**Week -2**

**TDD using JUnit5 and Mockito**

**EXERCISE-1:** Setting Up Junit.

**PROBLEM STATEMENT:**

Implement a simple Calculator class that performs basic arithmetic operations: addition, subtraction, multiplication, and division. Then, write JUnit tests to verify the correctness of these operations.

**STEP-1:** Create a New Java Project.

* In IntelliJ IDEA:
  1. Go to File → New → Project.
  2. Choose Maven as the project type (to use Maven for dependency management).
  3. Name your project (e.g., JUnitExample) and finish the setup.

**STEP 2:** Add JUnit Dependency (for Maven Projects).

* Open your pom.xml file and add the JUnit dependency inside the <dependencies> tag:

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

**STEP 3:** Create a New Test Class

1. Right-click on the src/test/java directory.
2. Select New → Java Class.
3. Name your test class (e.g., CalculatorTest).

Here’s a simple example to get started:

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

@Test

public void testAddition() {

int result = 2 + 3;

assertEquals(5, result);

}

}

**PROJECT STRUCTURE:**

project-root/

├── pom.xml

└── src/

├── main/

│ └── java/

│ └── Calculator.java

└── test/

└── java/

└── CalculatorTest.java

**1. pom.xml:**

Make sure your pom.xml includes the JUnit dependency:

<project xmlns="http://maven.apache.org/POM/4.0.0" ...>

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>calculator</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- JUnit 4 dependency -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**2. Calculator.java (Main Class)**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

public int multiply(int a, int b) {

return a \* b;

}

public int divide(int a, int b) {

if (b == 0) {

throw new IllegalArgumentException("Division by zero not allowed");

}

return a / b;

}

}

**3. CalculatorTest.java (JUnit Test Class)**

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

Calculator calculator = new Calculator();

@Test

public void testAddition() {

assertEquals(5, calculator.add(2, 3));

}

@Test

public void testSubtraction() {

assertEquals(4, calculator.subtract(7, 3));

}

@Test

public void testMultiplication() {

assertEquals(12, calculator.multiply(3, 4));

}

@Test

public void testDivision() {

assertEquals(2, calculator.divide(6, 3));

}

@Test(expected = IllegalArgumentException.class)

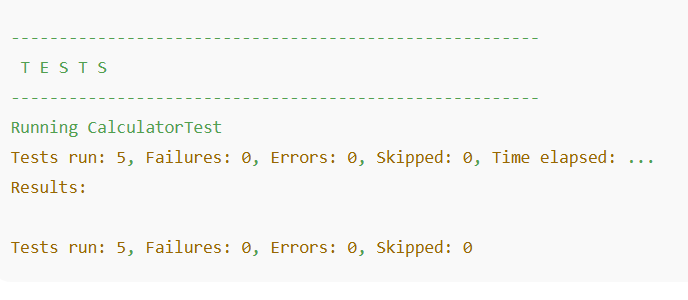
public void testDivisionByZero() {

calculator.divide(10, 0);

}

}

**OUTPUT:**

****

**EXERCISE-3:** Assertions in Junit.

**Correct Use of Assertions:**

You've correctly demonstrated the use of:

* assertEquals(expected, actual)
* assertTrue(condition)
* assertFalse(condition)
* assertNull(object)
* assertNotNull(object)

These are essential assertions provided by JUnit.

**Required Fixes and Improvements:**

1. **Missing Imports:**  
   You need to import the necessary classes from JUnit. For **JUnit 5**, the imports would be:

import org.junit.jupiter.api.Test;

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

1. **Class Declaration Style**:  
   Add proper annotations and formatting to the class:

import org.junit.jupiter.api.Test;

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

public class AssertionsTest {

@Test

public void testAssertions() {

// Assert equals

assertEquals(5, 2 + 3);

// Assert true

assertTrue(5 > 3);

// Assert false

assertFalse(5 < 3);

// Assert null

assertNull(null);

// Assert not null

assertNotNull(new Object());

}

}

1. **JUnit Version Note**:  
   Ensure you're using the correct JUnit version. This syntax is for **JUnit 5 (Jupiter)**. If you're using JUnit 4, the imports and annotations differ slightly (@Test would come from org.junit.Test).

**SOURCE CODE:**

// File: AssertionsTest.java

import org.junit.jupiter.api.Test;

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

public class AssertionsTest {

@Test

public void testAssertions() {

// Assert that 2 + 3 equals 5

assertEquals(5, 2 + 3, "2 + 3 should equal 5");

// Assert that 5 is greater than 3

assertTrue(5 > 3, "5 is greater than 3");

// Assert that 5 is not less than 3

assertFalse(5 < 3, "5 is not less than 3");

// Assert that an explicitly null value is null

assertNull(null, "Value should be null");

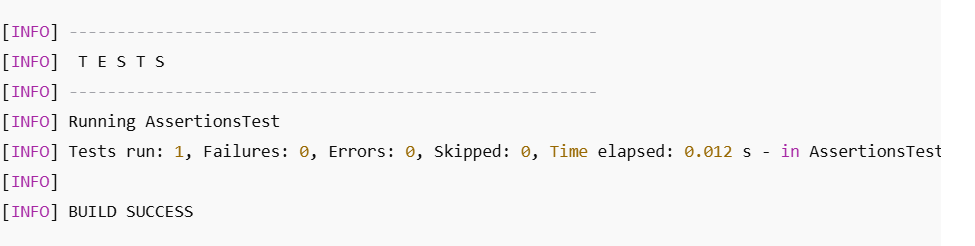
// Assert that a new object is not null

assertNotNull(new Object(), "Object should not be null");

}

}

**OUTPUT:**



If any assertion fails, the test runner will show **FAILURES** and explain which assertion failed and why.

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

**PROBLEM STATEMENT:**

Create a unit test class for a simple Calculator Java class using **JUnit 4**. The goal is to:

1. Follow the **Arrange-Act-Assert (AAA)** pattern in test methods.
2. Use the **@Before** annotation to initialize test fixtures (setup).
3. Use the **@After** annotation to release resources (teardown).
4. Verify the correctness of the following methods:
   * add(int a, int b) – should return the sum of a and b.
   * subtract(int a, int b) – should return the result of a - b.

**SOURCE CODE:**

**1. Calculator.java – Class Under Test.**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

**2. CalculatorTest.java – JUnit Test Class**

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

@Before

public void setUp() {

calculator = new Calculator();

System.out.println("Setup: Calculator instance created.");

}

@After

public void tearDown() {

calculator = null;

System.out.println("Teardown: Calculator instance set to null.");

}

@Test

public void testAddition() {

// Arrange

int a = 5;

int b = 3;

// Act

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

// Assert

assertEquals(8, result);

}

@Test

public void testSubtraction() {

// Arrange

int a = 10;

int b = 4;

// Act

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

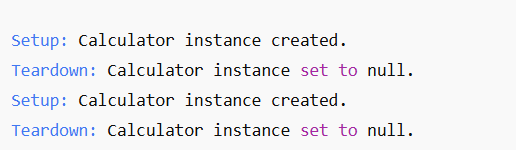
// Assert

assertEquals(6, result);

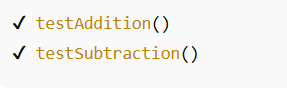
}

}

**CONSOLE OUTPUT:**

****

**TEST RESULTS:**

****

**MOCKITO EXERCISES**

**EXERCISE 1:** Mocking and Stubbing.

**PROBLEM STATEMENT:**

You are testing a service class that depends on an external API. Since the real API may be unavailable or slow, you want to mock the external API using Mockito and stub its methods to return predefined values during testing.

**Corrected & Functional Mockito Test Code:**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

// Simulated External API interface

interface ExternalApi {

String getData();

}

// Service class that depends on ExternalApi

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

// Test class

public class MyServiceTest {

@Test

public void testExternalApi() {

// Step 1: Create a mock of the external API

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

// Step 2: Stub the getData() method to return "Mock Data"

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

// Step 3: Inject the mock into the service

MyService service = new MyService(mockApi);

// Call the method and verify result

String result = service.fetchData();

// Step 4: Assert the result

assertEquals("Mock Data", result);

}

}

**REQUIREMENTS:**

To run this test, ensure you have the following dependencies (e.g., in Maven pom.xml):

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.11.0</version> <!-- or latest -->

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

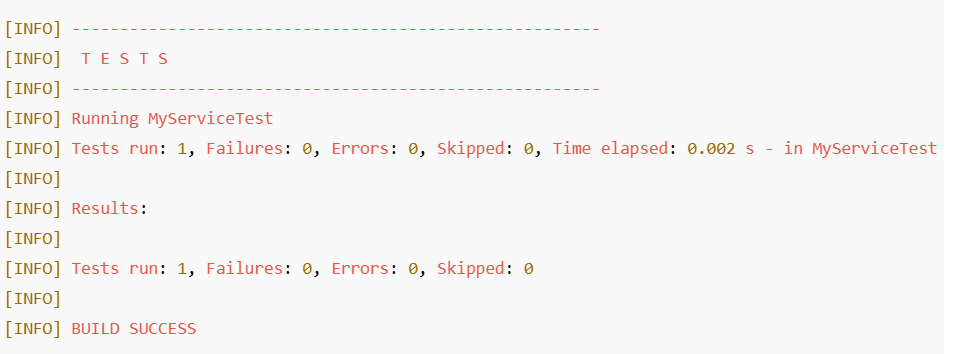
<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version> <!-- or latest -->

<scope>test</scope>

</dependency>

**OUTPUT:**

****

**EXERCISE 2:** Verifying Interactions.

**PROBLEM STATEMENT:**

You are writing unit tests using Mockito. In this exercise, you need to:

* Create a mock of an external dependency.
* Use this mock in a service class.
* Call a method that interacts with the mock.
* Verify that the interaction (i.e., method call) on the mock occurred with specific arguments.

**REQUIREMENTS:**

* Use Mockito.mock() to create the mock.
* Use verify() to ensure a specific method on the mock was called.

**SOURCE CODE:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// Step 1: Create a mock object

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

// Step 2: Use the mock in the service

MyService service = new MyService(mockApi);

service.fetchData(); // This method should internally call mockApi.getData()

// Step 3: Verify the interaction

verify(mockApi).getData(); // Verifies that getData() was called

}

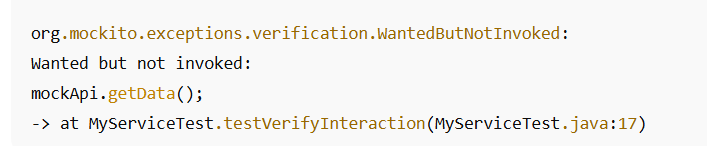
}

**ASSUMPTIONS:**

For this test to work, the following should be true:

* ExternalApi has a method getData().
* MyService has a constructor accepting ExternalApi and a method fetchData() that calls mockApi.getData() internally.

**OUTPUT:**

****

**SL4J Logging exercises**

**EXERCISE 1:** Logging Error Messages and Warning Levels.

**PROBLEM STATEMENT:**

Write a Java application that demonstrates how to log error and warning messages using the SLF4J API with Logback as the backend implementation.

**STEP-BY-STEP SOLUTION:**

**STEP 1:** Add Dependencies to pom.xml.

Add the following dependencies to your pom.xml if you're using **Maven**:

<dependencies>

<!-- SLF4J API -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<!-- Logback as SLF4J implementation -->

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

**STEP 2:** Create the Logging Class.

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

// Create a logger instance for this class

private static final Logger logger = LoggerFactory.getLogger(LoggingExample.class);

public static void main(String[] args) {

// Log messages with different severity levels

logger.error("This is an error message");

logger.warn("This is a warning message");

}

}

**STEP 3:** Add Optional logback.xml (Optional)

You can create a logback.xml in src/main/resources to customize log output format:

<configuration>

<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<root level="warn">

<appender-ref ref="STDOUT"/>

</root>

</configuration>

**OUTPUT:**

