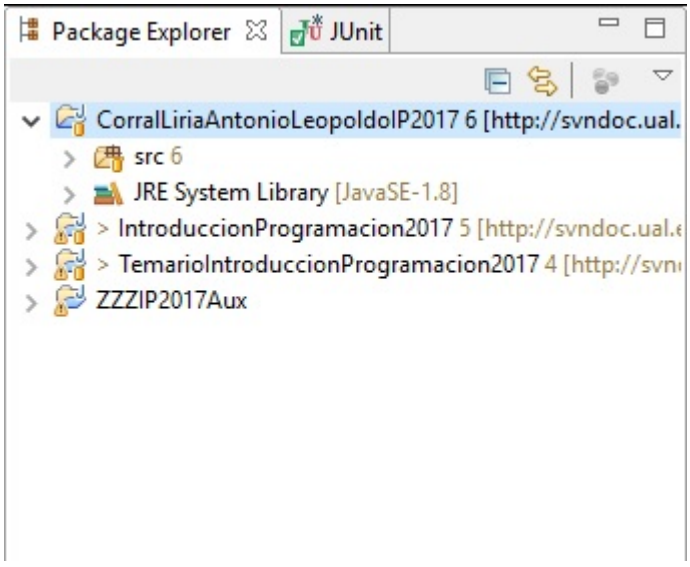


Por el contrario, si el test no funciona correctamente, tendremos la **no tan deseada** barra **roja**.

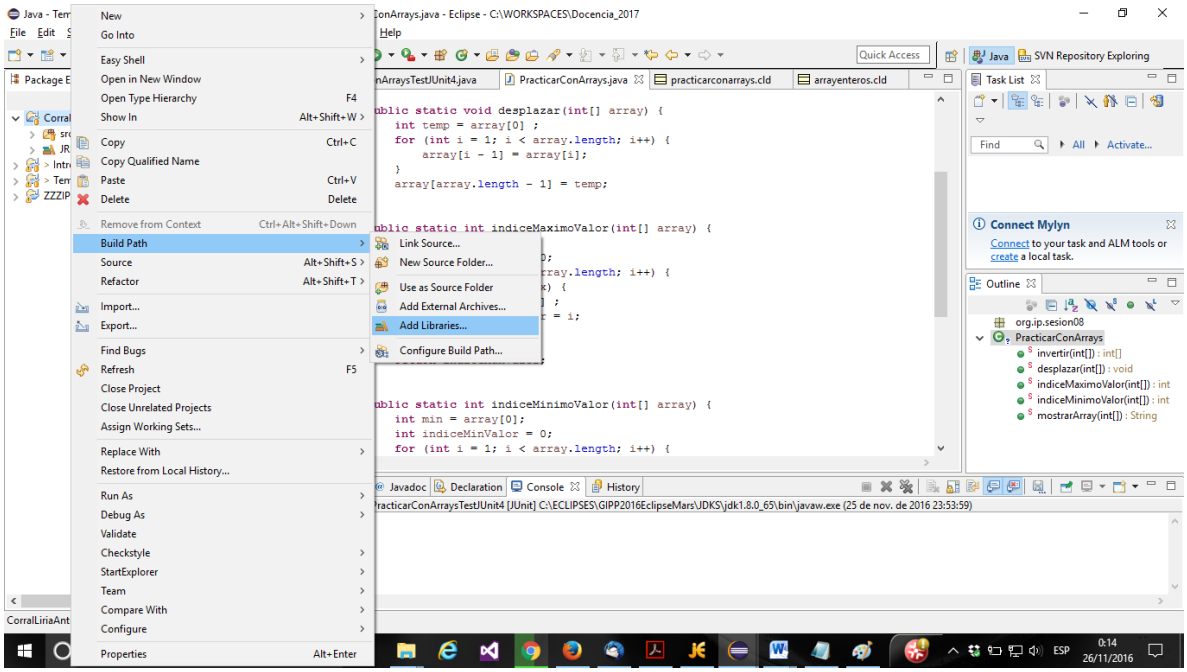


Como podréis imaginar, … el objetivo es **pasar el test**, y para ello la barra deberá estar en color **verde**.

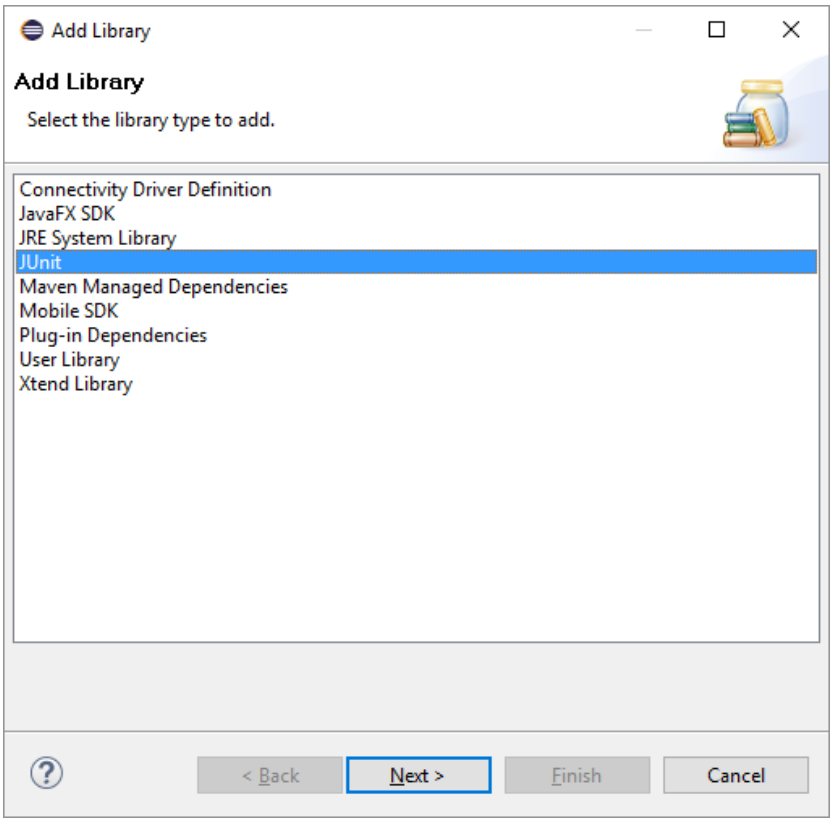
2. En segundo lugar, deberemos hacer que nuestro proyecto reconozca y pueda ejecutar los test unitarios. Para ello debemos hacer lo siguiente, partimos de nuestro proyecto personal desde la perspectiva Java en Eclipse, en mi caso **CorralLiriaAntonioLeopoldoIP2017**, tal y como se muestra en la siguiente figura



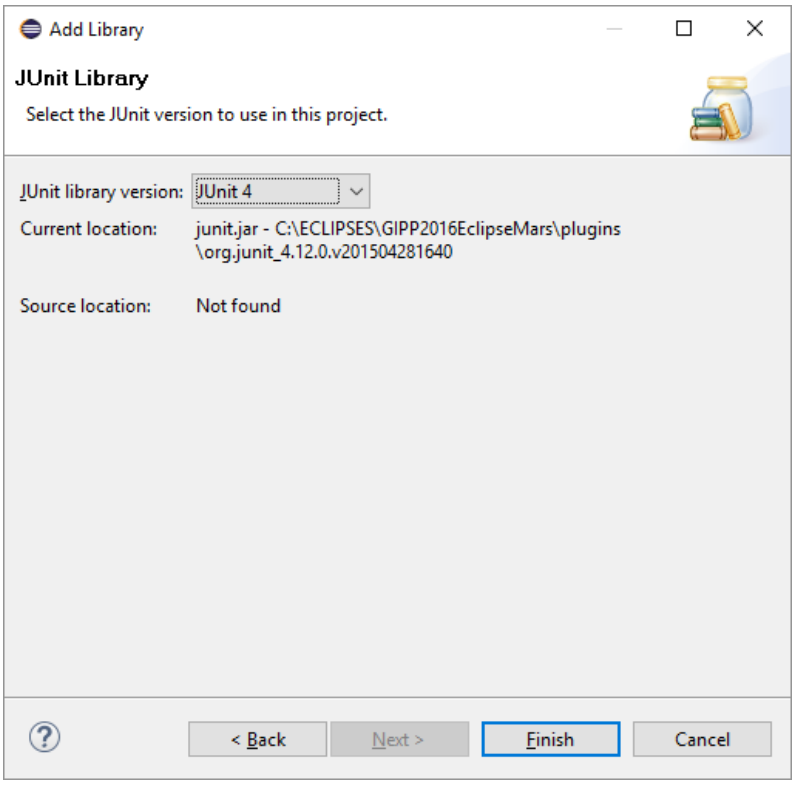
Ahora hacemos clic con el botón derecho del ratón sobre nuestro proyecto personal y después seleccionamos **Build Path > Add Libraries**, tal y como se muestra en la siguiente figura



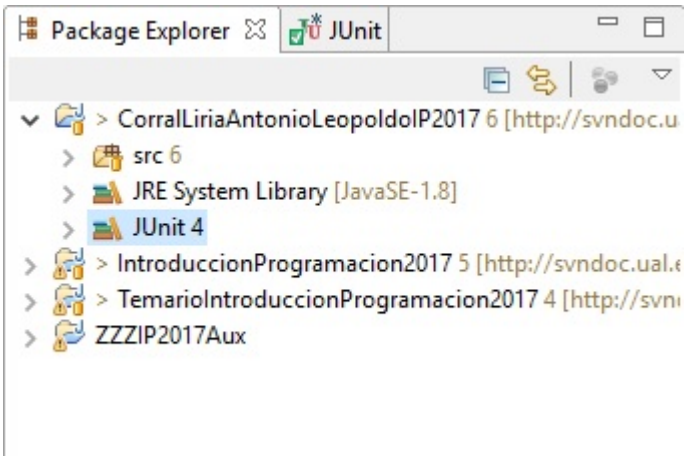
Después de realizar esta acción nos aparecerá la siguiente pantalla



Ahora seleccionamos **JUnit** y pulsamos **Next**.



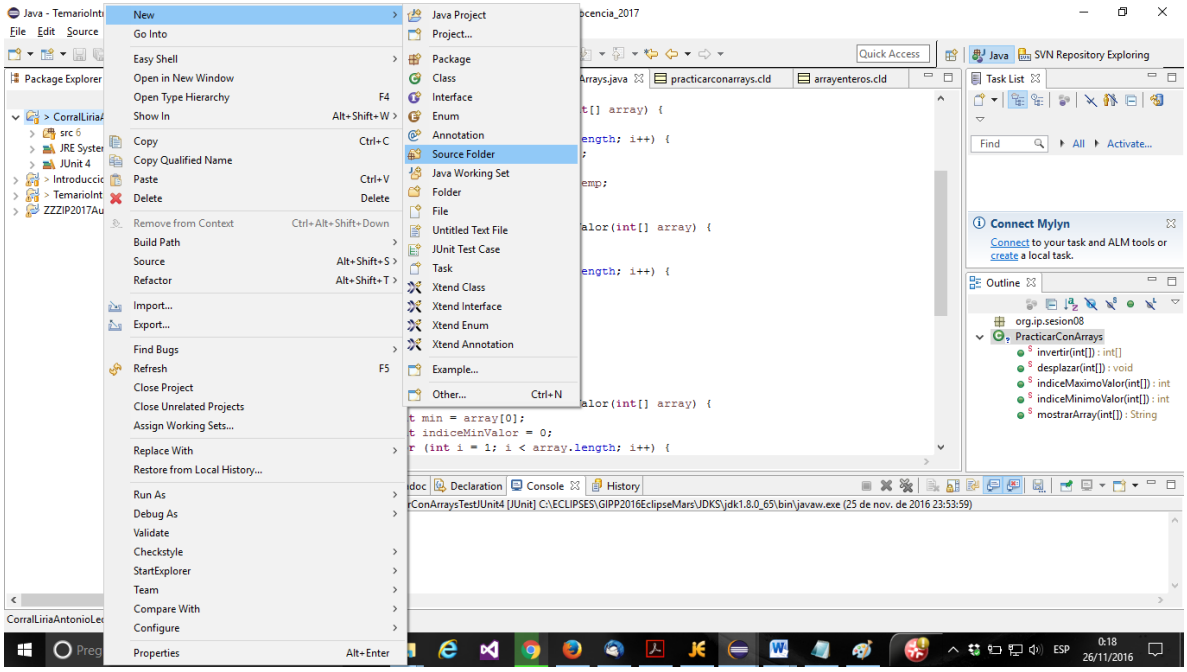
En este momento nos aseguramos de que en **JUnit library version:** está seleccionada **JUnit 4** y pulsamos en **Finish**. Al realizar esto, habremos incluido en nuestro proyecto personal la librería **JUnit 4**, tal y como se muestra en la siguiente pantalla (recordar que este es el proceso para añadir cualquier librería Java en Eclipse)



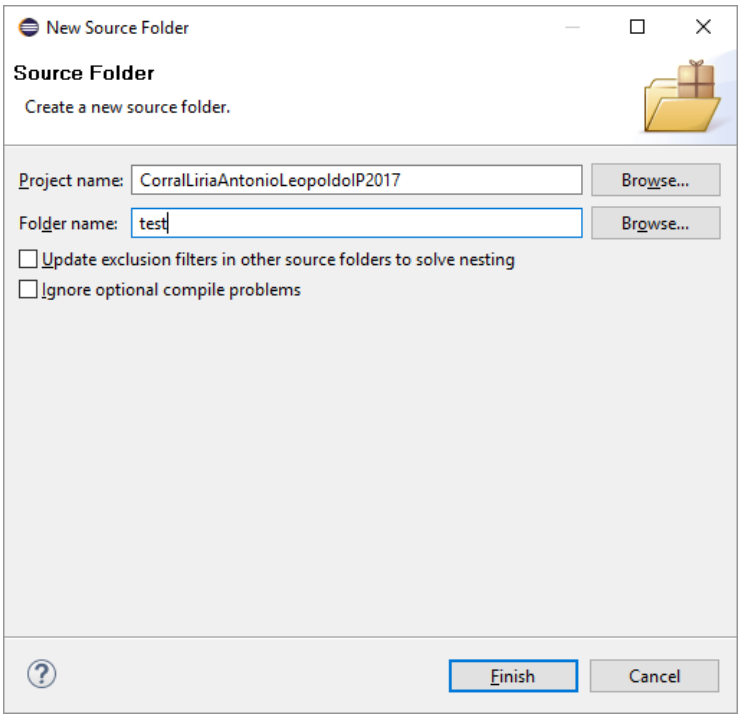
ahora podemos ejecutar en nuestro proyecto los test unitarios que se nos proporcionen en las sesiones de prácticas (prácticas de laboratorio).

3. En tercer lugar, y una vez hecho este proceso de incluir la librería **JUnit 4** en nuestro proyecto Java personal, vamos a ver los últimos pasos para poder ejecutar los test unitarios que se nos pasen en las sesiones de prácticas.

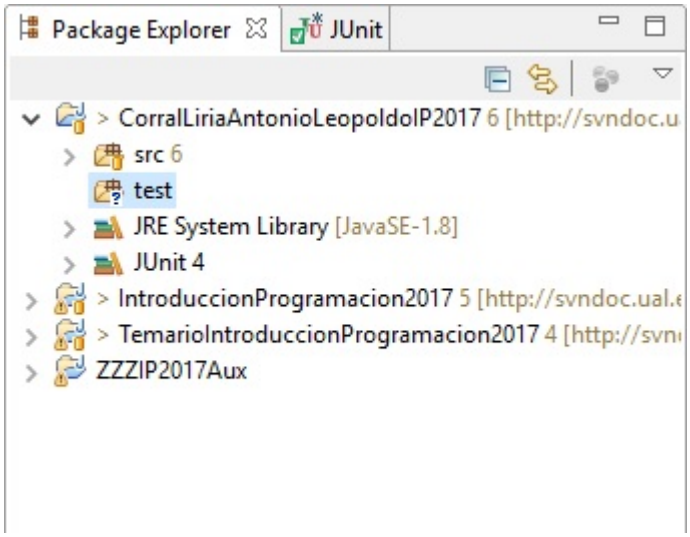
En primer lugar, creamos una nueva carpeta de código (**test**) que cuelgue de nuestro proyecto tal y como le sucede a la carpeta de código **src**. Para ello, hacemos clic con el botón derecho del ratón sobre nuestro proyecto personal y después seleccionamos **New** > **Source Folder**, tal y como se muestra en la siguiente figura



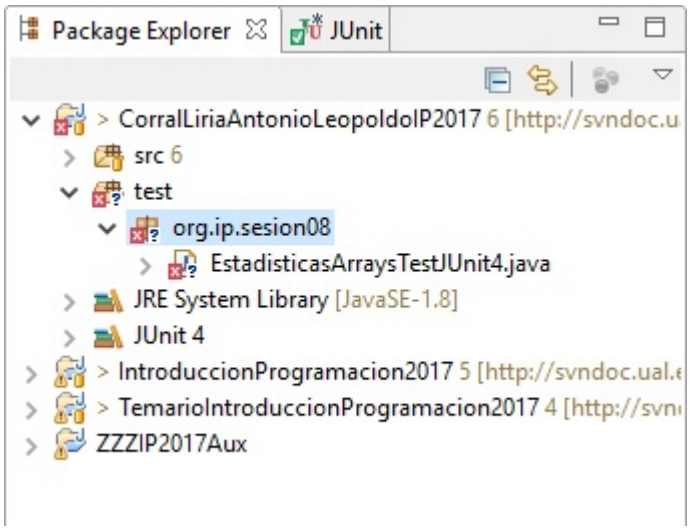
Después de realizar esta acción nos aparecerá la siguiente pantalla



Escribimos **test** en el campo asociado a **Folder name:** y pulsamos en **Finish**. Al realizar esto, habremos creado en nuestro proyecto personal una carpeta de código denominada **test**, donde almacenar los test unitarios que se nos proporcionen en cada sesión de prácticas, tal y como se muestra en la siguiente pantalla.

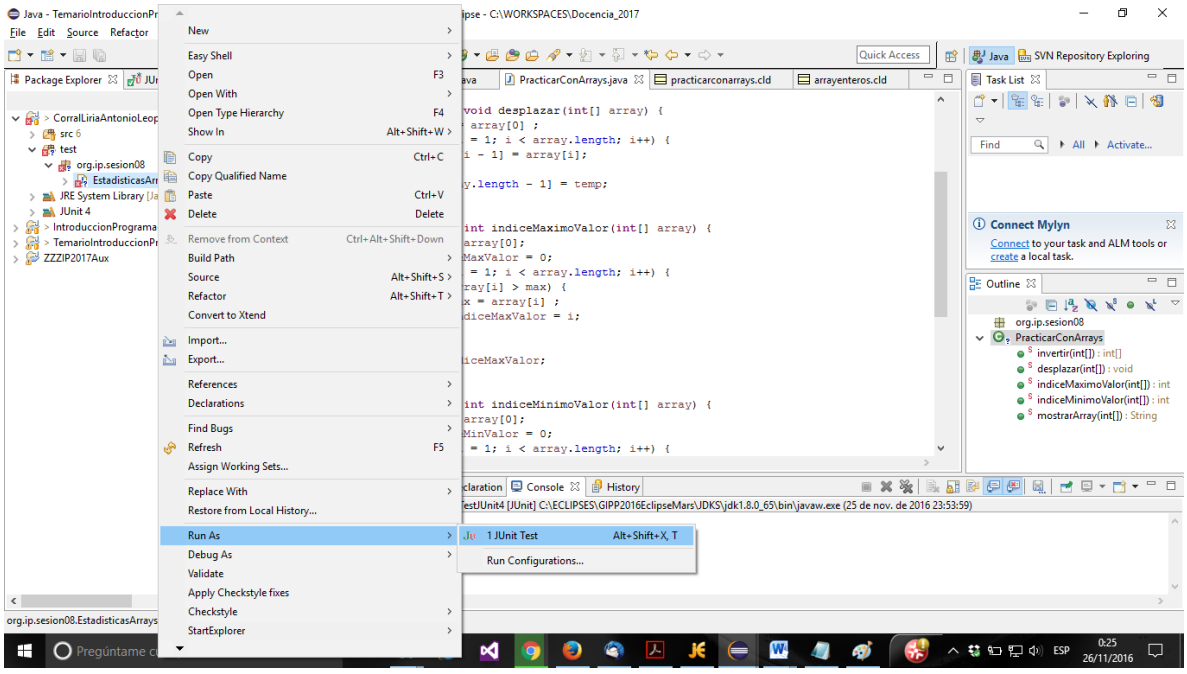


Ahora, dentro de la carpeta de código **test** vamos a crear los correspondientes paquetes, exactamente igual a como están en la carpeta **src** para que haya una correspondencia directa con los fuentes a testear. Por ejemplo, si nos han proporcionado en la sesión 08 el test unitario **EstadisticasArraysTestJUnit4.java**, debemos crear un nuevo paquete (**New > Package**) denominado **org.ip.sesion08** en la carpeta **test** (igual que tenemos en **src**) y luego copiar el archivo de test unitario en dicho paquete (**Copy & Paste**). En resumen, el resultado es el que se muestra en la siguiente figura

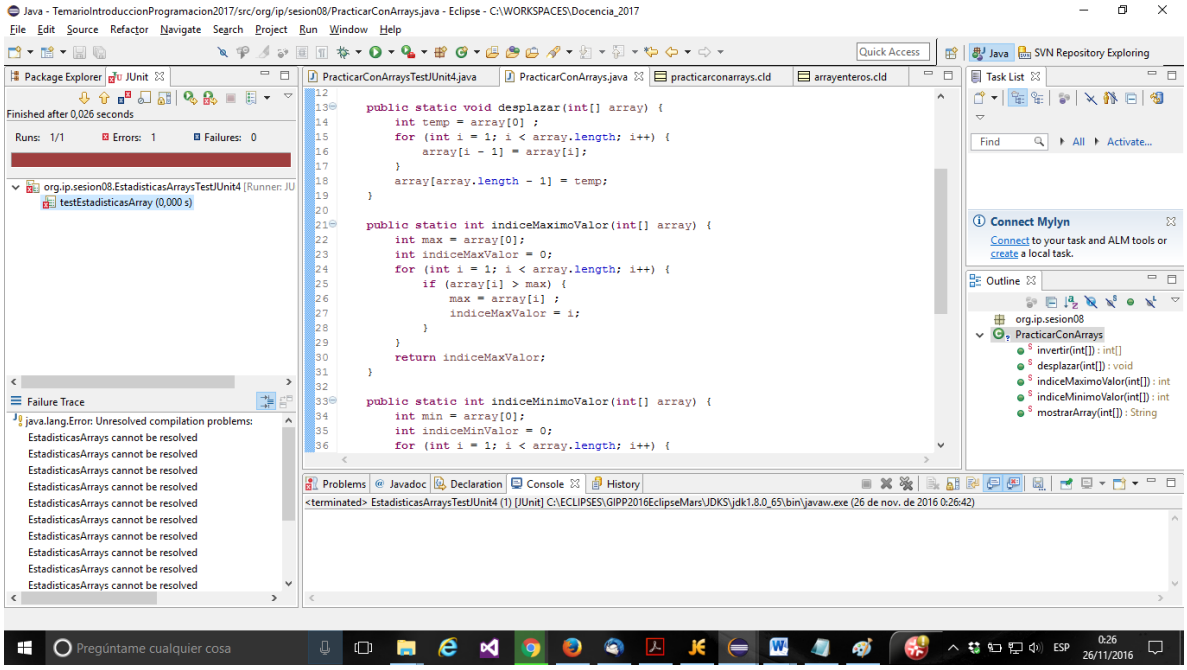


No debemos alarmarnos porque existan errores en el archivo de test unitarios (eso es normal al empezar a verificar un test unitario), esto es debido a que no han podido resolverse las clases y métodos que hay asociadas a los test.

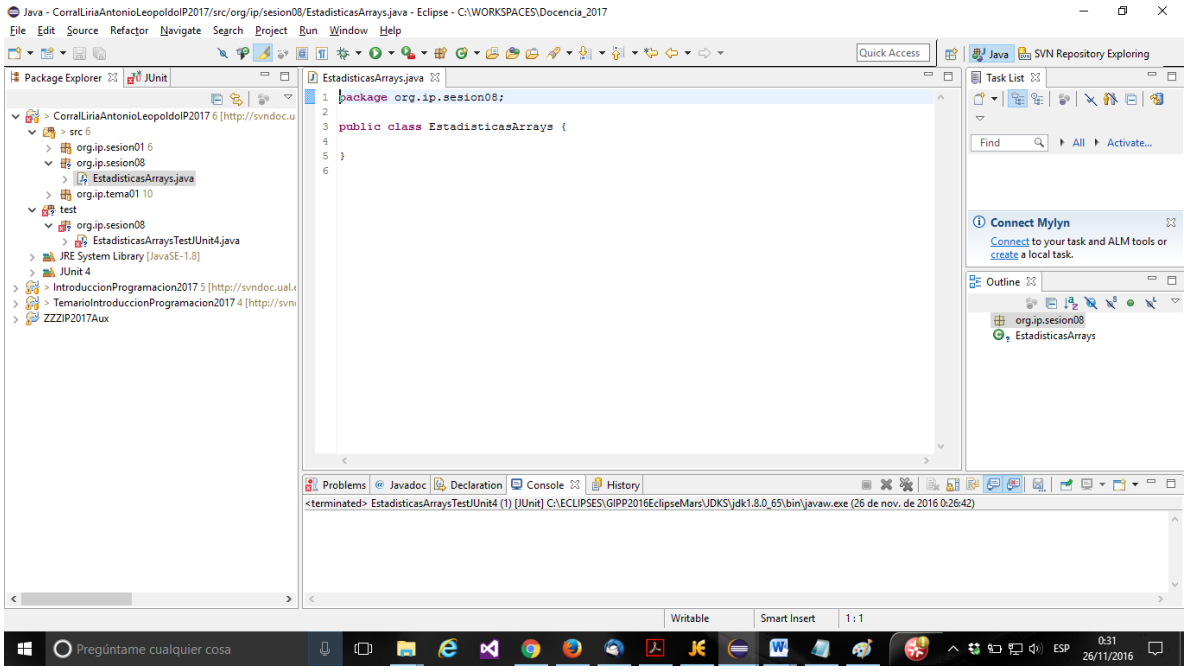
Obviamente, si ahora ejecutamos el test unitario. Para ello, nos situamos sobre el archivo de test unitario que queremos ejecutar (**EstadisticasArraysTestJUnit4.java**) y con el botón derecho de ratón hacemos clic en **Run As > JUnit Test**



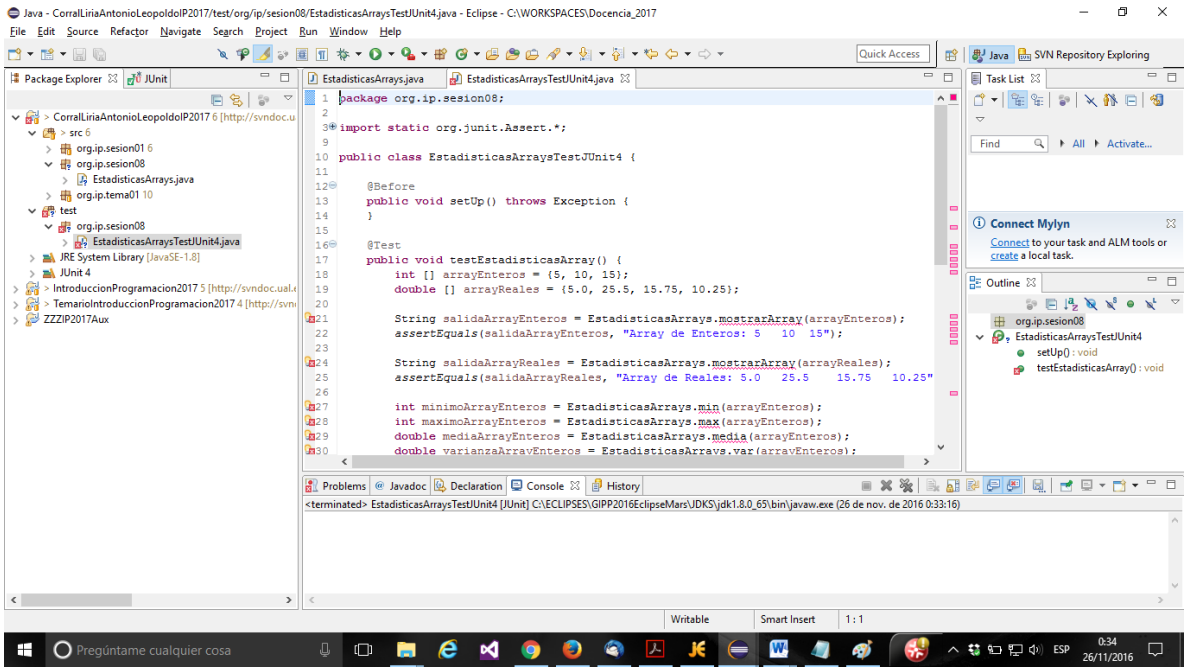
Nos aparecerá la no tan deseada barra **roja**.



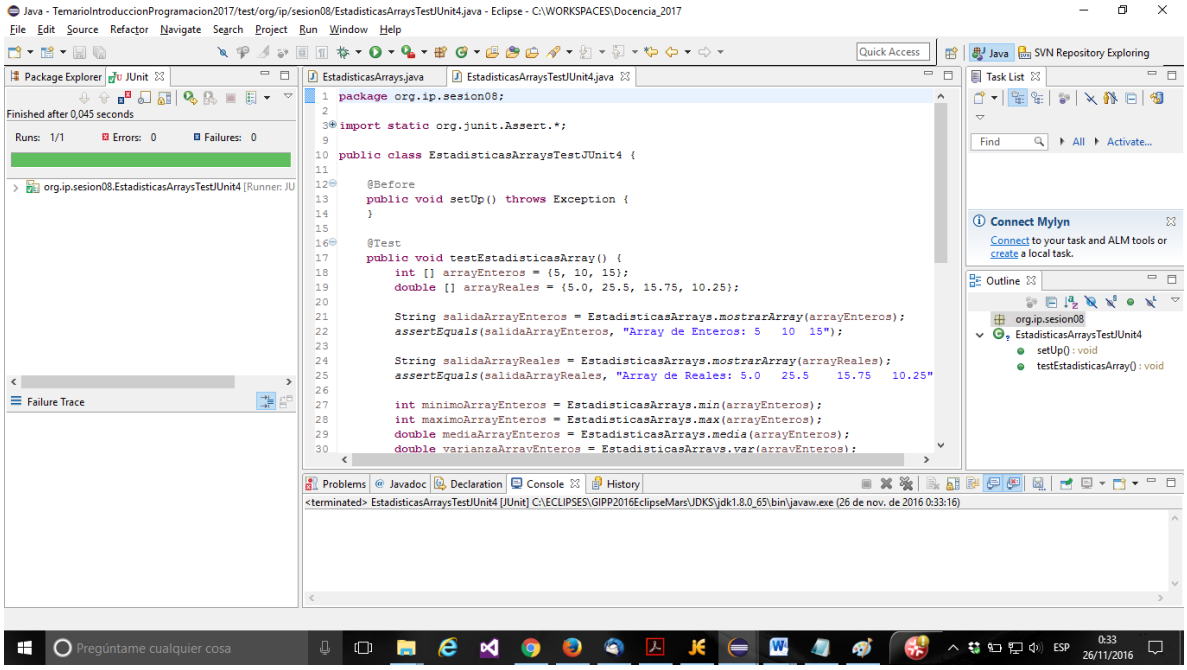
Para empezar a trabajar e ir progresivamente pasando los test unitarios debemos irnos a la pestaña **Package Explorer** y en la carpeta **src** crear el paquete (**New > Package**) **org.ip.sesion08** (si no lo tenemos creado ya), crear la clase (**New > Class**) que se nos pide en el test (**EstadisticasArrays**) e ir añadiendo en ella los métodos que se demandan en el test unitario o que se nos proporcionen mediante el diagrama de clases UML. Obviamente, los métodos deben de realizar la función para la que están diseñados y deben de verificar los **asserts** aparecen en el test unitario.



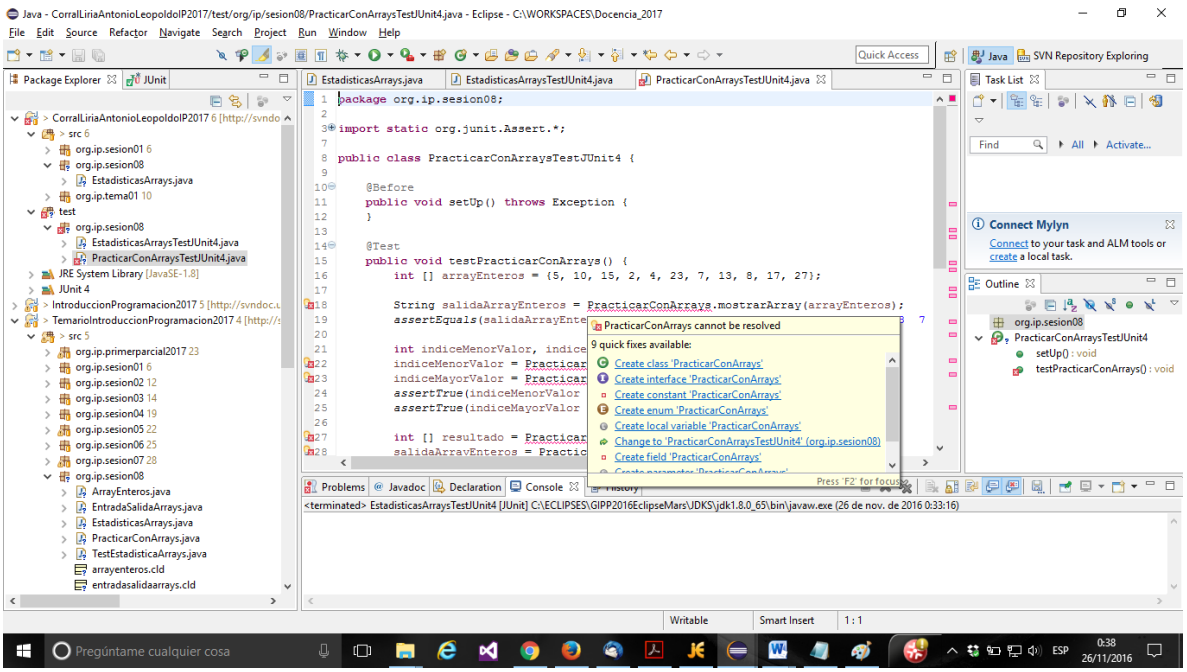
Y a partir de ahora, a desarrollar todos los métodos que hay que implementar en la clase **EstadisticasArrays** para que se verifiquen los test unitarios, es decir, para que desaparezcan todos los errores **rojos**



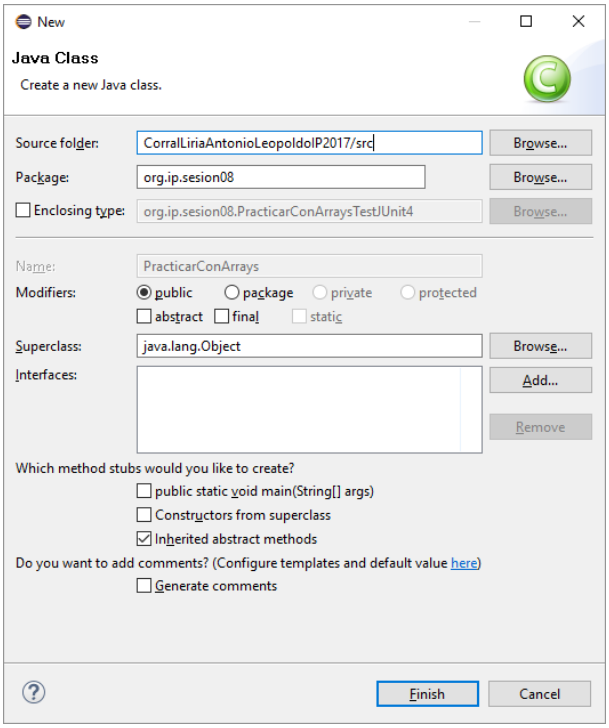
y la barra nos salga **verde** al ejecutarlo (**Run As > JUnit Test**).



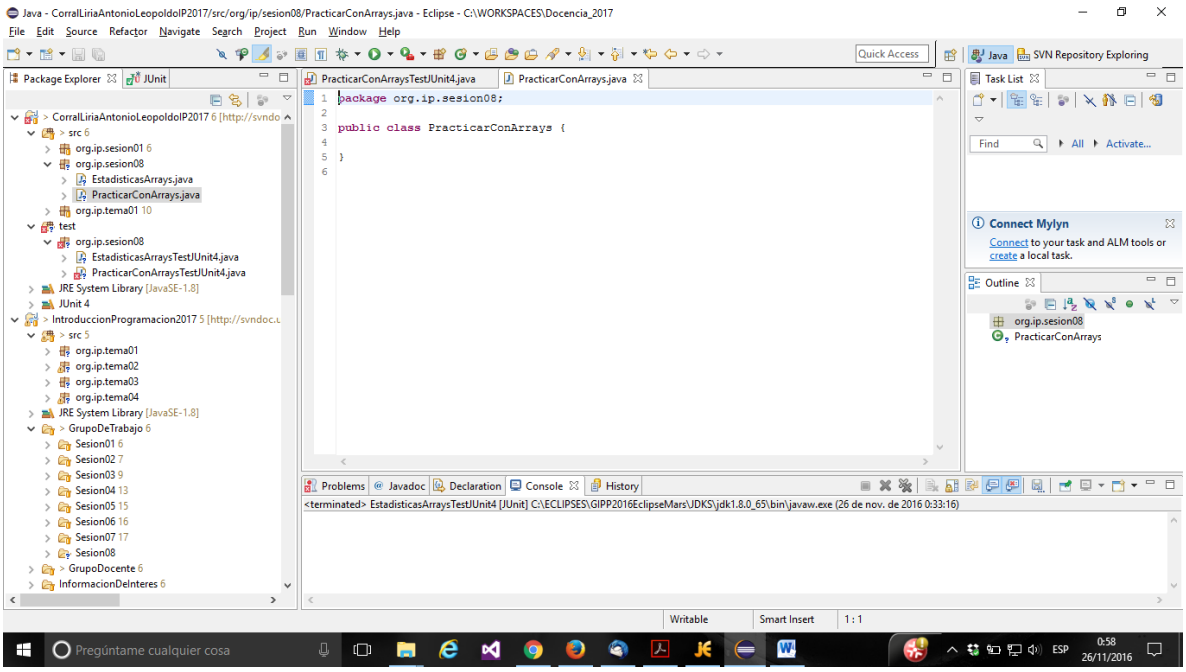
Eclipse nos facilita la labor de implementación en base a los test. Por ejemplo, para el caso del segundo test **PracticarConArraysTestJUnit4.java** podemos ver en la siguiente figura que nos aparece que la clase **PracticarConArrays** no está implementada, subrayada en color **rojo**. Si nos posicionamos sobre ella con el ratón nos aparecerá lo siguiente



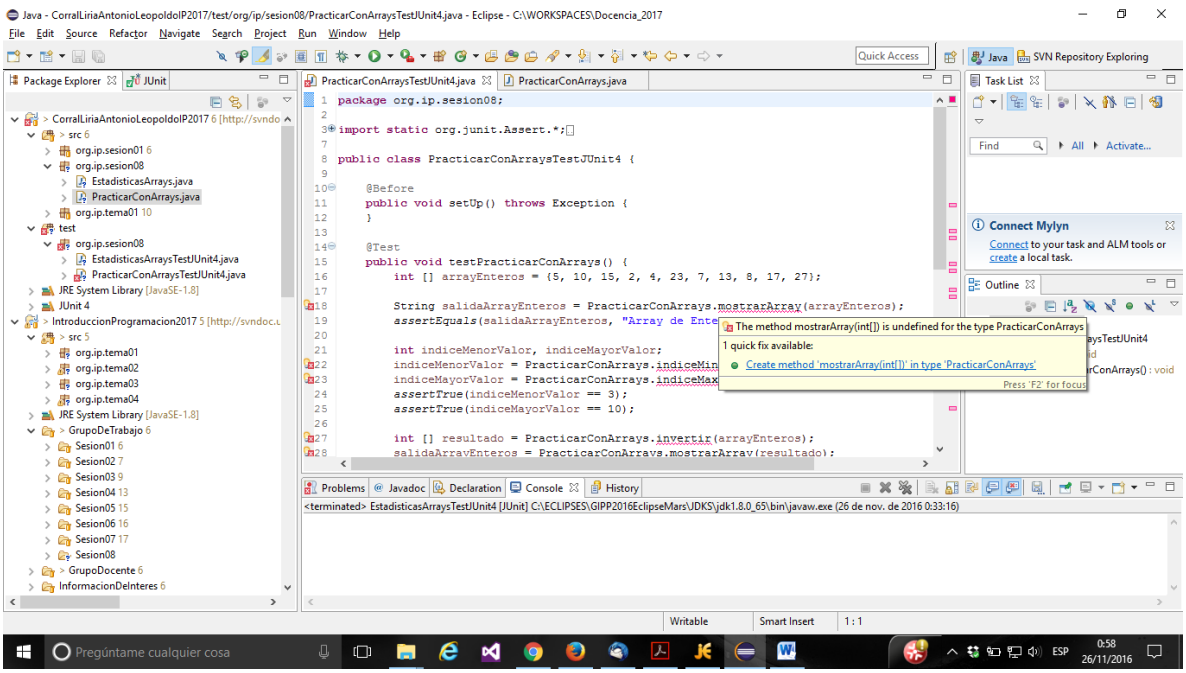
Si hacemos clic con el ratón en **Create class …** nos aparecerá la ventana para la creación de dicha clase, tal y como vemos en la siguiente figura. Ahora cambiamos en el campo **Source Folder** *CorralLiriaAntonioLeopoldoIP2017/test* por *CorralLiriaAntonioLeopoldoIP2017/src* hacemos clic en **Finish** y nos creará la clase en nuestro paquete en **src**.



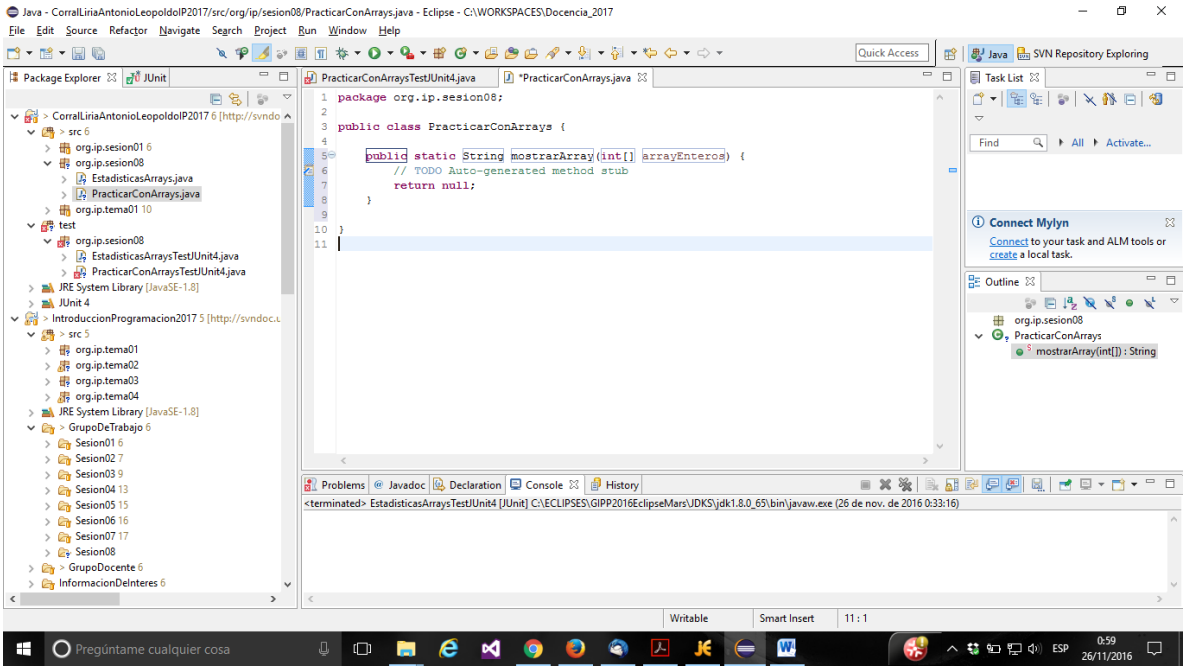
Es decir, nos aparecerá la clase creada (**PracticarConArrays**) en el paquete **src.org.ip.sesion08** tal y como se muestra en la siguiente pantalla.



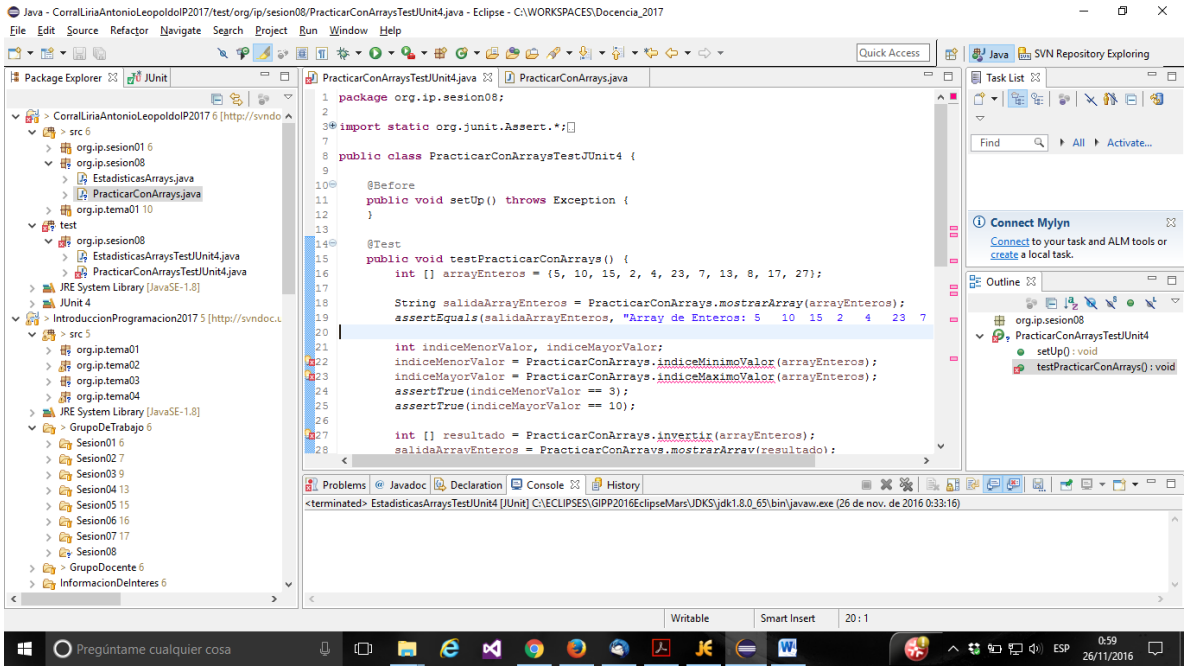
Ahora podremos comprobar que al crearse la clase ya no aparece marcada (subrayada) en **rojo**, ahora lo que aparece (subrayado) en **rojo** son los métodos que deben estar implementados en dicha clase (e implementados correctamente para que se pase el test). Si ahora nos volvemos a colocar sobre el método, Eclipse nos volverá a facilitar la tarea de creación tal y como se muestra a continuación



Es decir, Eclipse nos permite crear el método asociado en la clase a la que debe pertenecer. Haciendo clic con el ratón en la opción que nos aparece … **Create method …**



Ahora vemos que la cabecera del método estático **mostrarArray** nos aparece en su clase **PracticarConArrays**. Y ahora sólo nos queda implementar el método para que pueda pasar el **assert** asociado dentro del test. Es decir, si volvemos al test y guardamos el proyecto (no lo olvide) vemos que el método ya no aparece subrayado en **rojo**



Ahora debemos implementar el método **mostrarArray** para se verifique el **assert** que hay asociado

assertEquals(salidaArrayEnteros, "Array de Enteros: 5 10 15 2 4 23 7 13 8 17 27");

Podemos observar que dentro de **testPracticarConArrays()** hay una llamada al método **assertEquals(***esperado, real***)**. Este método comprueba si la expresión del resultado *esperado* (String **salidaArrayEnteros**), que es la salida del método **mostrarArray**, coincide con el resultado *real* (**"Array de Enteros: 5 10 15 2 4 23 7 13 8 17 27"**), y trasmite el resultado a JUnit. De esta forma, al terminar la ejecución de todos los tests, JUnit nos informa de aquellos que se han ejecutado correctamente y de aquellos que han producido algún error. En este ejemplo la comprobación se cumple y JUnit nos indica que todos los tests han sido superados con éxito. Otro tipo común de **assert** es **assertTrue()** que comprueba si la expresión que se le pasa como argumento es cierta. A continuación se exponen los métodos de que dispone JUnit para hacer comprobaciones

Método assertxxx() de JUnit	**Qué comprueba**
assertTrue(expresión)	comprueba que *expresión* evalúe a **true**
assertFalse(expresión)	comprueba que *expresión* evalúe a **false**
assertEquals(esperado, real)	comprueba que *esperado* sea igual a *real*
assertNull(objeto)	comprueba que *objeto* sea **null**
assertNotNull(objeto)	comprueba que *objeto* no sea *null*
assertSame(objeto_esperado, objeto_real)	comprueba que *objeto_esperado* y *objeto_real* sean el mismo objeto
assertNotSame(objeto_esperado, objeto_real)	comprueba que *objeto_esperado* no sea el mismo objeto que *objeto_real*
fail()	hace que el test termine con fallo

Todo este proceso se debe repetir para cada uno de los **métodos** que aparezcan en el test, y hay que implementarlos correctamente para que verifiquen todos los **asserts** que en el test son requeridos. Una vez implementados todos los métodos correctamente, el test se pasará y nos aparecerá la tan deseada barra **verde**.

Nosotros, en esta sesión y en las que quedan para terminar el programa de prácticas de la asignatura de Introducción a la Programación, veremos ejemplos muy sencillos de test unitarios (como si de *main* se tratara), pero lo realmente importante es comprender el concepto de pruebas o test unitarios y aprender a manejar **JUnit** dentro de **Eclipse**. Para obtener más información de cómo hacer los test, es recomendable la lectura de la API de JUnit que podemos encontrar en *http://junit.sourceforge.net/javadoc/org/junit/package-summary.html*