



Eusko Jaurlaritzaren Informatika Elkarte
Sociedad Informática del Gobierno Vasco

JUnit:

Manual de usuario

Fecha:

Referencia:

EJIE S.A.
Mediterráneo, 3
Tel. 945 01 73 00*
Fax. 945 01 73 01
01010 Vitoria-Gasteiz
Posta-kutxatila / Apartado: 809
01080 Vitoria-Gasteiz
www.ejie.es

Control de documentación

Título de documento: JUnit

Histórico de versiones

Código:

Versión: 1.0

Fecha:

Resumen de cambios:

Cambios producidos desde la última versión

Primera versión.

Control de difusión

Responsable: Ander Martínez

Aprobado por: Ander Martínez

Firma:

Fecha:

Distribución:

Referencias de archivo

Autor: Consultoría de áreas de conocimiento

Nombre archivo: JUnit Manual de Usuario v1.0.doc

Localización:

Contenido

	Capítulo/sección	Página
1	Introducción	4
2	Conceptos básicos	4
3	Estructura	5
4	Objeto TestCase	10
4.1	Método setUp ()	13
4.2	Método tearDown ()	14
4.3	Método Assert	15
5	Objeto TestSuite	17
6	Anexo 1 – Prueba de Test	20

1 Introducción

El presente manual es una guía de iniciación en el manejo de JUnit. El documento se estructura en varios apartados, en los que se han incluido los principales aspectos que debe saber un usuario novato de JUnit para empezar a trabajar con la herramienta.

2 Conceptos básicos

JUnit es un paquete Java utilizado para automatizar los procesos de prueba. Mediante la creación de Tests, JUnit realizará una prueba en el código que indique el usuario.

Siempre que vayamos a desarrollar algún tipo de software, habrá que tener en cuenta las pruebas a realizar, con esto nos cercioraremos de que nuestro programa o librería funcionen correctamente.

Normalmente las pruebas se realizan por parte del programador, esto incluye que el orden elegido podría no ser correcto y con ello alargar demasiado el trabajo. Aunque inicialmente el proceso sea mas rápido, con el avance de la aplicación podría complicarse el trabajo, ya que cada vez que se fuera a probar algo, habría que volver a escribir el código para realizar la prueba ya que no se tiene la certeza de cuales serán los módulos afectados con varios cambios, ni podremos adivinar exactamente la línea donde se ha generado el error.

Existen varias razones para utilizar JUnit a la hora de hacer pruebas de código:

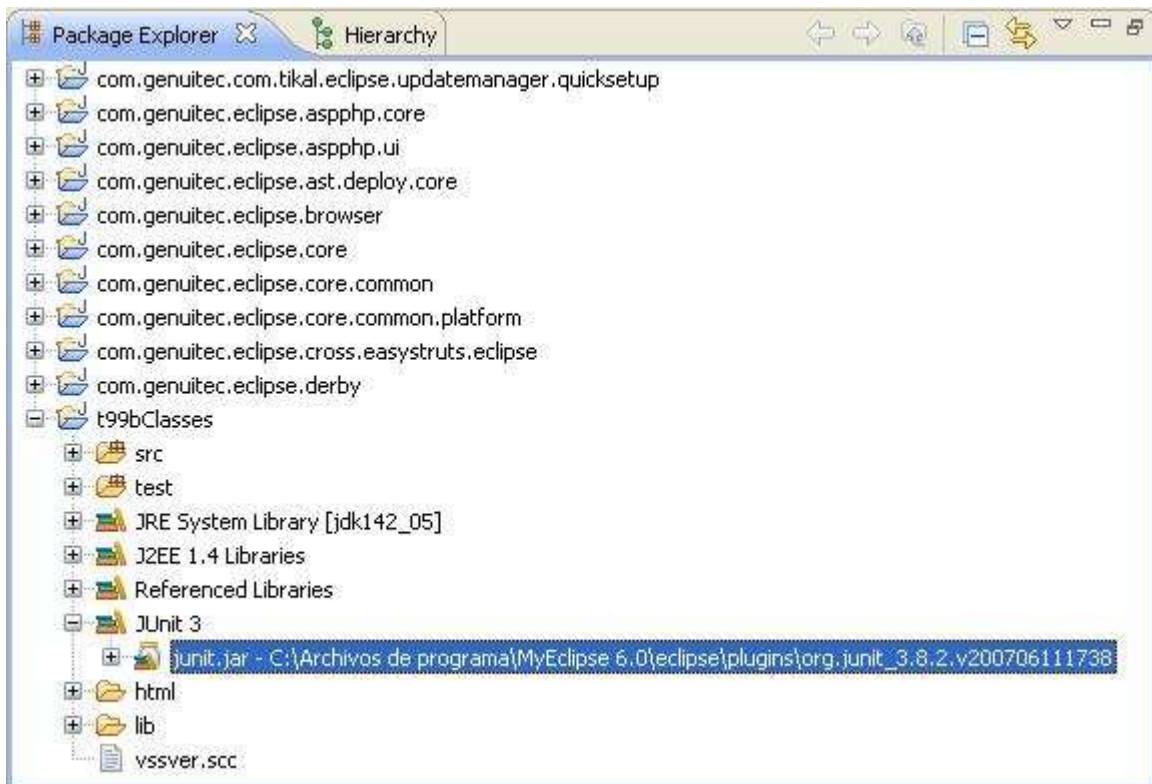
- La herramienta no tiene coste alguno, podremos descargarla directamente desde la Web oficial.
- Es una herramienta muy utilizada, por lo que no será complicado buscar documentación.
- Existen varios plugins para poder utilizar con diferentes Entornos de Desarrollo Integrado (**IDE**).
- Existen también muchas herramientas de pruebas de cobertura que utilizaran como base JUnit.
- Con JUnit, ejecutar tests es tan fácil como compilar tu código. El compilador "**testea**" la sintaxis del código y los tests "**validan**" la integridad del código.
- Los resultados son chequeados por la propia aplicación y dará los resultados inmediatamente.
- Los tests JUnit se pueden organizar en suites, las que contendrán ejemplares de tests, incluso podrán contener otras suites.
- Utilizando los tests programados en JUnit, la estabilidad de nuestras aplicaciones mejorará sustancialmente.
- Los tests realizados se podrán presentar junto con el código, para validar el trabajo realizado.

Para obtener información adicional sobre el producto acceder a su página Web:

<http://www.junit.org/>

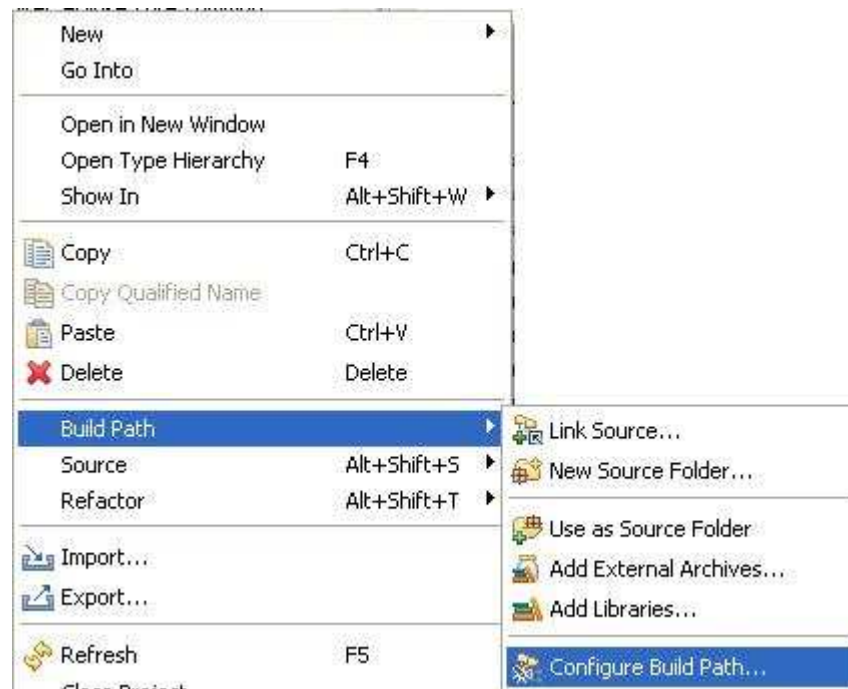
3 Estructura

En primer lugar debemos añadir el archivo **junit.jar** dentro del proyecto. En este caso utilizaremos la aplicación **MyEclipse**, por lo tanto el archivo tendrá que situarse en la carpeta correspondiente. Podremos ver en la imagen el ejemplo utilizado.

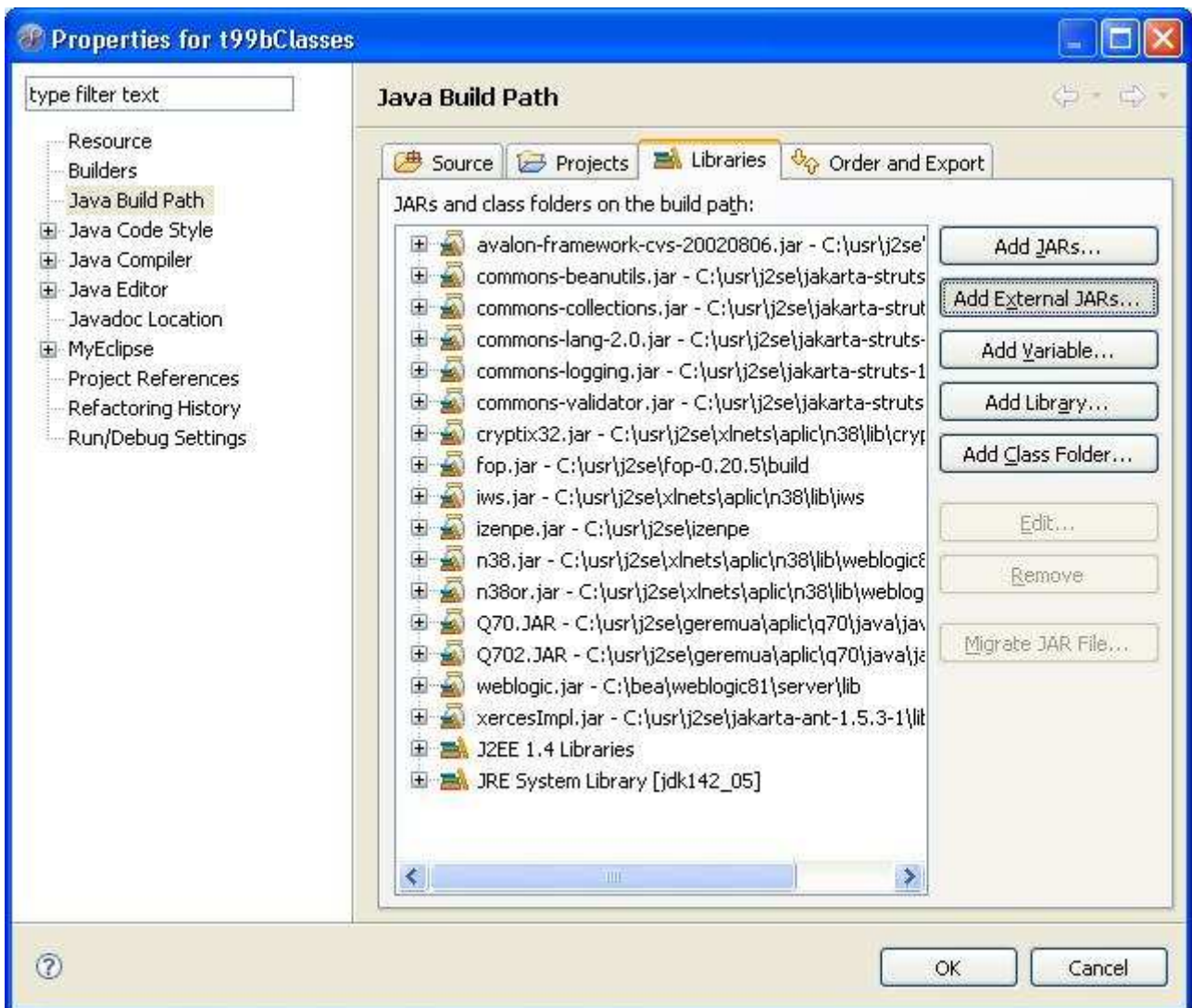


Una vez concluido este paso, crearemos un archivo que contendrá la clase de pruebas. Por defecto, los archivos creados comenzarán con el prefijo Test. Por ejemplo en una clase llamada **Fechas.java**, la prueba se llamará **TestFechas.java**.

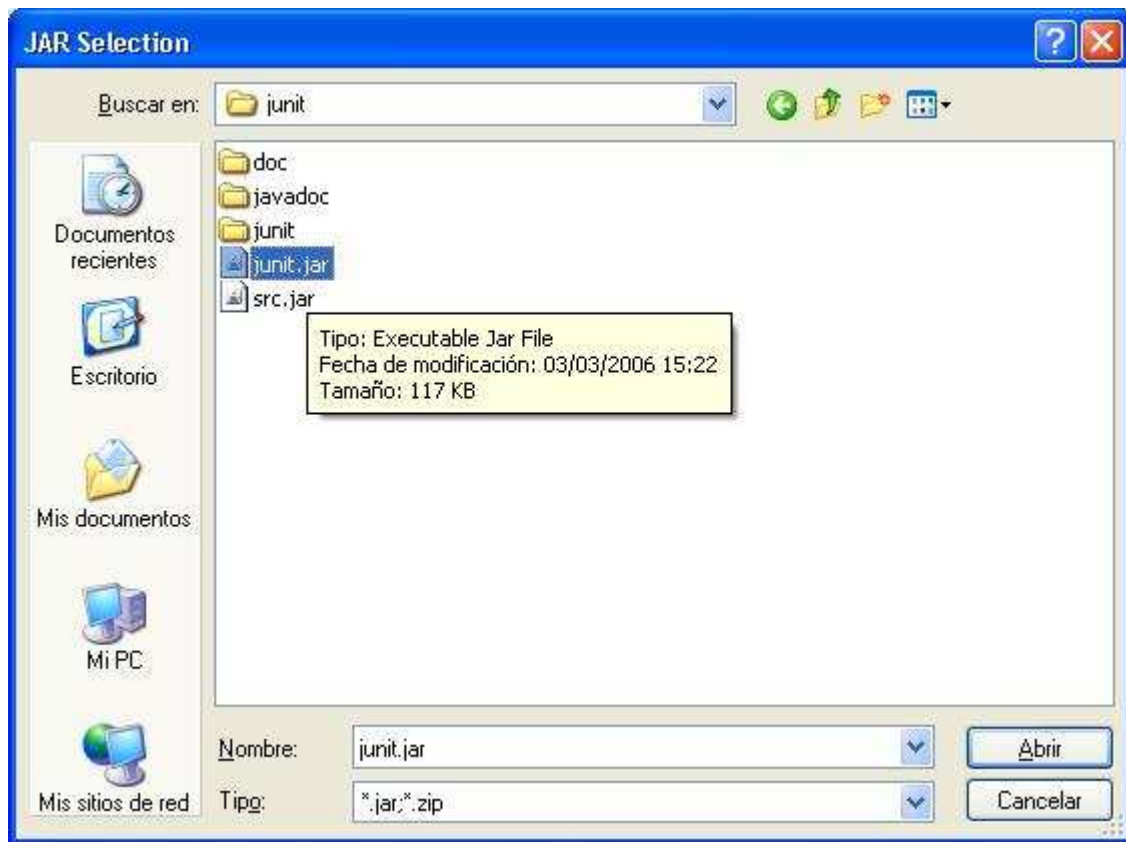
A continuación añadir al classpath el jar de la librería junit (en este caso junit.jar). Para ello haremos clic con el botón derecho encima del proyecto seleccionado, en este caso **t99bClasses**. Podemos apreciar en la imagen los pasos que seguiremos, nos situaremos sobre **Build Path** y haremos clic sobre **Configure Build Path**.



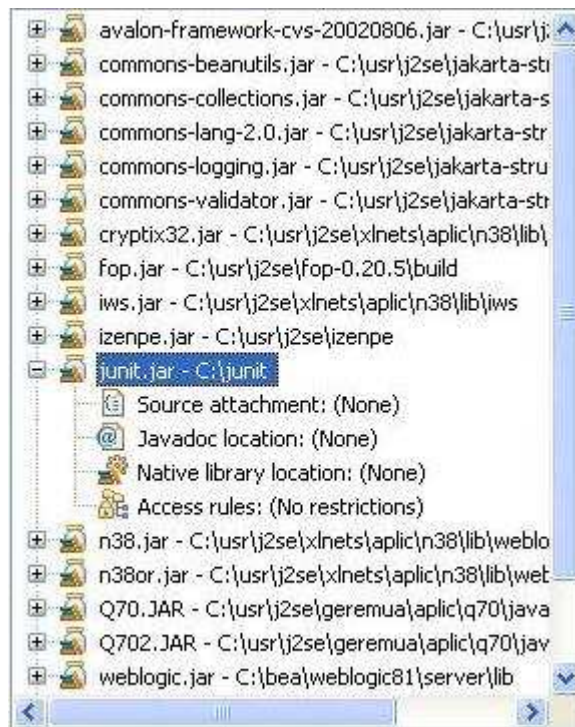
Nos aparecerá la siguiente pantalla donde nos mostraran las librerías que disponemos. Lo que haremos será añadir la librería junit.jar, y para ello pulsaremos sobre el botón llamado **Add External JARs**.



Y nos mostrara una pantalla donde buscaremos el archivo que vamos a añadir, en este caso, como hemos dicho anteriormente será el **JUnit.jar**. En la imagen mostrada a continuación podremos ver el ejemplo realizado.



Una vez buscado y seleccionado el archivo, el mismo, se añadirá a la lista. En la captura podremos observar la estructura de la que se compone.



Una vez realizado el paso anterior extenderemos la clase de prueba **TestCase**, para después crear un método público de tipo **void** para cada prueba que queramos realizar. Como acabamos de explicar, la prueba tendrá que llevar el prefijo **Test**.

Para saber si la prueba ha sido un éxito o ha fallado tendremos una sentencia que se incluye en los casos de prueba que es del tipo **assert** y tiene varios tipos diferentes, que veremos más adelante.

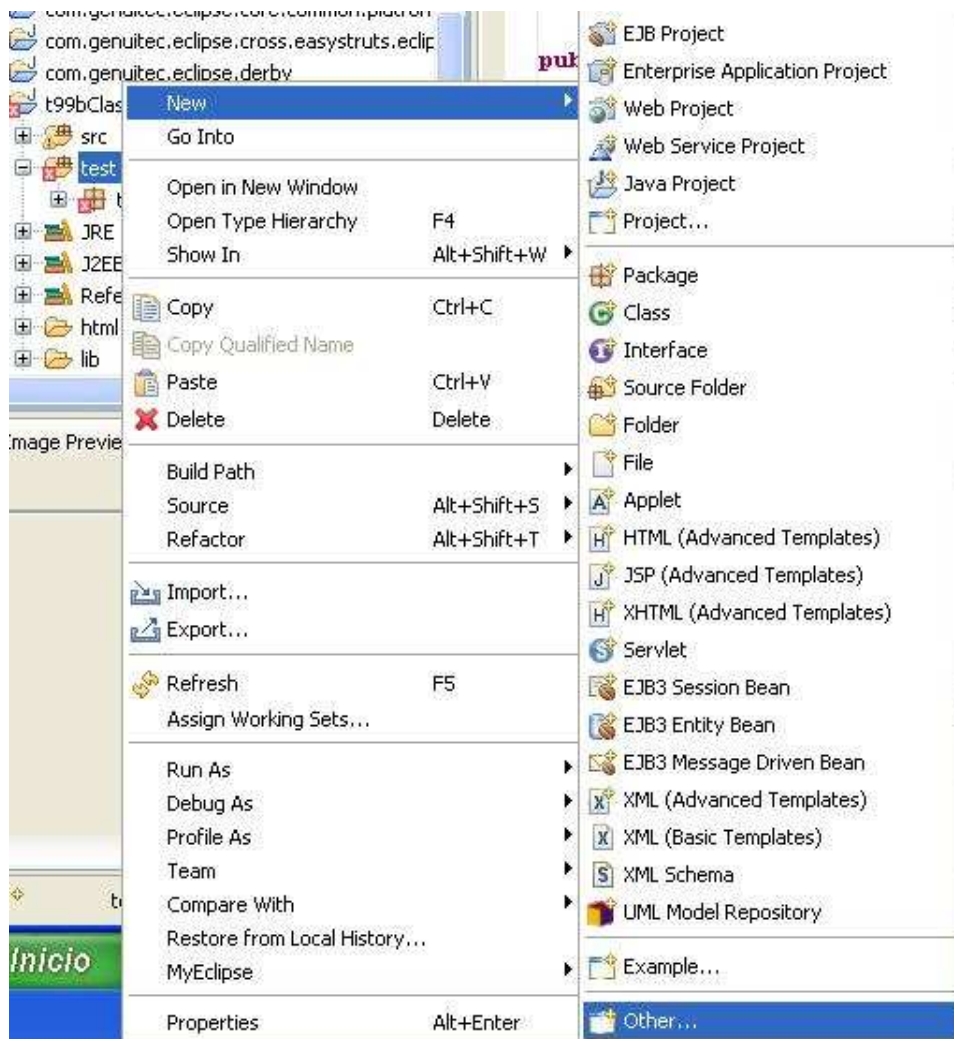
4 Objeto TestCase

Dentro de JUnit existen dos patrones de diseño principales: un de ellos es el patrón **Command** y el segundo es el patrón **Composite**.

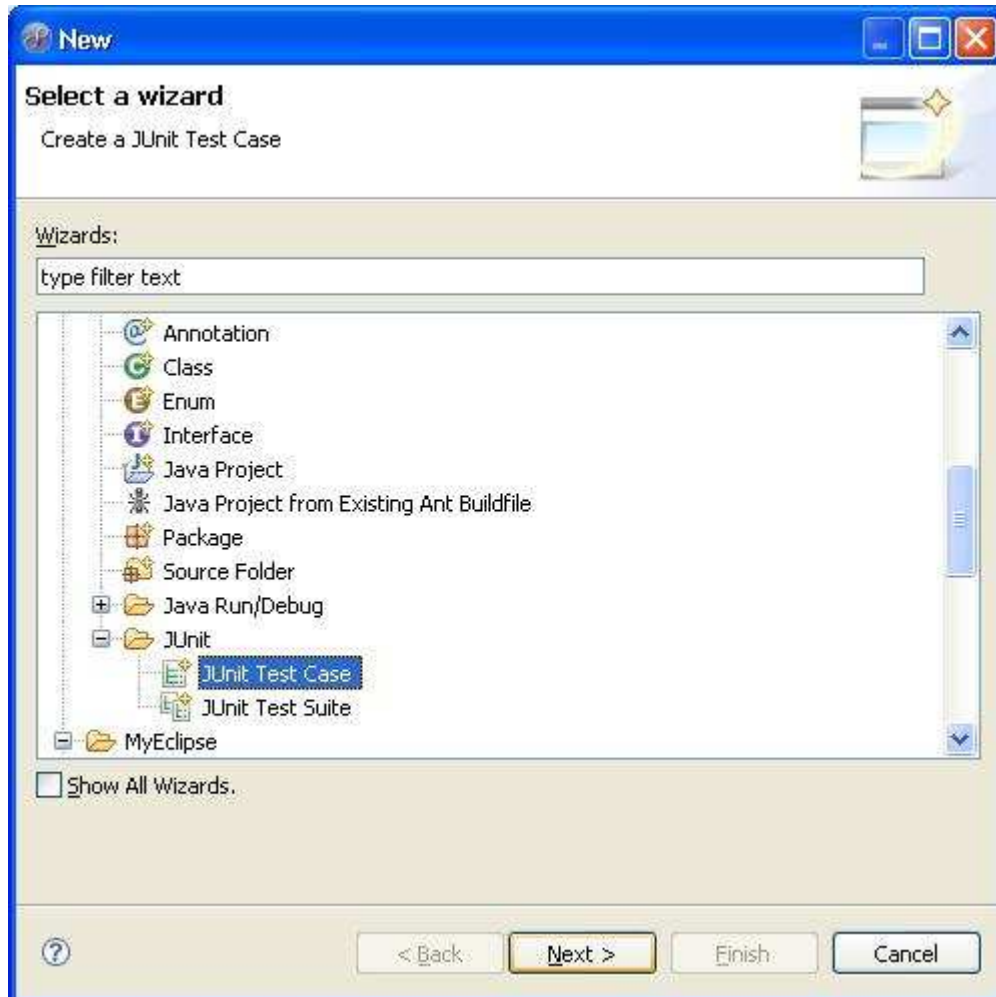
Los **TestCase** entran dentro del patrón **Command**. Cualquier clase que contenga métodos de testeo debería extender la clase **TestCase**. Con un **TestCase** podremos definir un número indefinido de métodos públicos **testXXX ()**. En el momento que queramos comprobar el resultado real y el esperado, invocaremos una variante del método **assert ()**.

Mostraremos con un ejemplo, como crear un método **TestCase**. En este caso la aplicación utilizada será **MyEclipse**, como hemos utilizado anteriormente.

Para ello haremos clic con el botón derecho sobre el proyecto deseado, seguido nos situaremos encima de **New** para después dar un clic sobre **Other**. Podemos observar en la imagen la estructura.



Una vez seleccionada la opción, la aplicación nos mostrara la siguiente ventana, donde elegiremos el método que utilizaremos, como podemos apreciar en la imagen elegimos la opción **JUnit TestCase** y le daremos a **Next**. En la siguiente imagen podemos ver el proceso realizado.



Aparecerá la siguiente pantalla, en la que añadiremos el paquete, el nombre de la clase y la clase creada bajo el test. Una vez agregadas estas opciones, le daremos a **Next**.

New JUnit Test Case

Type name is discouraged. By convention, Java type names usually start with an uppercase letter

New JUnit 3 test New JUnit 4 test

Source folder:

Package:

Name:

Superclass:

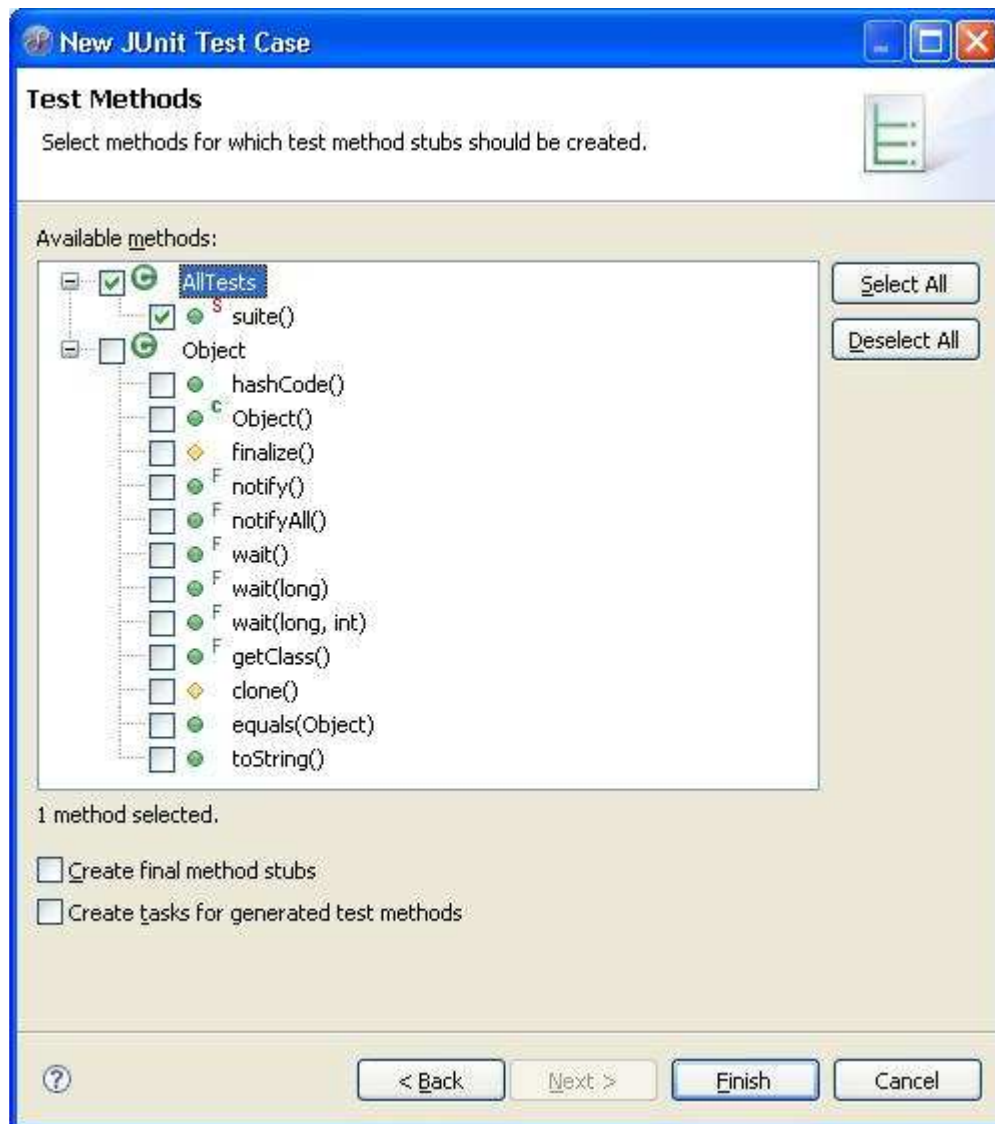
Which method stubs would you like to create?

setUpBeforeClass() tearDownAfterClass()
 setUp() tearDown()
 constructor

Do you want to add comments as configured in the [properties](#) of the current project?
 Generate comments

Class under test:

Una vez que pulsemos siguiente, aparecerá una pantalla donde marcaremos para qué métodos se generan los esqueletos de los casos de prueba. Aparecerá la siguiente imagen. En este ejemplo marcamos la primera opción y le damos a **Finish**. Con esto tendremos el **TestCase** creado.



4.1 Método setUp ()

Podremos encontrar casos en los que necesitemos ejecutar algún tipo de tarea específica, antes de lanzar la prueba.

Por ejemplo vendría muy bien, para abrir una base de datos antes de comenzar las pruebas.

Pasaremos a hacer un ejemplo básico viendo el código, para una mejor comprensión:

```
public class TestClass extends TestCase
{
    public void testPrueba1 ()
    {
        System.out.println ("Prueba1");
    }
    public void testPrueba2 ()
    {
        System.out.println ("Prueba2");
    }
    public void setUp ()
    {
        // Aquí se ejecutara setUp () antes de que se empiece a ejecutar un método.
    }
}
```

4.2 Método tearDown ()

Este método tiene una función similar a **setUp ()**, con la diferencia de que éste se ejecutará al final del código, para finalizar el tipo de tarea especificada.

Esta opción la podríamos utilizar por ejemplo para desconectar una base de datos abierta anteriormente con el método **setUp ()**

```
public class TestClass extends TestCase
{
    public void testPrueba1 ()
    {
        System.out.println ("Prueba1");
    }
    public void testPrueba2 ()
    {
        System.out.println ("Prueba2");
    }
    public void setUp ()
    {
        // Aquí se ejecutara setUp () antes de que se empiece a ejecutar un método.
    }
    public void tearDown ()
    {
        //Aquí se ejecutara tearDown () después de terminar de ejecutar el metodo.
    }
}
```

4.3 Método Assert

En este caso veremos el método **assert**, para saber cómo debemos utilizarlo. Lo normal es que los métodos de una clase de prueba creen una instancia de la clase principal. Después invocará al método que vayamos a probar y seleccionaremos el método **assert** que más nos convenga.

La definición del método **assert**, viene a ser que realiza una comparación de cualquier tipo de valor y en caso de que la prueba no sea exitosa, detendrá el proceso.

El método **assert** se ocupará de comparar el resultado de ejecutar el método con el valor que esperemos de retorno.

A continuación, añadiremos una lista con todas las funciones del método **assert** en JUnit.

Static void	assertEquals (boolean expected, boolean actual) Verifica que los valores proporcionados sean iguales.
Static void	assertEquals (byte expected, byte actual) Verifica que los valores proporcionados sean iguales.
Static void	assertEquals (char expected, char actual) Verifica que los valores proporcionados sean iguales.
Static void	assertEquals (double expected, double actual, double delta) Verifica que los valores proporcionados sean iguales tomando en cuenta la tolerancia indicada por el parámetro delta.
Static void	assertEquals (float expected, float actual, float delta) Verifica que los valores proporcionados sean iguales tomando en cuenta la tolerancia indicada por el parámetro delta.
Static void	assertEquals (int expected, int actual) Verifica que los valores proporcionados sean iguales.
Static void	assertEquals (long expected, long actual) Verifica que los valores proporcionados sean iguales.
Static void	assertEquals (java.lang.Object expected, java.lang.Object actual) Verifica que los valores proporcionados sean iguales.
Static void	assertEquals (short expected, short actual) Verifica que los valores proporcionados sean iguales.
Static void	assertEquals (java.lang.String message, boolean expected, boolean actual) Verifica que los valores proporcionados sean iguales. Si no lo son, envía el mensaje indicado por el parámetro message.
Static void	assertEquals (java.lang.String message, byte expected, byte actual) Verifica que los valores proporcionados sean iguales. Si no lo son, envía el mensaje indicado por el parámetro message.
Static void	assertEquals (java.lang.String message, char expected, char actual) Verifica que los valores proporcionados sean iguales. Si no lo son, envía el mensaje indicado por el parámetro message.
Static void	assertEquals (java.lang.string message, double expected, double actual, double delta) Verificar que los valores proporcionados sean iguales dentro de la tolerancia especificada por el parámetro delta. Si no lo son, envía el mensaje indicado por el parámetro message.
Static void	assertEquals (java.lang.string message, float expected, float actual, float delta) Verificar que los valores proporcionados sean iguales dentro de la tolerancia especificada por el parámetro delta. Si no lo son, envía el mensaje

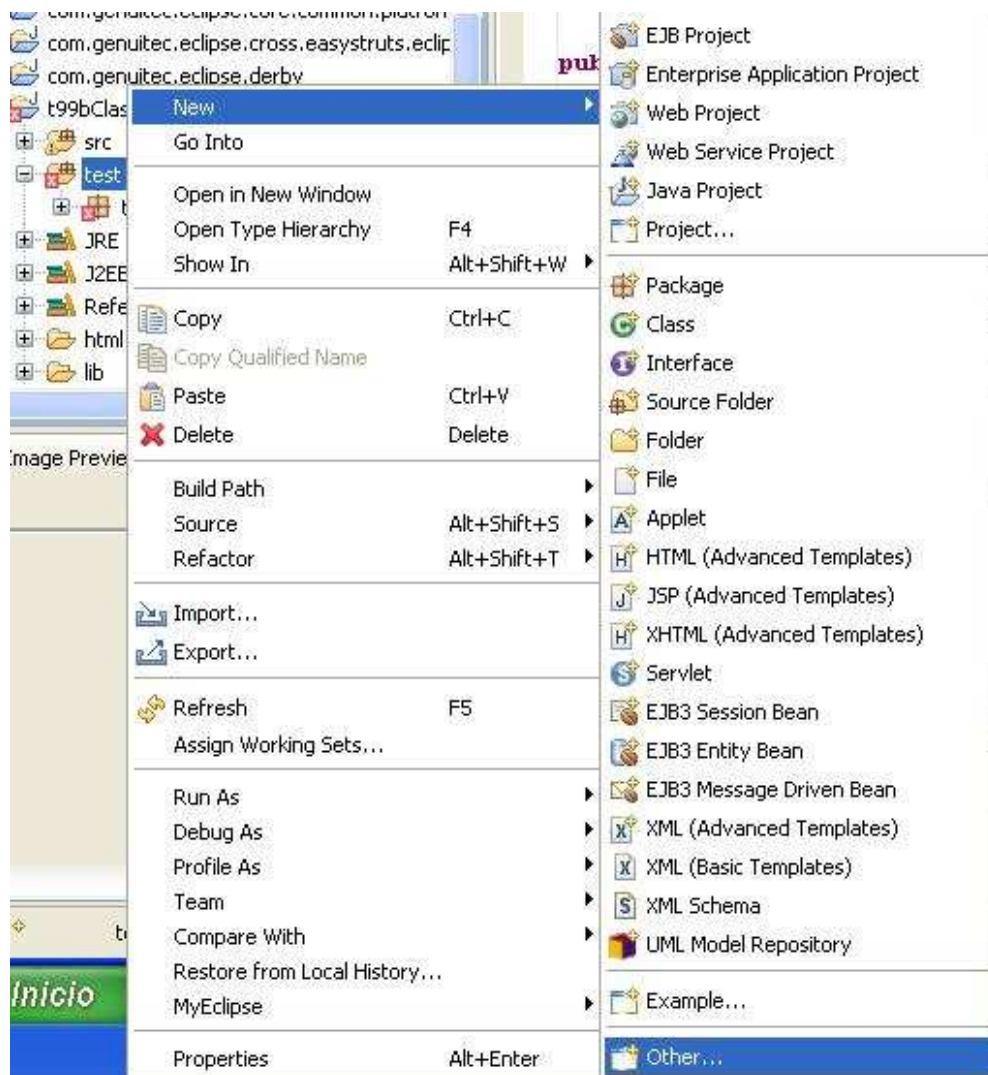
	indicado por el parámetro message.
Static void	assertEquals (java.lang.String message, int expected, int actual) Verifica que los valores proporcionados sean iguales. Si no lo son, envía el mensaje indicado por el parámetro message.
Static void	assertEquals (java.lang.String message, long expected, long actual) Verifica que los valores proporcionados sean iguales. Si no lo son, envía el mensaje indicado por el parámetro message.
Static void	assertEquals (java.lang.String message, java.lang.Object expected, java.lang.Object actual) Verifica que los valores proporcionados sean iguales. Si no lo son, envía el mensaje indicado por el parámetro message.
Static void	assertEquals (java.lang.String message, short expected, short actual) Verifica que los valores proporcionados sean iguales. Si no lo son, envía el mensaje indicado por el parámetro message.
Static void	assertEquals (java.lang.String expected, java.lang.String actual) Verifica que los valores proporcionados sean iguales.
Static void	assertEquals (java.lang.String message, java.lang.String expected, java.lang.String actual) Verifica que los valores proporcionados sean iguales. Si no lo son, envía el mensaje indicado por el parámetro message.
Static void	assertFalse (boolean condition) Verifica que el objeto boolean seleccionado sea falso.
Static void	assertFalse (java.lang.String message, boolean condition) Verifica que el objeto boolean seleccionado sea falso.
Static void	assertNotNull (java.lang.Object object) Verifica que el objeto proporcionado no sea null.
Static void	assertNotNull (java.lang.String message, java.lang.Object object) Verifica que el objeto proporcionado no sea null y envía un mensaje en caso de que sea cierto.
Static void	assertNotSame (java.lang.Object expected, java.lang.Object actual) Verifica que los objetos proporcionados no son los mismos.
Static void	assertNotSame (java.lang.String message, java.lang.Object expected, java.lang.Object actual) Verifica que los objetos proporcionados no son los mismos.
Static void	assertNull (java.lang.Object object) Verifica que el objeto proporcionado es null.
Static void	assertNull (java.lang.String message, java.lang.Object object) Verifica que el objeto proporcionado es null enviando un mensaje en caso de que no se cumpla la condición.
Static void	assertSame (java.lang.Object expected, java.lang.Object actual) Verifica que los objetos proporcionados son los mismos.
Static void	assertSame (java.lang.String message, java.lang.Object expected, java.lang.Object actual) Verifica que los objetos proporcionados son los mismos enviando un mensaje en caso de que no se cumpla la condición.
Static void	assertTrue (boolean condition) Verifica que el parámetro proporcionado true.
Static void	assertTrue (java.lang.String message, boolean condition) Verifica que el parámetro proporcionado sea true enviando un mensaje de error en caso de que no sea así.

5 Objeto TestSuite

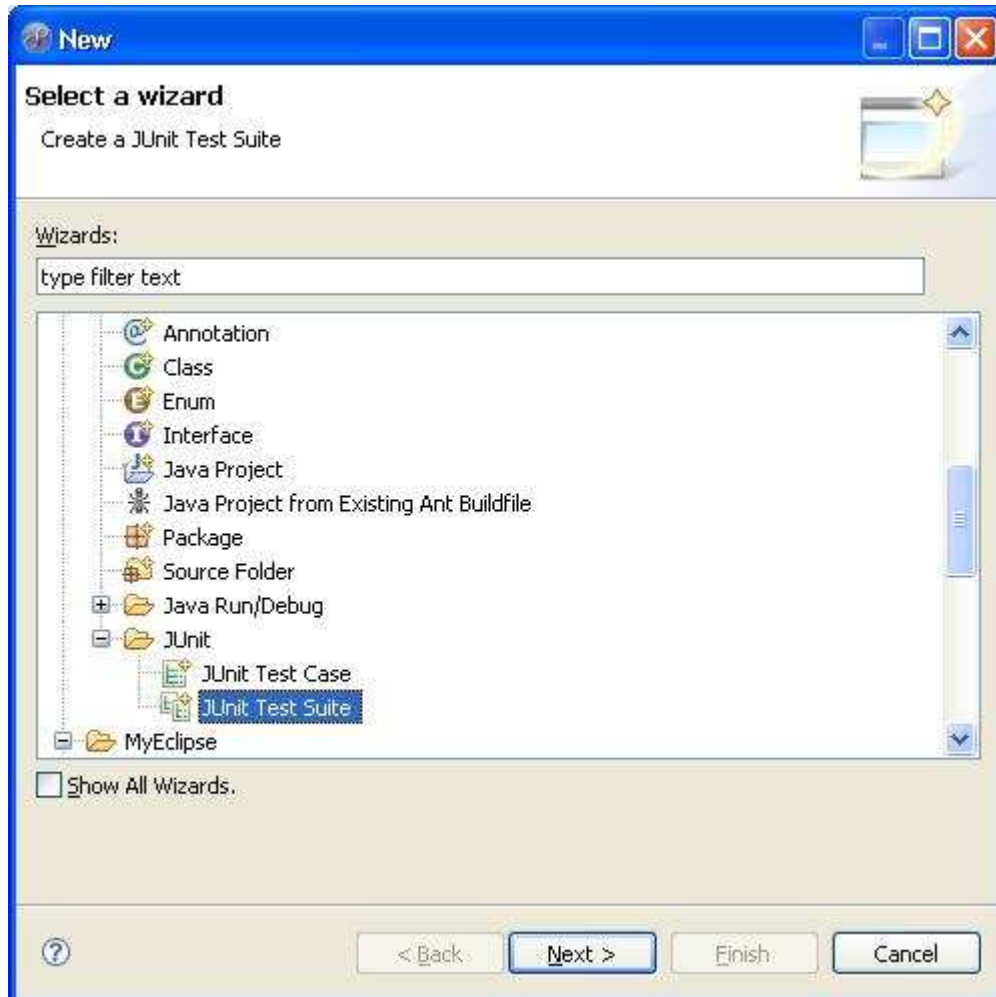
Podremos crear varios **TestCase** uniéndolos todos en árboles de **TestSuite**, estos últimos invocarán automáticamente todos los métodos **testXXX ()** definidos en cada **TestCase**.

Con un **TestSuite** tendremos la posibilidad de juntar varios tests, incluso los TestSuite que queramos, para añadirlos todos en uno solo. Después ejecutando el **TestSuite** apropiado, podremos ver los resultados de todos los diferentes tests unidos.

Procederemos ahora a crear un **TestSuite** con dos **TestCase** en su interior. Para ello haremos clic con el botón derecho sobre la carpeta donde estén incluidos los **TestCase**, nos situaremos sobre **New** y como hemos hecho antes elegiremos **Other**.

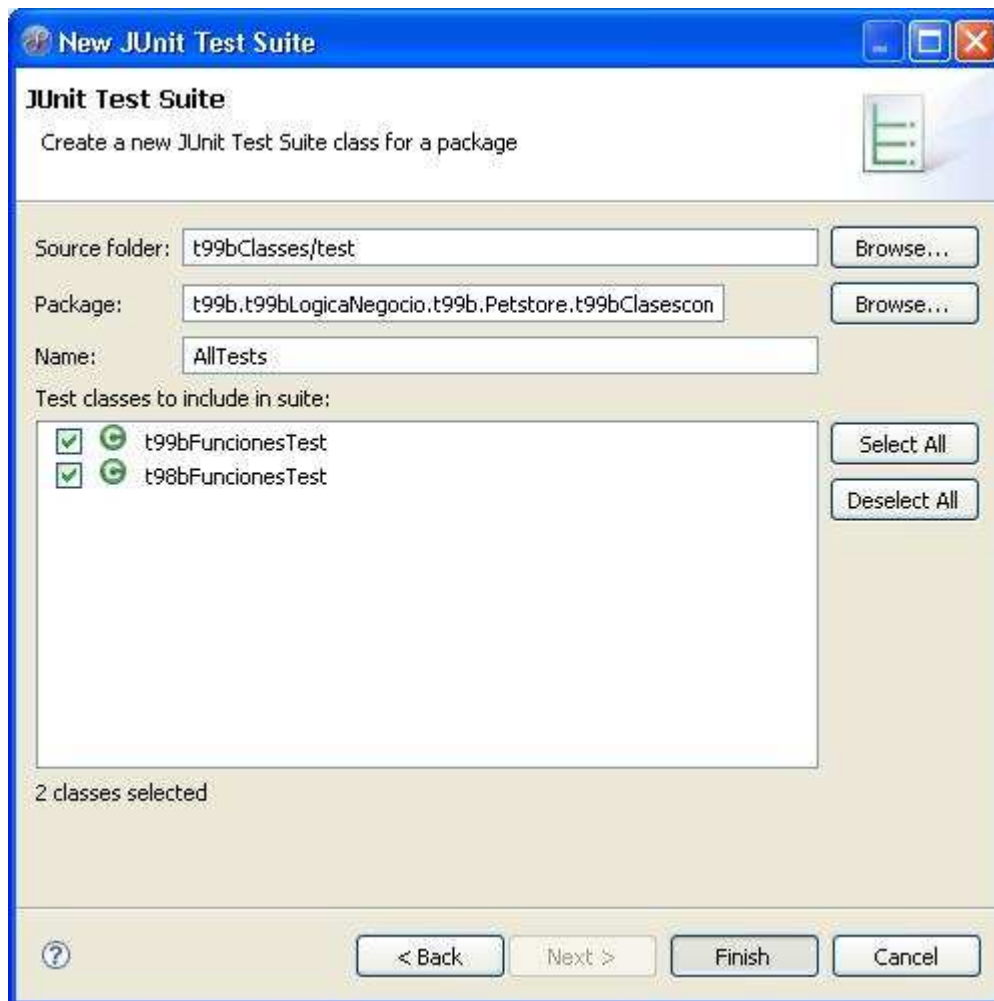


Una vez elegida la opción mencionada anteriormente, aparecerá la pantalla que ya hemos comentado antes, pero esta vez tendremos que pulsar sobre **JUnit TestSuite** y darle a **Next**.

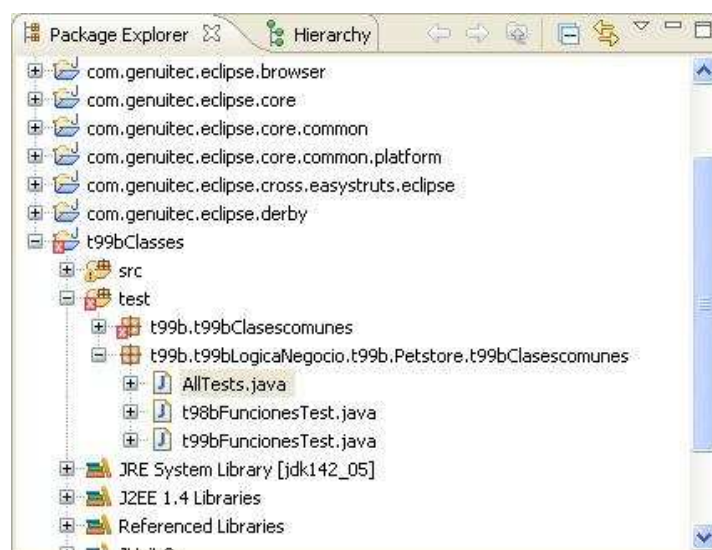


Pasaremos a la siguiente ventana, donde tendremos la opción de añadirle un nombre a nuestra suite. También elegiremos todos los **TestCase** y **TestSuite** que deseemos, para que luego se ejecuten todos seguidos.

Cuando elijamos los tests deseados daremos clic a Finish para que se cree el nuevo **TestSuite**.



Cuando acabemos podremos observar la estructura creada por la herramienta que estamos utilizando.



6 Utilidad Practica

JUnit, se trata de una herramienta que en la fase de pruebas proporcionara un modo de validar la correcta ejecución de las funcionalidades implementadas en un proyecto.

Así mismo, permitirá validar el correcto proceso de paso entre entornos, ya si los procesos funcionan correctamente en un entorno deberían hacerlo en el siguiente, así como proporcionara valiosa información en el proceso de realización de pruebas de regresión.

7 Anexo 1 – Prueba de Test

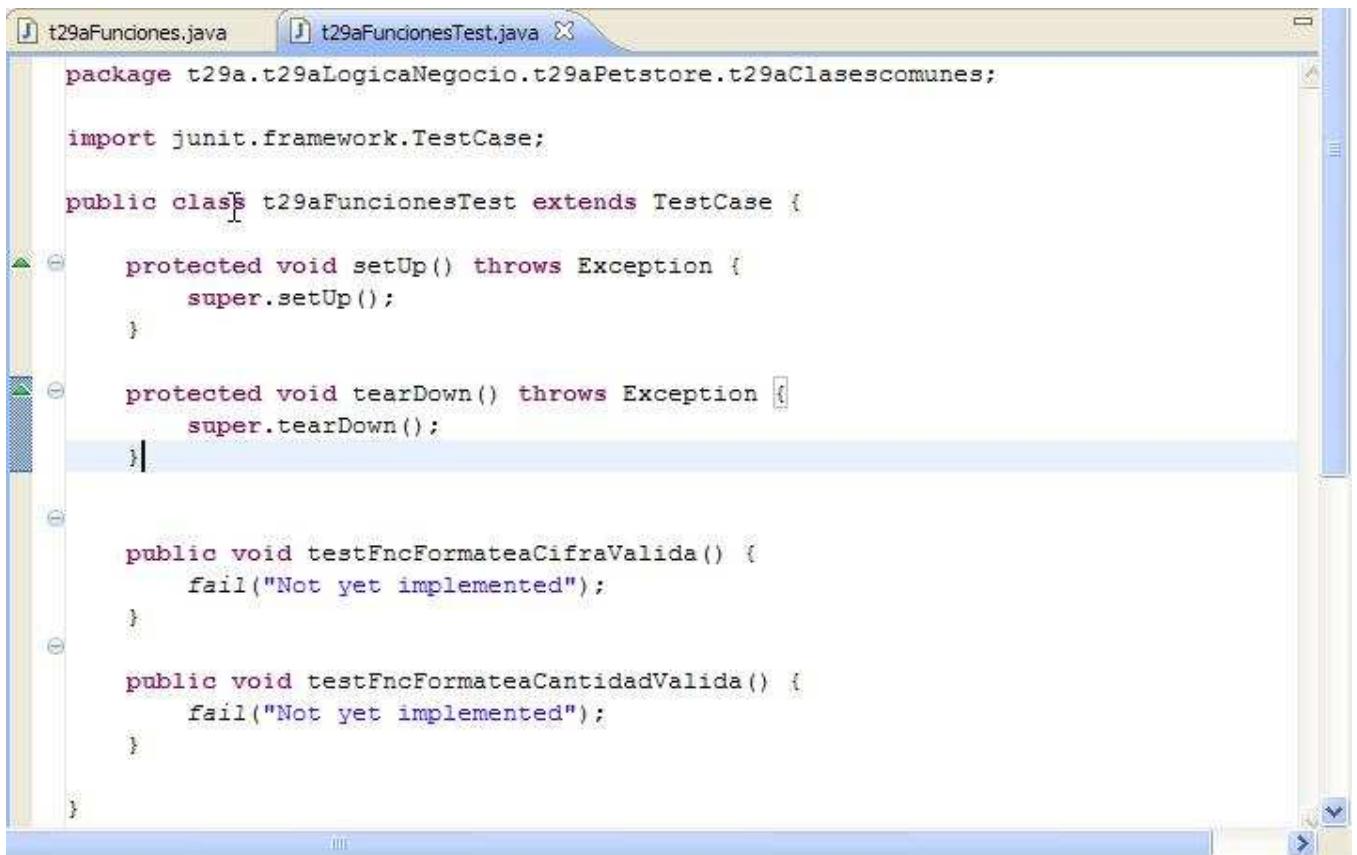
Vamos a proceder a realizar un test mediante JUnit para verificar su funcionamiento.

Realizaremos un **TestCase** mediante **MyEclipse** para después lanzarlo y ver los resultados. También crearemos un **TestSuite** donde incluiremos el **TestCase** realizado anteriormente para después observar los resultados.

7.1 Resolución

Comenzaremos realizando una prueba con **MyEclipse**, para cerciorarnos de que los tests funcionarán correctamente.

Abriremos la aplicación y ejecutaremos una clase, sobre la que haremos la prueba mas adelante.



```
package t29a.t29aLogicaNegocio.t29aPetstore.t29aClasescomunes;

import junit.framework.TestCase;

public class t29aFuncionesTest extends TestCase {

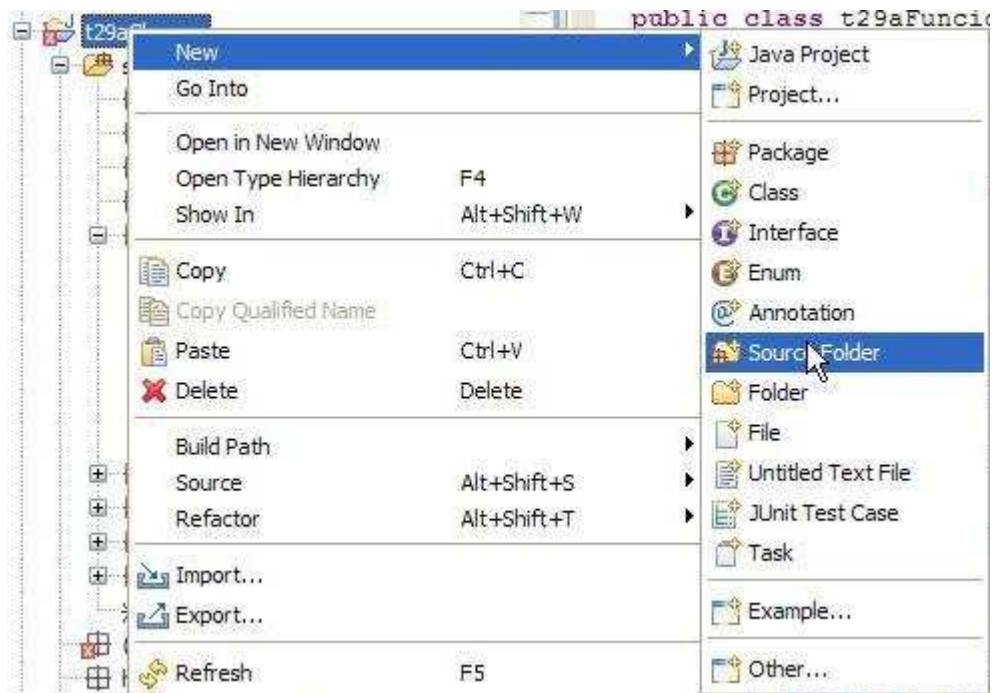
    protected void setUp() throws Exception {
        super.setUp();
    }

    protected void tearDown() throws Exception {
        super.tearDown();
    }

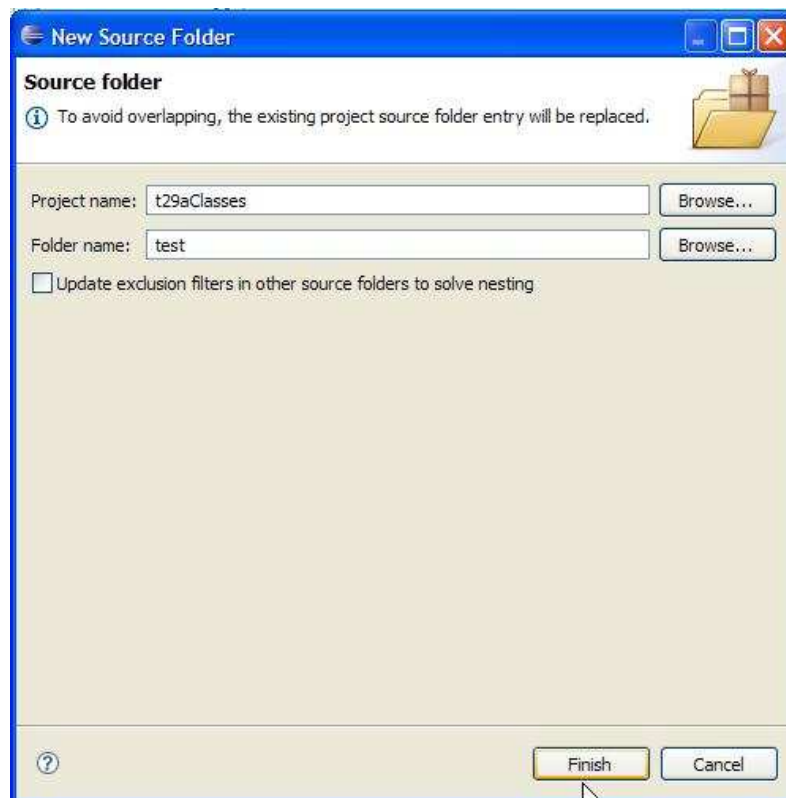
    public void testFncFormateaCifraValida() {
        fail("Not yet implemented");
    }

    public void testFncFormateaCantidadValida() {
        fail("Not yet implemented");
    }
}
```

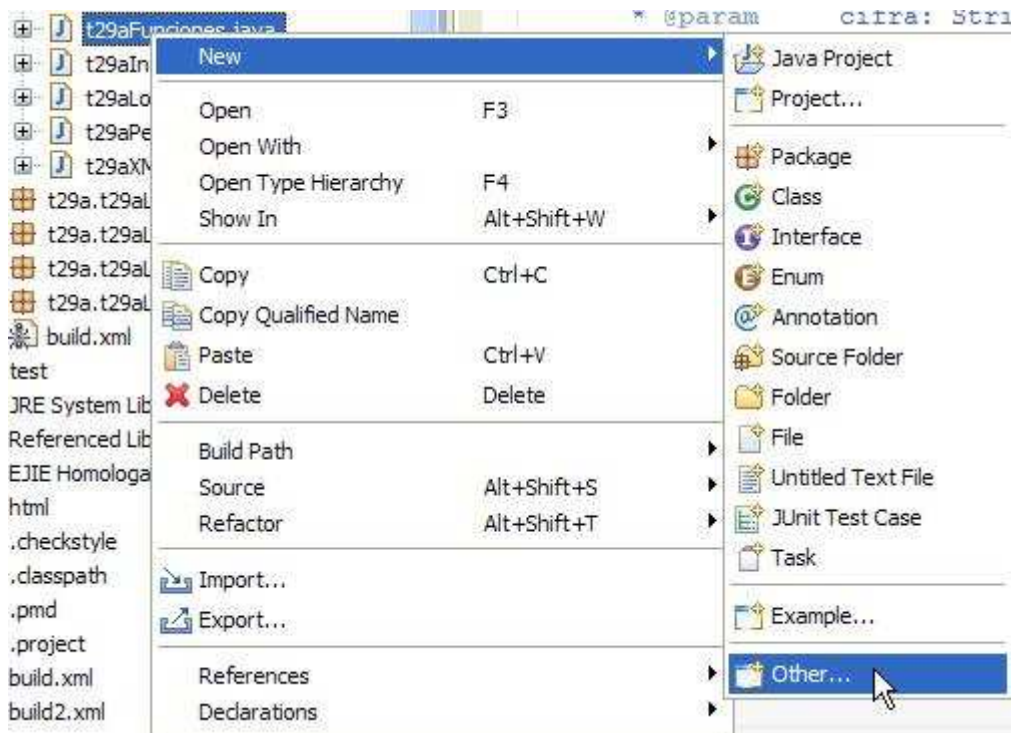
Aquí podemos observar una clase abierta, donde le añadiremos el test que ejecutaremos más adelante. El siguiente paso que realizaremos será, crear una carpeta para almacenar dentro los códigos fuente. Para ello, haremos clic con el botón derecho sobre el proyecto, en este caso llamado **"t29aClasses"**, nos situaremos sobre **New** y después daremos clic encima de **Source Folder** como podemos apreciar en la imagen.



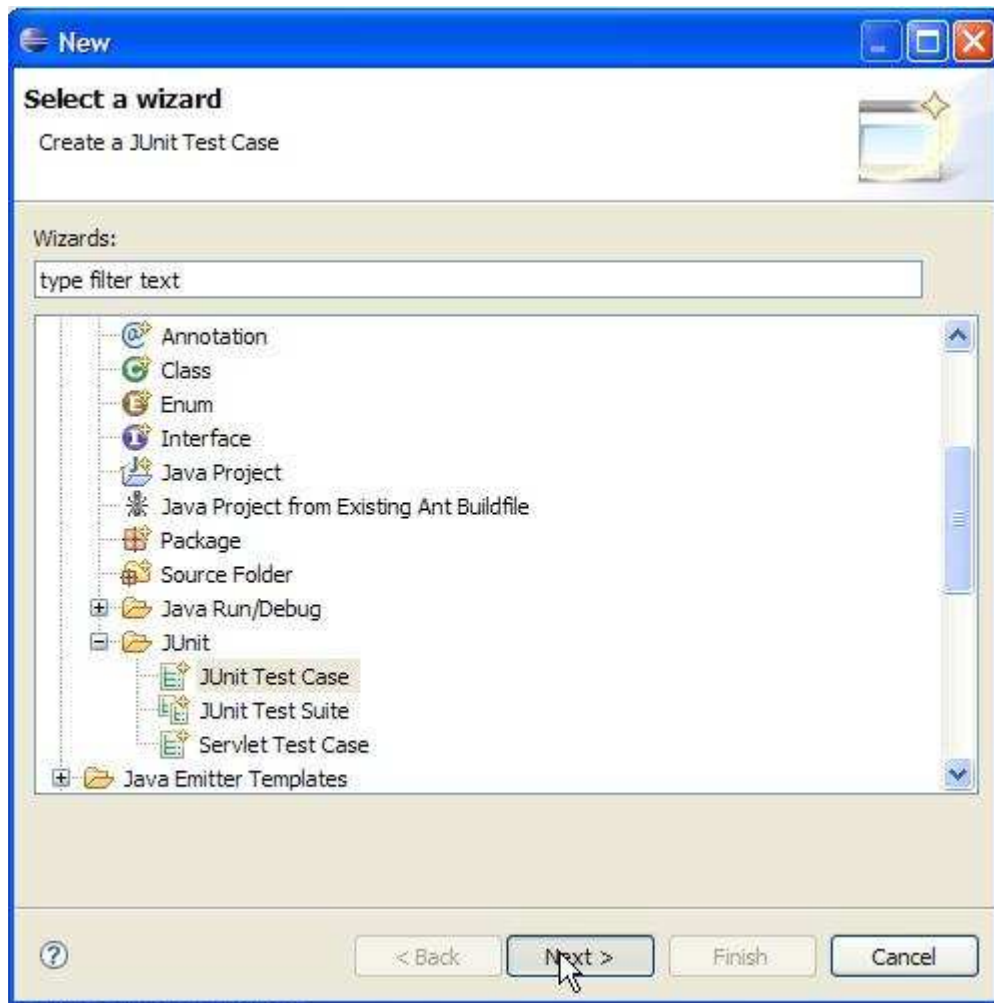
Una vez presionemos sobre el botón, nos saldrá la siguiente pantalla, donde indicaremos el nombre de la carpeta, seguido pulsaremos sobre **Finish** como aparece en la imagen. Con esto nuestra carpeta estará creada.



El siguiente paso será crear un **TestCase**, para ello presionaremos sobre la clase, en este caso la del ejemplo. En la sección **New** pulsaremos sobre **Other** como podemos apreciar en la imagen.



Una vez hecho esto, aparecerá la siguiente pantalla, donde extenderemos la carpeta **java** y después elegiremos la opción **JUnit** para desplegar el siguiente menú.



Aquí elegiremos la opción **JUnit Test Case** y seguido le daremos a **Next** para que aparezca la siguiente pantalla.

New JUnit Test Case

Type name is discouraged. By convention, Java type names usually start with an uppercase letter

New JUnit 3 test New JUnit 4 test

Source folder: t29aClasses/test

Package: t29a.t29aLogicaNegocio.t29aPetstore.t29aClasescomunes

Name: t29aFuncionesTest

Superclass: junit.framework.TestCase

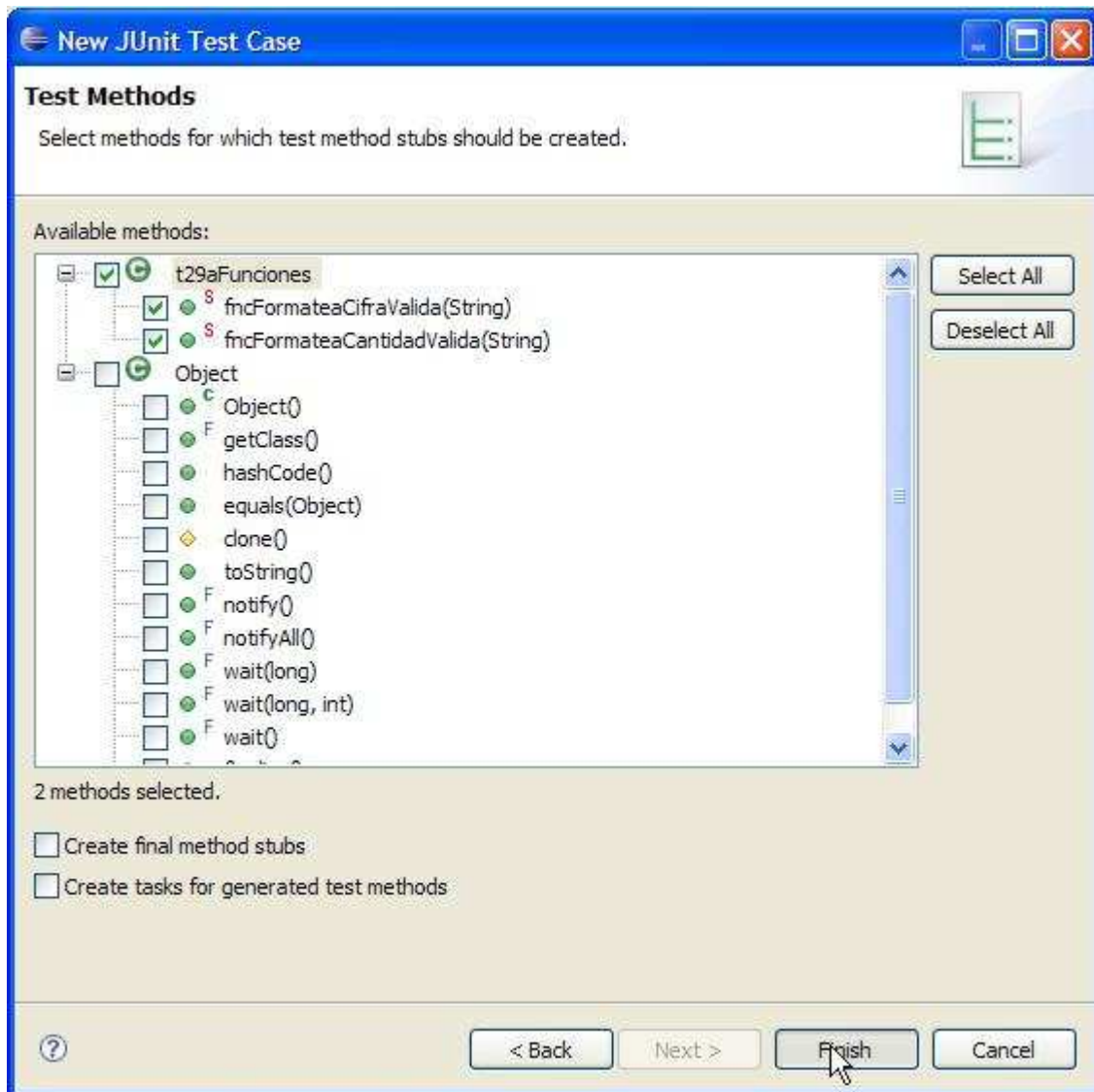
Which method stubs would you like to create?

setUpBeforeClass() tearDownAfterClass()
 setUp() tearDown()
 constructor

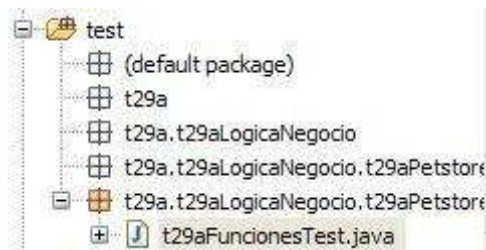
Do you want to add comments as configured in the [properties](#) of the current project?
 Generate comments

Class under test: t29a.t29aLogicaNegocio.t29aPetstore.t29aClasescomunes.t29aFunciones

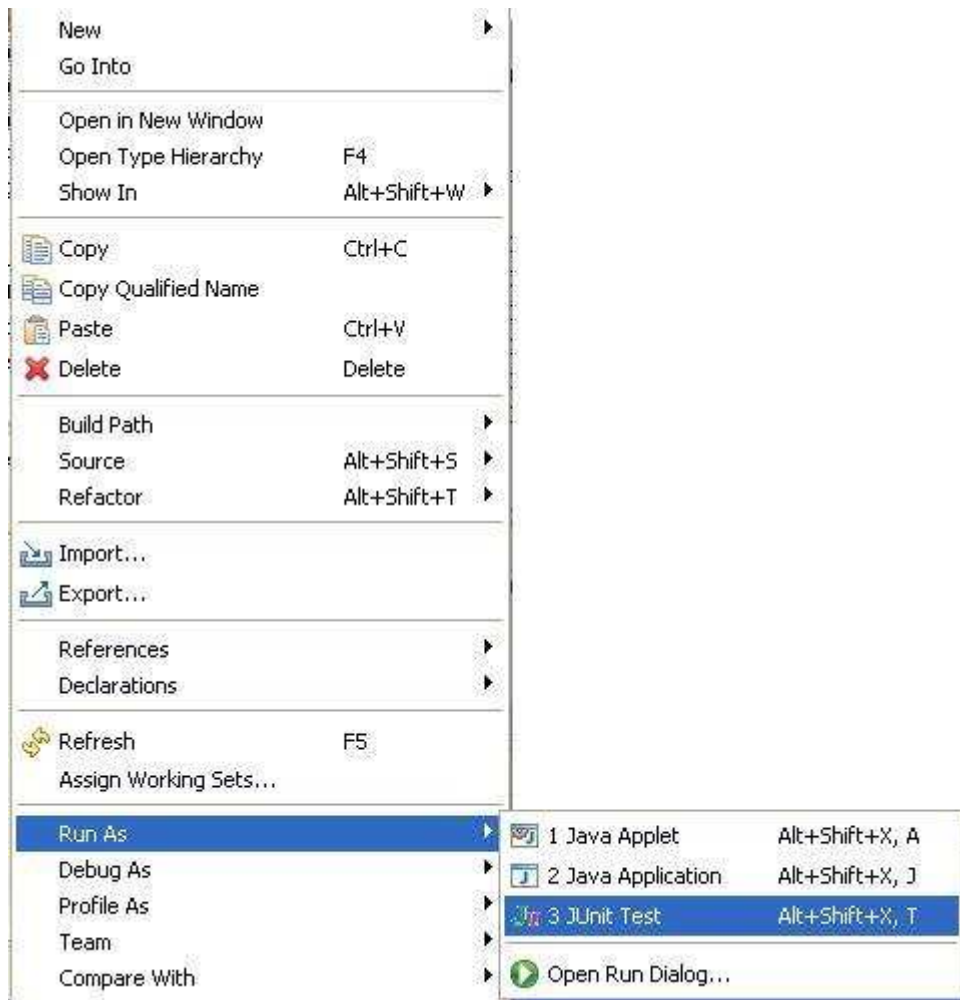
En esta ventana tendremos varias opciones para elegir, nosotros cambiaremos la **Source Folder** y elegiremos la carpeta creada anteriormente, llamada test. Y después hemos marcado las opciones **setUp ()** y **tearDown ()** por si necesitamos ejecutar esos métodos en algún momento.



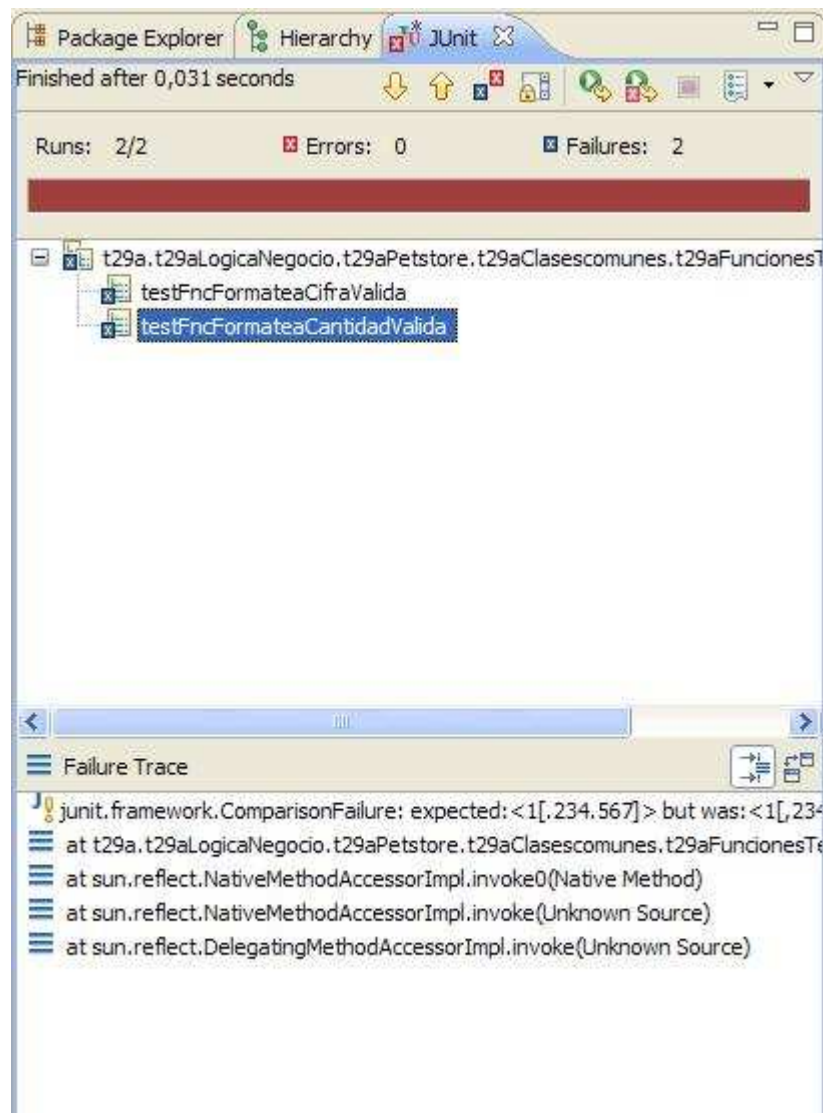
En la siguiente pantalla lo que haremos será, marcar para que métodos se generen los esqueletos de cada caso de pruebas, en este caso marcaremos la primera opción. Una vez creado el TestCase aparecerá de la siguiente manera en el menú izquierdo.



El próximo paso, será escribir el test dependiendo de lo que estemos buscando. Una vez este listo pasaremos a ejecutar el test de la manera que vemos en la imagen. Pulsaremos con el botón derecho sobre el test recién creado y buscaremos la opción **Run As** y seguido pulsaremos sobre la opción **JUnit Test**, para que de comienzo nuestro test.



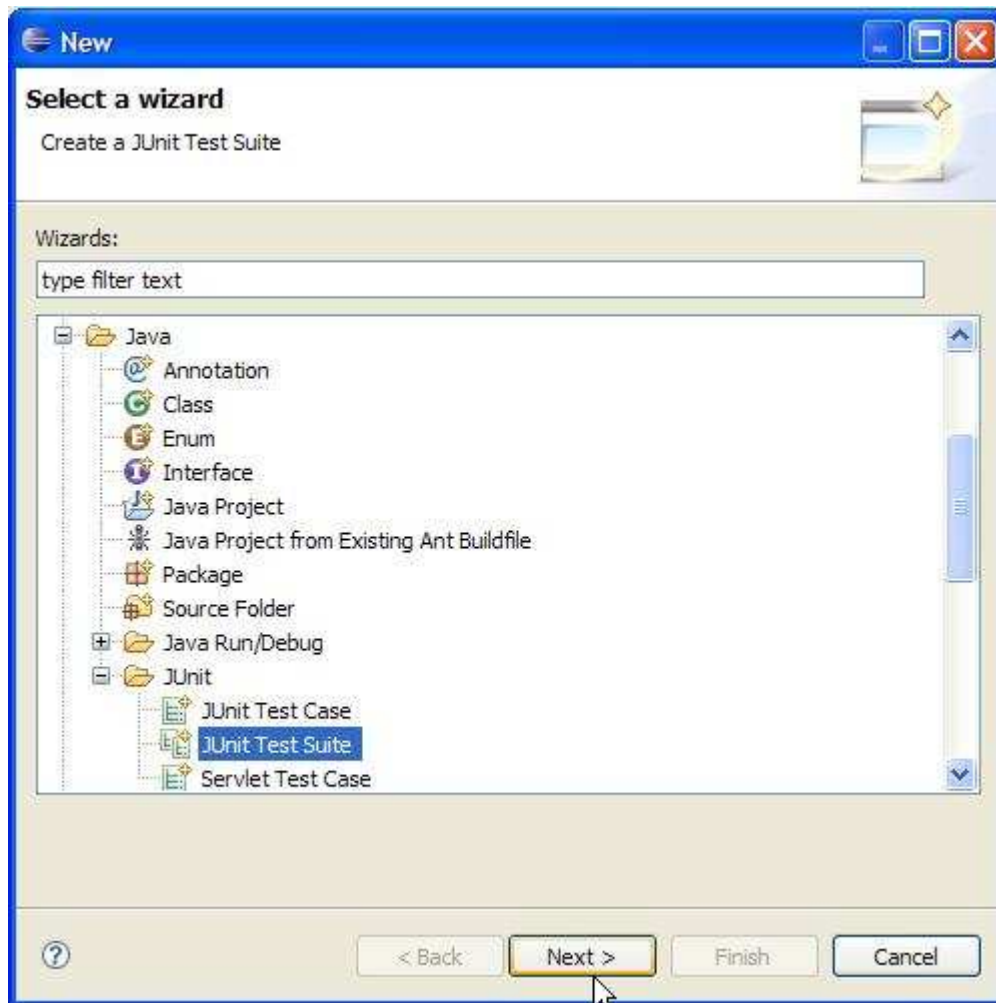
Cuando lo ejecutemos nos dará los resultados del test de la siguiente manera gráfica, pudiendo observar que este test ha dado dos fallos, como se observa en la imagen.



Podemos apreciar el test completo que acaba de realizar la herramienta con los dos fallos encontrados y explicando donde se encuentran estos.

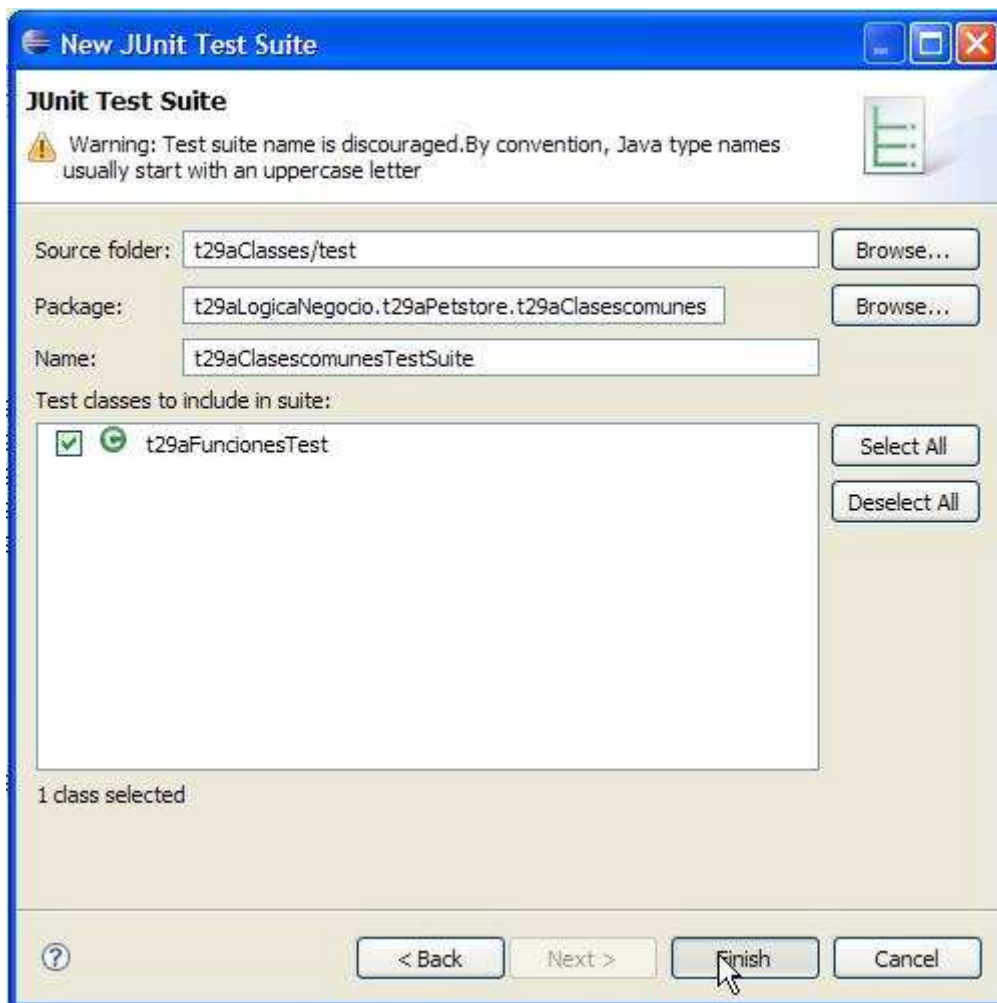
Ahora proseguiremos creando un **TestSuite**, donde podremos insertar todos los **TestCase**, incluso otros **TestSuite** que nos puedan interesar. Para ello volveremos a presionar con el botón derecho encima del test seguido nos situaremos encima de **New** y como hemos hecho al principio, pulsaremos sobre el botón **Other**.

Al aparecer la pantalla que hemos visto anteriormente, esta vez elegiremos la opción **JUnit Test Suite** como se aprecia en la imagen.



Cuando le demos a **Next**, aparecerá la siguiente pantalla. En este caso solo tenemos un **TestCase**, así que solamente podremos añadir ese, pero se podrían añadir todos los test deseados.

Como apreciamos en la siguiente captura, le insertaremos un nombre a este **TestSuite** y en la parte inferior, sería donde añadiríamos los test que nos gustaría unir.



Una vez terminada esta parte, estaremos listo para ejecutar este nuevo TestSuite, en este caso con un solo TestCase dentro.