TUTORIAL: Test unitarios en Java (JUnit con Eclipse)

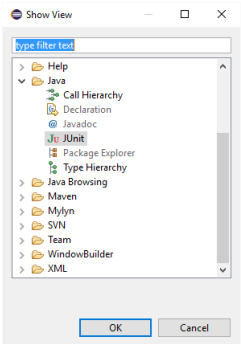
Una **prueba unitaria** o **test unitario** según *Wikipedia* es *una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado*. Para algunos, este tipo de pruebas es una pérdida de tiempo, aunque en muchas 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 la **sesión 08** utilizaremos test unitarios para que nos guíen en la comprobación del correcto funcionamiento de determinados ejercicios. En cursos anteriores eran las sesiones 09 y 10 las que utilizaban test unitarios, y por esa razón, en el texto y en muchas figuras aparece la **`sesion09`**, con lo que solo habrá que sustituir **`sesion09`** por **`sesion08`** para que todo funcione correctamente. 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 Ingeniería del Software. Además, será habitual el uso de test unitarios en sucesivas asignaturas de programación como: Metodología de la Programación, EDA 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.

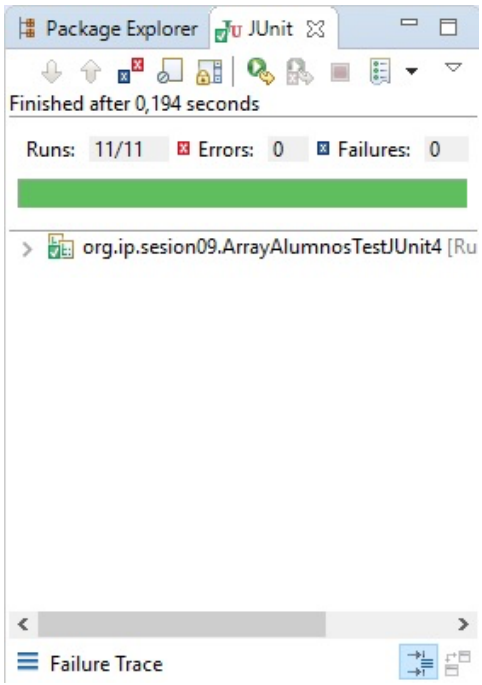
Para que nuestro proyecto personal Java de IP pueda ejecutar los test unitarios que nos proporcione el profesor, se deberán de realizar los siguientes pasos.

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

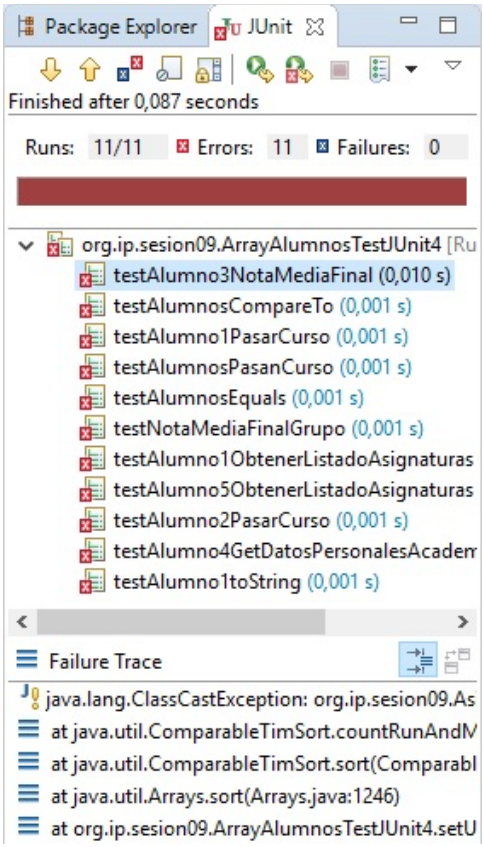


Y nos aparecerá una nueva vista, **JUnit**, al lado del **Package Explorer**, tal y como se indica en la siguiente figura

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

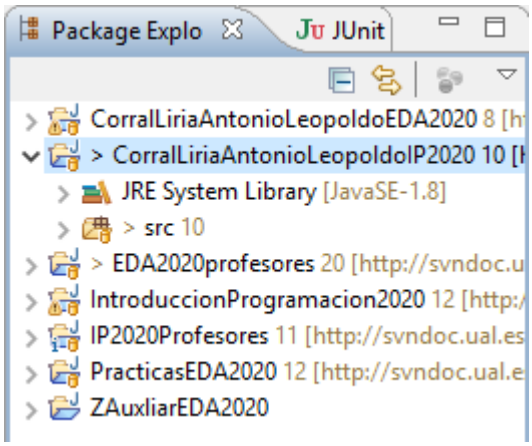


Por el contrario, si el test no funciona correctamente, tendremos la **no tan deseada** barra **roja**.

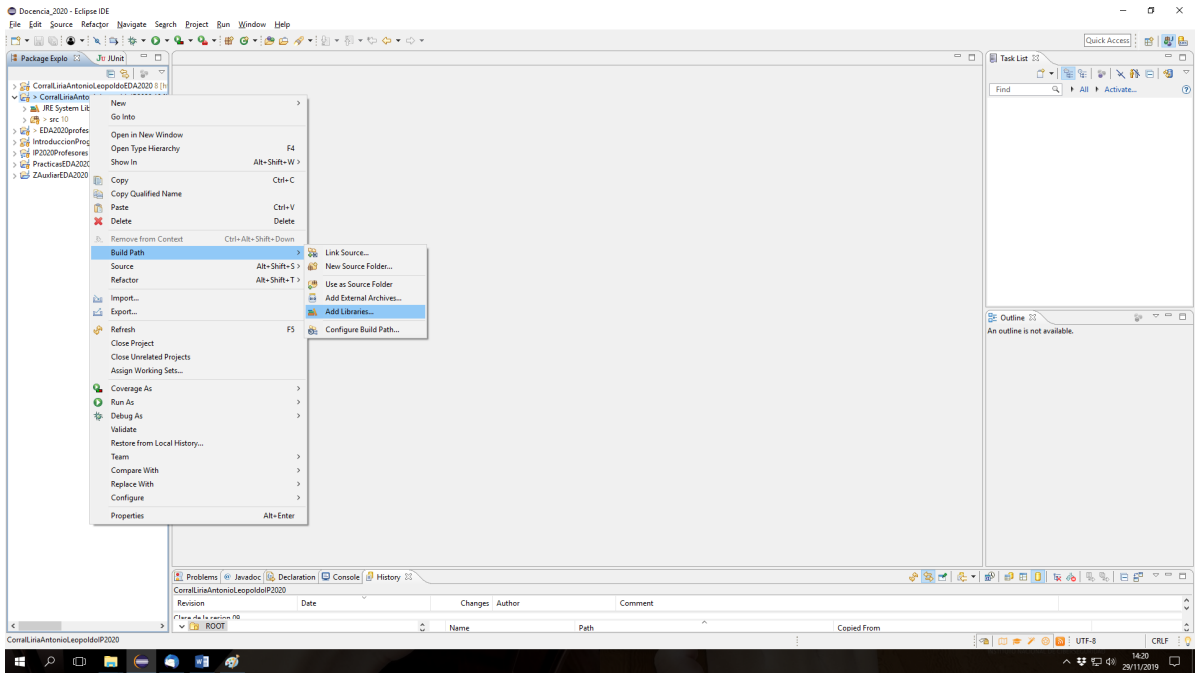


Como podréis imaginar, … el objetivo es **pasar el test**, y para ello la barra deberá estar en color **verde**.

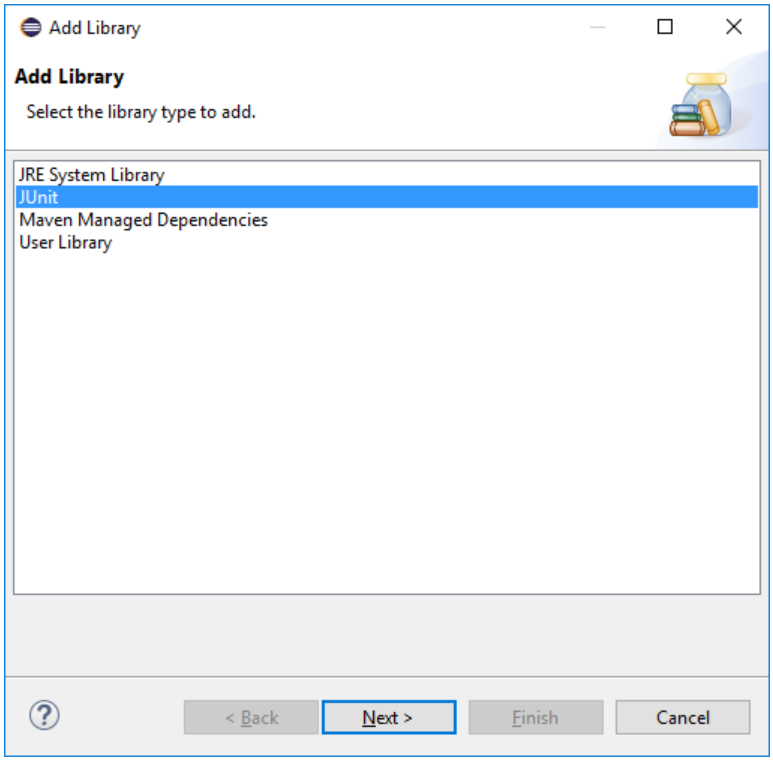
2. En segundo lugar, deberemos hacer que nuestro proyecto reconozca y pueda ejecutar los test unitarios. Para ello debemos hacer lo siguiente, partimos de **nuestro proyecto personal** desde la perspectiva Java en Eclipse, en mi caso **CorralLiriaAntonioLeopoldoIP2020**, tal y como se muestra en la siguiente figura



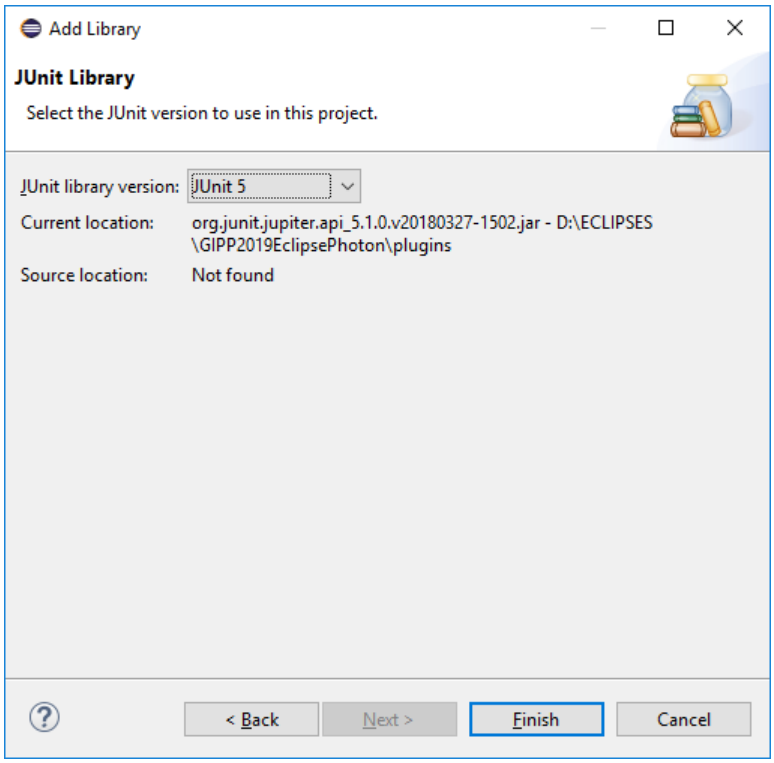
Ahora hacemos clic con el botón derecho del ratón sobre nuestro proyecto personal y después seleccionamos **Build Path > Add Libraries**, tal y como se muestra en la siguiente figura



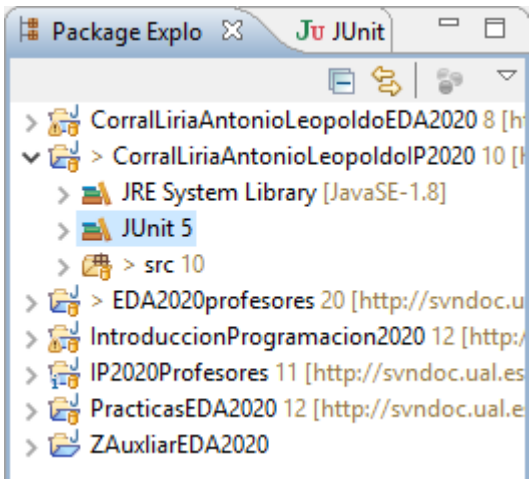
Después de realizar esta acción nos aparecerá la siguiente pantalla



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



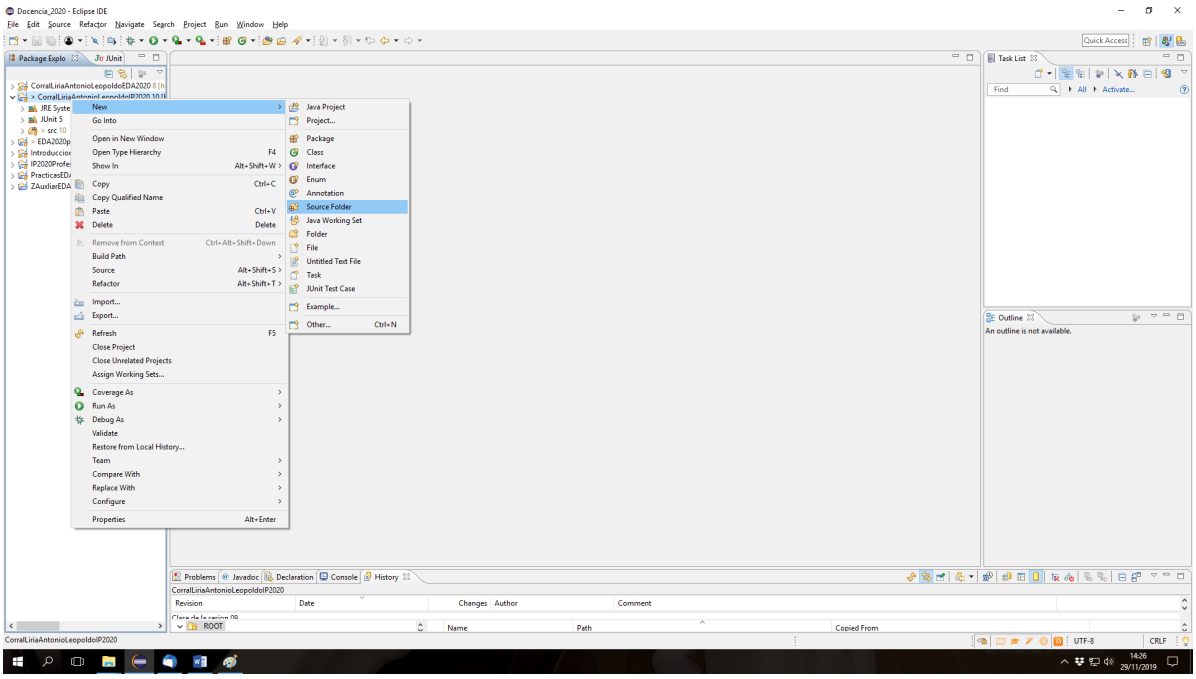
En este momento nos aseguramos de que en **JUnit library version:** está seleccionada **JUnit 5** y pulsamos en **Finish**. Al realizar esto, habremos incluido en nuestro proyecto personal la librería **JUnit 5**, tal y como se muestra en la siguiente pantalla (recordar que éste es el proceso para añadir cualquier librería Java en Eclipse)



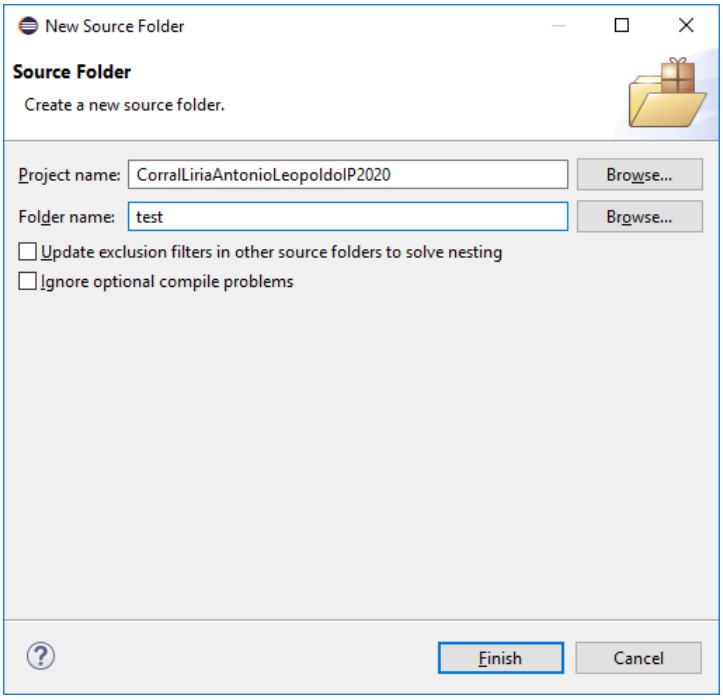
ahora podemos ejecutar en nuestro proyecto los test unitarios que se nos proporcionen en las sesiones de prácticas (prácticas del grupo de trabajo).

3. En tercer lugar, y una vez hecho este proceso de incluir la librería **JUnit 5** 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.

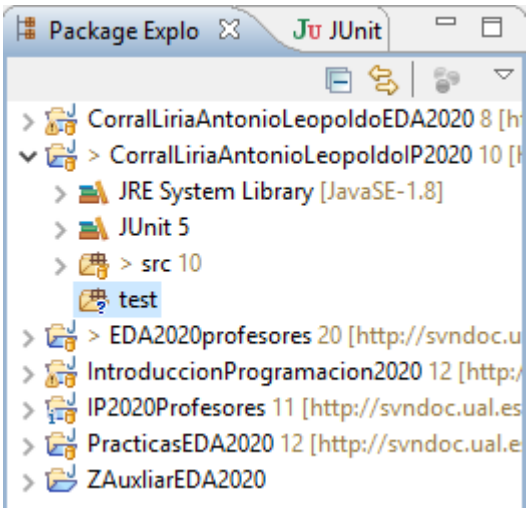
En primer lugar, creamos una nueva carpeta de código (**`test`**) que cuelgue de nuestro proyecto tal y como le sucede a la carpeta de código **`src`**. Para ello, hacemos clic con el botón derecho del ratón sobre nuestro proyecto personal y después seleccionamos **New > Source Folder**, tal y como se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



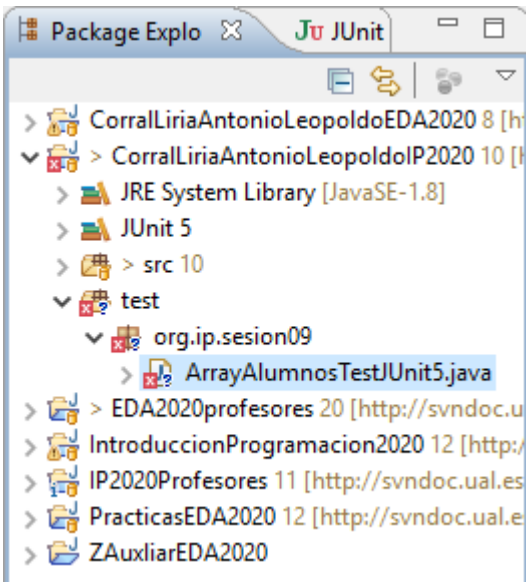
Después de realizar esta acción nos aparecerá la siguiente pantalla



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.

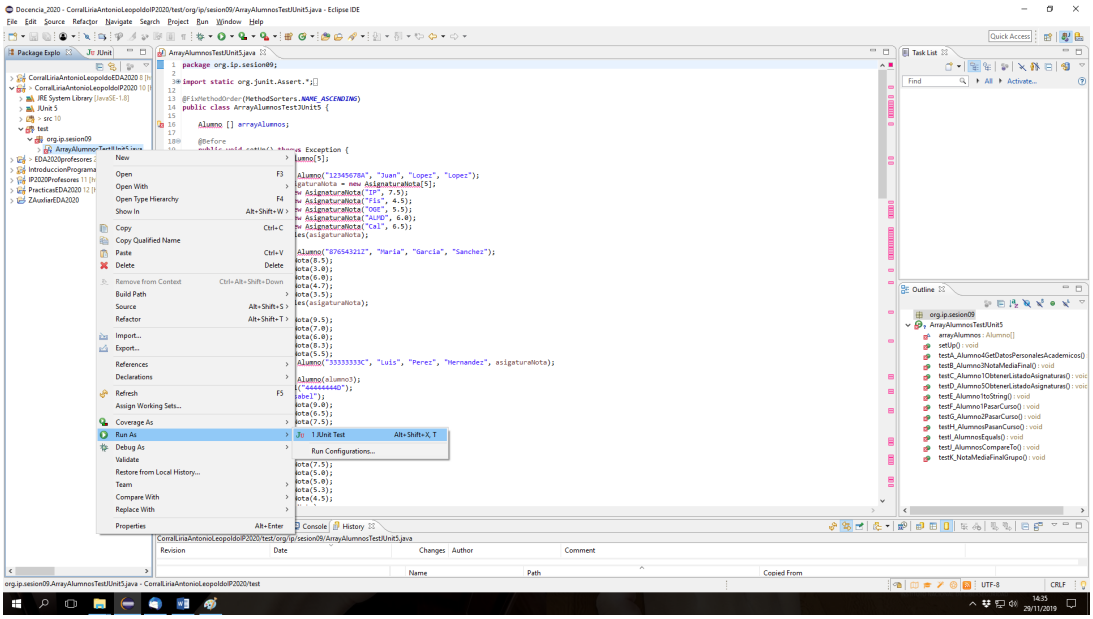


Ahora, dentro de la carpeta de código **`test`** vamos a crear los correspondientes paquetes, exactamente igual a como están en la carpeta **`src`** para que haya una correspondencia directa con los fuentes a testear. Por ejemplo, si nos han proporcionado en la sesión 09 el test unitario **ArrayAlumnosTestJUnit5.java**, debemos crear un nuevo paquete (**New > Package**) denominado **`org.ip.sesion09`** 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

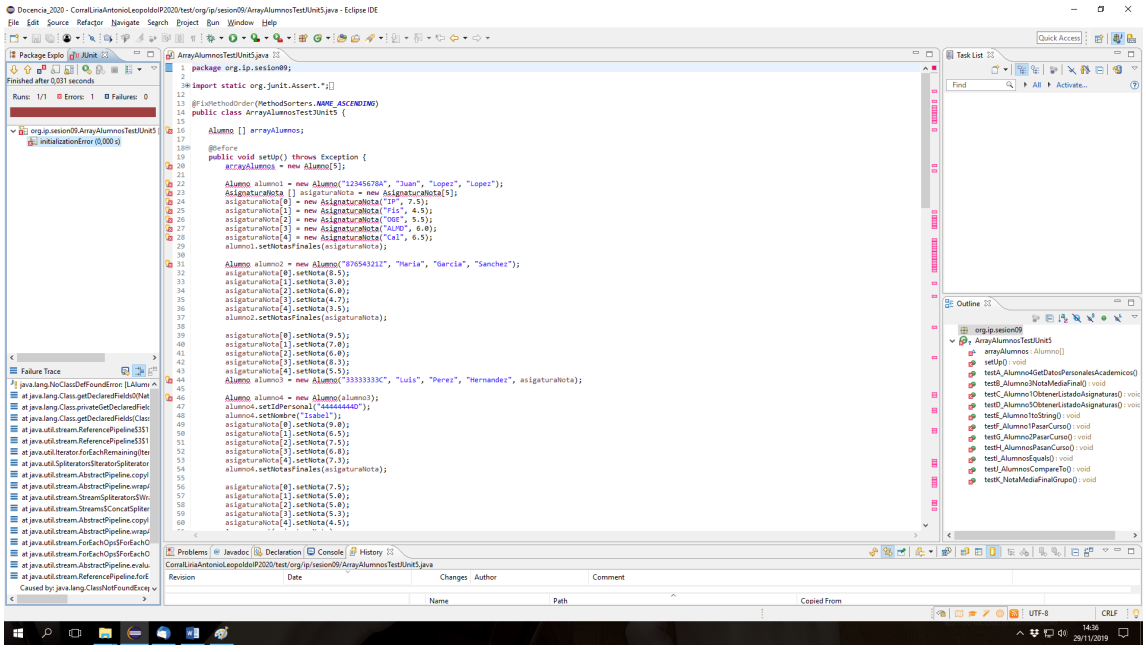


No debemos alarmarnos porque existan errores en el archivo de test unitarios (eso es normal al empezar a verificar un test unitario), esto es debido a que no han podido resolverse las clases y métodos que hay asociados a los test.

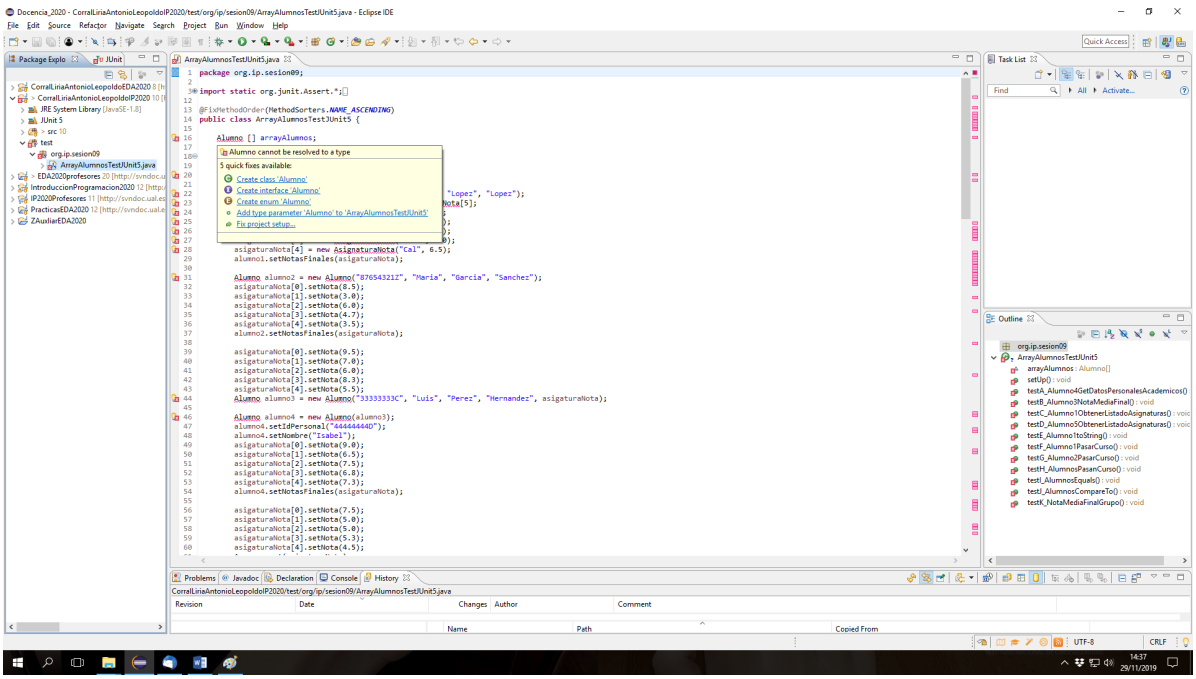
Obviamente, si ahora ejecutamos el test unitario. Para ello, nos situamos sobre el archivo de test unitario que queremos ejecutar (**ArrayAlumnosTestJUnit5.java**) y con el botón derecho de ratón hacemos clic en **Run As > JUnit Test**



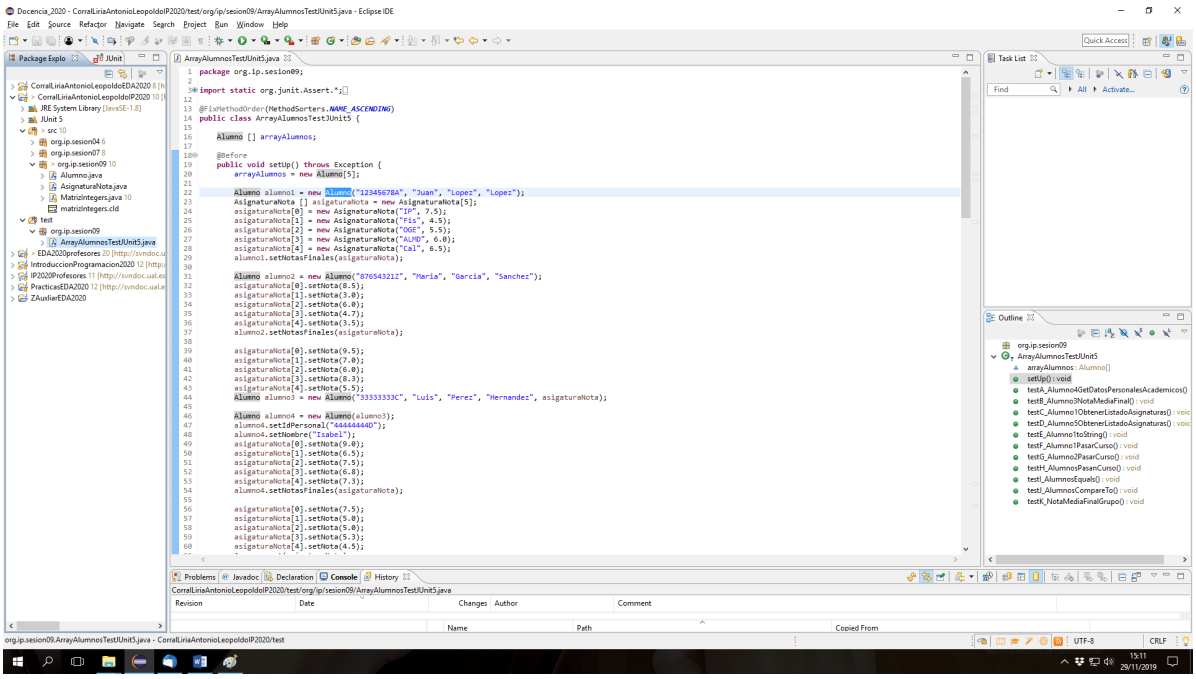
Nos aparecerá la no tan deseada barra **roja**.



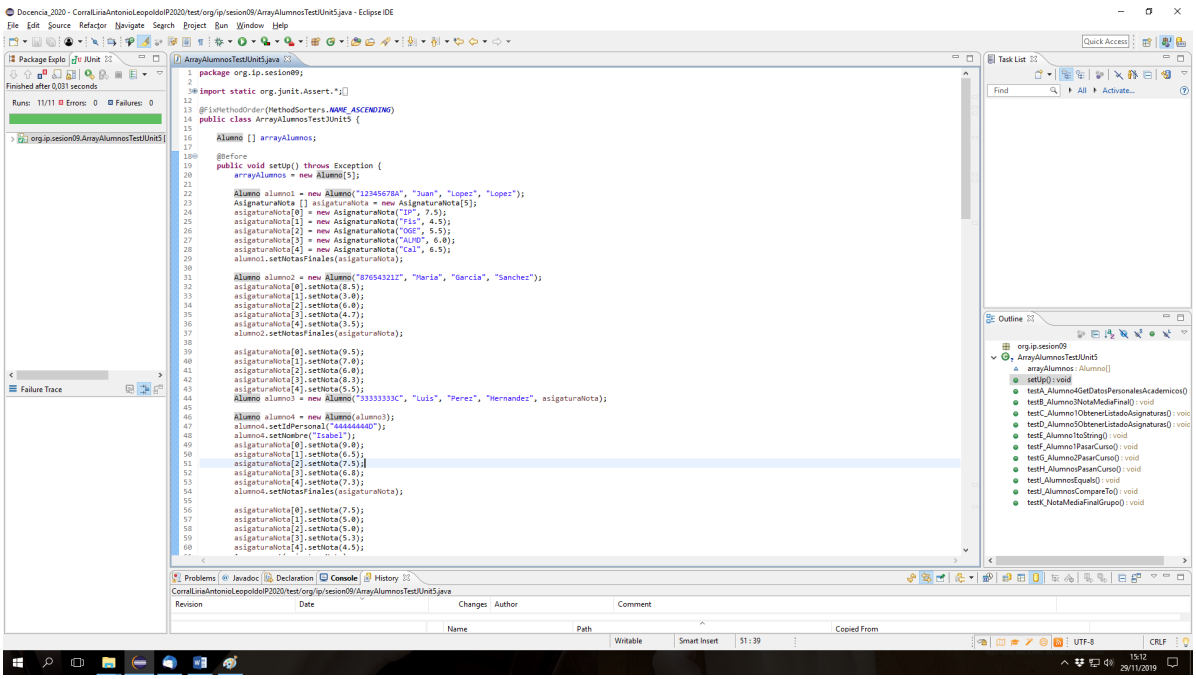
Para empezar a trabajar e ir progresivamente pasando los test unitarios debemos irnos a la pestaña **Package Explorer** y en la carpeta **`src`** crear el paquete (**New > Package**) **`org.ip.sesion09`** (si no lo tenemos creado ya), crear la clase (**New > Class**) que se nos pide en el test (**ArrayAlumnosTestJUnit5**) 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.



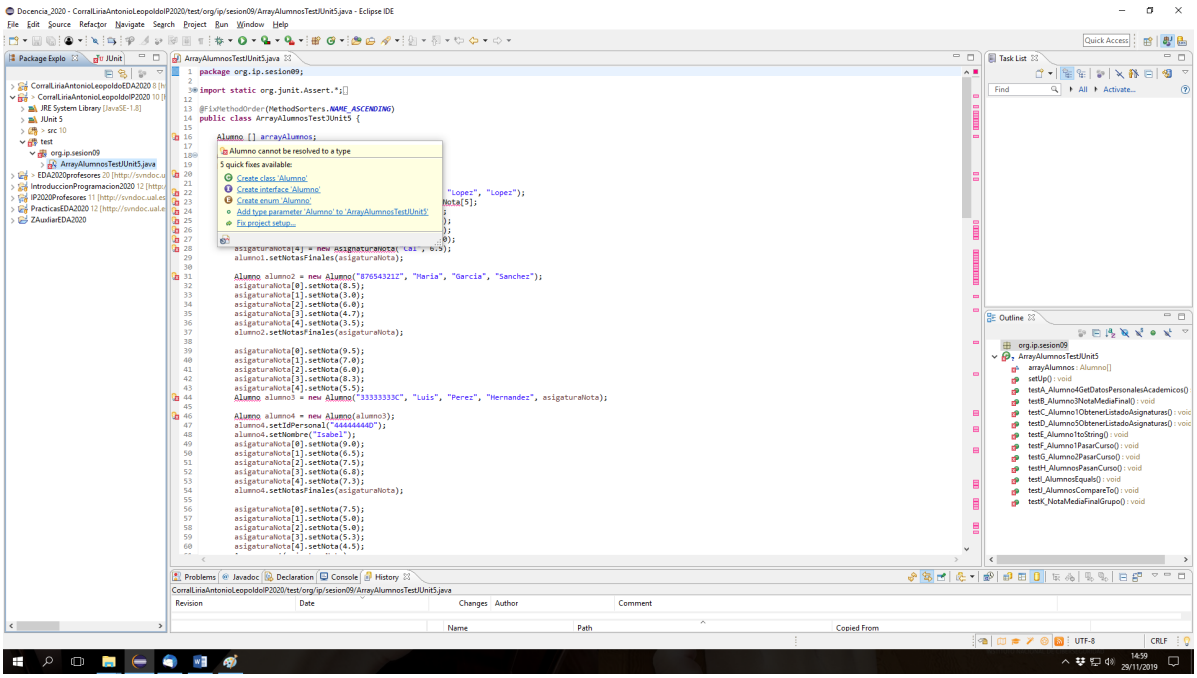
Y a partir de ahora, a desarrollar todos los métodos que hay que implementar en la clase **ArrayAlumnosTestJUnit5** para que se verifiquen los test unitarios, es decir, para que desaparezcan todos los errores **rojos**



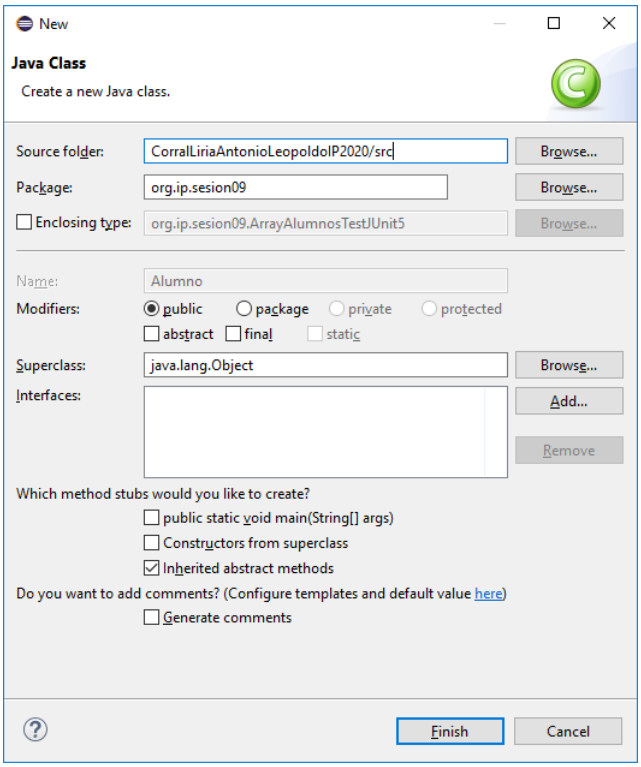
y la barra nos salga **verde** al ejecutarlo (**Run As > JUnit Test**).



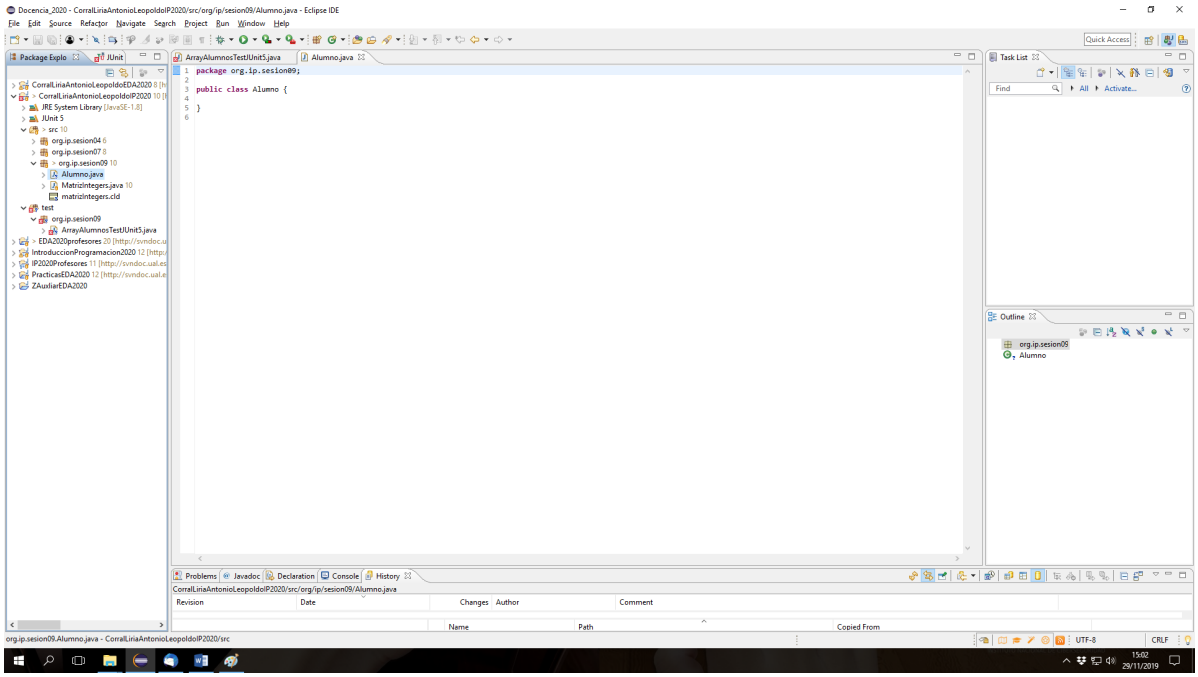
Eclipse nos facilita la labor de implementación en base a los test. Por ejemplo, para este test **ArrayAlumnosTestJUnit5.java** podemos ver en la siguiente figura que nos aparece que la *clase* `Alumno` no está implementada, subrayada en color **rojo**. Si nos posicionamos sobre ella con el ratón nos aparecerá lo siguiente



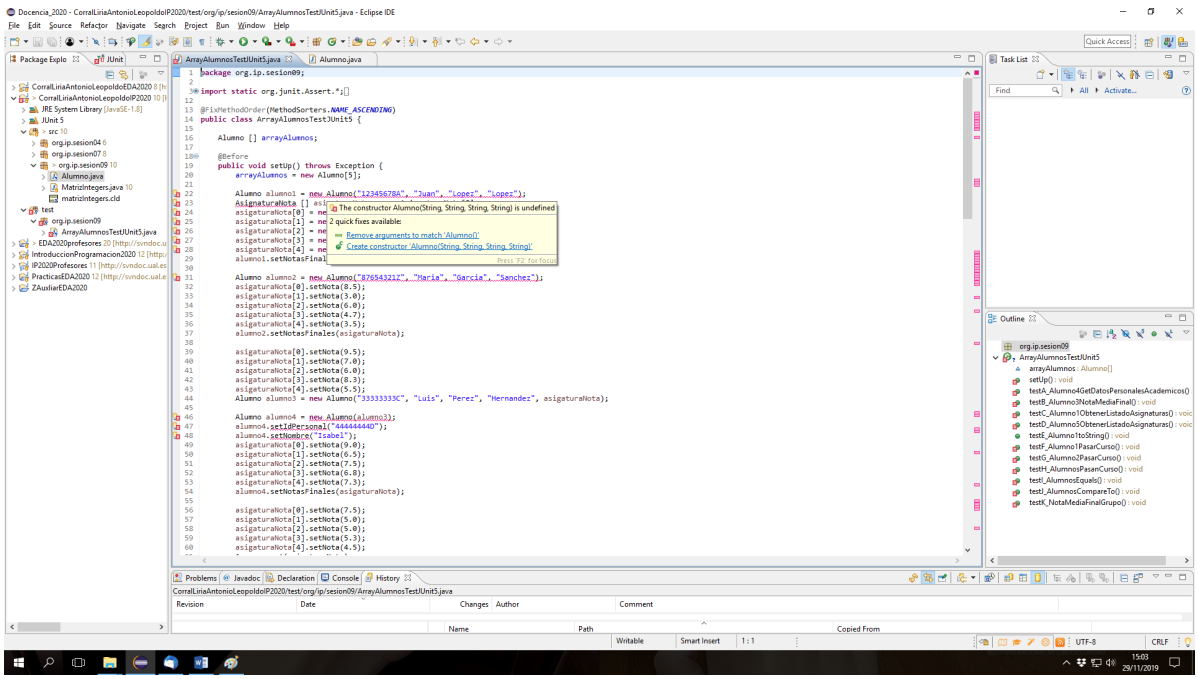
Si hacemos clic con el ratón en **Create class …** nos aparecerá la ventana para la creación de dicha clase, tal y como vemos en la siguiente figura. Ahora cambiamos en el campo **Source Folder** *CorralLiriaAntonioLeopoldoIP2020/test* por *CorralLiriaAntonioLeopoldoIP2020/src* hacemos clic en **Finish** y nos creará la clase en nuestro paquete que está en la carpeta de fuentes **`src`**.



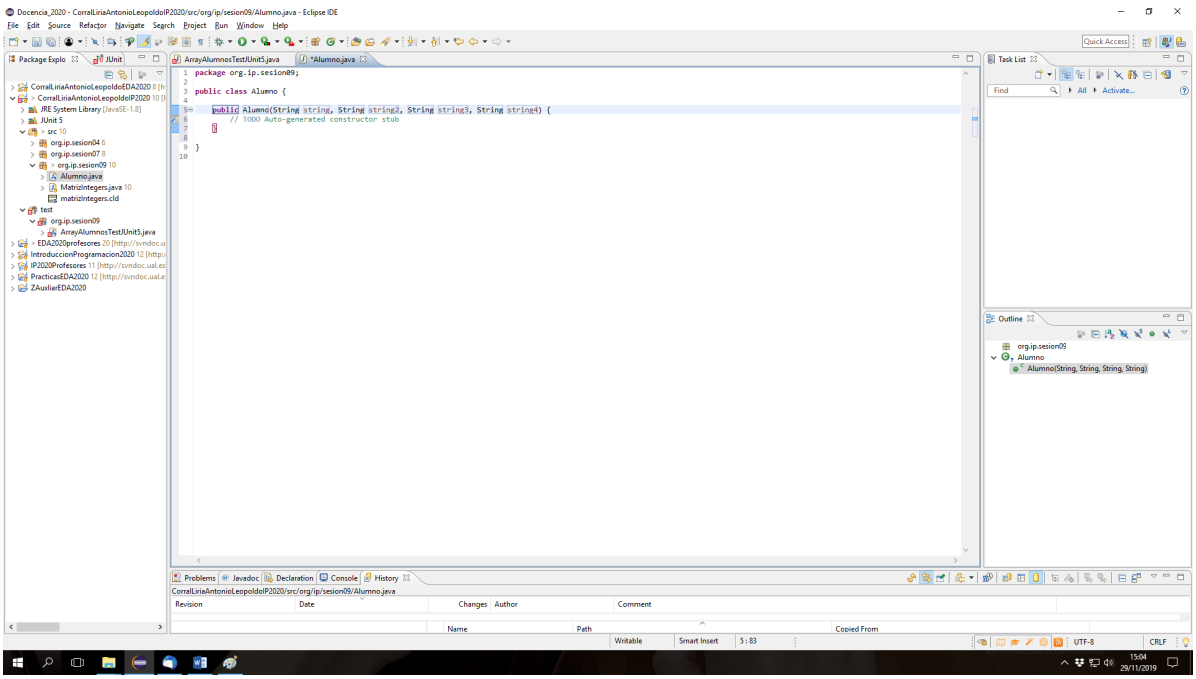
Es decir, nos aparecerá la clase creada (`Alumno`) en el paquete **`src.org.ip.sesion09`** tal y como se muestra en la siguiente pantalla.



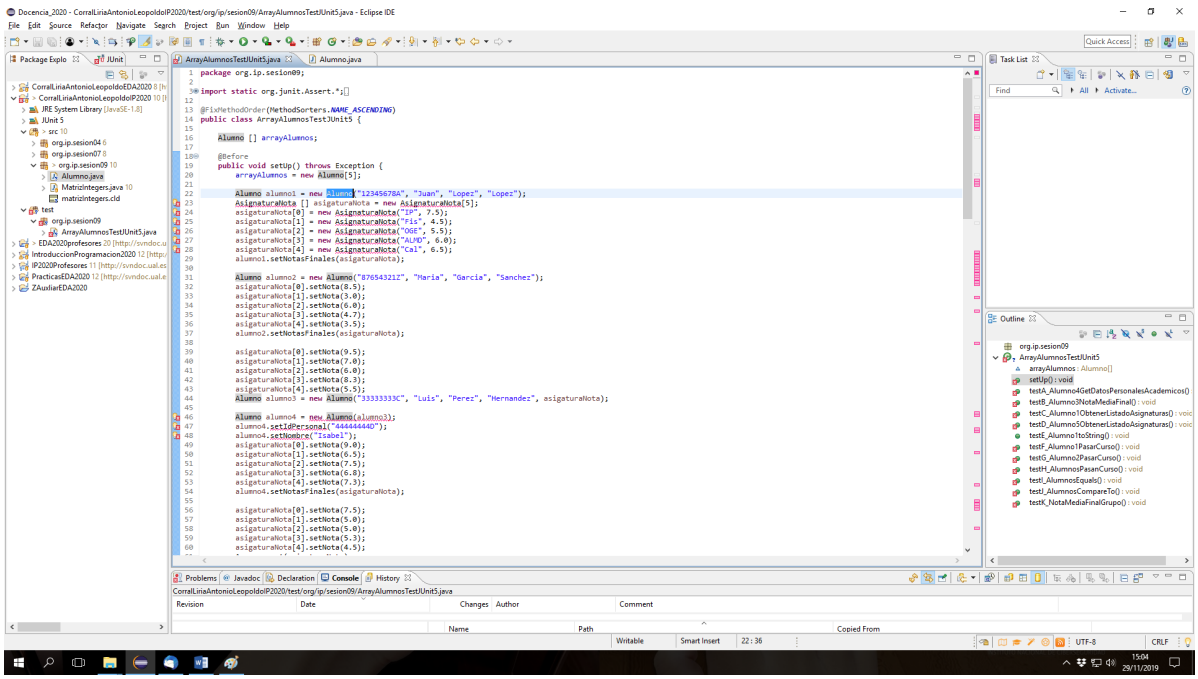
Ahora podremos comprobar que al crearse la clase ya no aparece marcada (subrayada) en **rojo**, ahora lo que aparece (subrayado) en **rojo** son los *métodos* que deben estar implementados en dicha clase (e implementados correctamente para que se pase el test). Si ahora nos volvemos a colocar sobre el ***método***, Eclipse nos volverá a facilitar la tarea de creación (en este caso de un constructor de la clase `Alumno`) 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 *constructor* `Alumno` nos aparece en su clase `Alumno`. Y ahora sólo nos queda implementar este y otros métodos 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 constructor `Alumno` para se verifique el `assert` que hay asociado

Podemos observar que dentro de `testAlumno1ObtenerListadoAsignaturas()` 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 `obtenerListadoNotasAsignaturas()`, coincide con el resultado *real*, y trasmite el resultado a JUnit. De esta forma, al terminar la ejecución de todos los test, 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 test 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 veremos ejemplos sencillos de test unitarios (como si de un `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://www.jtech.ua.es/j2ee/publico/lja-2012-13/sesion04-apuntes.html

http://junit.sourceforge.net/javadoc/org/junit/package-summary.html