**TDD using JUnit5 and Mockito**

**Exercise 1: Setting Up Junit**

//pom.xml - Added JUnit Dependency

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

//Java class Calculator.java file

package com.example;

public class Calculator {

// Method to add two numbers a and b

public int add(int a, int b) {

return a + b;

}

}

//JUnit Test Class CalculatorTest.java file

package com.example;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

int result = calc.add(2, 3);

assertEquals(5, result); // Expected: 5

}

}

**OUTPUT:**

-------------------------------------------------------

T E S T S

-------------------------------------------------------

Running com.example.CalculatorTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.004 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

**Exercise 3: Assertions in Junit**

**CODE:**

// AssertionsTest.java Class

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.Test;

public class AssertionsTest {

@Test

public void testAssertions() {

// Assert that the sum of 4 and 6 is 10

assertEquals(10, 4 + 6, "Sum should be 10");

// Assert that 8 is greater than 2

assertTrue(8 > 2, "8 should be greater than 2");

// Assert that 2 is not greater than 9

assertFalse(2 > 9, "2 should not be greater than 9");

// Assert that a null reference is null

String message = null;

assertNull(message, "Message should be null");

// Assert that a string is not null

String name = "JUnit";

assertNotNull(name, "Name should not be null");

}

}

**OUTPUT:**

-------------------------------------------------------

T E S T S

-------------------------------------------------------

Running AssertionsTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.004 sec

Results :

Tests run: 1, Failures: 0

**Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in Junit**

**CODE:**

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.\*;

public class CalculatorTest {

// Calculator logic implemented as an inner static class

static class Calculator {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

public boolean isEven(int number) {

return number % 2 == 0;

}

}

private Calculator calculator;

@BeforeEach

public void setUp() {

// Setup: Initialize resources before each test

calculator = new Calculator();

System.out.println("Setup complete.");

}

@AfterEach

public void tearDown() {

// Teardown: Clean up after each test

calculator = null;

System.out.println("Teardown complete.");

}

@Test

public void testAddition() {

// Arrange

int a = 5;

int b = 7;

// Act

int result = calculator.add(a, b);

// Assert

assertEquals(12, result, "Sum should be 12");

}

@Test

public void testMultiplication() {

// Arrange

int x = 4;

int y = 6;

// Act

int result = calculator.multiply(x, y);

// Assert

assertEquals(24, result, "Product should be 24");

}

@Test

public void testIsEvenWithEvenNumber() {

// Arrange

int number = 8;

// Act

boolean result = calculator.isEven(number);

// Assert

assertTrue(result, "8 should be even");

}

@Test

public void testIsEvenWithOddNumber() {

// Arrange

int number = 5;

// Act

boolean result = calculator.isEven(number);

// Assert

assertFalse(result, "5 should be odd");

}

}

**OUTPUT:**

✔ testAddition() — Passed

✔ testMultiplication() — Passed

✔ testIsEvenWithEvenNumber() — Passed

✔ testIsEvenWithOddNumber() — Passed

Total tests run: 4

All tests passed successfully.

Runs: 4/4 Errors: 0 Failures: 0

**Mockito exercises**

**Exercise 1: Mocking and Stubbing**

Scenario: You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.

**CODE:**

import static org.junit.jupiter.api.Assertions.\*;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

// Define an external API interface

interface ExternalApi {

String getData();

}

// Create a service that depends on the external API

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

// Test the service using Mockito

public class MyServiceTest {

@Test

public void testExternalApiUsingMock() {

ExternalApi mockApi = mock(ExternalApi.class);

when(mockApi.getData()).thenReturn("Mock Data");

MyService service = new MyService(mockApi);

String result = service.fetchData();

// Result

assertEquals("Mock Data", result);

System.out.println("Test Passed: " + result);

}

}

**OUTPUT:**

Test Passed: Mock Data

**Exercise 2: Verifying Interactions**

Scenario: You need to ensure that a method is called with specific arguments

**CODE:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

// External API interface

interface ExternalApi {

String getData(String query);

}

// Service class that uses ExternalApi

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void fetchData() {

api.getData("user123");

}

}

// Unit test

public class MyServiceTest {

@Test

public void testVerifyInteractionWithArgument() {

// Create mock

ExternalApi mockApi = mock(ExternalApi.class);

// Call method

MyService service = new MyService(mockApi);

service.fetchData();

// Verify interaction with specific argument

verify(mockApi).getData("user123");

}

}

**OUTPUT :**

✔ testVerifyInteractionWithArgument() passed