PRUEBAS DE COBERTURA

PRUEBAS DE TEST UNITARIAS

Es distinto realizar un test parametrizado con Junit versión 4 a la versión 5. NetBeans 8 usa el 4, mientras que eclipse ya usa el 5 así como la versión de Apache de NetBeans.

**VERSIÓN 4**

Hay que importar las clases:

import org.junit.runner.RunWith;

import org.junit.runners.Parameterized;

import org.junit.runners.Parameterized.Parameters;

import java.util.Arrays;

import java.util.Collection;

Después, tendremos que indicar antes de la clase:

@RunWith(Parameterized.class)

Creamos un método que contenga los parámetros a usar:

@Parameters

public static Collection<Object[]> data(){

return Arrays.asList(new Object[][]{

{10, 40, 50}, {10, 10, 20}

});

}

Declaramos las variables que se instanciarán a continuación en el constructor de la clase de prueba:

private int valorA, valorB, esperado;

public SumaTest(int valorA, int valorB, int esperado){

this.valorA = valorA;

this.valorB = valorB;

this.esperado = esperado;

}

Luego solo hay que crear el método de prueba teniendo en cuenta los valores previamente indicados:

@Test

public void testSumar() {

System.out.println("sumar");

Suma instance = new Suma(valorA, valorB);

int result = instance.sumar();

assertEquals(esperado, result);

}

**VERSIÓN 5**

Importamos las siguientes clases:

**import** org.junit.jupiter.params.ParameterizedTest;

**import** org.junit.jupiter.params.provider.CsvSource;

Cada test, tendrá el siguiente aspecto:

@ParameterizedTest

@CsvSource({"0, 0, 0", "100, 0, 100", "100, 100, 200", "50, 50, 100"})

**void** sumarTest(**int** a, **int** b, **int** valorEsperado) {

Calculadora c = **new** Calculadora(a, b);

**int** resultado = c.sumar();

*assertEquals*(valorEsperado, resultado);

}

Dentro de CsvSource se indican los valores se pasarán al método y se probarán. Esto se tiene que hacer por cada método que queramos probar.

**TEST NG**

Se importan los paquetes de TestNG:

import static org.testng.Assert.\*;

import org.testng.annotations.AfterClass;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.Test;

import org.testng.annotations.DataProvider;

Se crea un método que contenga los valores a probar, dándole un nombre a dicha colección de la siguiente manera:

@DataProvider(name = "suma")

public Object[][] createData1(){

return new Object[][]{{20, 20, 40}, {10, 20, 50}};

}

A continuación, al método que contenga la prueba le pasamos estos valores:

@Test(dataProvider = "suma")

public void testSumar(int a, int b, int esperado) {

Suma s = new Suma(a, b);

assertEquals(esperado, s.sumar());

}

DOCUMENTACIÓN

**Plan de pruebas**: Documento exhaustivo en donde se identifican los requisitos existentes, las pruebas que se realizarán y su resultado. Se indican los casos de uso, los valores usados, y el valor esperado para luego probarlos. Este documento se usará como guion para el equipo de pruebas.

Un plan de pruebas no es válido si algún requisito funcional no se prueba en ningún caso de prueba.

Trazabilidad de casos de pruebas: Se indican los distintos requisitos funcionales y los casos de pruebas que cubren dichos requisitos.

Estrategia de ejecución de planes: Se muestra qué casos de pruebas dan fallos y los distintos ciclos que se han tomado para ir arreglando cada fallo necesario.

**Informe de pruebas**: Similar al anterior, pero se centra en los resultados propuestos en el plan de pruebas.

**Manuales de sistema, instalación, usuarios, etc…**