Guía Laboratorio de Ingeniería del Software 2

Guía 02 - Pruebas Unitarias

Prof. Hernan Nina Hanco

OBJETIVO:

- Conocer el desarrollo de pruebas unitarias paramétricas utilizando el framework Junit4 en Java.

TRABAJO PREPARATORIO:

Pruebas Paramétricas Unitarias (Pruebas Parametrizadas):

Las pruebas paramétricas unitarias en Java, también conocidas como pruebas parametrizadas, son una técnica de pruebas que te permite ejecutar la misma prueba con diferentes conjuntos de datos de entrada o parámetros. Esta técnica es especialmente útil cuando deseas probar un método o función con una variedad de casos de prueba sin tener que escribir pruebas individuales para cada uno de ellos. Las pruebas paramétricas unitarias son una característica que se encuentra en varios marcos de pruebas unitarias, incluido JUnit.

DESARROLLO DE LA GUÍA:

Paso 1: Configuración del entorno

Toma en cuenta el proyecto de la Guía 1, sobre prueba de una Calculadora.java.

Paso 2: Crear una clase de prueba paramétrica

Crea una nueva clase de prueba paramétrica en tu proyecto. Puedes nombrarla, por ejemplo, CalculadoraParametricaTest.

```
import java.util.ArrayList;
import java.util.List;
import junit.framework.Assert;
import org.junit.Test;
import static org.junit.Assert.*;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;

@RunWith(value = Parameterized.class)
public class CalculadoraParametricaTest {

    @Parameters
    public static Iterable<Object[]> getData(){
        List<Object[]> obj = new ArrayList();
        obj.add(new Object[]{2,3,5});
        obj.add(new Object[]{0,0,0});
        obj.add(new Object[]{-2,-8,-10});
        obj.add(new Object[]{-2,8,6});
        obj.add(new Object[]{2,-8,-6});
        return obj;
    }

    private int a, b, esp;
```

```
    public CalculadoraParametricaTest(int a, int b, int esp){
        this.a = a;
        this.b = b;
        this.esp = esp;
    }

    @Test
    public void sumaTest(){
        Calculadora cal = new Calculadora();
        int resultado = cal.add(a, b);
        int esperado = esp;
        Assert.assertEquals(esperado, resultado);
    }

}
```

Paso 3: Anotar la clase con @RunWith(Parameterized.class)

La anotación @RunWith(Parameterized.class) indica que esta clase es una prueba paramétrica que utilizará los datos proporcionados en el método getData().

Paso 4: Crear el método getData() con los conjuntos de datos

El método getData() es un método estático que proporciona los conjuntos de datos para la prueba. Cada conjunto de datos se representa como un arreglo de objetos, y cada objeto en el arreglo corresponde a un parámetro utilizado en la prueba. En este caso, estamos probando la suma de dos números, por lo que cada conjunto de datos contiene tres valores: a, b y el resultado esperado esp.

Paso 5: Crear un constructor en la clase de prueba

Debes crear un constructor en la clase de prueba que tome los mismos parámetros que se utilizan en los conjuntos de datos. En este caso, el constructor toma tres enteros: a, b y esp.

Paso 6: Escribir el método de prueba

Escribe un método de prueba (en este caso, sumaTest()) que utilice los parámetros a y b para realizar una suma utilizando la clase Calculadora. Luego, compara el resultado con el valor esperado esp utilizando Assert.assertEquals().

Paso 7: Ejecutar la prueba paramétrica

Ejecuta la prueba paramétrica desde tu entorno de desarrollo. Verás que la prueba se ejecuta varias veces, una vez por cada conjunto de datos proporcionado en getData(). La prueba se completará exitosamente si todas las comparaciones assertEquals son verdaderas para cada conjunto de datos.

Con estos pasos, habrás aprendido a realizar pruebas paramétricas en JUnit, lo que te permite probar una función o método con múltiples conjuntos de datos de manera más eficiente y reutilizable.

EJERCICIOS PROPUESTOS

Ejercicio 1: Pruebas de Multiplicación Paramétricas

Escribe pruebas paramétricas para el método de multiplicación de la calculadora. Utiliza diferentes combinaciones de números para verificar que la multiplicación funcione correctamente. Asegúrate de probar la multiplicación por cero y números negativos.

Ejercicio 2: Pruebas de División Paramétricas

Crea pruebas paramétricas para el método de división de la calculadora. Genera conjuntos de datos que incluyan casos de división exacta y no exacta, así como casos donde la división por cero deba manejar excepciones adecuadamente.

```
@RunWith(value = Parameterized.class)
public class CalculadoraParametricaTest {

    @Parameters
    public static Iterable<Object[]> getData() {
        List<Object[]> obj = new ArrayList();
        obj.add(new Object[] { 6, 0 }); // División por cero,
debe lanzar una excepción
        obj.add(new Object[] { 10, 2 }); // Prueba sin excepción
        // Agrega más conjuntos de datos paramétricos aquí
        return obj;
    }

    private int a, b;
    public CalculadoraParametricaTest(int a, int b) {
        this.a = a;
        this.b = b;
    }

    @Test
    public void divisionTest() {
        Calculadora cal = new Calculadora();
        try {
            int resultado = cal.dividir(a, b);
            // En este punto, si no se lanzó una excepción, la
prueba fallará
            fail("Se esperaba una excepción
ArithmeticException.");
        } catch (ArithmeticException e) {
            // La prueba es exitosa si se lanza una excepción
ArithmeticException
            assertTrue(b == 0);
        }
    }
}
```

Ejercicio 3: Pruebas Paramétricas Avanzadas

Desarrolla pruebas paramétricas que combinan múltiples operaciones en una secuencia. Por ejemplo, puedes probar una secuencia de operaciones como (a + b) * c - d / e, donde a, b, c, d y e son valores parametrizados. Verifica que el resultado de la secuencia sea el esperado.