**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**

<?xml version="1.0" encoding="UTF-8"?>

<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>org.example</groupId>

<artifactId>Exercise\_1</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>22</maven.compiler.source>

<maven.compiler.target>22</maven.compiler.target>

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

</properties>

<dependencies>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**src/main/java/org/example/Calculator.java**

package org.example;

public class Calculator {

public int add(int a, int b){

return a+b;

}

}

**src/test/java/CalculatorTest.java**

import org.example.Calculator;

import org.junit.jupiter.api.Test;

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

public class CalculatorTest {

@Test

void testAdd() {

Calculator calc = new Calculator();

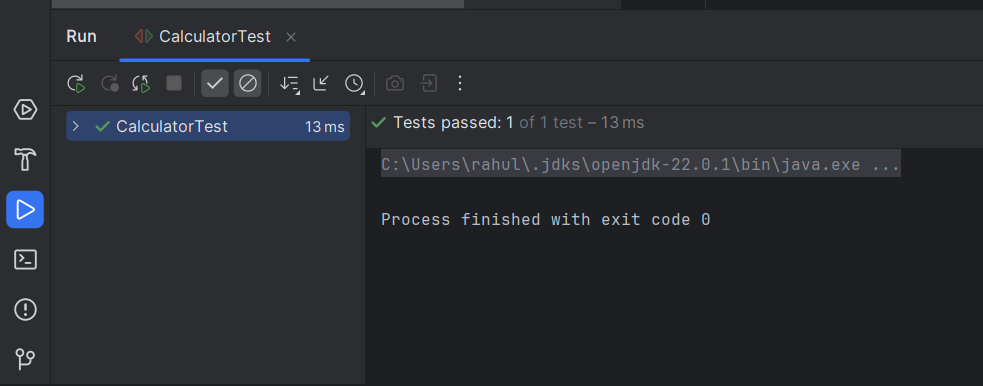
int result = calc.add(2, 3);

*assertEquals*(5, result, "2 + 3 should be 5");

}

}

**Output:**

****

**Exercise 3: Assertions in JUnit**

Scenario: You need to use different assertions in JUnit to validate your test results.

**src/test/java/AssertionsTest.java**

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, "2 + 3 should equal 5");

// Assert true

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

// Assert false

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

// Assert null

Object obj1 = null;

*assertNull*(obj1, "Object should be null");

// Assert not null

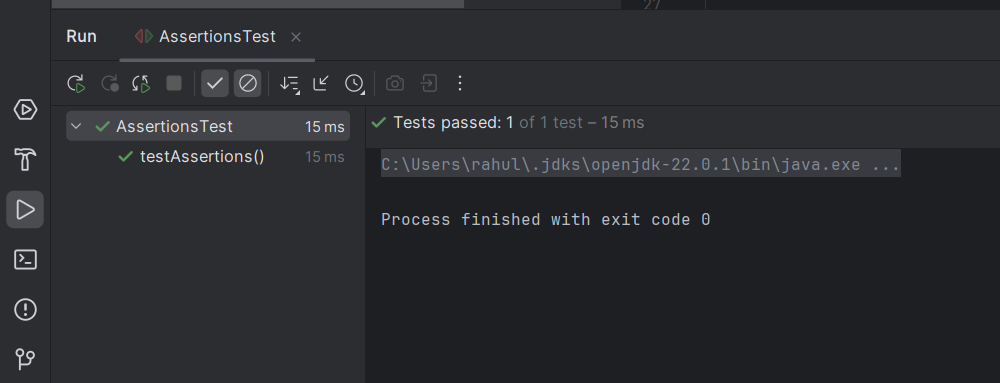
Object obj2 = new Object();

*assertNotNull*(obj2, "Object should not be null");

}

}

**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.

**src/main/java/org/example/Calculator.java**

package org.example;

public class Calculator {

public int add(int a, int b){

return a+b;

}

public void clearMemory() {

System.*out*.println("----Clearing memory----");

}

}

**src/test/java/CalculatorTest.java**

import org.example.Calculator;

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

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

public class CalculatorTest {

private Calculator calculator;

//runs before each test

@BeforeEach

public void setUp() {

calculator = new Calculator();

System.*out*.println("Creating new Calculator instance");

}

//runs after each test

@AfterEach

public void tearDown() {

calculator.clearMemory();

System.*out*.println("Calculator memory cleared");

}

@Test

public void testAddition() {

//Arrange

int a = 5;

int b = 3;

// Act

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

// Assert

*assertEquals*(8, result, "5 + 3 should equal 8");

}

@Test

public void testAdditionWithNegativeNumbers() {

// Arrange

int a = -4;

int b = -6;

// Act

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

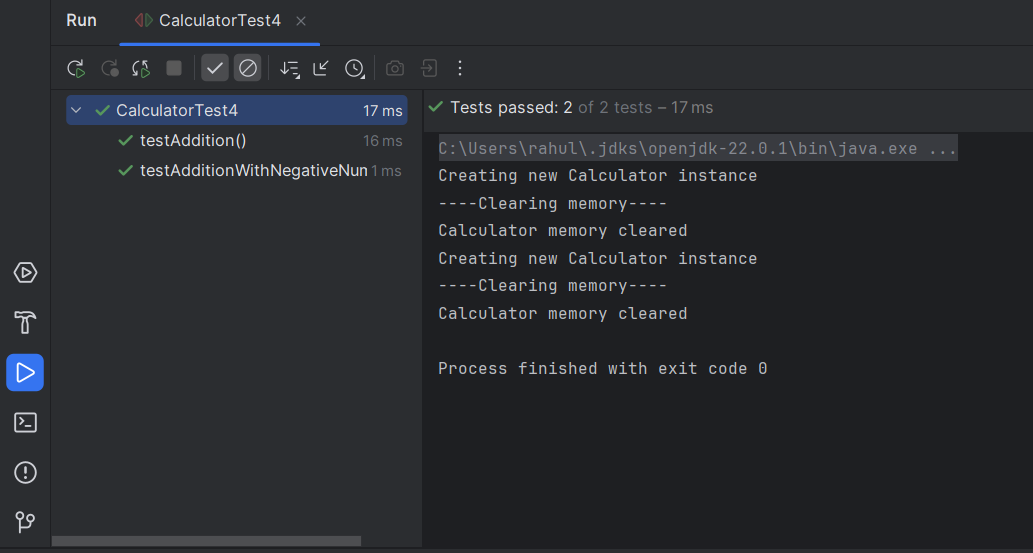
// Assert

*assertEquals*(-10, result, "-4 + (-6) should equal -10");

}

}

**Output:**



**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.

**src/main/java/org/example/ExternalApi.java**

package org.example;

public interface ExternalApi {

String getData();

}

**src/main/java/org/example/MyService.java**

package org.example;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**src/test/java/MyServiceTest.java**

import org.example.ExternalApi;

import org.example.MyService;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testExternalApi() {

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

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

MyService service = new MyService(mockApi);

String result = service.fetchData();

System.*out*.println("Fetched Data: " + result);

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

}

}

**pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<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>org.example</groupId>

<artifactId>4\_Exercise\_1</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>22</maven.compiler.source>

<maven.compiler.target>22</maven.compiler.target>

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

</properties>

<dependencies>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.12.0</version>

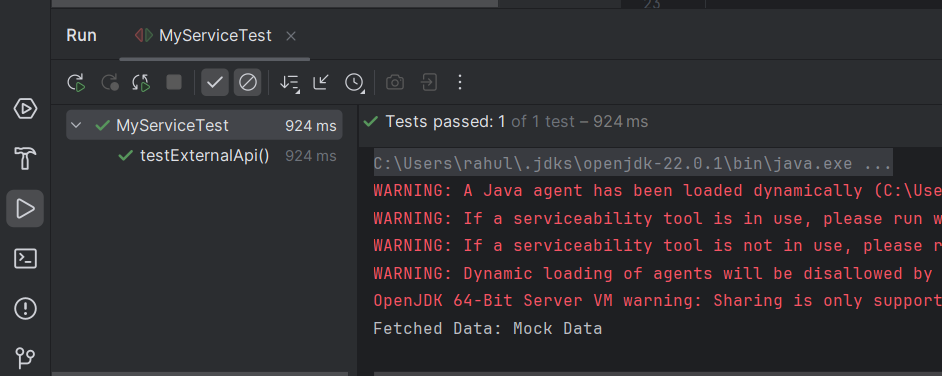
<scope>test</scope>

</dependency>

</dependencies>

</project>

**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.

**src/main/java/org/example/ExternalApi.java**

package org.example;

public interface ExternalApi {

String getData();

}

**src/main/java/org/example/MyService.java**

package org.example;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**src/test/java/MyServiceTest.java**

import org.example.ExternalApi;

import org.example.MyService;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest2 {

@Test

public void testVerifyInteraction() {

//Create mock

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

//Inject into service and call method

MyService service = new MyService(mockApi);

service.fetchData();

//Verify that getData() was called once

*verify*(mockApi).getData();

}

}

**Output:**

