Manual de usuario JUnit

Fecha: 17 Diciembre 2014

Referencia:

EJIE S.A.

Mediterráneo, 14

01010 Vitoria-Gasteiz

Posta-kutxatila / Apartado: 809

01080 Vitoria-Gasteiz

Tel. 945 01 73 00*

Fax. 945 01 73 01

www.ejie.es

Control de documentación

Título de documento: Proyecto			
	Histórico de versiones		
Código:	Versión:	Fecha:	Resumen de cambios:
	V1.0		
	V2.0		Introducido Junit4.0
	Cambios producidos desde la última versión		
Primera versión.			
	Control de difusión		
Responsable: Ander Martínez			
Aprobado por:			
Firma:		Fecha: 12/09/2014	
Distribución:			
	Referencias de archivo		
Autor: Grupo de Consultoría de Áreas de Conocimiento			
Nombre archivo: JUnit. Manual de Usuario v2.0.doc			
Localización:			

Contenido

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. Se referencian las dos versiones de JUnit que se pueden utilizar en EJIE (JUnit 3 para weblogic8 y JUnit 4 para weblogic11)

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 más 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 cuáles 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.

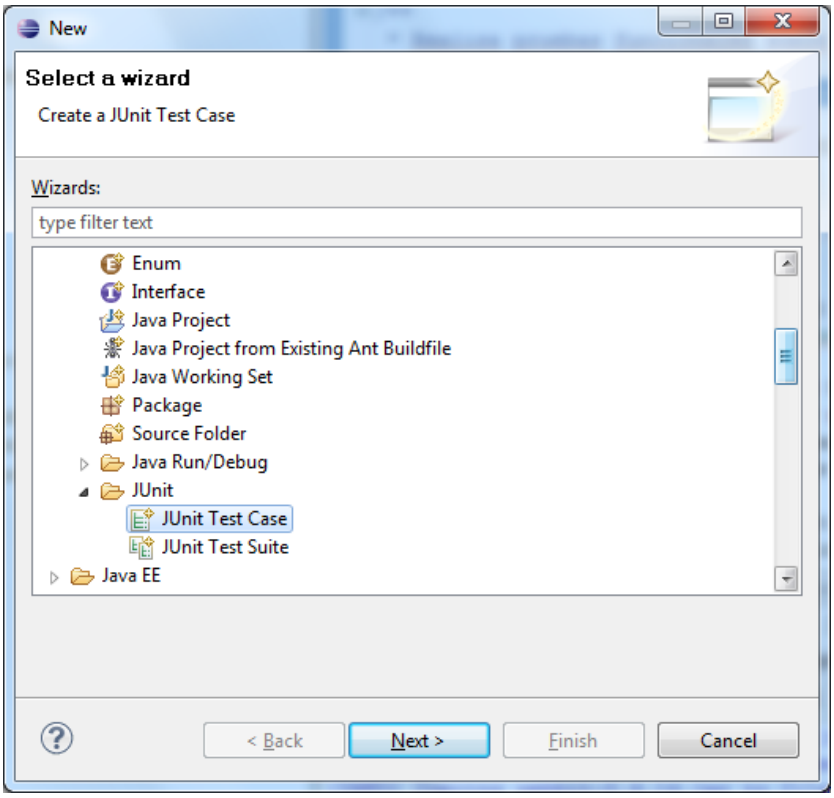
Para obtener información adicional sobre el producto acceder a su página Web:

http://www.junit.org/

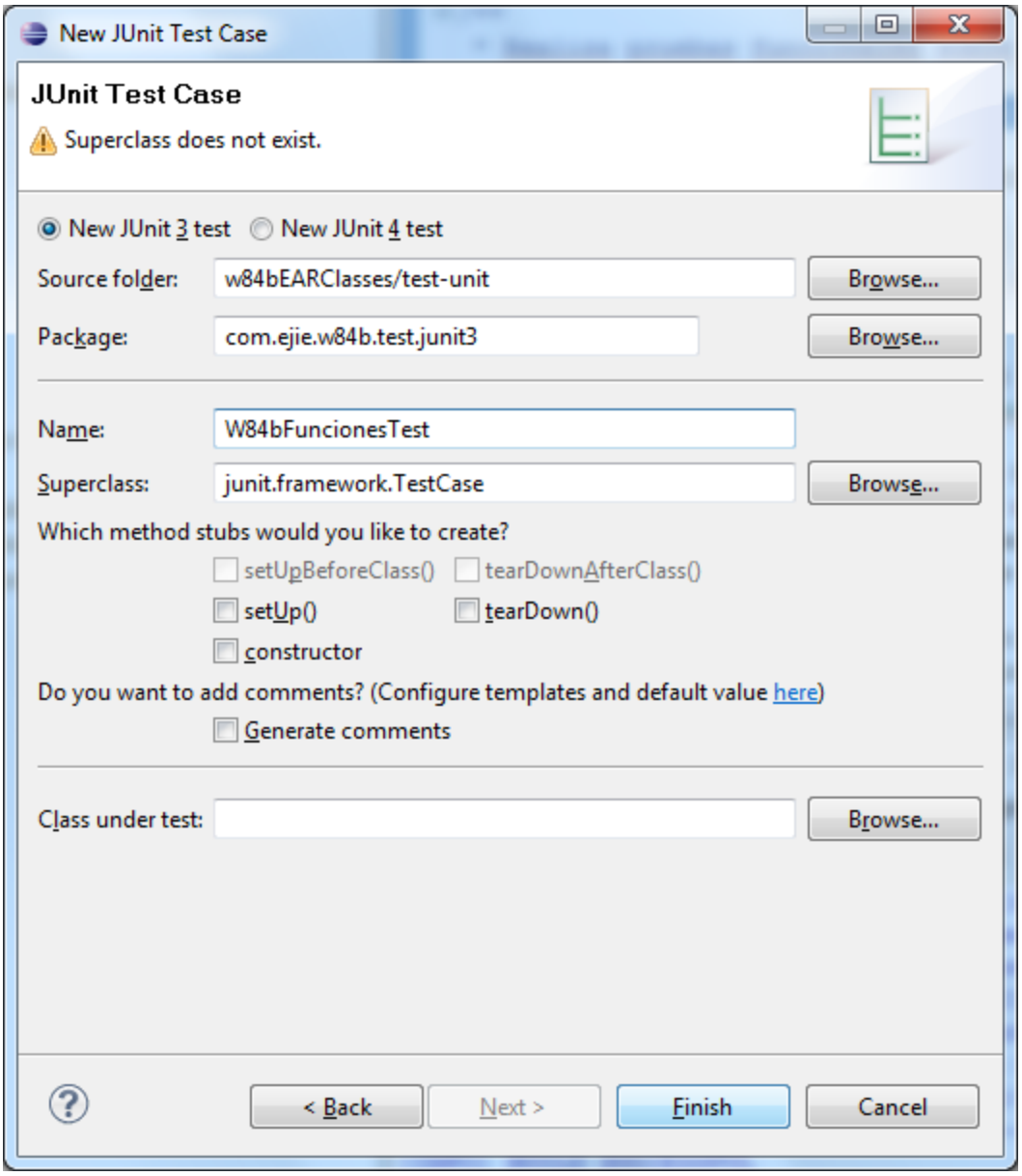
3 Utilizando JUnit en Eclipse

Utilizando el IDE Eclipse se pueden crear clases JUnit Test Case y Test Suites. Cuando se cree el primer Test Case se pedirá la versión de JUnit a utilizar (versión 3 o 4) y dependiendo de la utilizada sugerirá la introducción de la librería en el classpath del proyecto. A continuación se mostrarán ejemplos de cómo crear pruebas utilizando las dos versiones.

El primer paso sería crear un TestCase. Para ello sobre el paquete deseado pulsamos botón derecho del ratón y damos a New > Other y seleccionamos dentro de la carpeta Java > JUnit > JUnit TestCase.



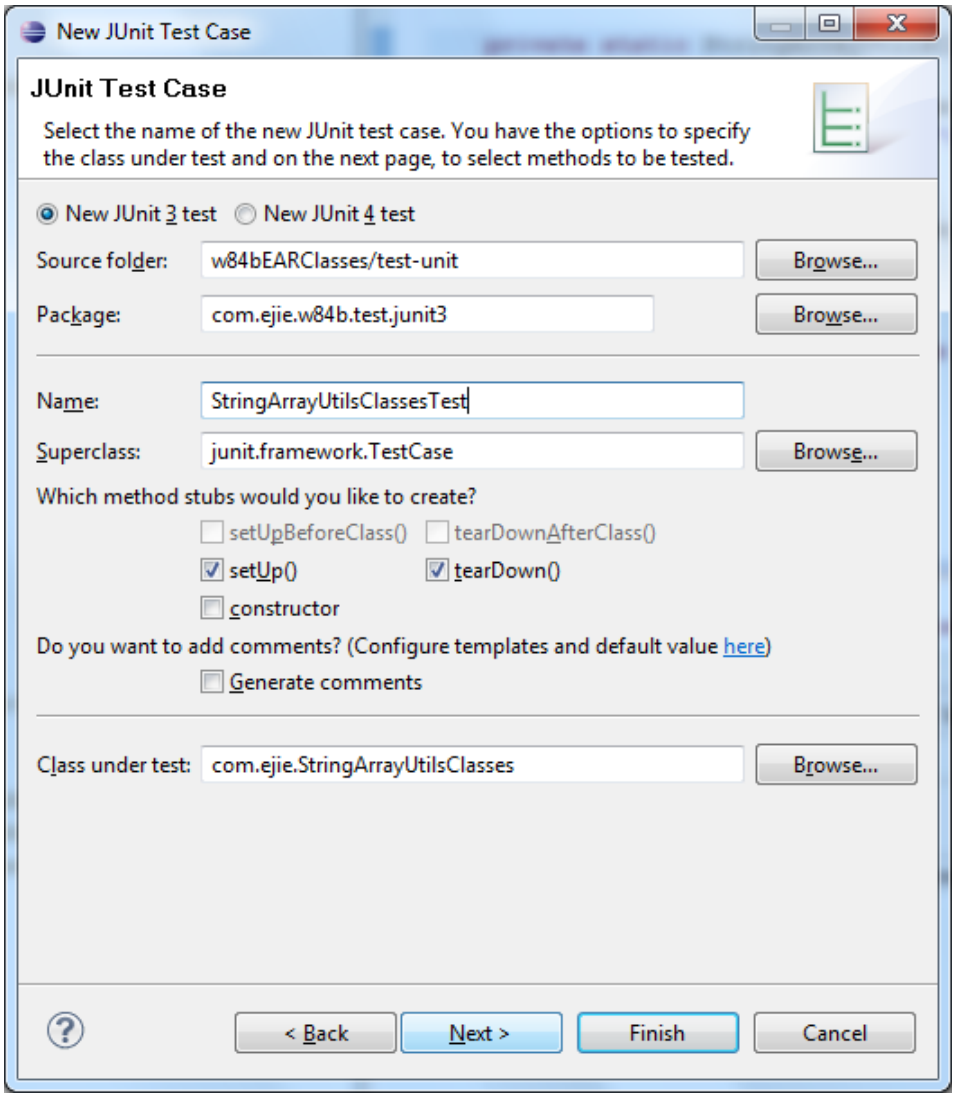
Una vez ahí se decidirá la versión de JUnit a utilizar:



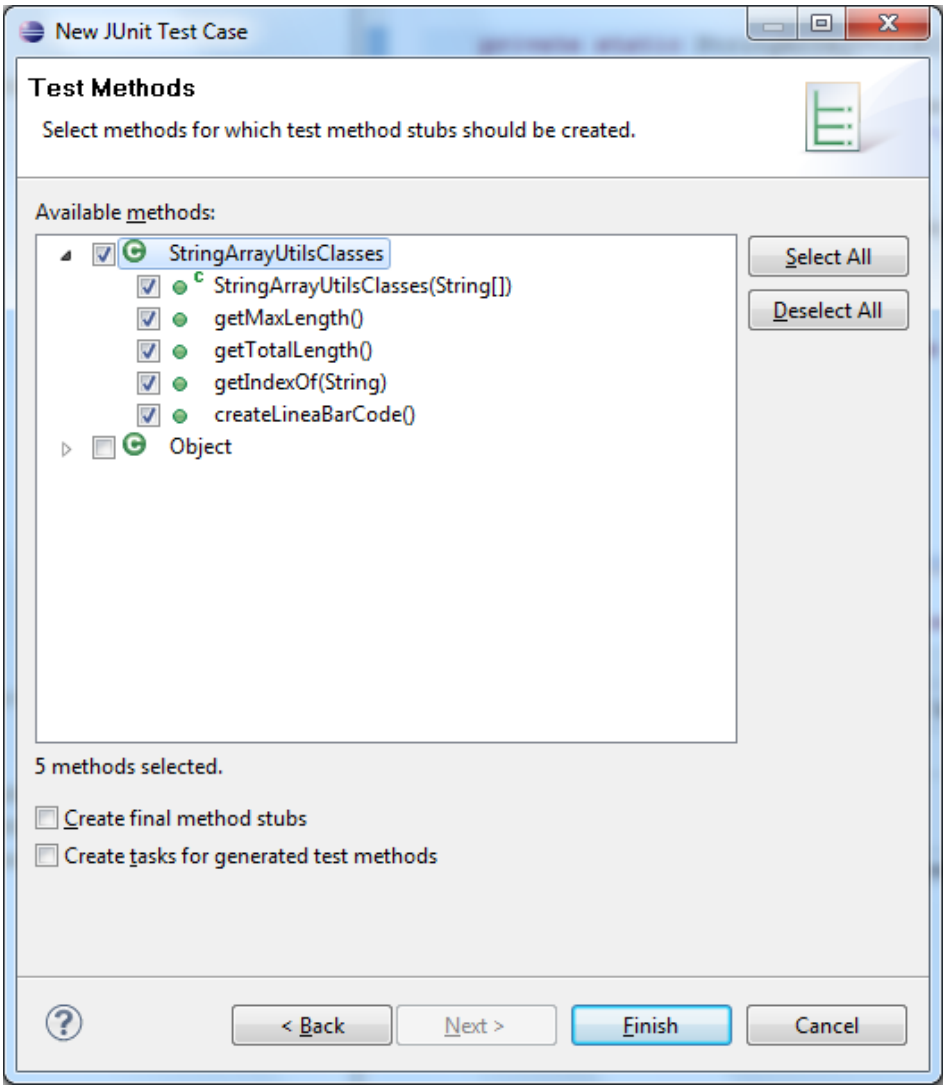
A partir de aquí se crearán las diferentes clases dependiendo de la librería seleccionada.

3.1 JUnit 3

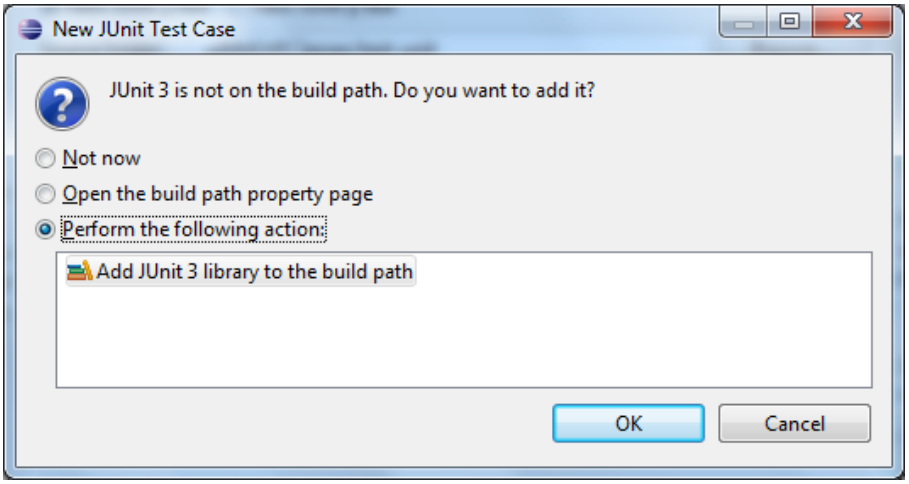
Se seleccionará crear el Test Case con JUnit 3 como se ve a continuación.



Si se le da a Next se podrán seleccionar los métodos a testear



Si no se tiene añadida la librería al classpath se sugerirá añadirla.



Te creará un esqueleto de la clase para implementar los test a posteriori.

```
package com.ejie.w84b.test.junit3;

import junit.framework.TestCase;

public class StringArrayUtilsClassesTest extends TestCase {

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

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

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

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

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

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

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

}
```

3.1.1. 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.
```

```
        }
}
```

3.1.2. 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.
        }
}
```

3.1.3. Métodos de aserciones

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	fail () Falla el test sin ningún mensaje.
Static void	fail (java.lang.String message) Falla el test con el mensaje indicado.
Static void	assertEquals (bolean excepted, bolean 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, flota 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, bolean expected, bolean 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 excepted, long actual) Verifica que los valores proporcionados sean iguales. Si no lo son, envia el mensaje indicado por el parámetro message.
Static void	assertEquals (java.lang.String message, java.lang.Object excepted, 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 excepted, 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 excepted, 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 (bolean condition) Verifica que el objeto bolean seleccionado sea falso.
Static void	assertFalse (java.lang.String message, bolean condition) Verifica que el objeto bolean 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 (bolean 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.

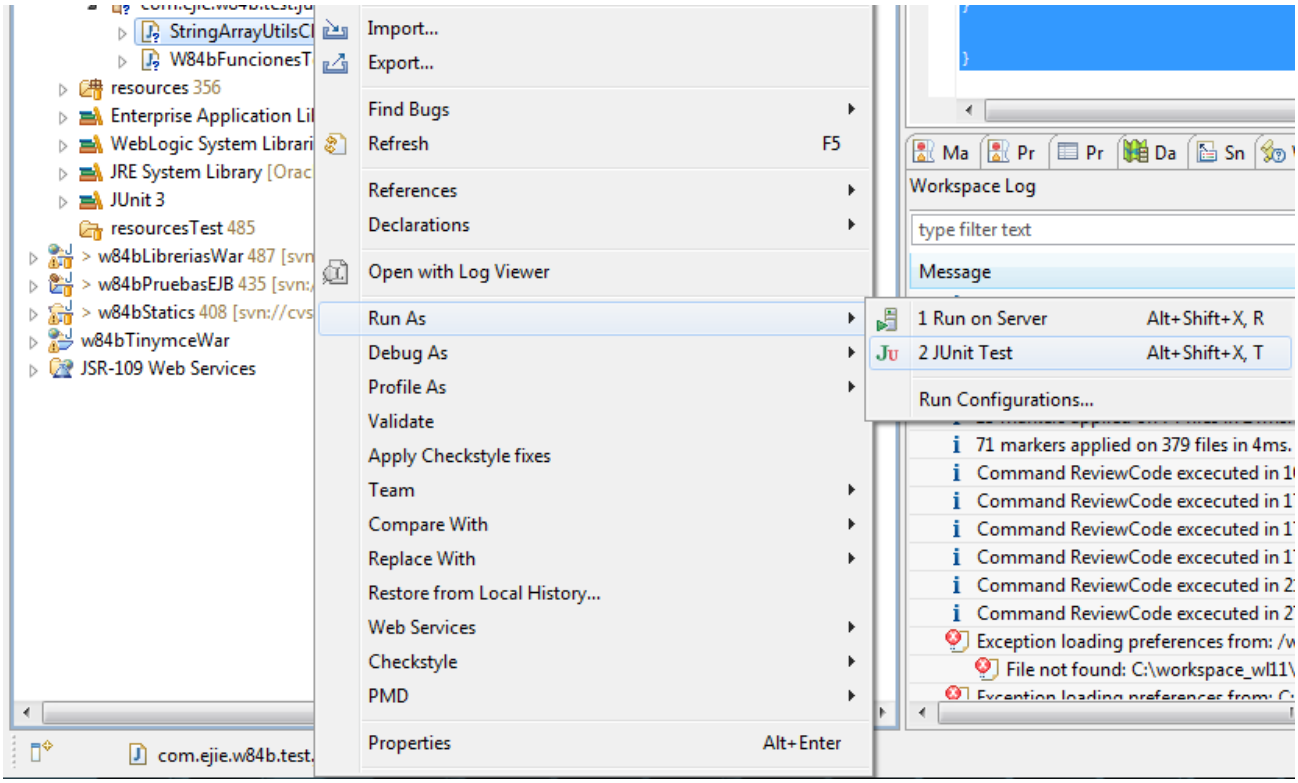
3.1.1. Ejemplo de Test Case

A continuación se muestra un ejemplo de Test Case con JUnit 3 que realiza pruebas unitarias de la clase `com.ejie.StringArrayUtilsClasses`.

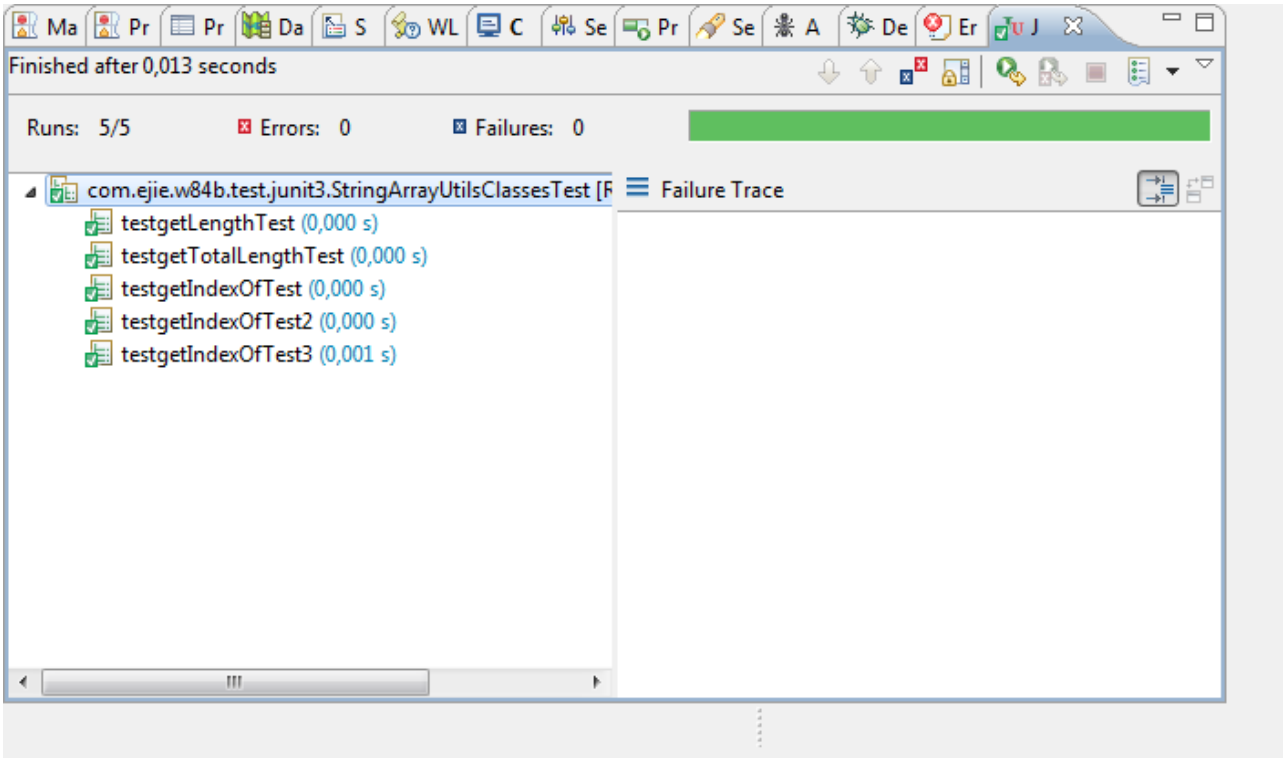
```
package com.ejie.w84b.test.junit3;

import com.ejie.StringArrayUtilsClasses;

import junit.framework.TestCase;

public class StringArrayUtilsClassesTest extends TestCase {

	private static StringArrayUtilsClasses stringUtils;

	protected void setUp() throws Exception {
		String[] elements = {"uno","dos","tres"};


	  stringUtils = new StringArrayUtilsClasses(elements);
	}


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



/**
 * Verificamos que la cadena más larga sea la cadena "Tres"
 */
public void testgetLengthTest() throws Exception {
    assertEquals("tres", stringUtils.getMaxLength());
}

/**
 * Prueba sobre el método que devuelve la suma total de todas las cadenas
 * almacenadas
 */
public void testgetTotalLengthTest() {
    assertEquals(10, stringUtils.getTotalLength());
}

/**
 * Prueba sobre el método que devuelve la posición de una cadena.
 * Verificamos que si le pasamos null como argumento lanza la excepción
```

```
 * correcta
 */
public void testgetIndexOfTest() {
      try{
            stringUtils.getIndexOf(null);
            fail("Debería haber lanzado una excepción:
java.lang.IllegalArgumentException");
      }catch (RuntimeException re){
            //Tratar excepcion
      }
}

/**
 * Prueba sobre el método que devuelve la posición de una cadena Verificamos
 * que si le pasamos una cadena que no existe como argumento lanza la
 * excepción correcta
 */
public void testgetIndexOfTest2() {
      try{
            assertEquals(0, stringUtils.getIndexOf("EsteElementoNoExiste"));
            fail("Debería haber lanzado una excepción:
java.util.NoSuchElementException");
      }catch (RuntimeException re){
            //Tratar excepcion
      }
}

/**
 * Prueba sobre el método que devuelve la posición de una cadena.
 * Verificamos que si le pasamos una cadena que existe devuelve la posición
correcta
 */
public void testgetIndexOfTest3() {
    assertEquals(1, stringUtils.getIndexOf("dos"));
}

}
```

Para ejecutar la prueba unitaria en el menu contextual de la clase JUnit a ejecutar se selecciona Run As JUnit Test.

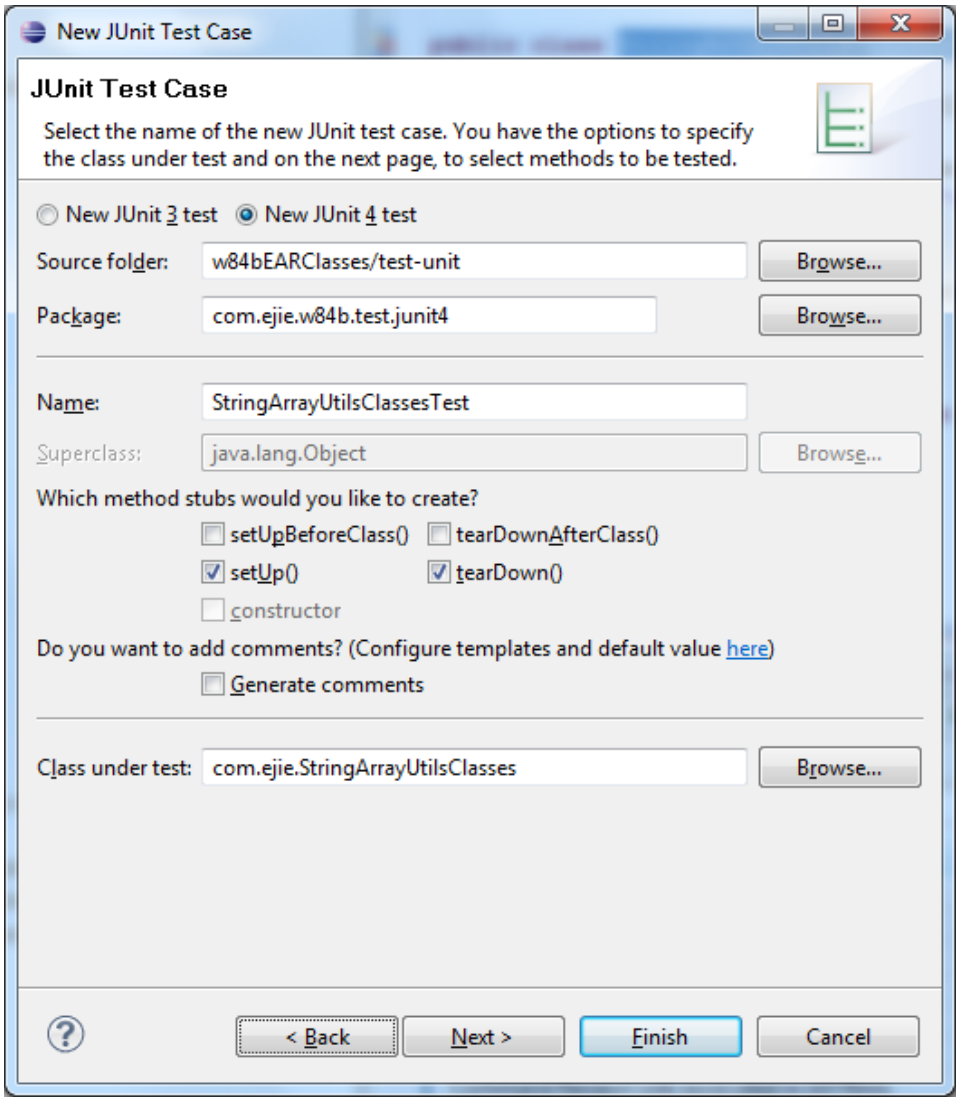


Tras la ejecución se mostrarán los resultados de las pruebas en la vista de JUnit:

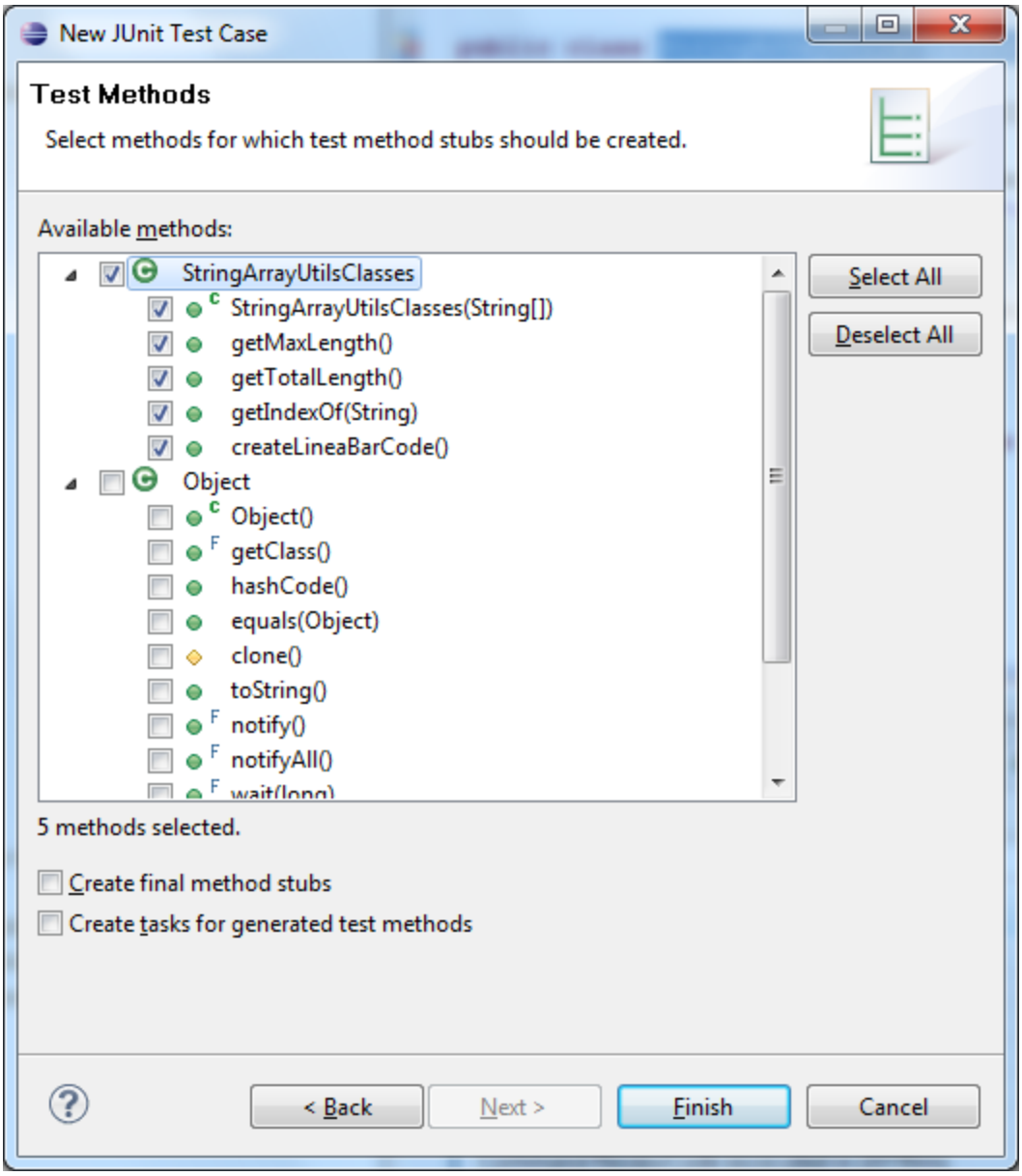


3.1 JUnit 4

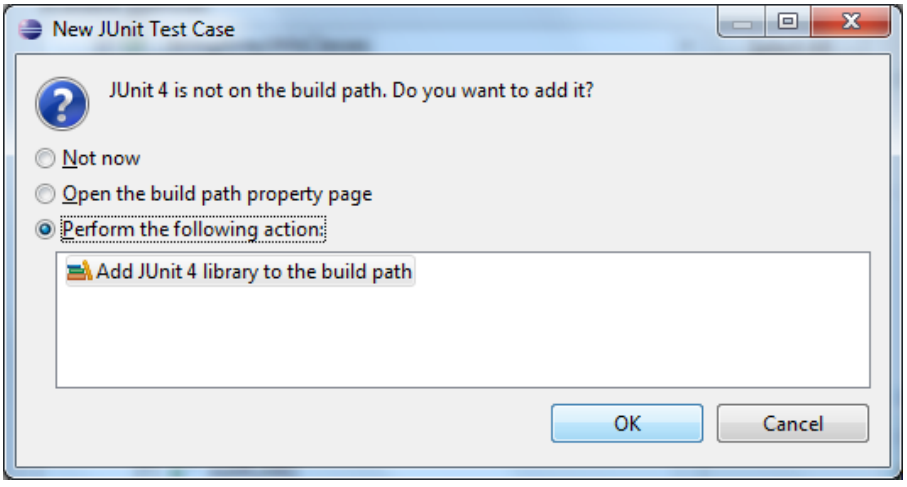
Se seleccionará crear el Test Case con JUnit 4 como se ve a continuación.



Si se le da a Next se podrán seleccionar los métodos a testear



Si no se tiene añadida la librería al classpath se sugerirá añadirla.



Te creará un esqueleto de la clase para implementar los test a posteriori.

```
package com.ejie.w84b.test.junit4;

import static org.junit.Assert.*;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class StringArrayUtilsClassesTest {

	@Before
	public void setUp() throws Exception {
	}

	@After
	public void tearDown() throws Exception {
	}

	@Test
	public void testStringArrayUtilsClasses() {
		fail("Not yet implemented"); // TODO
	}

	@Test
	public void testGetMaxLength() {
		fail("Not yet implemented"); // TODO
	}

	@Test
	public void testGetTotalLength() {
		fail("Not yet implemented"); // TODO
	}

	@Test
	public void testGetIndexOf() {
		fail("Not yet implemented"); // TODO
	}

	@Test
	public void testCreateLineaBarCode() {
		fail("Not yet implemented"); // TODO
	}

}
```

3.1.1. Anotaciones en JUnit 4

JUnit 4 se basa en anotaciones para determinar los métodos de a testear así como para ejecutar código previo a los tests a realizar. En la tabla a continuación se muestran las anotaciones disponibles:

Anotación	Descripción
@Test public void method()	La anotación @Test identifica el método como método de test.
@Test (expected = Exception.class)	Falla si el método no lanza la excepción esperada.
@Test(timeout=100)	Falla si el método tarda más de 100 milisegundos.
@Before public void method()	Este método es ejecutado antes de cada test. Se usa para preparar el entorno de test (p.ej., leer datos de entrada, inicializar la clase).
@After public void method()	Este método es ejecutado después de cada test. Se usa para limpiar el entorno de test (p.ej., borrar datos temporales, restaurar valores por defecto). Se puede usar también para ahorrar memoria limpiando estructuras de memoria pesadas.
@BeforeClass public static void method()	Este método es ejecutado una vez antes de ejecutar todos los test. Se usa para ejecutar actividades intensivas como conectar a una base de datos. Los métodos marcados con esta anotación necesitan ser definidos como static para trabajar con JUnit.
@AfterClass public static void method()	Este método es ejecutado una vez después que todos los tests hayan terminado.Se usa para actividades de limpieza, como por ejemplo, desconectar de la base de datos. Los métodos marcados con esta anotación necesitan ser definidos como static para trabajar con JUnit.
@Ignore	Ignora el método de test. Es útil cuando el código a probar ha cambiado y el caso de uso no ha sido todavía adaptado. O si el tiempo de ejecución del método de test es demasiado largo para ser incluído.

3.1.2. Métodos de aserciones

Los métodos para las aserciones son como los de la versión anterior:

Statement	Descripción
fail(String)	Hace que el método falle. Debe ser usado para probar que cierta parte del código no es alcanzable para que el test devuelva fallo hasta que se implemente el método de prueba. El parámero String es opcional.
assertTrue([message], boolean condition)	Verifica que la condición booleana sea true.
assertFalse([message], boolean condition)	Verifica que la condición booleana sea false.
assertEquals([String message], expected, actual)	Verifica que los dos valores son idénticos. Nota: Para arrays se verifica la referencia no el contenido de éstos.
assertEquals([String message], expected, actual, tolerance)	Verifica que los valores float o double coincidan. La tolerancia es el número de decimales que deben ser iguales.
assertNull([message], object)	Verifica que el objeto sea nulo.
assertNotNull([message], object)	Verifica que el objeto no sea nulo.
assertSame([String], expected, actual)	Verifica que las dos variables referencien al mismo objeto.
assertNotSame([String], expected, actual)	Verifica que las dos variables no referencien al mismo objeto.

3.1.3. Ejemplo de Test Case

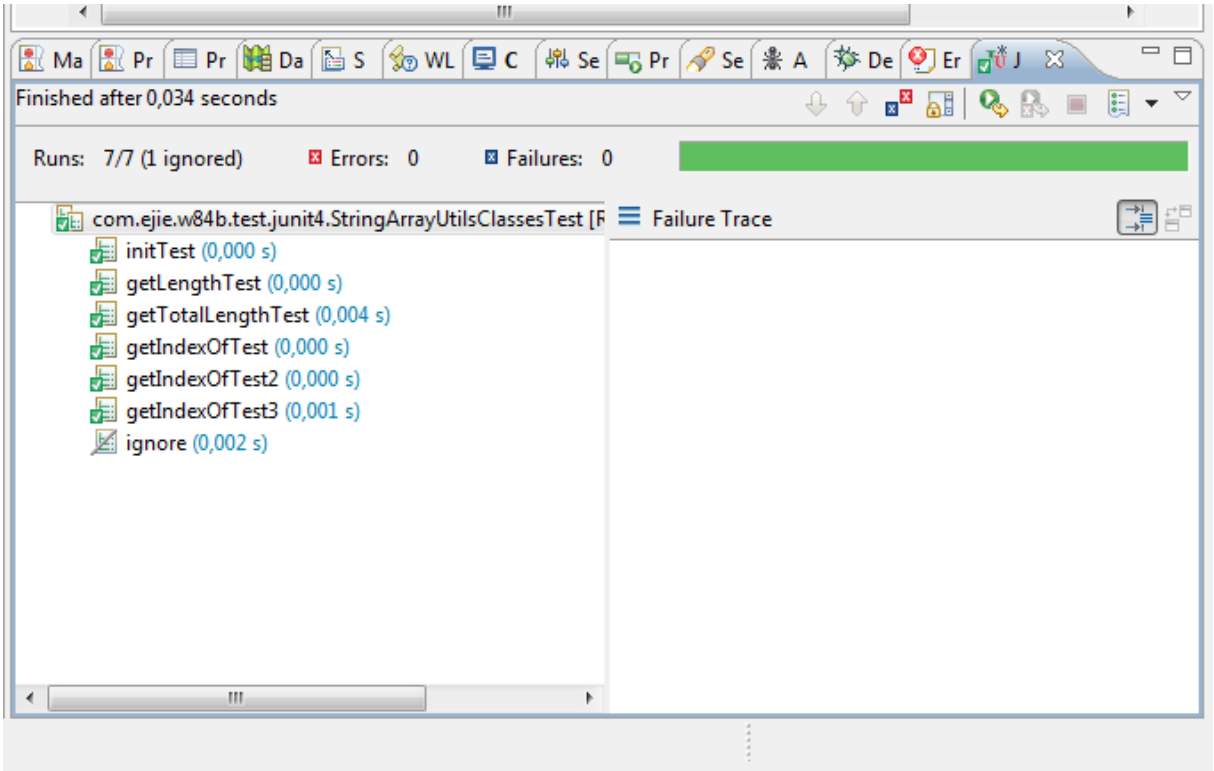
A continuación se muestra un ejemplo de Test Case con JUnit 4 que realiza pruebas unitarias de la clase `com.ejie.StringArrayUtilsClasses`.

```
package com.ejie.w84b.test.junit4;

import static org.junit.Assert.*;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Vector;

import junit.framework.Assert;
import junit.framework.JUnit4TestAdapter;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Ignore;
import org.junit.Test;
```

```java
import org.junit.runner.JUnitCore;

import com.ejie.StringArrayUtilsClasses;
import com.ejie.test.JUnit4UnitClasesTestStringArrayUtilsTest;

public class StringArrayUtilsClassesTest {

    private static StringArrayUtilsClasses stringUtils;

    /**
     * Leemos desde un fichero, los datos de prueba con los que vamos a realizar las pruebas
     */
    @BeforeClass
    public static void inicioClase() {
      String[] elements = {"uno","dos","tres"};
        stringUtils = new StringArrayUtilsClasses(elements);
      }

    /**
     * Este método liberará los recursos reservados en BeforeClass
     */
    @AfterClass
    public static void finClase() {
        // Para este ejemplo no hacemos nada, pero exponemos el m todo por
        // motivos didácticos exclusivamente
    }

    /**
     * Este método se ejecuta para cada prueba ANTES de invocar el código de cada prueba
     */
    @Before
    public void testStart() {
        // Para este ejemplo no hacemos nada, pero exponemos el m todo por
        // motivos didácticos exclusivamente
    }

    /**
     * Este método se ejecuta para cada prueba DESPUÉS de invocar el código de cada prueba.
     */
    @After
    public void testEnd() {
        // Para este ejemplo no hacemos nada, pero exponemos el m todo por
        // motivos didácticos exclusivamente
    }

    /**
     * Verificamos que en caso de recibir un null como argumento en el
     * constructor la clase lanza una IllegalArgumentException
     */
    @Test(expected = java.lang.IllegalArgumentException.class)
    public void initTest() {
        new StringArrayUtilsClasses(null);
    }

    /**
     * Verificamos que la cadena más larga sea la cadena "Tres"
     */
    @Test
    public void getLengthTest() {
        Assert.assertEquals("tres", stringUtils.getMaxLength());
    }

    /**
```

```
     * Prueba sobre el método que devuelve la suma total de todas las cadenas
     * almacenadas Suponemos que el calculo del tamaño total es un método
     * crítico que debe realizarse antes de 25 milisegundos
     */
    @Test(timeout = 25)
    public void getTotalLengthTest() {
        Assert.assertEquals(10, stringUtils.getTotalLength());
    }

    /**
     * Prueba sobre el método que devuelve la posición de una cadena.
     * Verificamos que si le pasamos null como argumento lanza la excepción
     * correcta
     */
    @Test(expected = java.lang.IllegalArgumentException.class)
    public void getIndexOfTest() {
        stringUtils.getIndexOf(null);
    }

    /**
     * Prueba sobre el método que devuelve la posición de una cadena Verificamos
     * que si le pasamos una cadena que no existe como argumento lanza la
     * excepción correcta
     */
    @Test(expected = java.util.NoSuchElementException.class)
    public void getIndexOfTest2() {
        Assert.assertEquals(0, stringUtils.getIndexOf("EsteElementoNoExiste"));
    }

    /**
     * Prueba sobre el método que devuelve la posición de una cadena.
     * Verificamos que si le pasamos una cadena que existe devuelve la posición
correcta
     */
    @Test
    public void getIndexOfTest3() {
        Assert.assertEquals(1, stringUtils.getIndexOf("dos"));
    }

    @Ignore("Este test no se hace, se expone como ejemplo")
    @Test
    public void ignore() {
        // Código que compone la prueba
        // ...
        // ...
    }
}
```

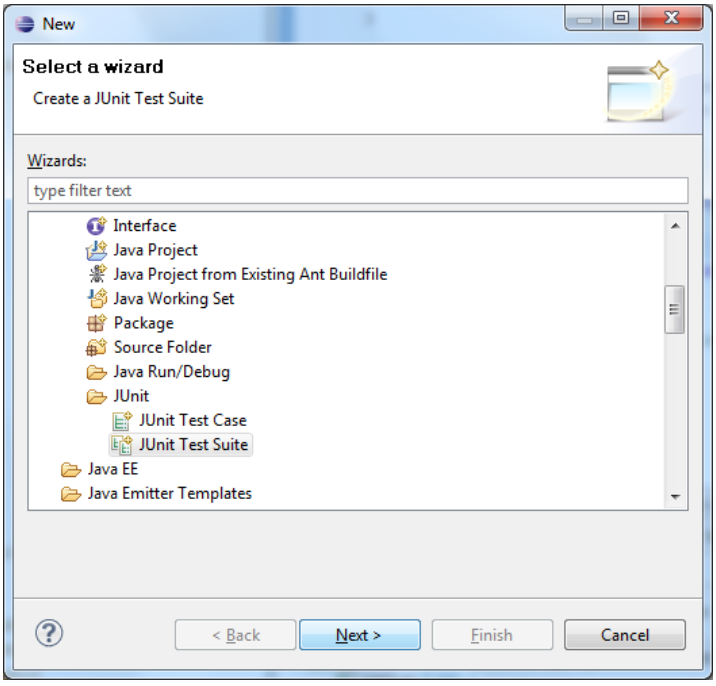
Para ejecutar la prueba unitaria se procedería de manera igual que para el ejemplo de JUnit 3, en el menu contextual de la clase JUnit a ejecutar se selecciona Run As JUnit Test y obtendríamos el resultado de la ejecución.



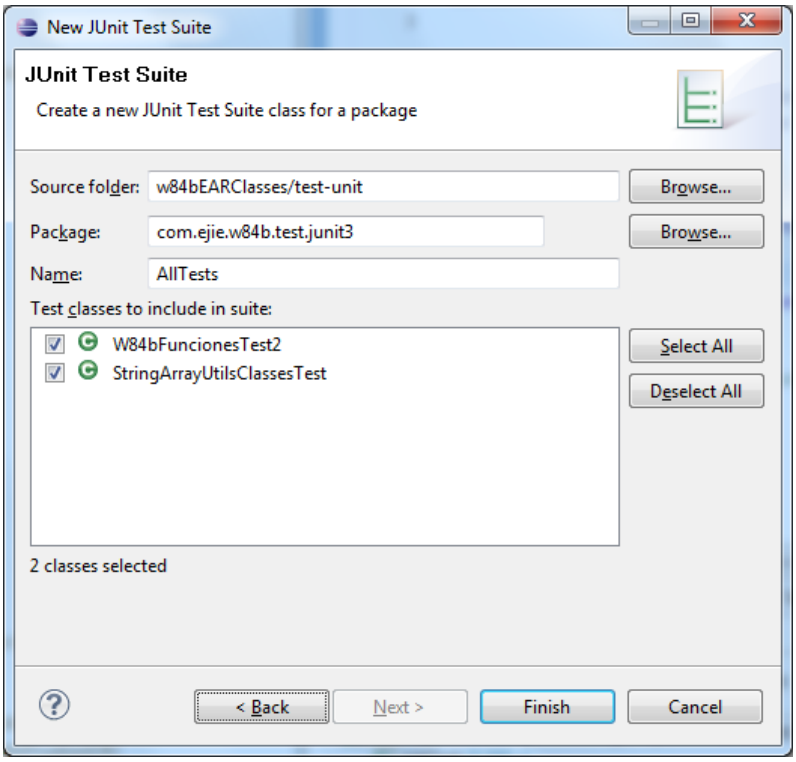
3.2 Crear un Test Suite

Una vez creados varios **TestCase** se podrán agrupar en árboles de **TestSuite**, estos últimos serán los que invocquen automáticamente todos los métodos **testXXX ( )** definidos en cada **TestCase**. En otras palabras con un **TestSuite** tendremos la posibilidad de juntar varios TestCase, e incluso otros TestSuites. De esta forma ejecutando el **TestSuite** apropiado, podremos ver los resultados de todos los diferentes tests unidos.

Para crear un TestCase sobre el paquete donde residen las pruebas Junit pulsamos botón derecho del ratón y damos a New > Other y seleccionamos dentro de la carpeta Java > JUnit > JUnit Test Suite.



Al darle a siguiente se mostrarán las clases a incluir en el Test Suite.



Creara una clase como la que sigue:

```
package com.ejie.w84b.test.junit3;

import junit.framework.Test;
import junit.framework.TestSuite;

public class AllTests {

	public static Test suite() {
		TestSuite suite = new TestSuite(AllTests.class.getName());
		//$JUnit-BEGIN$
		suite.addTestSuite(W84bFuncionesTest2.class);
		suite.addTestSuite(StringArrayUtilsClassesTest.class);
		//$JUnit-END$
		return suite;
	}
	}
```

Una vez creado se podrá ejecutar el TestSuite creado analogamente a los TestCasesmostrandose el resultado siguiente: