**Candidate Name**:- Amrutha H

**Superset ID**:-6421402

**WEEK – 2 HANDS ON EXERCISE (JAVA FSE DEEPSKILLING)**

**Module 4 – Test driven development and Logging framework**

**(JUnit Testing)**

**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:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

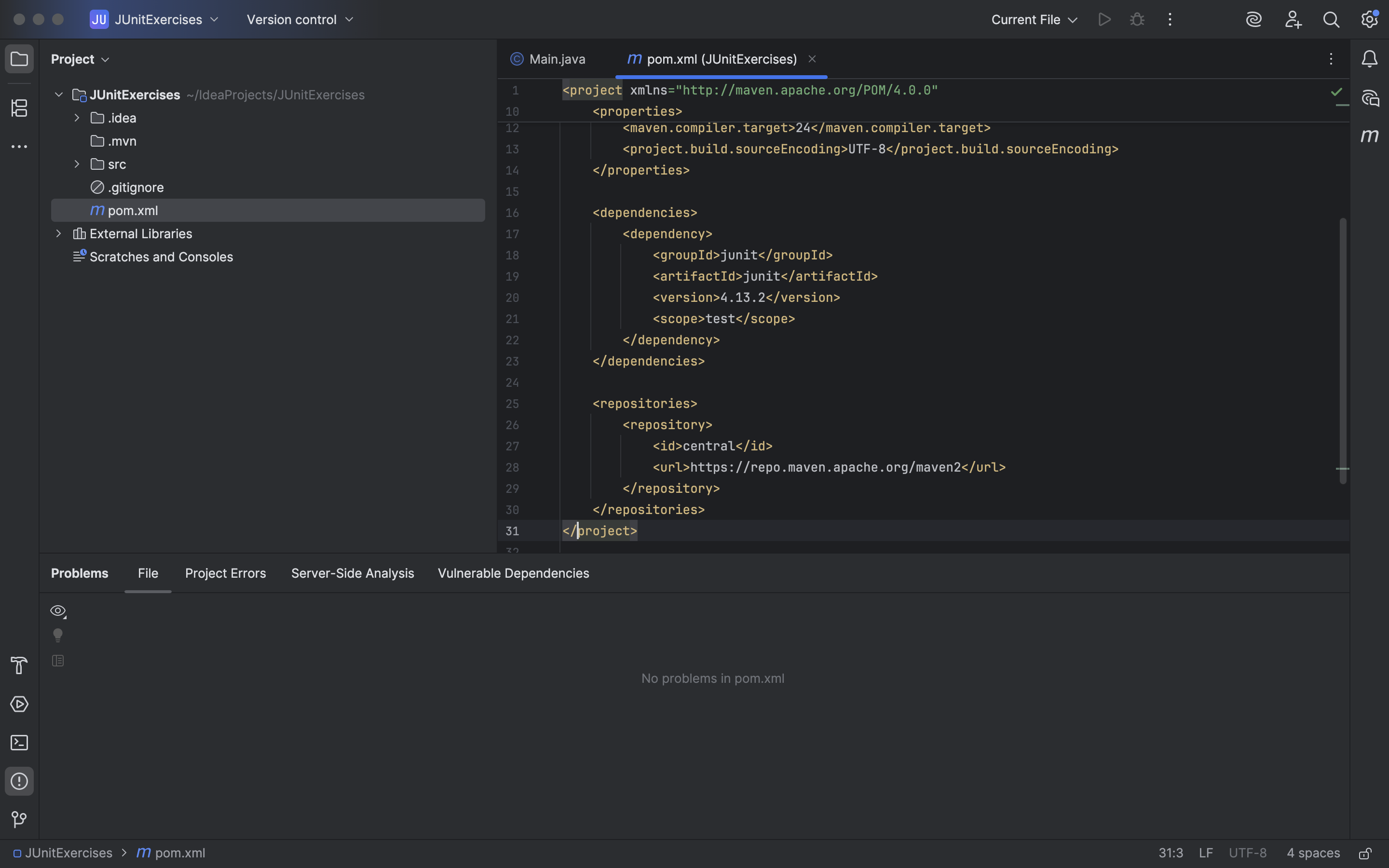
<version>4.13.2</version>

<scope>test</scope>

</dependency>

3. Create a new test class in your project.

**Dependency added in pom.xml**

****

**Code part**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

**Testing For the code part**

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

*assertEquals*(5, calc.add(2, 3));

}

@Test

public void testSubtract() {

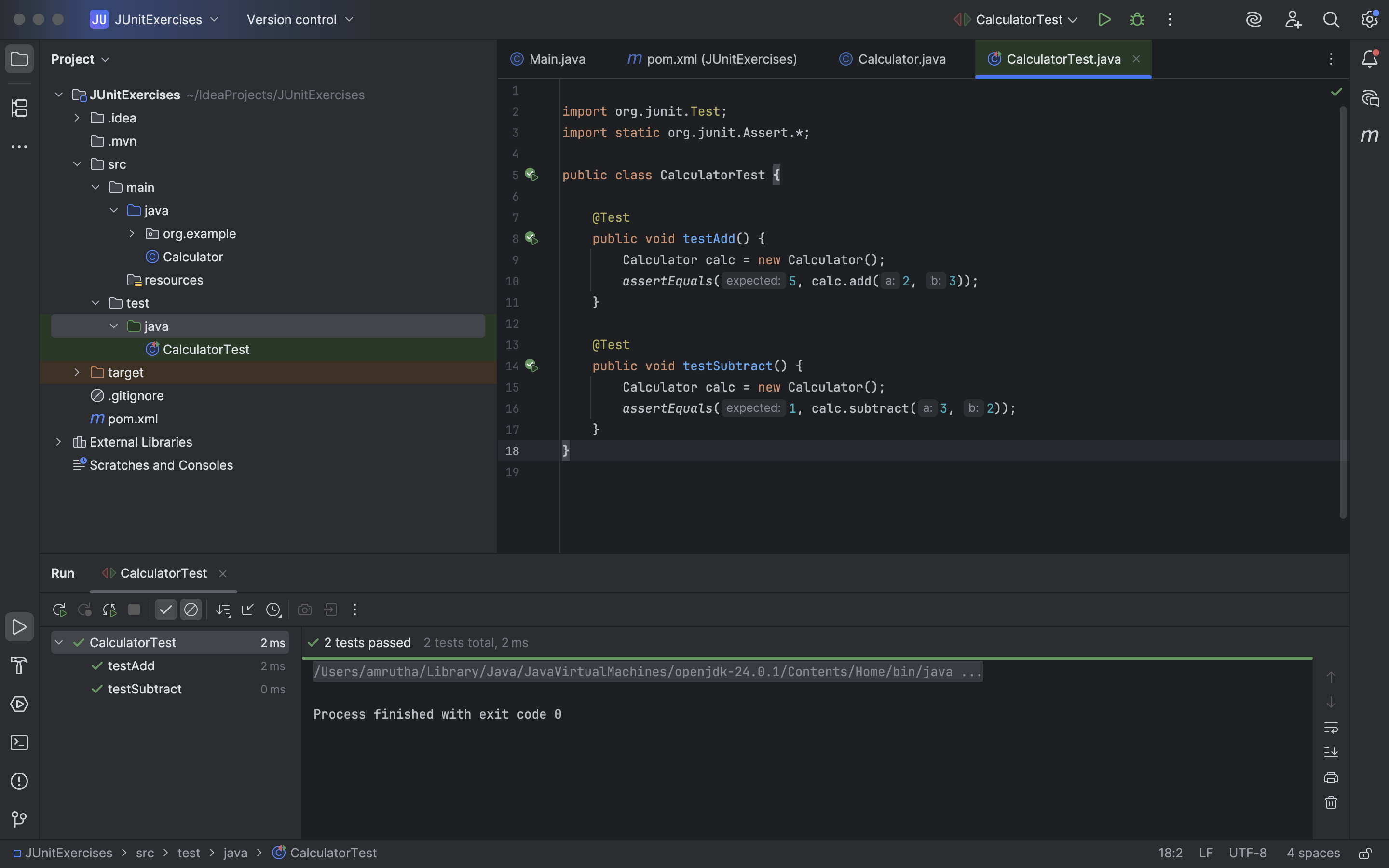
Calculator calc = new Calculator();

*assertEquals*(1, calc.subtract(3, 2));

}

}

**Terminal Output For Test Pass Confirmation**

****

**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 part**

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

// Assert equals

*assertEquals*("Sum check failed", 5, 2 + 3);

// Assert true

*assertTrue*("Expected true condition", 5 > 3);

// Assert false

*assertFalse*("Expected false condition", 5 < 3);

// Assert null

Object obj1 = null;

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

// Assert not null

Object obj2 = new Object();

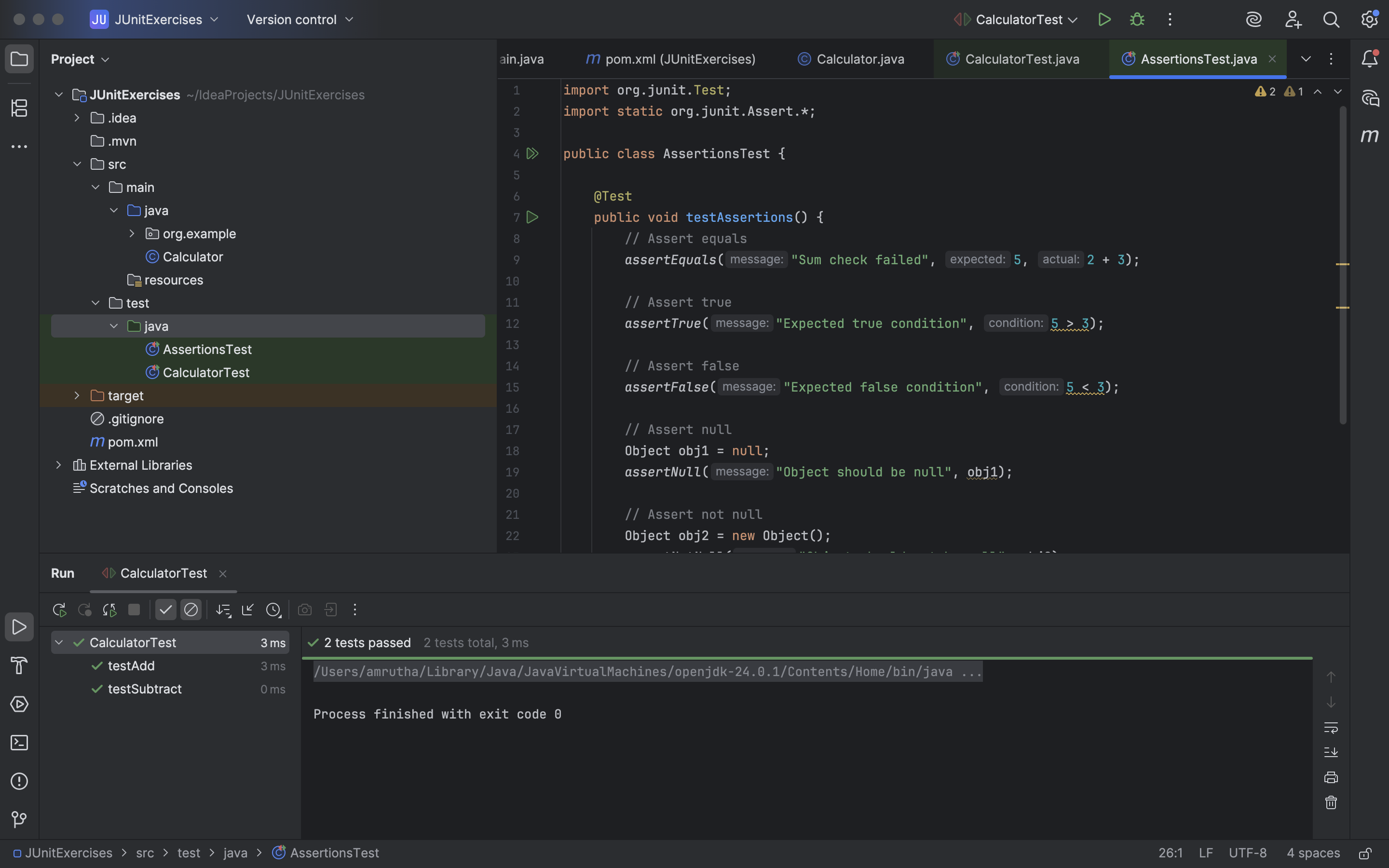
*assertNotNull*("Object should not be null", obj2);

}

}

| **Assertion** | **Purpose** |
| --- | --- |
| assertEquals | Validates arithmetic or logic matches expected |
| assertTrue | Confirms condition is true |
| assertFalse | Confirms condition is false |
| assertNull | Ensures an object is null |
| assertNotNull | Ensures an object is not 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.

**Code Part**

**SimpleCalculation.java**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

**SimpleCalculationTest.java**

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorLifecycleTest {

private Calculator calculator;

// Setup (Before each test)

@Before

public void setUp() {

calculator = new Calculator();

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

}

// Teardown (After each test)

@After

public void tearDown() {

calculator = null;

System.*out*.println(" Teardown: Calculator object cleared");

}

@Test

public void testAdd() {

// Arrange is done in setUp()

// Act

int result = calculator.add(10, 5);

// Assert

*assertEquals*(15, result);

System.*out*.println(" testAdd passed");

}

@Test

public void testSubtract() {

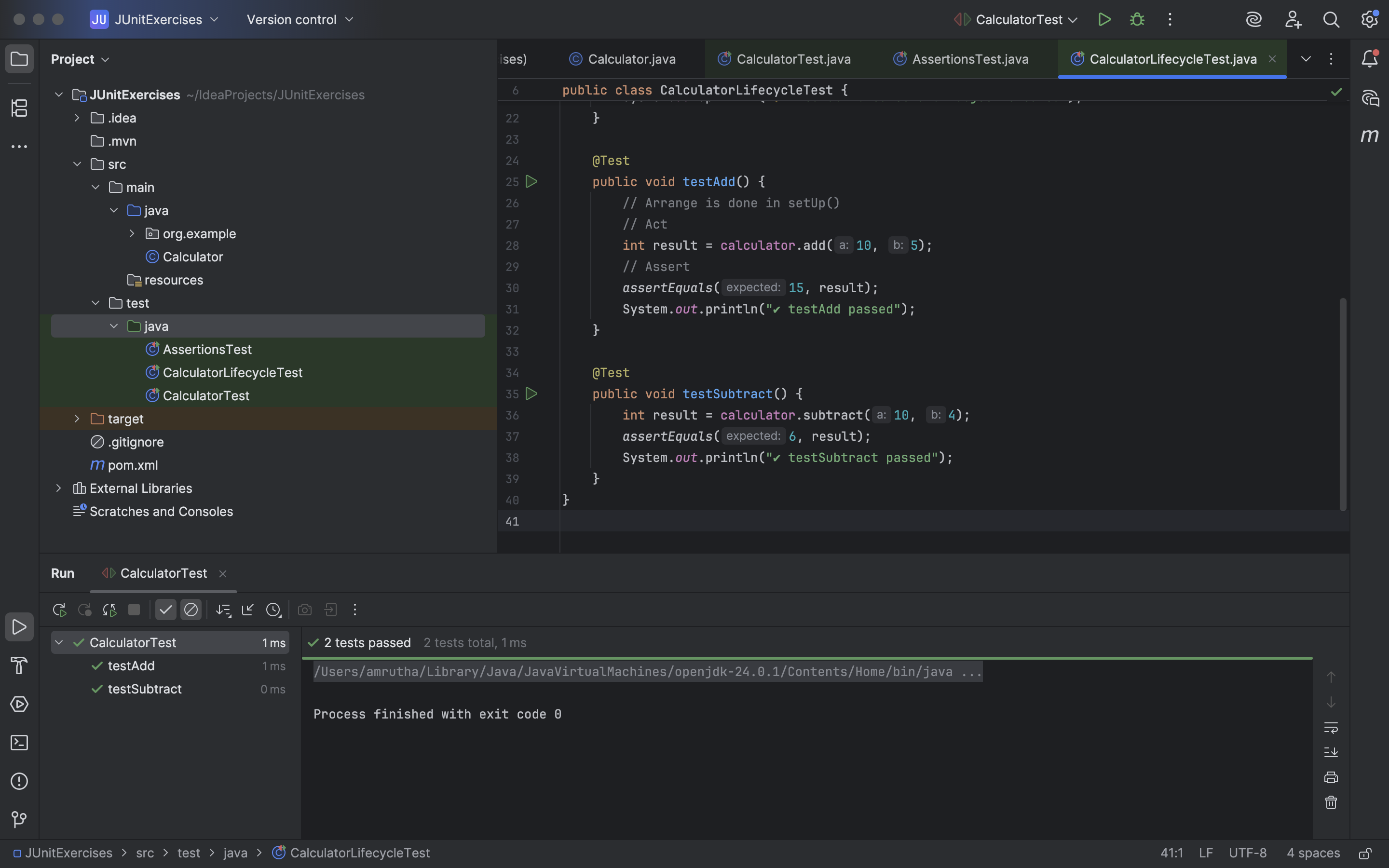
int result = calculator.subtract(10, 4);

*assertEquals*(6, result);

System.*out*.println("testSubtract passed");

}

}



**(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.

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:

import static org.mockito.Mockito.\*;

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);

}

}

**Solution Code:**

package org.example;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**Test.java**

package org.example;

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);

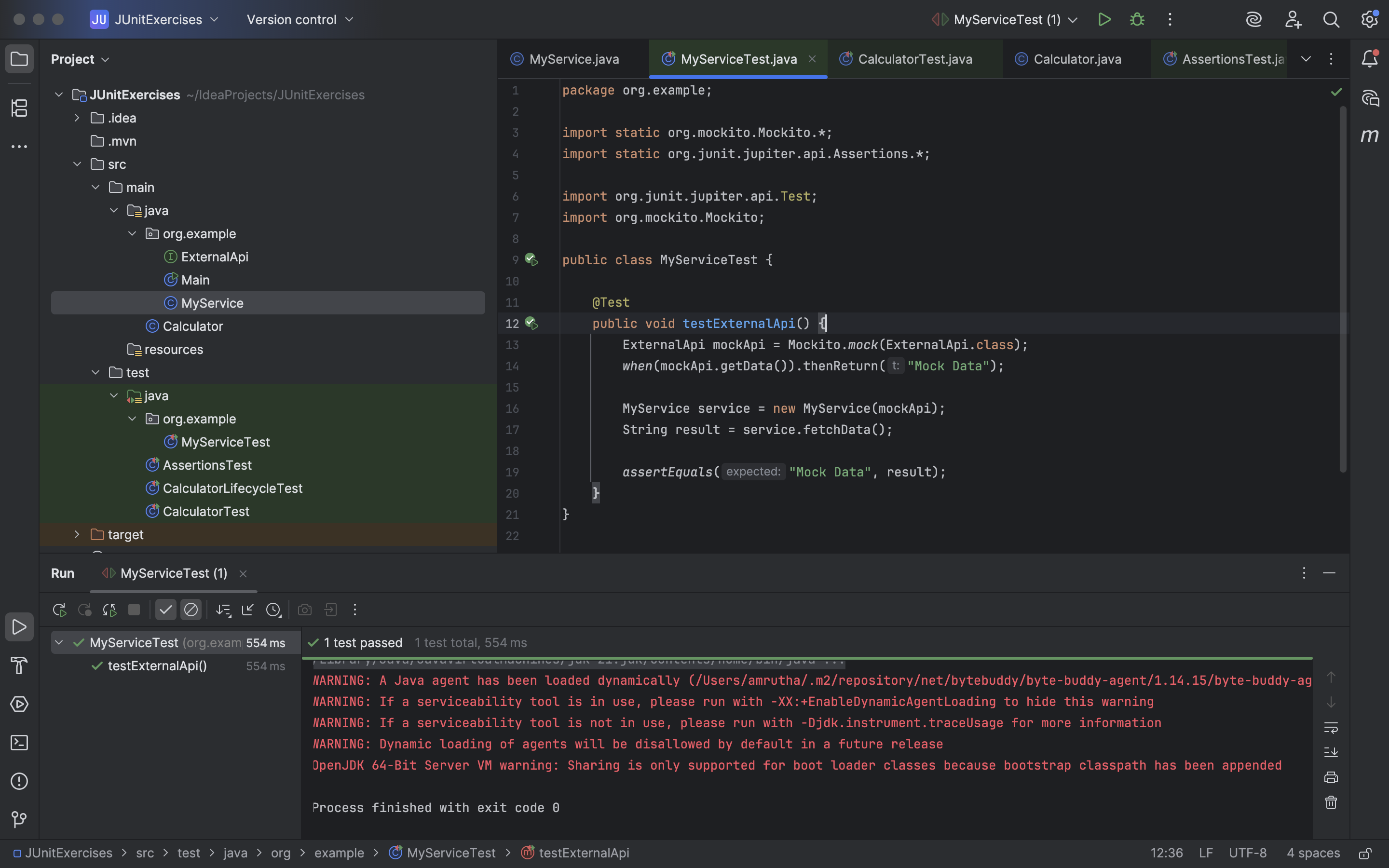
String result = service.fetchData();

*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:

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

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

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

**Solution Code:**

**ApiClient.java**

package com.example.junitdemo;

public interface ApiClient {

    void fetchUserDetails();

}

**UserProcessor.java**

package com.example.junitdemo;

public class UserProcessor {

    private final ApiClient apiClient;

    public UserProcessor(ApiClient apiClient) {

        this.apiClient = apiClient;

    }

    public void processUser() {

        apiClient.fetchUserDetails();

    }

}

**MyserviceTest.java**

package org.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// Step 1: Create mock

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

// Step 2: Create service with mock

MyService service = new MyService(mockApi);

// Step 3: Call method on service

service.fetchData();

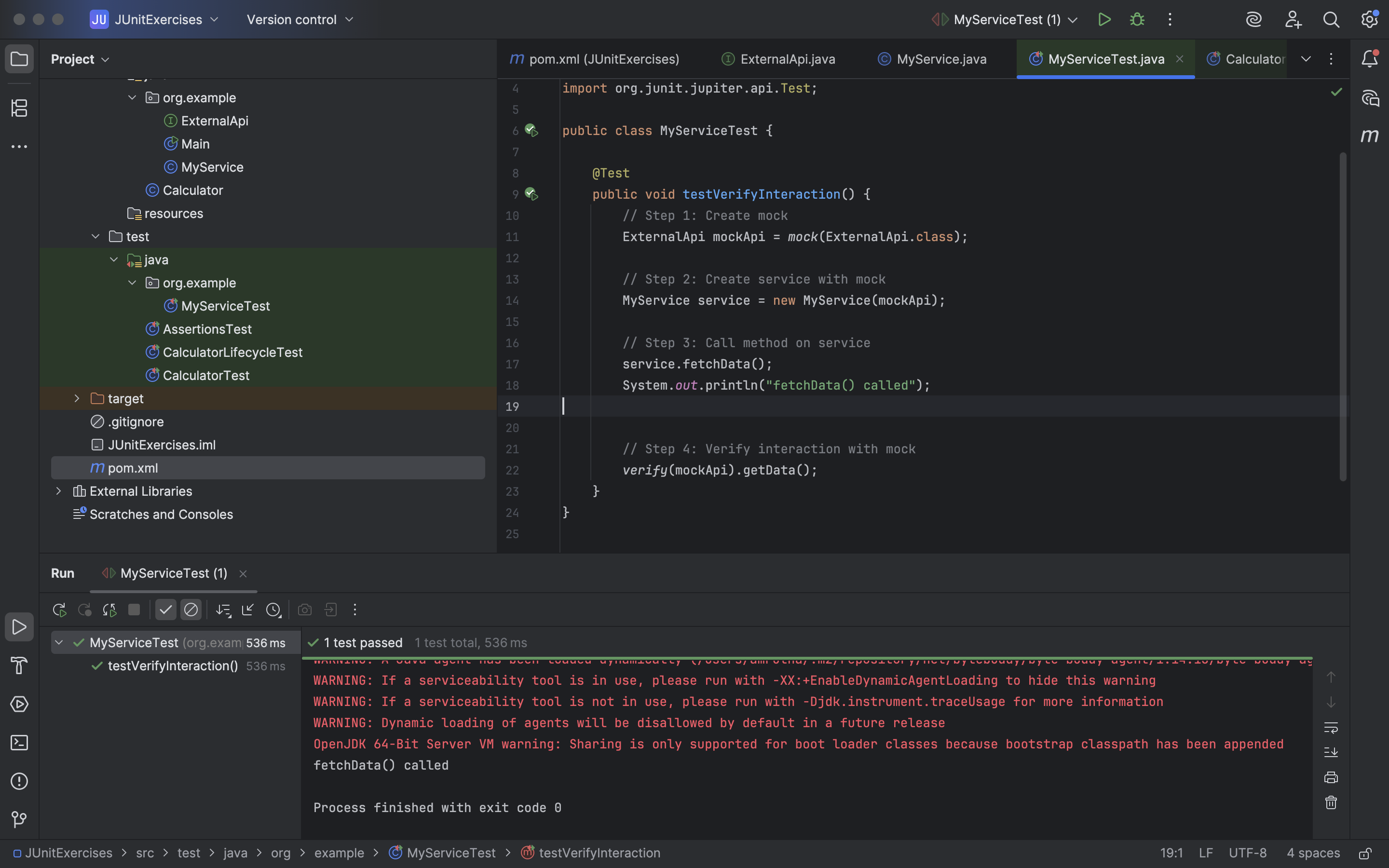
System.*out*.println("fetchData() called");

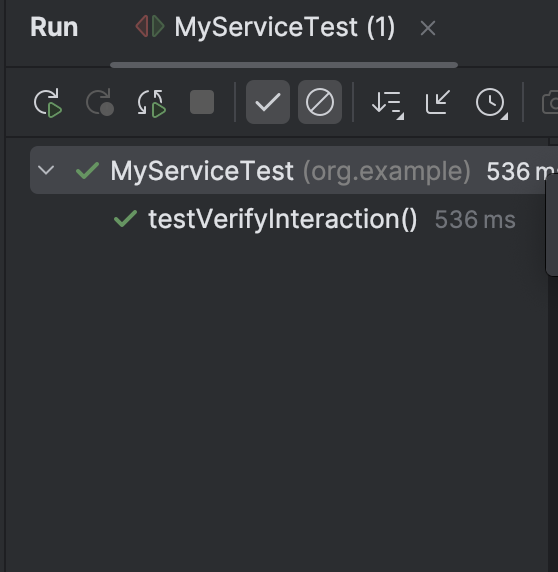
// Step 4: Verify interaction with mock

*verify*(mockApi).getData();

}

}





**(SL4J Logging exercises)**

**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:

<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>

2. Create a Java class that uses SLF4J for logging:

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");

}

}

**Solution Code:**

**LoggingExample.java**

package org.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");

}

}

