**Module -4**

**Topic:-TDD using JUnit5 and Mockito**

**Exercise 1: Setting Up JUnit**

Scenario: You need to set up JUnit in your Java project to start writing unit tests. Steps: 1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).

2. Add JUnit dependency to your project. If you are using Maven, add the following to your pom.xml:

3. Create a new test class in your project.

**Solution:**

**Code:**

**Calculator.java**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorTest.java**

package com.example;

import static org.junit.Assert.*assertEquals*;

import org.junit.Test;

public class CalculatorTest {

@Test

public void testAdd() {

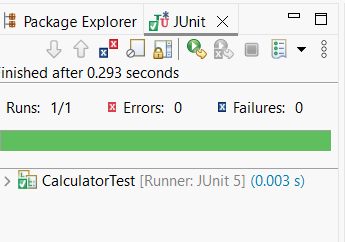
Calculator calc = new Calculator();

int result = calc.add(5, 7);

*assertEquals*(12, result);

}

}

**OUTPUT:-**

**Exercise 3: Assertions in JUnit Scenario:** You need to use different assertions in JUnit to validate your test results.

Steps: 1. Write tests using various JUnit assertions.

**Solution:**

**Code:**

package com.example;

import static org.junit.Assert.\*;

import org.junit.Test;

public class AssertionsTest {

@Test

public void testAssertions() {

*assertEquals*(5, 2 + 3);

*assertTrue*(5 > 3);

*assertFalse*(5 < 3);

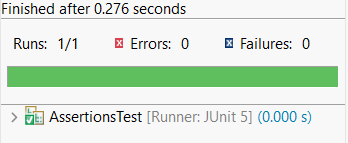
*assertNull*(null);

*assertNotNull*(new Object());

}

}

**OUTPUT:-**

****

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

Scenario: You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.

Steps:

1. Write tests using the AAA pattern.

2. Use @Before and @After annotations for setup and teardown methods.

**Solution:**

**Code:**

**Calculator.java**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

}

**CalculatorTest.java**

package com.example;

import static org.junit.Assert.\*;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

public class CalculatorTest {

private Calculator calc;

@Before

public void setUp() {

calc = new Calculator();

System.*out*.println("Setup completed.");

}

@After

public void tearDown() {

calc = null;

System.*out*.println("Teardown completed.");

}

@Test

public void testAdd() {

int result = calc.add(10, 20);

*assertEquals*(30, result);

}

@Test

public void testMultiply() {

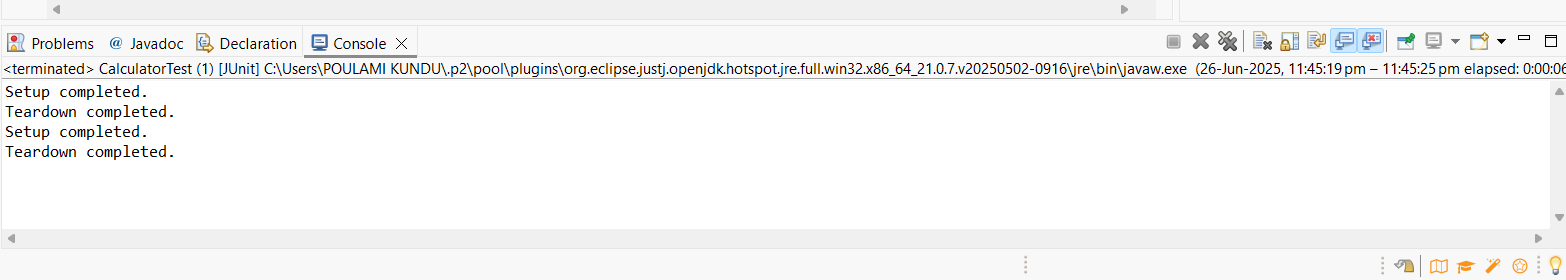
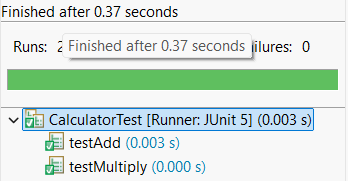
int result = calc.multiply(5, 4);

*assertEquals*(20, result);

}

}

**OUTPUT:-**

****

**Topic:-Mockito Hands-On 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.

Steps:

1. Create a mock object for the external API.

2. Stub the methods to return predefined values.

3. Write a test case that uses the mock object.

**Solution:**

**Code:**

**pom.xml**

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

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example.mocktest</groupId>

<artifactId>mockito-demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>mockito-demo</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

</properties>

<dependencies>

<dependency>

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

<artifactId>junit-jupiter-api</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.12.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.1.2</version>

<configuration>

<useModulePath>false</useModulePath>

</configuration>

</plugin>

</plugins>

</build>

</project>

**ExternalApi.java**

package com.example.mocktest.mockito\_demo;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example.mocktest.mockito\_demo;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

package com.example.mocktest.mockito\_demo;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testExternalApi() {

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

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

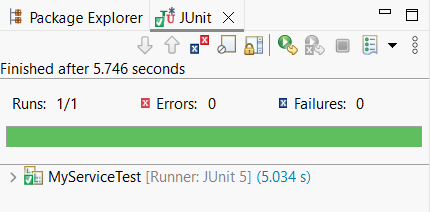
MyService service = new MyService(mockApi);

String result = service.fetchData();

*assertEquals*("Mock Data", result);

}

}

****

**OUTPUT:-**

**Exercise 2: Verifying Interactions**

Scenario:

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

Steps:

1. Create a mock object.

2. Call the method with specific arguments.

3. Verify the interaction.

**Solution:**

**Code:**

**pom.xml**

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

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>mockitotest</artifactId>

<version>0.0.1-SNAPSHOT</version>

<properties>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

</properties>

<dependencies>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.12.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.1.2</version>

<configuration>

<useModulePath>false</useModulePath>

</configuration>

</plugin>

</plugins>

</build>

</project>

**ExternalApi.java**

package com.example.mockitotest;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example.mockitotest;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void fetchData() {

api.getData();

}

}

**MyServiceTest.java**

package com.example.mockitotest;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

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

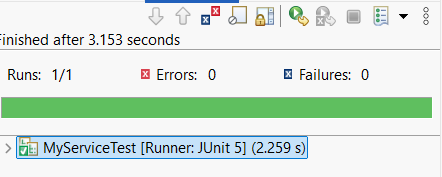
MyService service = new MyService(mockApi);

service.fetchData();

*verify*(mockApi).getData();

}

}

****

**Topic:-SLF4J logging framework**

**Exercise 1: Logging Error Messages and Warning Levels Task:** Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

Step-by-Step

Solution:

1. Add SLF4J and Logback dependencies to your `pom.xml` file

2. Create a Java class that uses SLF4J for logging

**Solution:**

**Code:**

**Maven Dependencies**

**pom.xml**

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

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>LoggingDemo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>LoggingDemo</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<dependency>

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

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

</project>

**LoggingExample.java**

package com.example;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

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

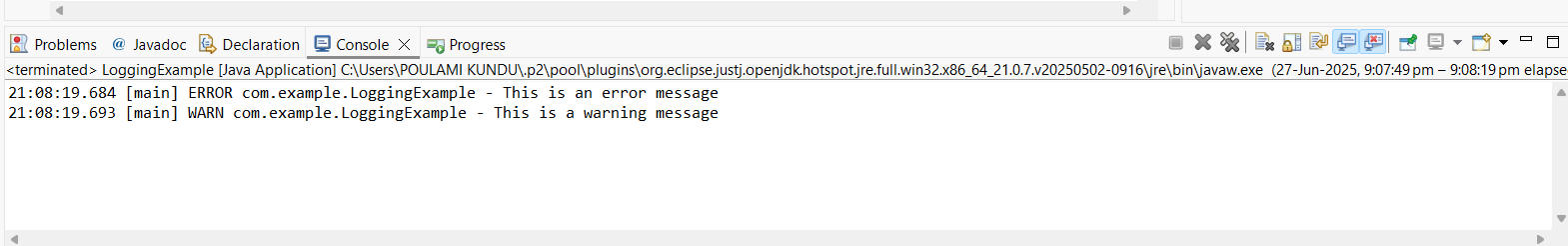
public static void main(String[] args) {

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

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

}

}

**OUTPUT:**