

## Pruebas en eclipse con JUnit y EclEmma

### Introducción

En este laboratorio se presentará la herramienta de pruebas *JUnit* a través de varios ejemplos. A continuación veremos cómo se puede medir la cobertura de las pruebas realizadas utilizando la herramienta *EclEmma*.

### Objetivos

Los objetivos de este laboratorio son los siguientes:

- Conocer las herramientas *JUnit* y *EclEmma*.
- Desarrollar varios casos de prueba utilizando la herramienta *JUnit*.
- Medir la cobertura de las pruebas realizadas utilizando la herramienta *EclEmma*.

### ¿Qué es JUnit?

*JUnit* es un conjunto de bibliotecas creadas por Erich Gamma y Kent Beck que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java. Entre otras funcionalidades contiene una herramienta (llamada test runner) que ejecuta las pruebas. *JUnit* no es una herramienta de pruebas automática: los casos de prueba se tienen que escribir manualmente. *JUnit* te ofrece algunas guías para poder definirlos y por lo tanto te permite escribirlos de manera más adecuada.

Supongamos que queremos probar la clase C. Escribimos una nueva clase CTest donde definiremos los casos de prueba para testear la clase C. Una vez escrita la clase de prueba CTest, debemos ejecutarla para comprobar si la clase C está definida correctamente. Esto se realiza ejecutando la herramienta *JUnit* test runner y pasándole como argumento nuestra clase de prueba CTest. Esto es todo lo necesario. *JUnit* ejecutará los casos de prueba por nosotros.

Una vez ejecutados los casos de prueba, *JUnit* genera un informe con las pruebas que se han ejecutado satisfactoriamente y cuáles han fallado. Para las pruebas que han fallado, se añade información adicional indicando de los detalles del fallo, que permitirán darnos guías para resolver el error.

*JUnit* se puede ejecutar desde la línea de comando o como un plug-in de Eclipse. En esta laboratorio exploraremos la segunda opción.

### Pasos a seguir

#### Paso 1. Crear un proyecto con la clase que queremos probar.

Abrimos Eclipse, creamos un proyecto *JUnitLab* y a continuación la clase Java que queremos probar.

En este ejemplo vamos a probar la clase *Subscripcion* que viene a continuación:

```
public class Suscripcion {
    private int precio ; // precio total de la suscripción euro-cent
    private int periodo ; // periodo de suscripcion en meses

    /**
     * El constructor para crear una suscripción.
     */
    public Suscripcion(int p, int n) {
        precio = p ;
        periodo = n ;
    }

    /**
     * Calcula el precio de la suscripción mensual en euros,
     * redondeándolo por arriba al céntimo más cercano.
     */
    public double precioPorMes() {
        if (periodo<=0 || precio<=0) return 0 ;
        double r = (double) precio / (double) periodo ;
        double resto = r%1;

        if (resto > 0)
            return r+1;
        else
            return r;
    }

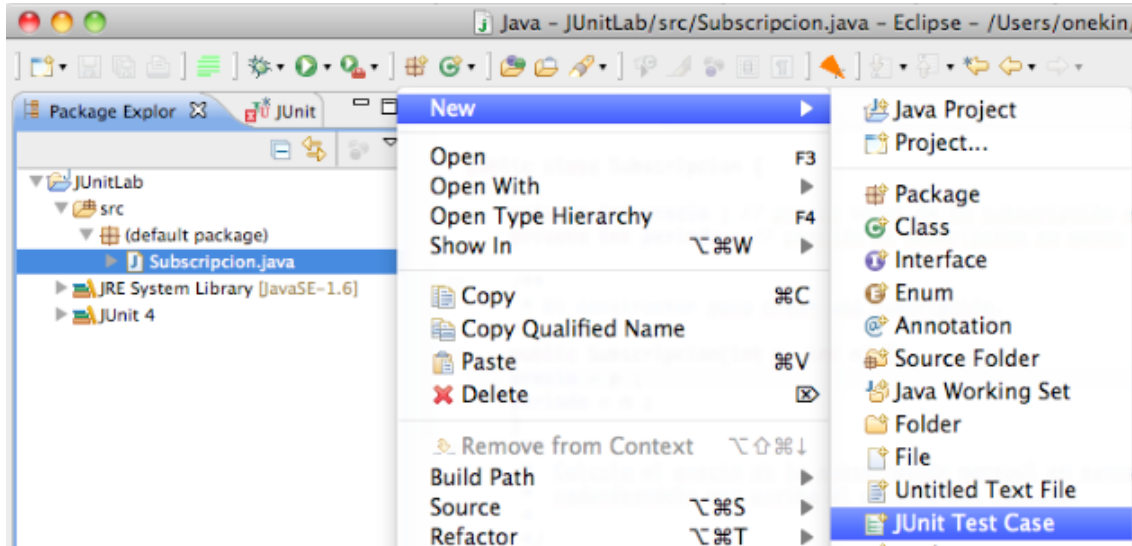
    /**
     * Este método cancela la suscripción.
     */
    public void cancel() {
        periodo = 0 ;
    }
}
```

Una instancia de esta clase representa una suscripción a una publicación. Cada suscripción contiene el precio total de la suscripción en la variable precio. El precio recoge el total en euros y céntimos (2 decimales) (p.ej 1245 representa, 12 euros y 45 céntimos, o 1300 representa 13 euros). El periodo de suscripción se representa en meses y se recoge en la variable periodo.

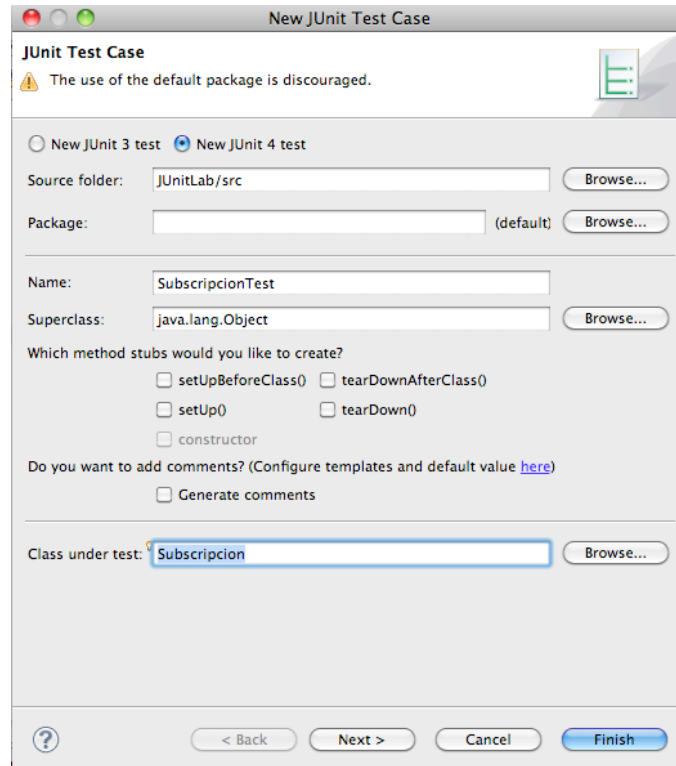
Antes de continuar estudiar el código y entender el método precioPorMes() que aparece en la clase Suscripción.

## Paso 2. Crear una clase de prueba.

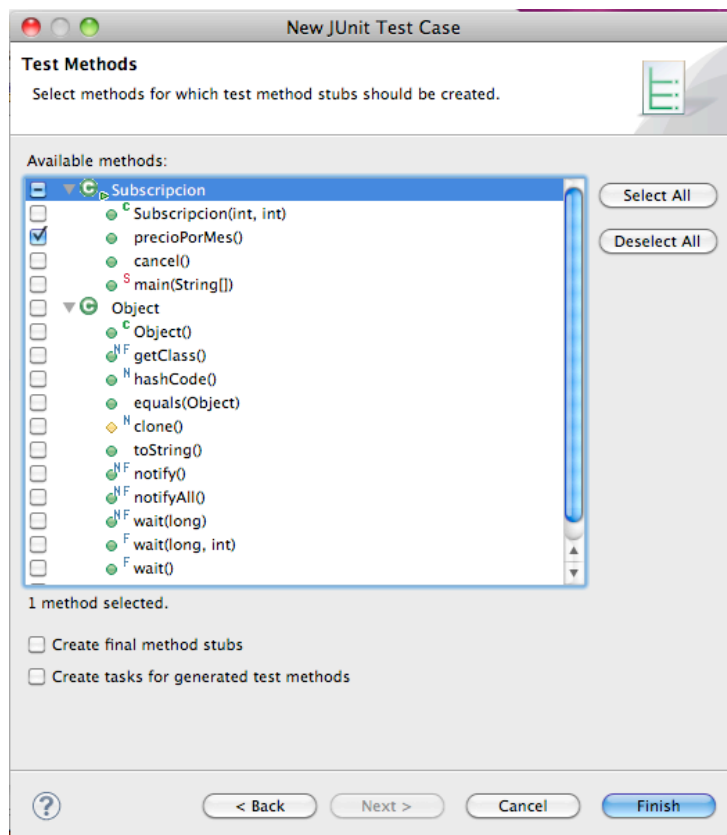
Para crear una clase de prueba que pruebe una clase ya existente (es decir, Suscripción), nos posicionamos sobre la clase que queremos probar, pulsamos el botón derecho y seleccionamos la opción “New”>>”JUnit Test Case” tal y como aparece en la siguiente figura.



A continuación nos aparecerá una pantalla, donde algunos de los campos estarán rellenos en base a la clase que queremos probar, tal y como aparece a continuación:



Pulsamos el botón Next, y en la última pantalla nos aparecerán los métodos de la clase sobre la cual queremos crear un caso de prueba. En nuestro caso, seleccionamos el método precioPorMes y pulsamos Finish.



Como podemos observar la herramienta nos crea una clase de prueba, donde incluye las librerías de JUnit necesarias para su ejecución, así como un método de prueba vacío (marcado con la etiqueta @Test) donde deberemos escribir nuestro caso de prueba.

```
import static org.junit.Assert.*;

import org.junit.Test;

public class SubscriptionTest {

    @Test
    public void testPrecioPorMes() {
        fail("Not yet implemented");
    }
}
```

### Paso 3. Crear los casos de prueba.

Como primer ejemplo, podemos rellenar la clase anterior con el siguiente código.

```
import static org.junit.Assert.*;

import org.junit.Test;

public class SubscripcionTest {

    @Test
    public void testprecioPorMes() {
        double esperado=1;
        Subscripcion s = new Subscripcion(200,2) ;
        double resultado = s.precioPorMes();
        assertEquals(esperado, resultado,0) ;
    }

    @Test
    public void testprecioPorMes2() {
        double esperado=67;
        Subscripcion s = new Subscripcion(200,3) ;
        double resultado= s.precioPorMes();
        assertEquals(esperado, resultado, 0) ;
    }

}
```

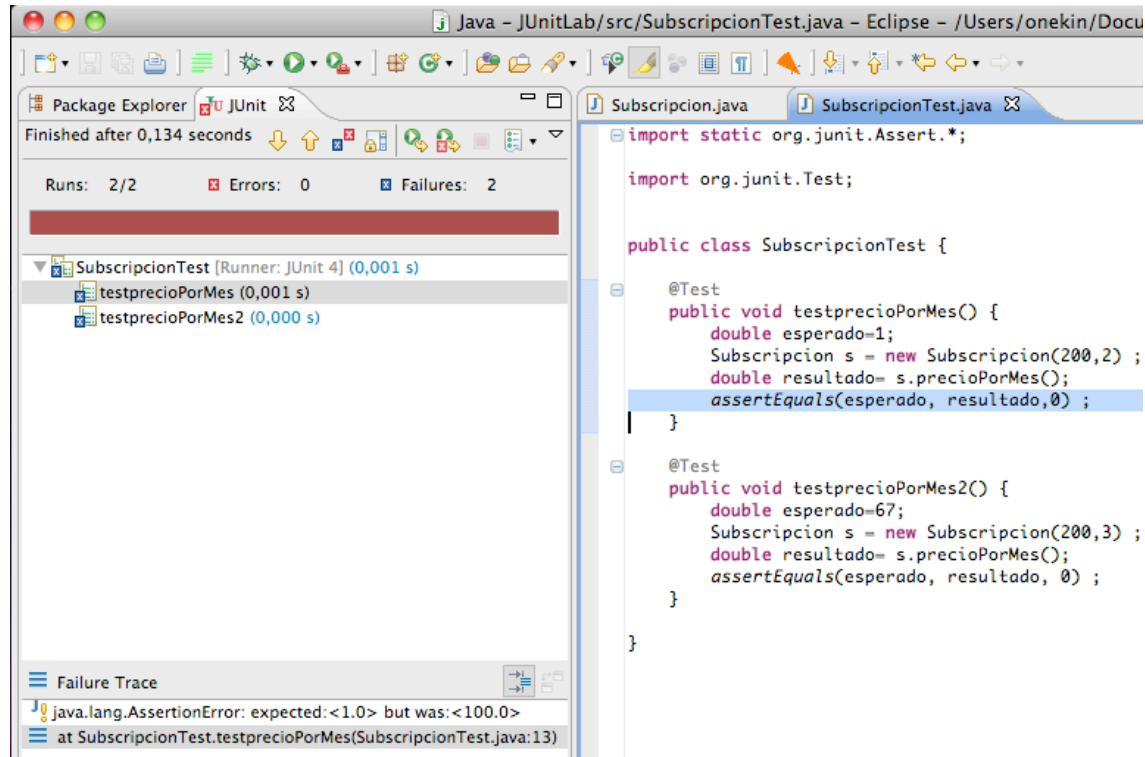
Todo caso de prueba se compone básicamente de 3 partes:

1. Se indica en una variable, cuál es el valor esperado por el método que queremos comprobar.
2. Se ejecuta el método que queremos probar con los datos de entrada adecuados y se guarda el resultado en otra variable.
3. Se comprueba la relación entre el valor esperado y el resultado del método, a través de una aserción. En este caso la aserción *assertEquals* indica que el caso de prueba es correcto sólo si el valor de las variables *esperado* y *resultado* son iguales. Nota: el tercer parámetro de *assertEquals* se puede especificar la diferencia admisible entre *esperado* y *resultado* para considerar que dichos valores son iguales, en este caso, al ser 0, deben ser iguales.

#### Paso 4. Ejecutar los casos de prueba

Para ejecutar la clase de prueba, seleccionar la clase de prueba con el botón derecho, y seleccionar la opción, **“Run As”>>JUnit Test**.

Después de ejecutar la clase de prueba, obtendréis en la parte izquierda de eclipse una pantalla con información acerca de la ejecución de los casos de prueba, tal y como aparece a continuación:



En la figura se puede observar que tenemos 2 fallos, indicando que los 2 test en la clase SubscriptionTest han fallado. Debajo de la clase SubscriptionTest aparecen todos los métodos de test, indicando a través de un icono en la parte izquierda si se han ejecutado con éxito o han fallado. Si te posicionas en el test que ha fallado, en el panel inferior puedes obtener información acerca del fallo y en la línea del caso de test donde se ha producido el fallo.

En nuestro primer caso de prueba, el panel inferior nos informa que ha fallado por :

```
java.lang.AssertionError: expected:<1.0> but was:<100.0>
```

Es decir, que esperaba 1.0 y ha recibido como respuesta 100.0. En este punto podemos observar que el valor esperado no estaba correctamente definido, y que en vez de un error en el programa, es un error de la definición del caso de prueba. Modificamos la variable esperado=100 en el caso de prueba y volvemos a ejecutarlo, y comprobaremos que efectivamente se ejecuta correctamente.

A continuación debéis resolver el fallo en el segundo caso de prueba.

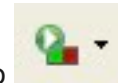
#### Paso 6: Medición de la cobertura de código

Después de solucionar los fallos, nuestra clase parece que se ejecuta de manera satisfactoria. Sin embargo, ¿podemos asegurar que nuestra clase está definida correctamente? Cubren nuestros casos de prueba todas las ramas de ejecución del programa (cobertura)?

La idoneidad (adequacy) de los test se mide en términos de su cobertura. La cobertura de líneas de código determina el porcentaje de líneas de código que son ejecutadas por los tests. De manera similar, la cobertura de ramas, es el porcentaje de ramas en los métodos que son atravesadas por los test (condicionales). El 100% de cobertura no implica la ausencia de errores, pero sí que indica un testeo riguroso. Por el contrario, una baja cobertura conlleva a una mayor posibilidad de que el código contenga errores no detectados.

Actualmente existen herramientas en eclipse que permiten medir la cobertura. **EclEmma** es una herramienta para **Eclipse** que permite visualizar la **cobertura del código** de los tests unitarios que hemos realizado. Esta herramienta se utiliza cuando estamos realizando pruebas de Caja Blanca, es decir, tenemos el código fuente que queremos probar disponible. **EclEmma** mide la cobertura de líneas de código, sin embargo, también nos ofrece información acerca de si una rama de código (p.ej if-the-else) ha sido total o parcialmente cubierta.

**EclEmma** se instala en **Eclipse** de forma sencilla, desde *Help -> Eclipse Marketplace*, en el buscador se pone la palabra **EclEmma** y se siguen los pasos del instalador del plug-in.



Una vez instalado el plugin y reiniciado **Eclipse**, aparece el siguiente icono en la barra de herramientas.

A través de este botón, podemos lanzar los tests de forma que **EclEmma** analizará qué parte del código se está cubriendo en ellos. Veamos un ejemplo práctico.

Crear una nueva clase *OperadorAritmético*, que realiza las operaciones de *suma* y *división*:

```
public class OperadorAritmetico {
    public static int suma(int a, int b) {
        return a + b;
    }
    public static int division(int a, int b) throws Exception {
        if(b==0) {
            throw new Exception();
        }
        return a / b;
    }
}
```

Para probar esta clase, crear una clase de test que, en principio, sólo pruebe la *suma*:

```
import static org.junit.Assert;
import org.junit.Test;

public class OperadorAritmeticoTest {

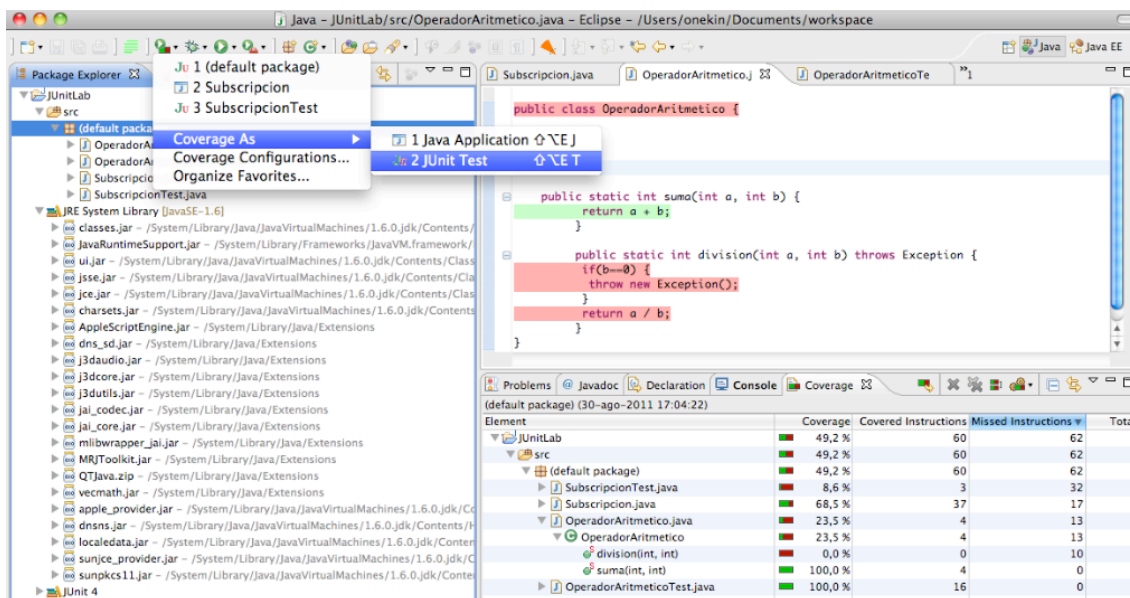
    @Test
    public void suma() {

        int esperado=8;
        int a = 5;
        int b = 3;

        int suma = OperadorAritmetico.suma(a, b);

        assertEquals(esperado, suma);
    }
}
```

Ahora lanzamos las pruebas para el proyecto, situándonos en la raíz del proyecto y yendo al botón de **EcEmma** antes mostrado, seleccionamos la opción *Coverage as -> JUnit Test*, tal y como aparece en la siguiente figura:



Vemos cómo las instrucciones de la clase *OperadorAritmetico* han sido coloreadas, con verde, aquellas que han sido cubiertas en las pruebas y con rojo aquellas que no.

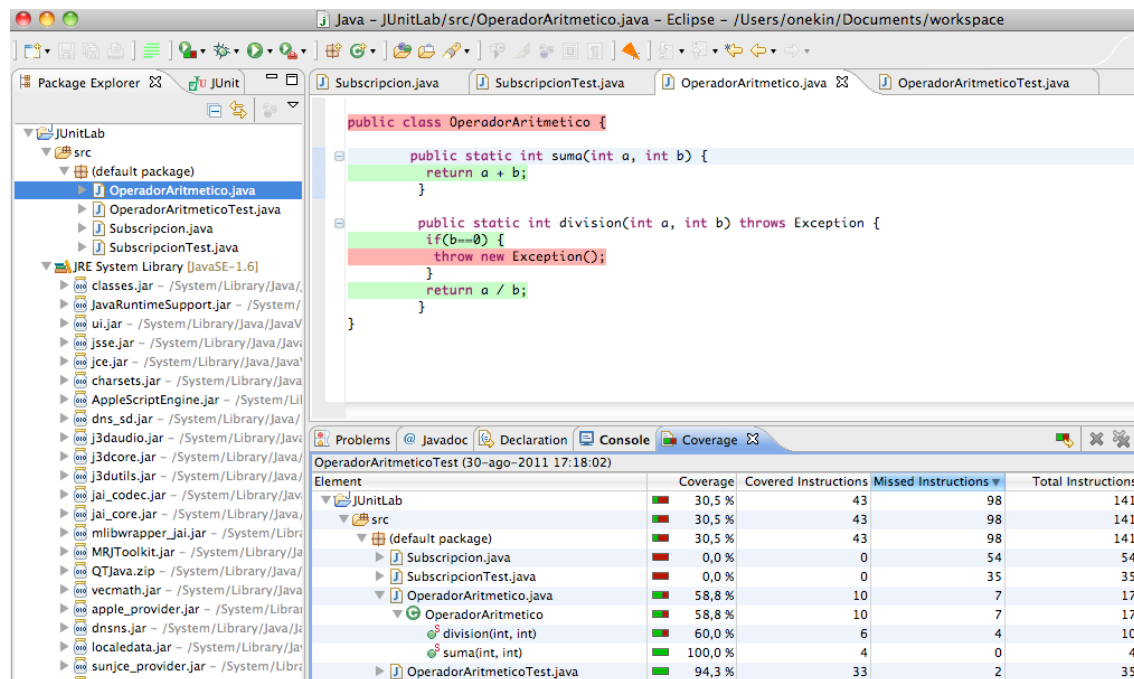


La pestaña *Coverage* muestra el informe de cobertura del proyecto.

A continuación creamos un caso de prueba para la división:

```
@Test
public void division() {
    int esperado=2;
    int a = 8;
    int b = 4;
    int division;
    try {
        division = OperadorAritmetico.division(a, b);
        assertEquals(esperado, division);
    } catch (Exception e) {
        fail();
    }
}
```

Si ahora volvemos a realizar las pruebas, vemos que la situación ha mejorado, aunque todavía queda código sin cubrir:



Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
JUnitLab	30,5 %	43	98	141
src	30,5 %	43	98	141
(default package)	30,5 %	43	98	141
SubscriptionTest.java	0,0 %	0	54	54
SubscriptionTest.java	0,0 %	0	35	35
OperadorAritmeticoTest.java	58,8 %	10	7	17
OperadorAritmetico	58,8 %	10	7	17
division(int, int)	60,0 %	6	4	10
suma(int, int)	100,0 %	4	0	4
OperadorAritmeticoTest.java	94,3 %	33	2	35

De esta forma, podemos ir mejorando progresivamente la cobertura de las pruebas para asegurar el correcto funcionamiento de nuestro código.

**NOTA: tened en cuenta que sólo estamos interesados en obtener una cobertura 100% del código de la clase a probar (OperadorAritmetico), no del código de la clase de pruebas (OperadorAritmeticoTest)**

## Referencias:

- [1] EclEmma: Visualizar la cobertura de los tests unitarios en Eclipse  
<http://tratandodeentenderlo.blogspot.com/2009/11/visualizar-la-cobertura-de-los-tests.html>
- [2] Unit Testing with JUnit - Tutorial  
<http://www.vogella.com/tutorials/JUnit/article.html>
- [3] EclEmma Home Page  
<http://www.eclEmma.org/index.html>

