Entornos de Desarrollo Unidad 3 - Caso 1

Introducción

En el presente trabajo se desarrollará la implementación de pruebas unitarias con JUnit en Eclipse para validar el correcto funcionamiento de operaciones matemáticas básicas. Las pruebas unitarias son esenciales en el desarrollo de software, ya que permiten detectar errores de manera temprana, asegurando la fiabilidad del código. Siguiendo las instrucciones de la tarea, se programarán tres clases: **Resta, Multiplicacion** y **Division**, cada una con atributos y métodos que permitan la realización de las respectivas operaciones matemáticas. Posteriormente, se definirán y ejecutarán pruebas unitarias sobre dichas clases, aplicando las mejores prácticas en la ejecución de pruebas automatizadas.

Desarrollo

1. Implementación de Clases

Cada clase contará con dos variables enteras (**num1** y **num2**), un constructor y un método que realice la operación correspondiente. Se ha seguido una estructura modular para garantizar la reutilización del código y facilitar futuras ampliaciones.

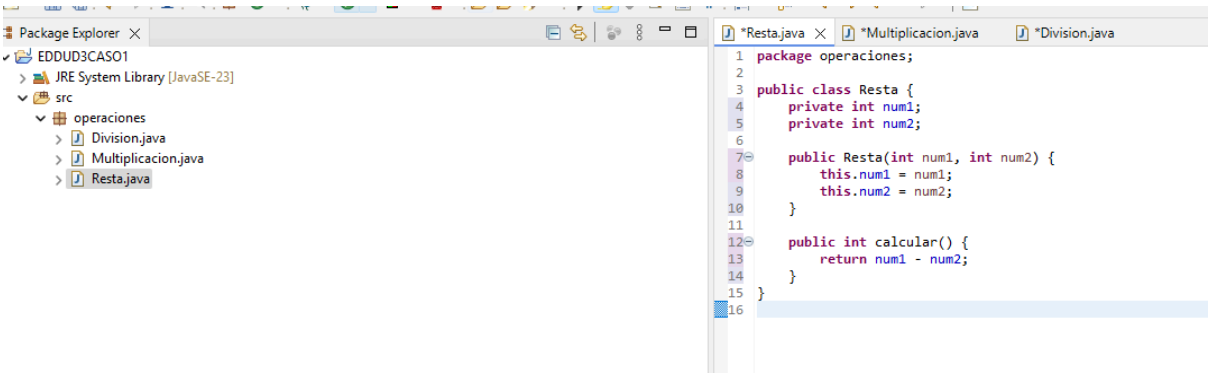
1.1 Clase Resta

Esta clase implementa una operación de resta simple, en la que se recibe dos números enteros como parámetros y se devuelve la diferencia entre ellos.

```
public class Resta {
    private int num1;
    private int num2;

    public Resta(int num1, int num2) {
        this.num1 = num1;
        this.num2 = num2;
    }

    public int calcular() {
        return num1 - num2;
    }
}
```



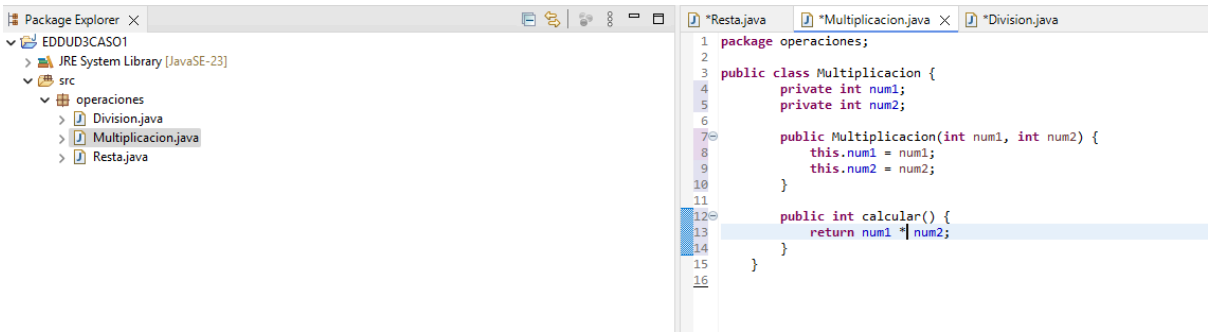
1.2 Clase Multiplicacion

La clase **Multiplicacion** se encarga de realizar la operación de producto entre dos valores enteros, retornando el resultado como salida.

```
public class Multiplicacion {
    private int num1;
    private int num2;

    public Multiplicacion(int num1, int num2) {
        this.num1 = num1;
        this.num2 = num2;
    }

    public int calcular() {
        return num1 * num2;
    }
}
```



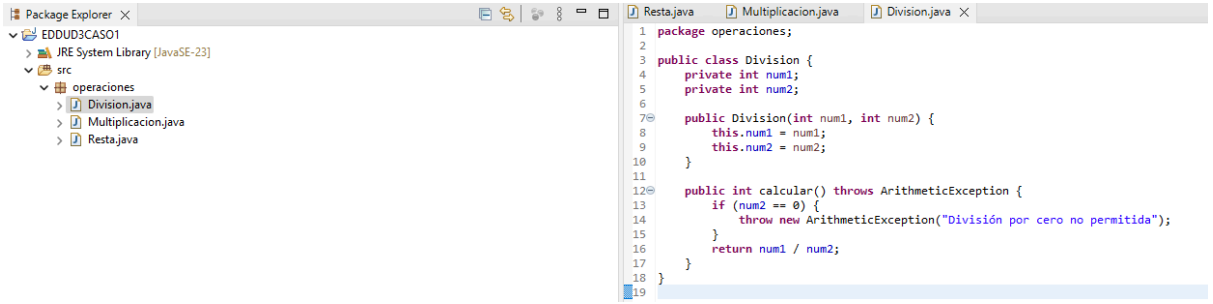
1.3 Clase Division

La división es una operación que puede generar errores si el divisor es cero. Por ello, en esta clase se implementa una validación para evitar la división por cero, lanzando una excepción cuando esto ocurre.

```
public class Division {
    private int num1;
    private int num2;

    public Division(int num1, int num2) {
        this.num1 = num1;
        this.num2 = num2;
    }

    public int calcular() throws ArithmeticException {
        if (num2 == 0) {
            throw new ArithmeticException("División por cero no permitida");
        }
        return num1 / num2;
    }
}
```



2. Pruebas Unitarias con JUnit

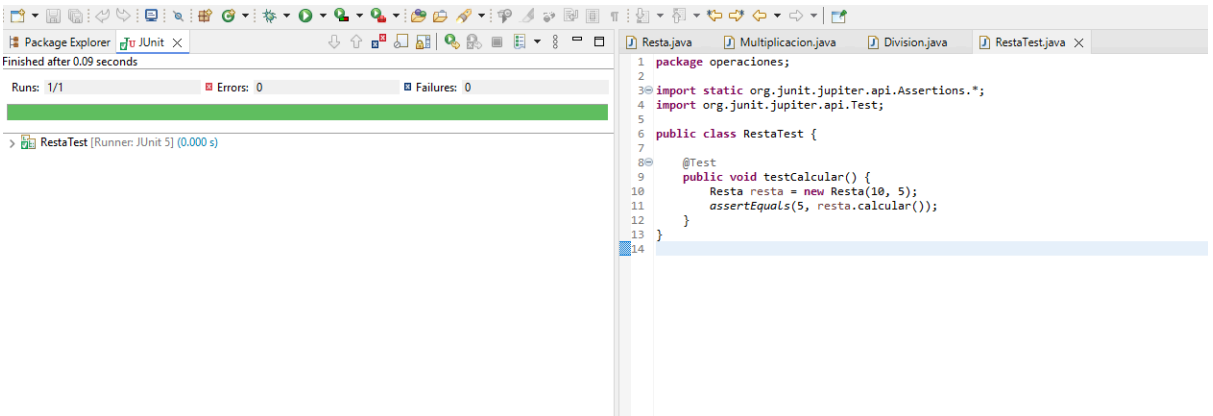
Para validar el funcionamiento de cada operación, se realizarán pruebas unitarias utilizando JUnit. Estas pruebas permiten verificar que los métodos implementados funcionan correctamente bajo diferentes condiciones, ayudando a mejorar la robustez del código.

2.1 Prueba para la Clase Resta

En esta prueba se verifica que la resta entre dos números devuelve el valor esperado. Se compara la salida del método **calcular()** con el resultado esperado utilizando **assertEquals**.

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class RestaTest {
    @Test
    public void testCalcular() {
        Resta resta = new Resta(10, 5);
        assertEquals(5, resta.calcular());
    }
}
```

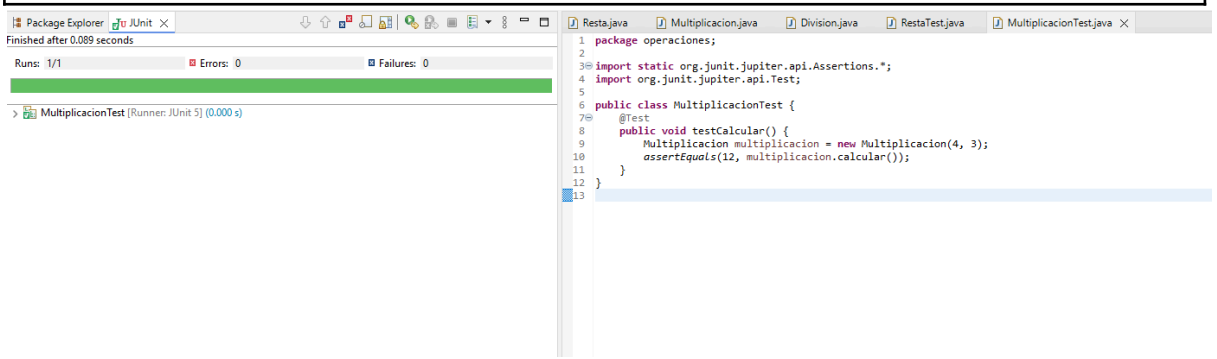


2.2 Prueba para la Clase Multiplicacion

Esta prueba valida que la multiplicación entre dos números retorne el valor esperado, comprobándolo con **assertEquals**.

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class MultiplicacionTest {
    @Test
    public void testCalcular() {
        Multiplicacion multiplicacion = new Multiplicacion(4, 3);
        assertEquals(12, multiplicacion.calcular());
    }
}
```

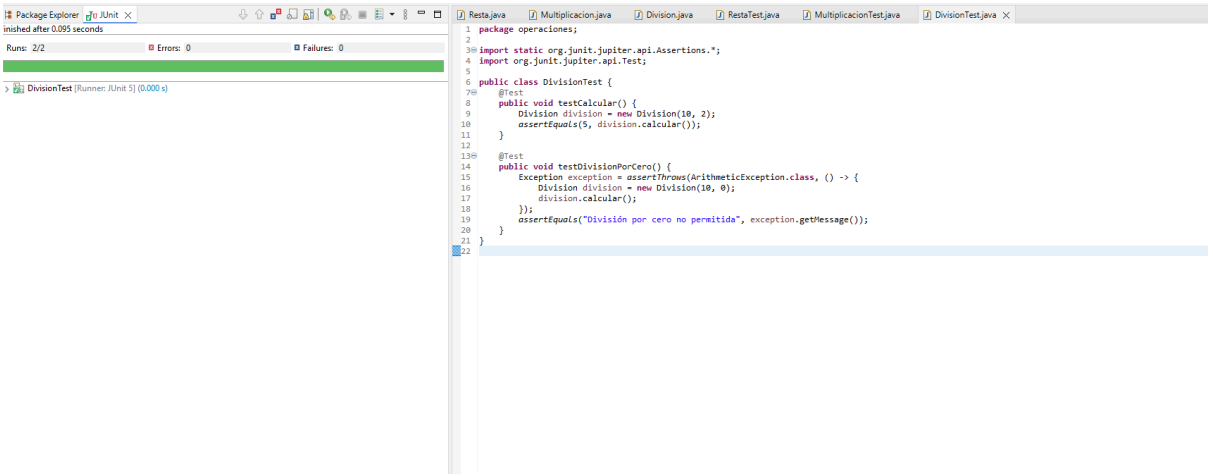


2.3 Prueba para la Clase Division

Las pruebas para la clase **Division** incluyen la validación de una operación de división normal y la gestión de excepciones en el caso de una división por cero.

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class DivisionTest {
    @Test
    public void testCalcular() {
        Division division = new Division(10, 2);
        assertEquals(5, division.calcular());
    }
    @Test
    public void testDivisionPorCero() {
        Exception exception = assertThrows(ArithmeticException.class, () -> {
            Division division = new Division(10, 0);
            division.calcular();
        });
        assertEquals("División por cero no permitida", exception.getMessage());
    }
}
```



Conclusiones

El desarrollo de pruebas unitarias es fundamental para garantizar la calidad del software. En este trabajo, se ha implementado correctamente la estructura de clases y pruebas unitarias con JUnit en Eclipse, verificando el correcto funcionamiento de operaciones matemáticas básicas. Se ha podido comprobar que cada operación matemática devuelve los resultados esperados, lo que permite validar su correcto funcionamiento antes de integrarlas en un sistema más complejo.

Además, se ha manejado adecuadamente el caso de división por cero, demostrando la importancia de capturar excepciones en la programación para evitar errores inesperados en tiempo de ejecución. Esta práctica mejora la fiabilidad del software y previene fallos en la aplicación.

Bibliografía

- https://youtu.be/74sCIDEYSQ4?si=F0pvYDs7buwDAS_y
- https://youtu.be/XlkMN_i8j5w?si=Sr2JZgeX5mmfQX7B
- Guía de contenidos de la unidad.