**TDD using JUnit5 And Mockito And SJF4L**

**JUNIT Basic Exercises:**

**Question 1**

**Exercise 1: Setting Up JUnit Scenario:**

You need to set up JUnit in your Java project to start writing unit tests. Steps:

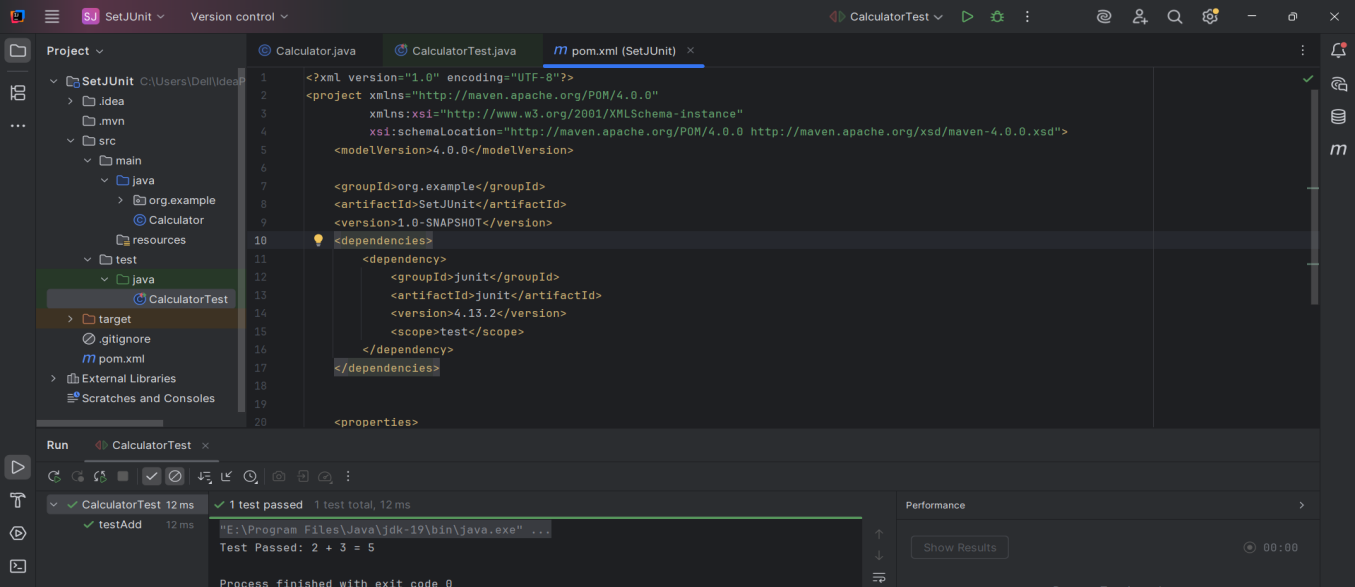
1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).

2. Add JUnit dependency to your project. If you are using Maven, add the following to your pom.xml: junit junit 4.13.2 test

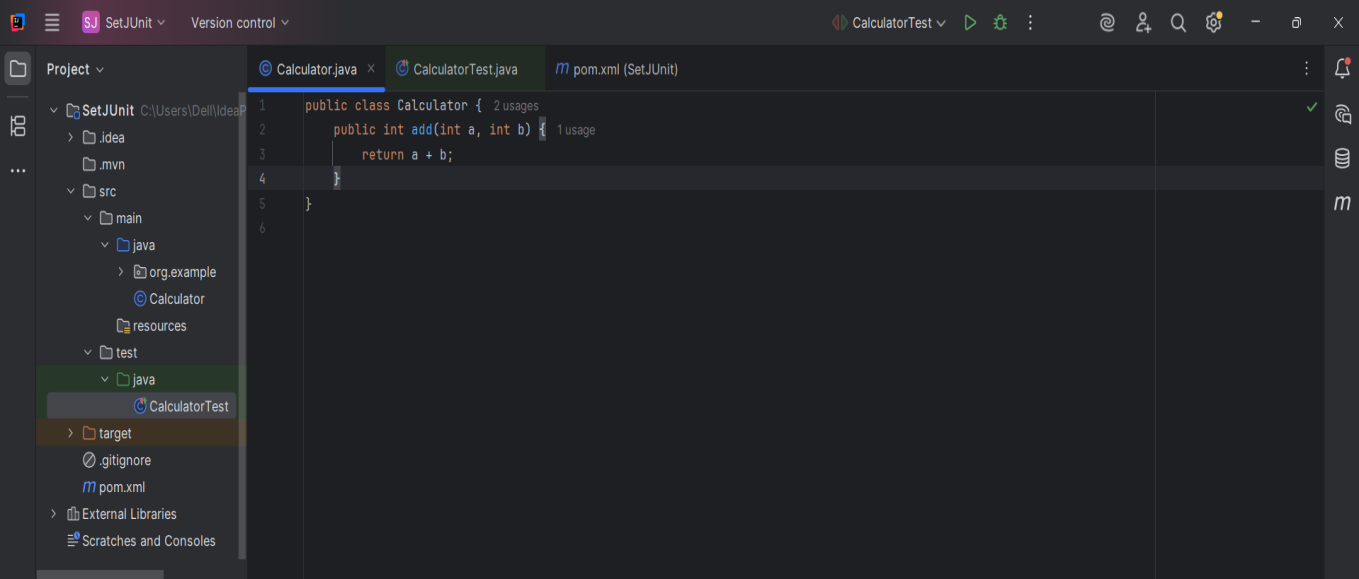
3. Create a new test class in your project.

**ANSWER:**

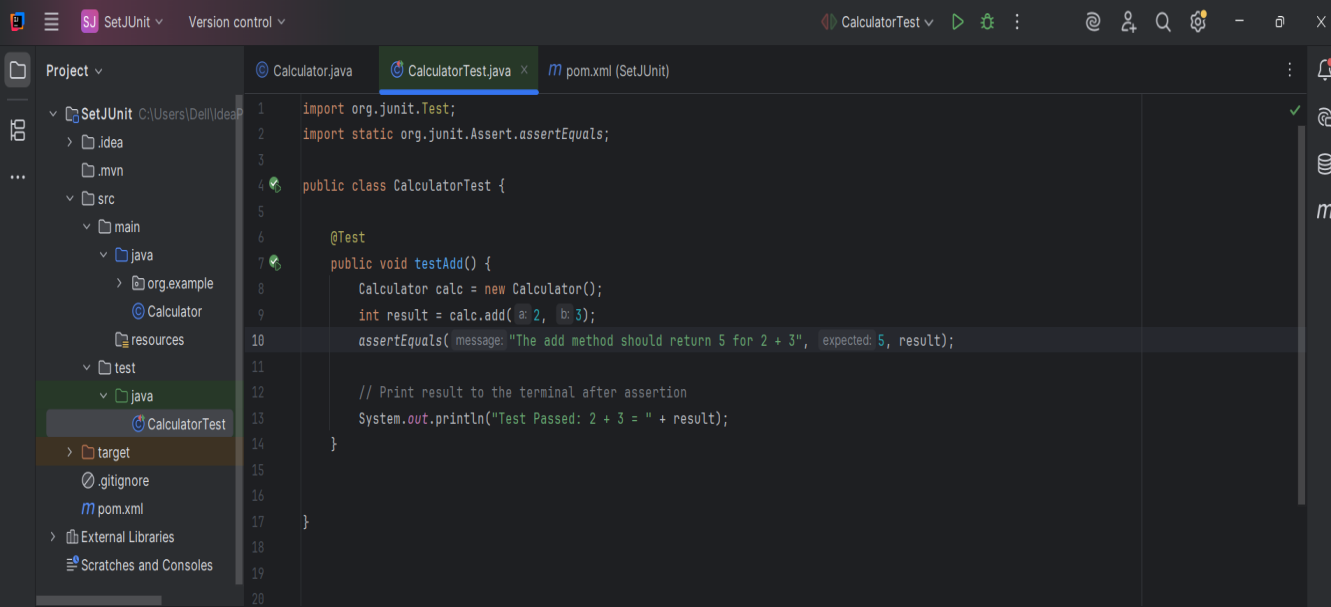
Step 1: Setup of java project and pom.xml



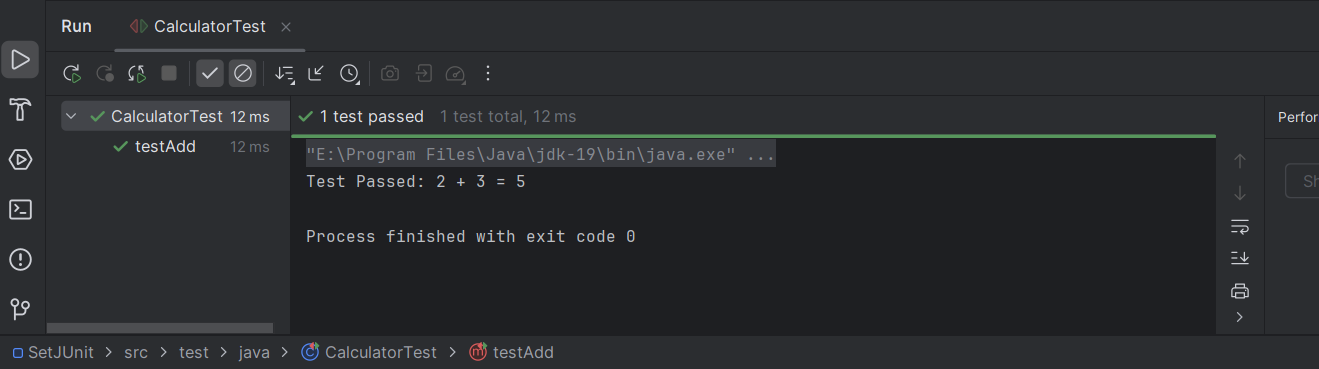
Step 2: calculator code



Step 3: Testing Calculator code



**Output:**



**Question 3**

**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:

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

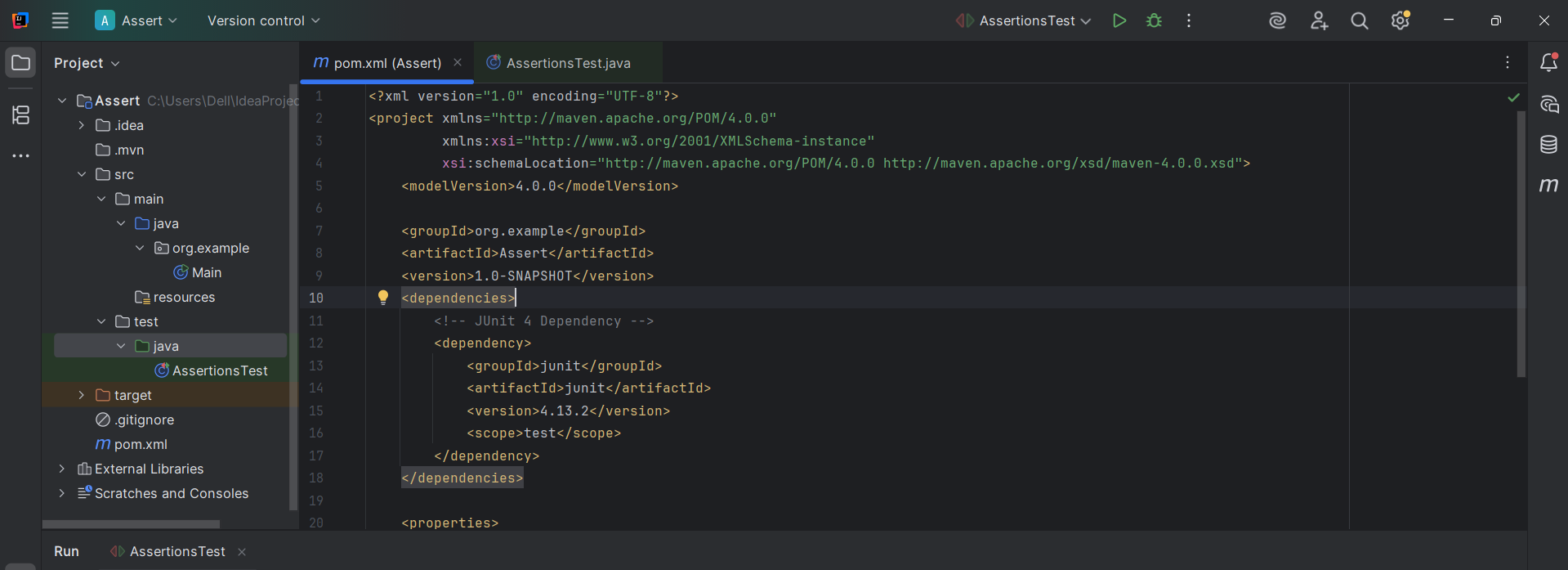
assertNotNull(new Object());

}

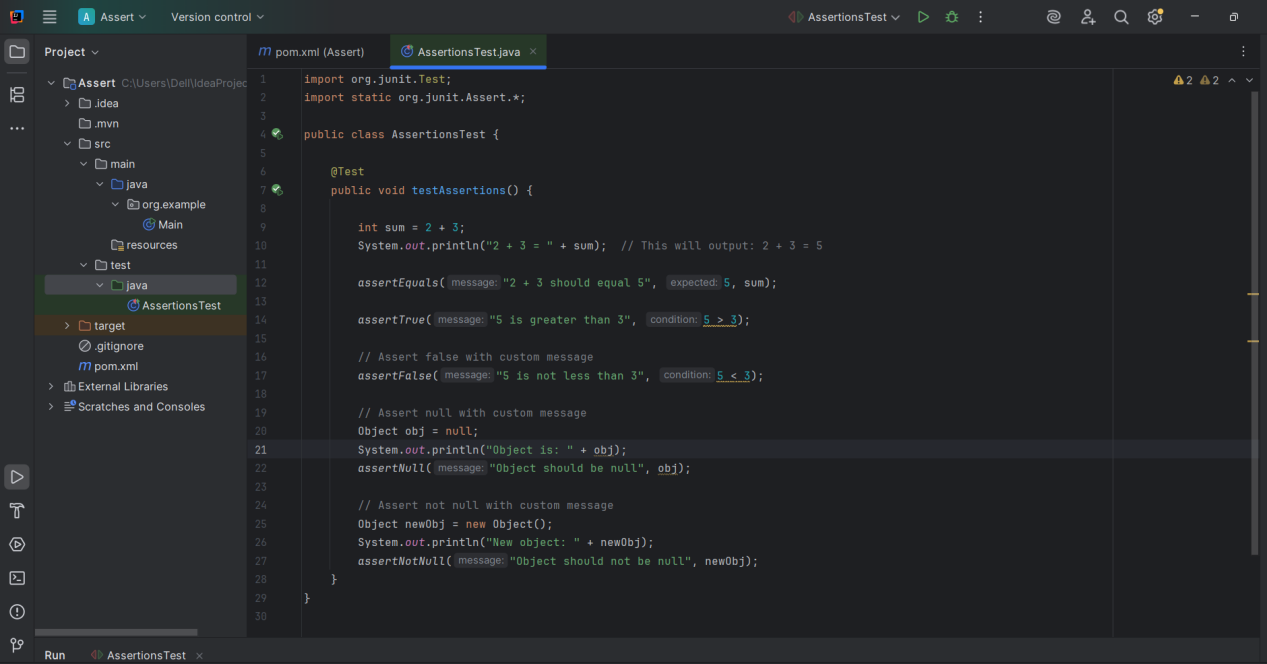
}

**Answer:**

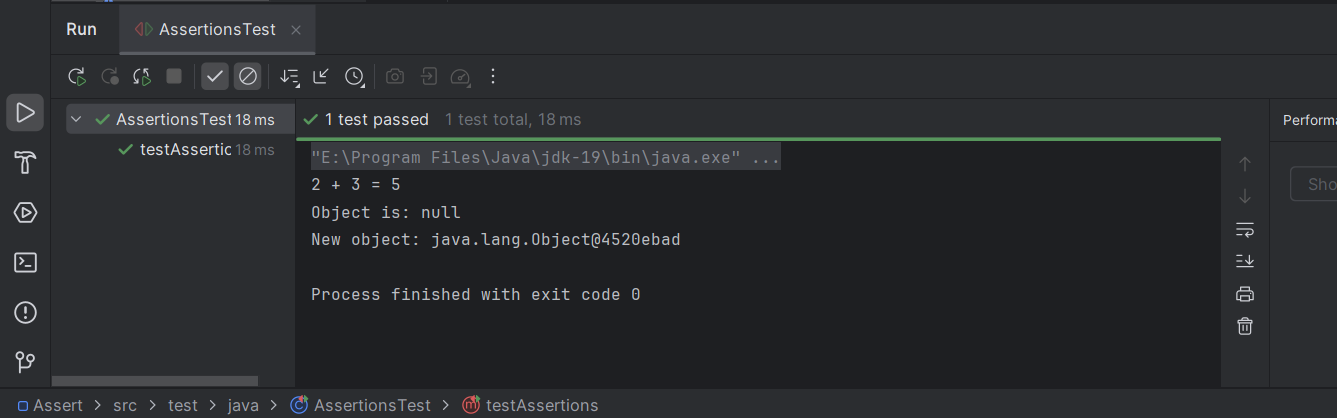
Step 1:

****

Step 2: Code



**Output:**



**Question 4**

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

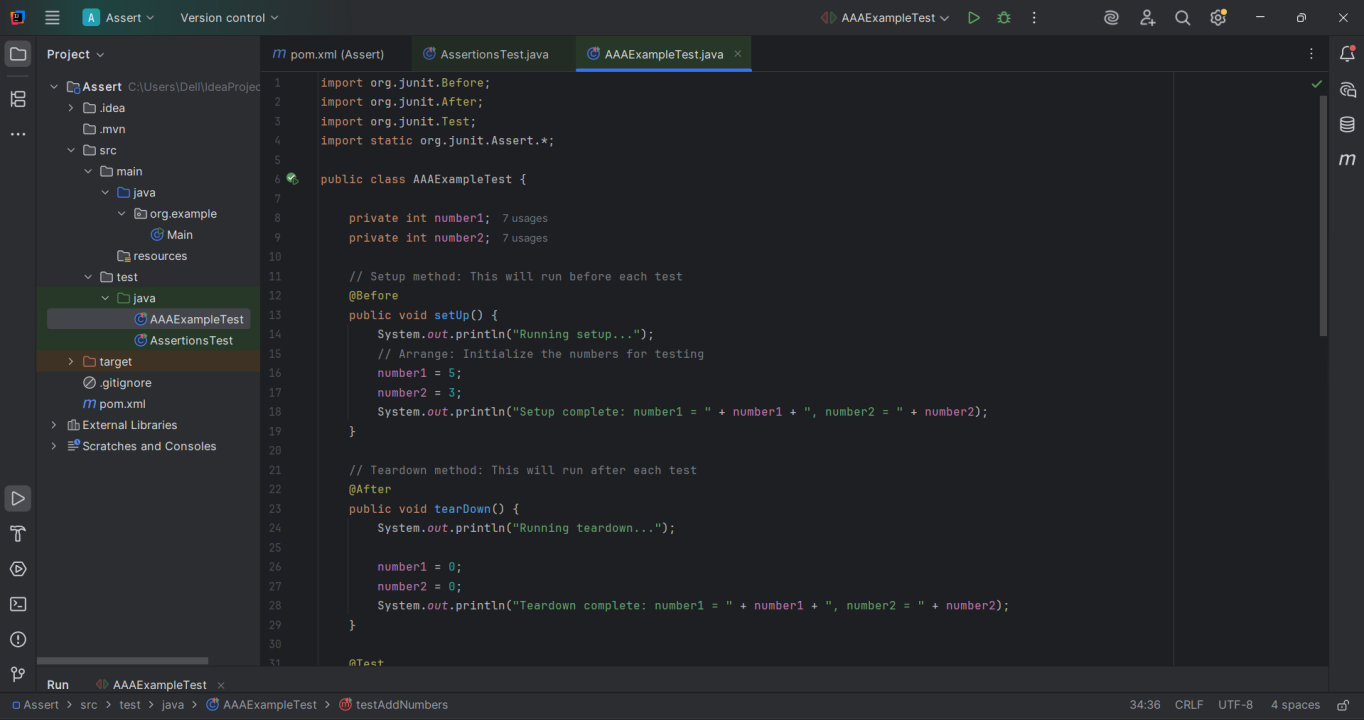
You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.

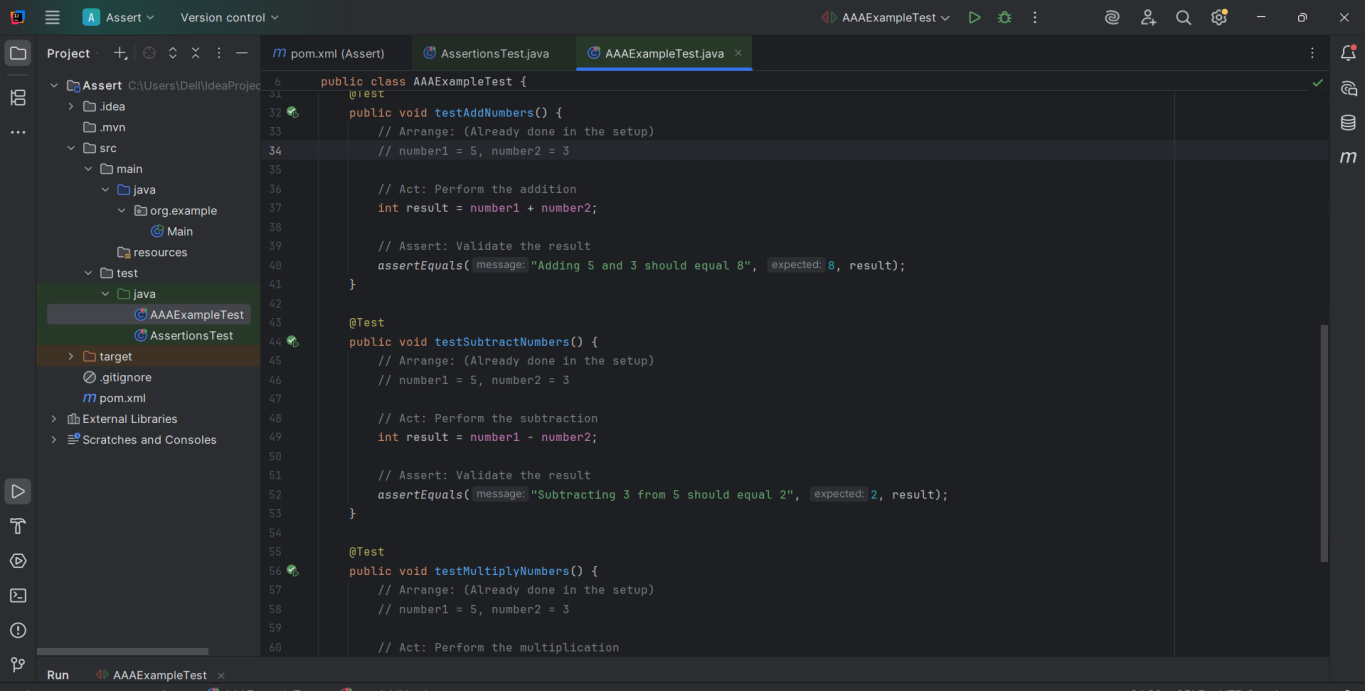
Steps:

1. Write tests using the AAA pattern.
2. Use @Before and @After annotations for setup and teardown methods.

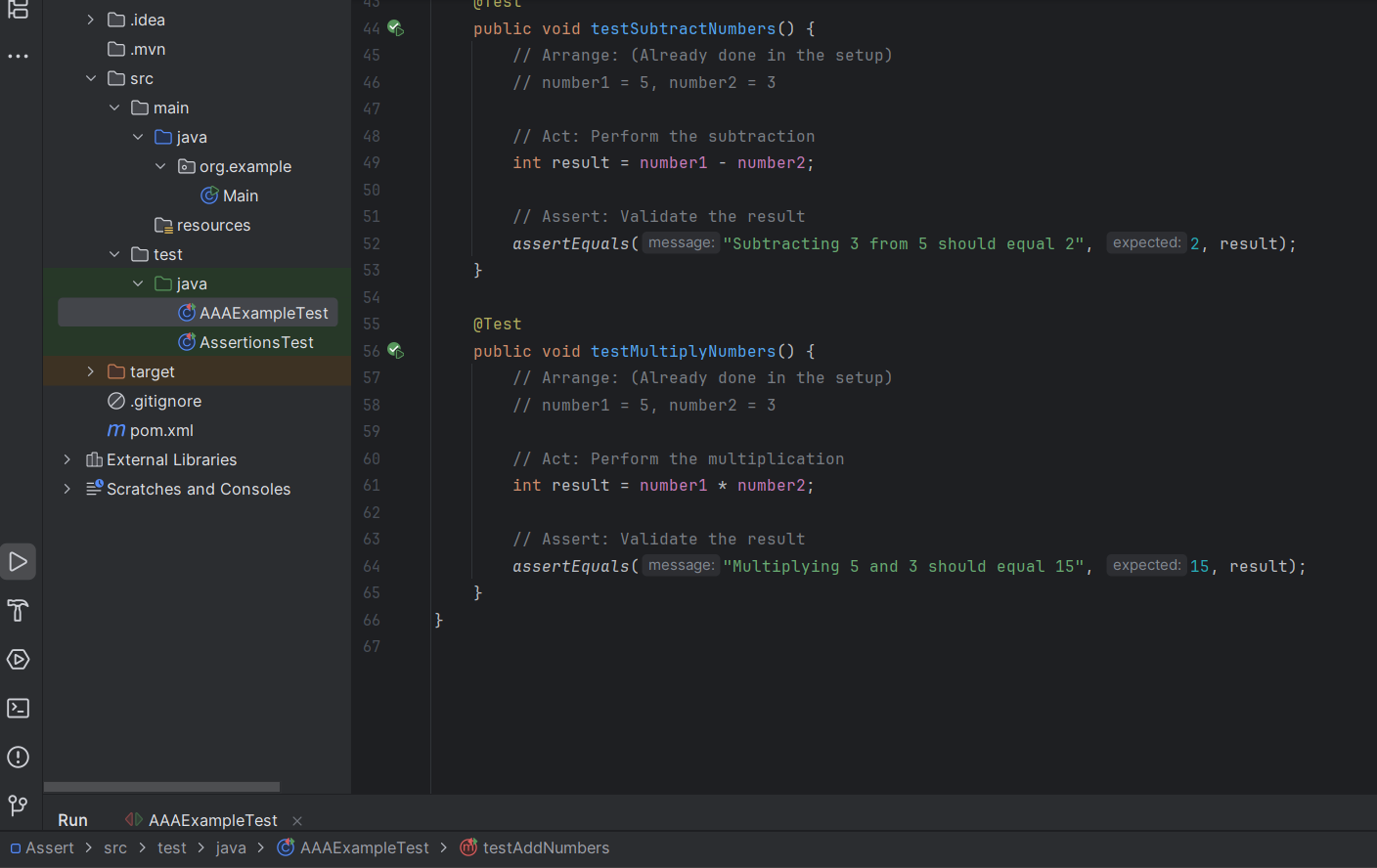
**Answer:**

Step 1: Tests using AAA pattern with @Before and @After

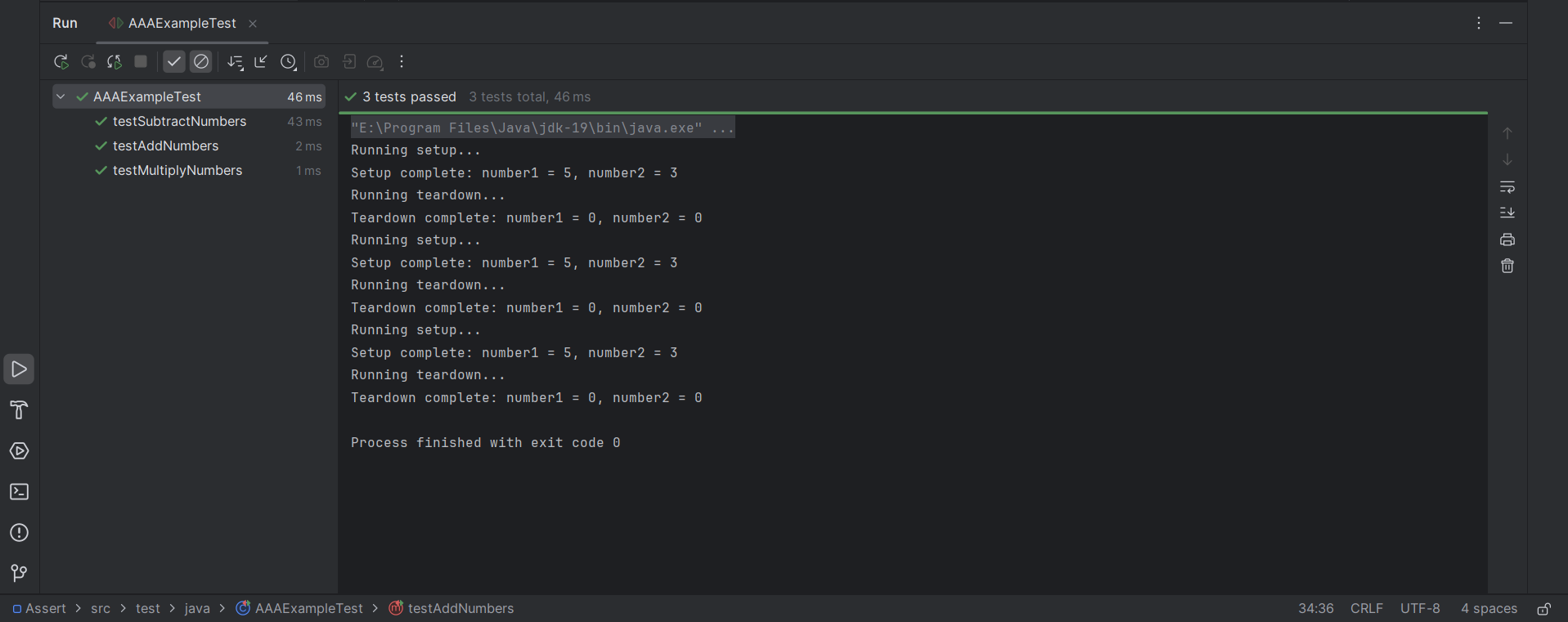




Step 2: Teardown method



**Output:**

****

**Mockito Exercises:**

**Question 1**

**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();

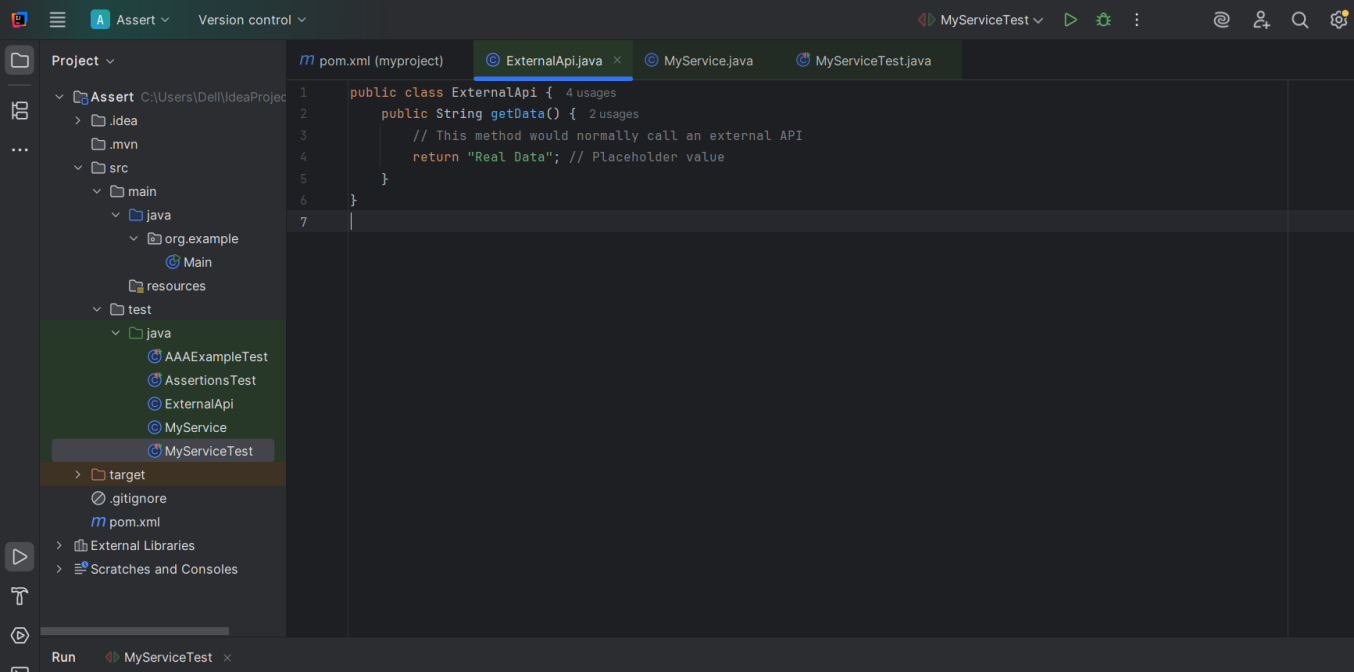
assertEquals("Mock Data", result);

}

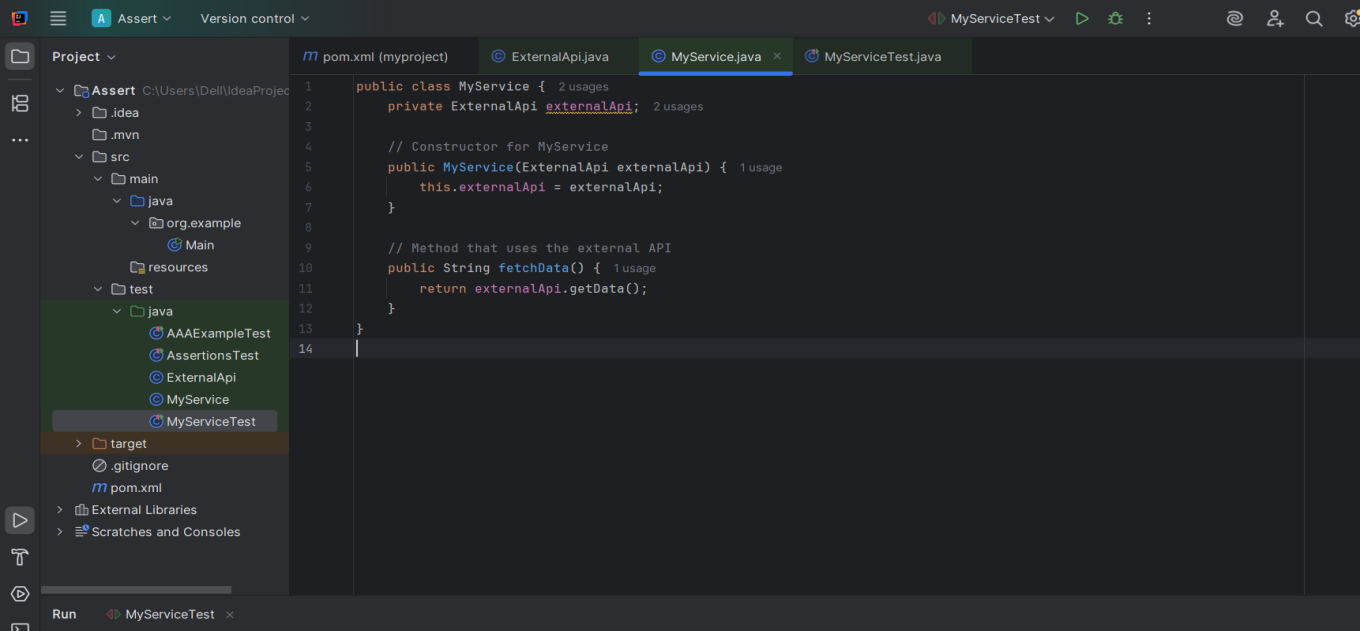
}

**Answer:**

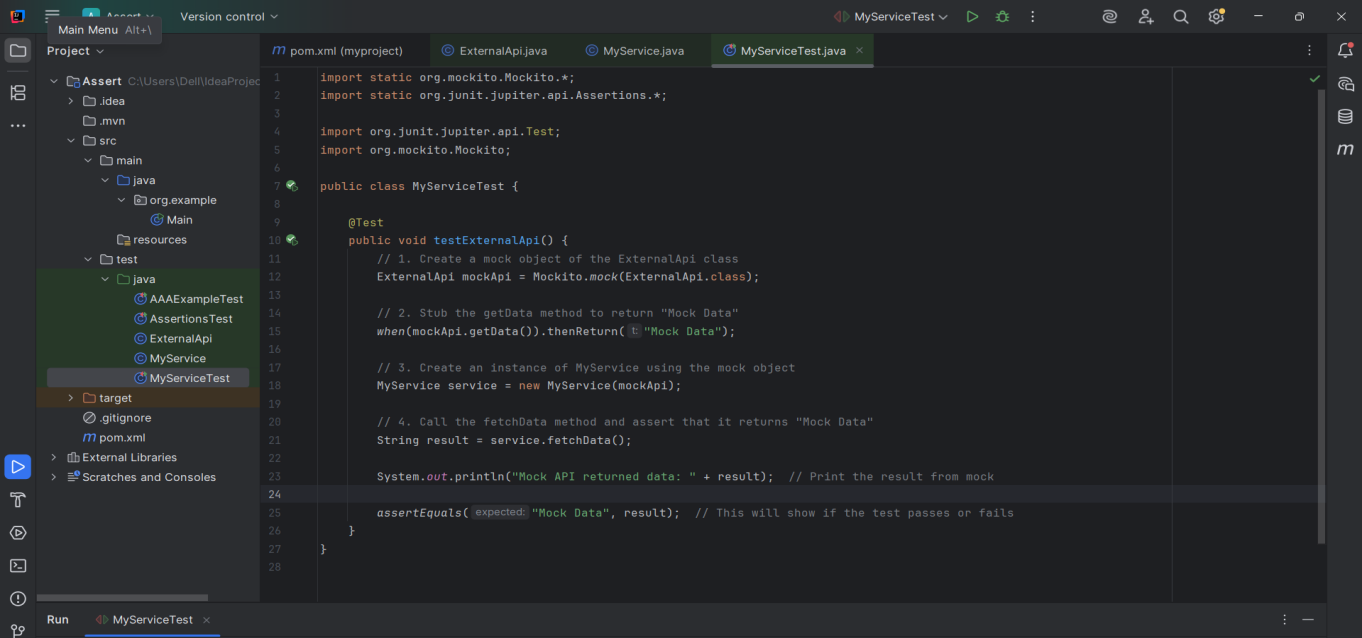
Step 1: API

****

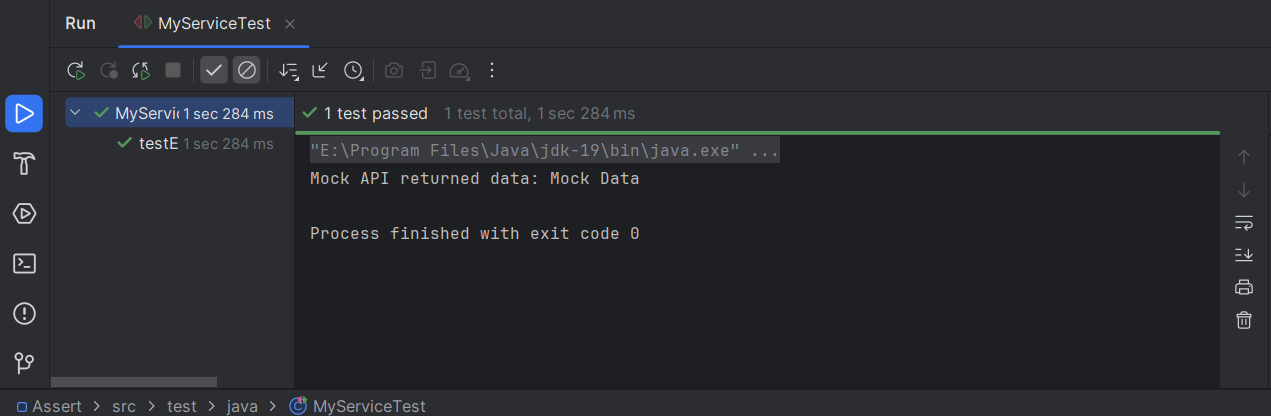
Step 2: methods

****

Step 3: Test case for mock project



**Output:**



**Question 2**

**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();

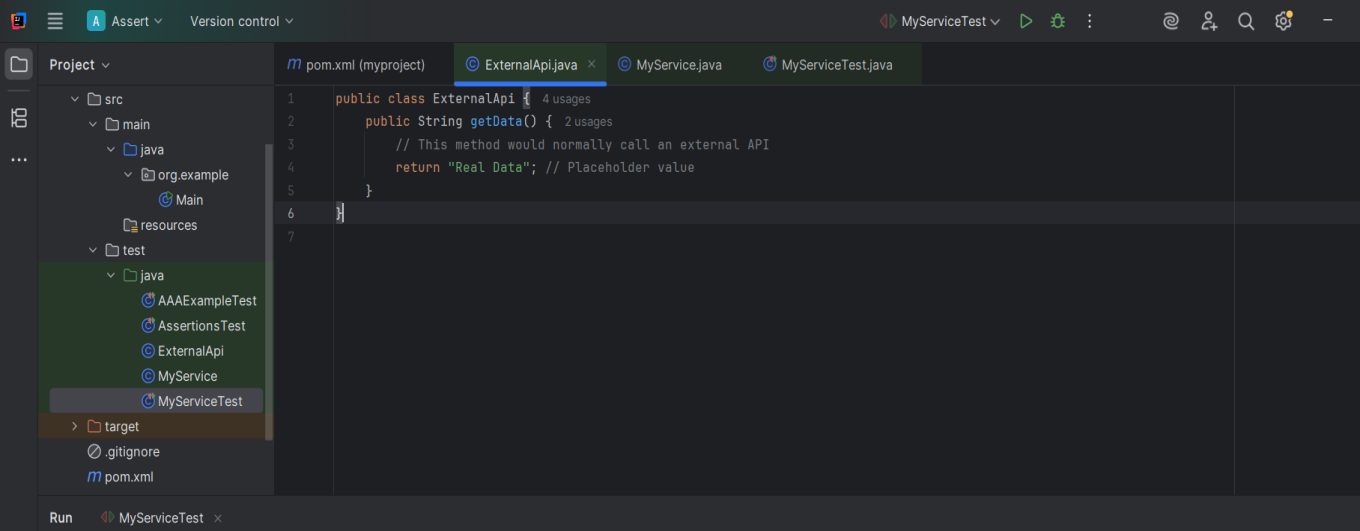
verify(mockApi).getData();

}

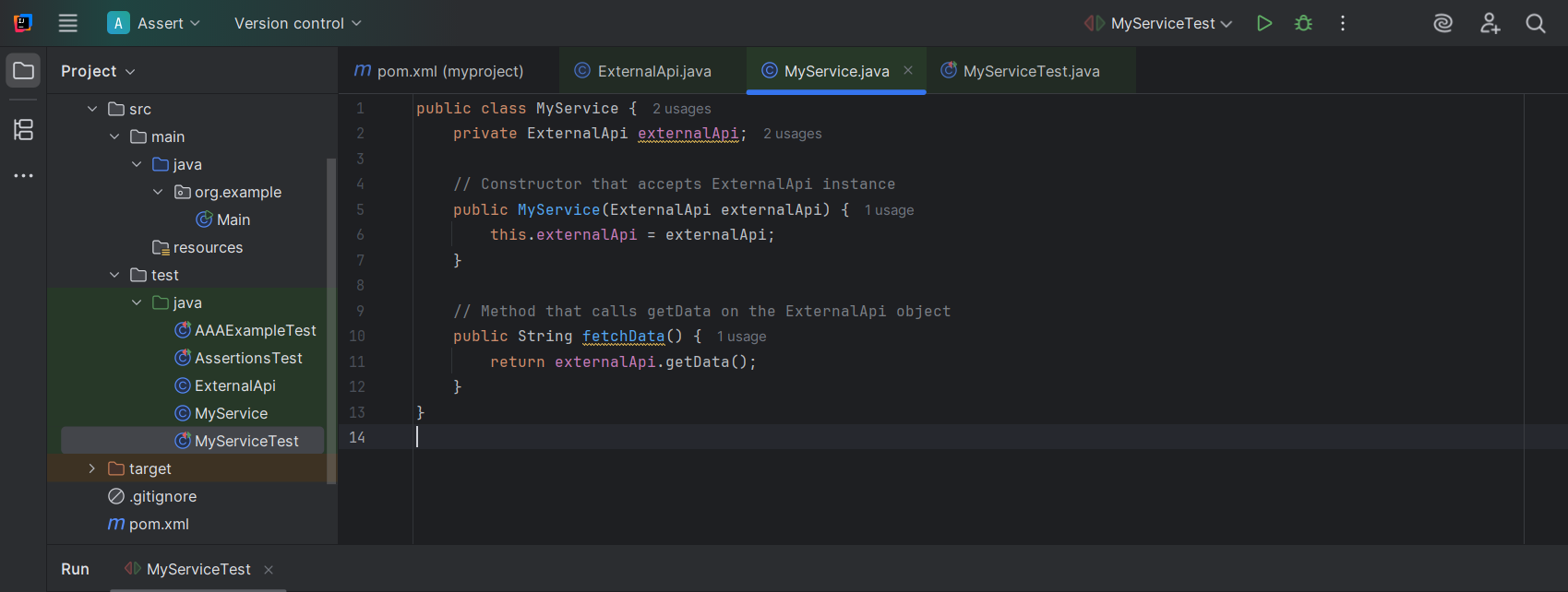
}

**Answer:**

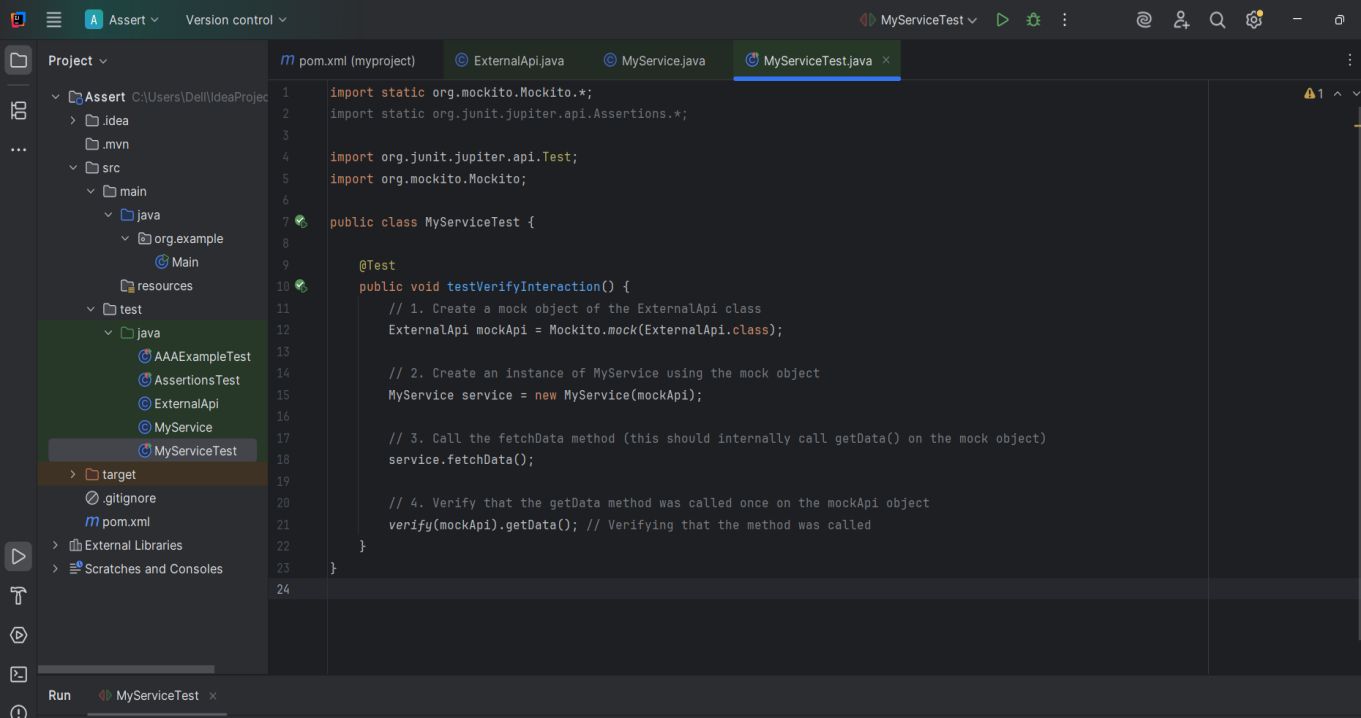
Step 1: API



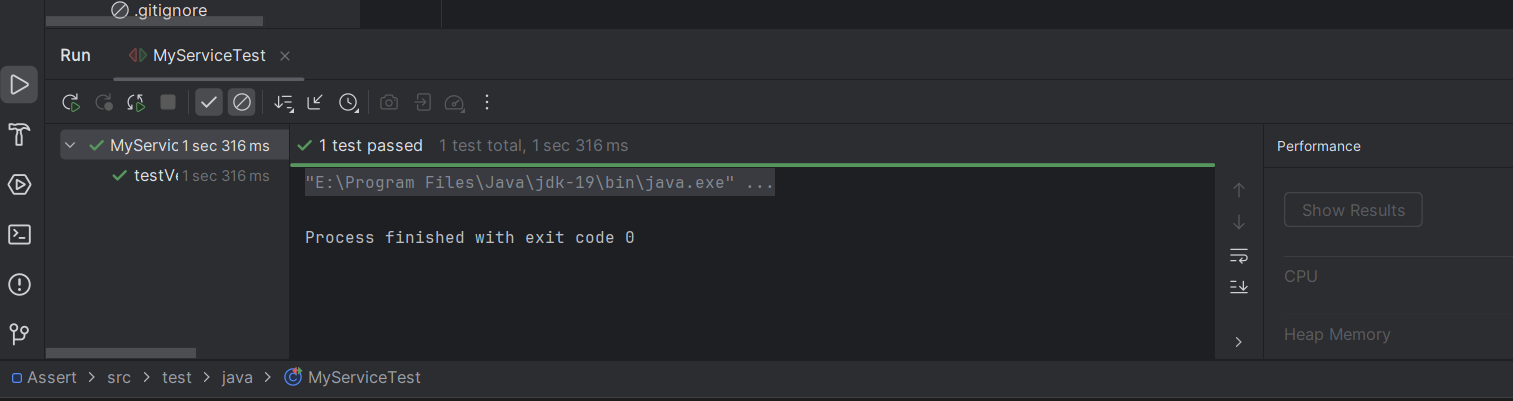
Step 2: method



Step 3: verifying interaction



**Output:**



**SJF4L Exercise:**

**Question 1**

**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");

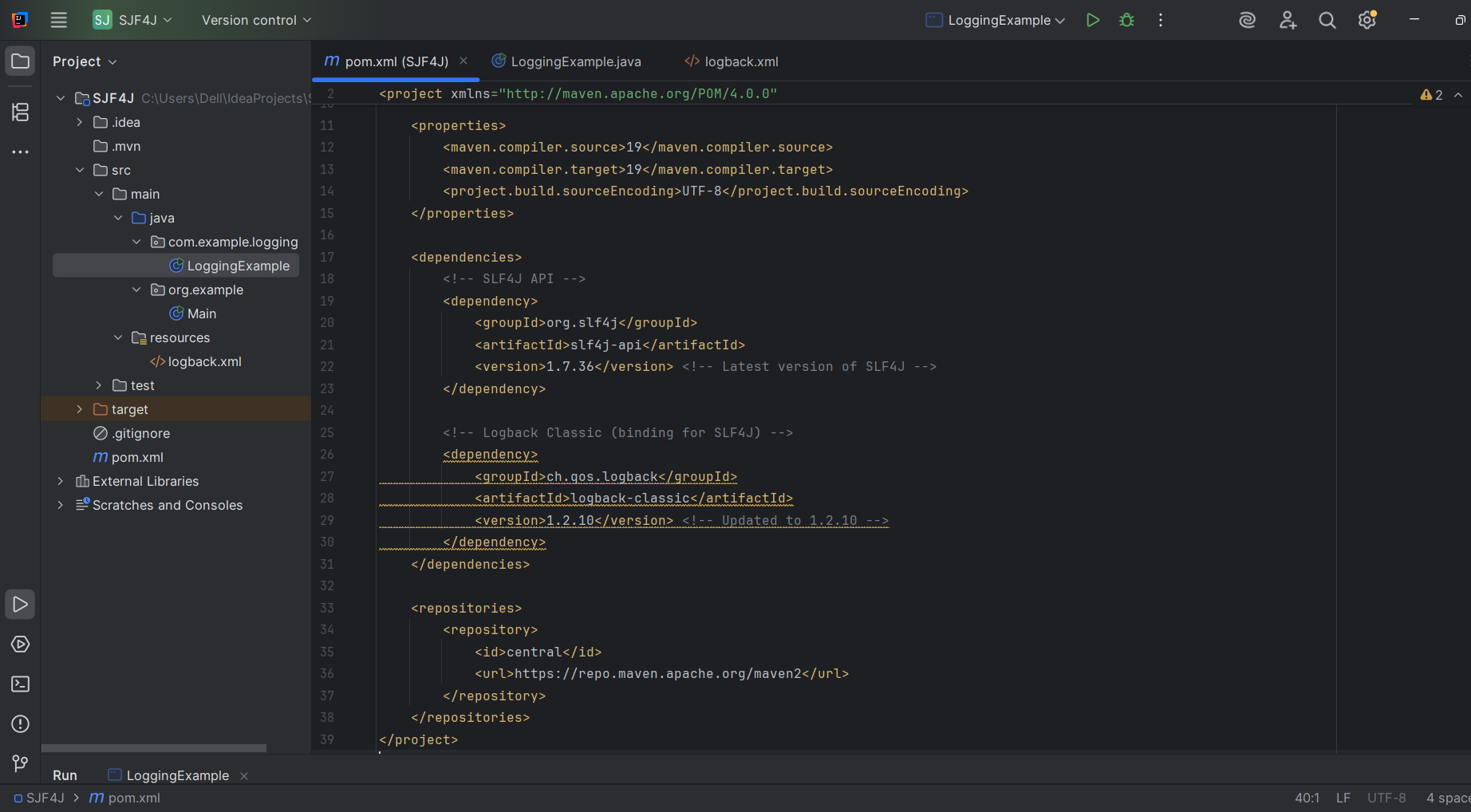
logger.warn("This is a warning message");

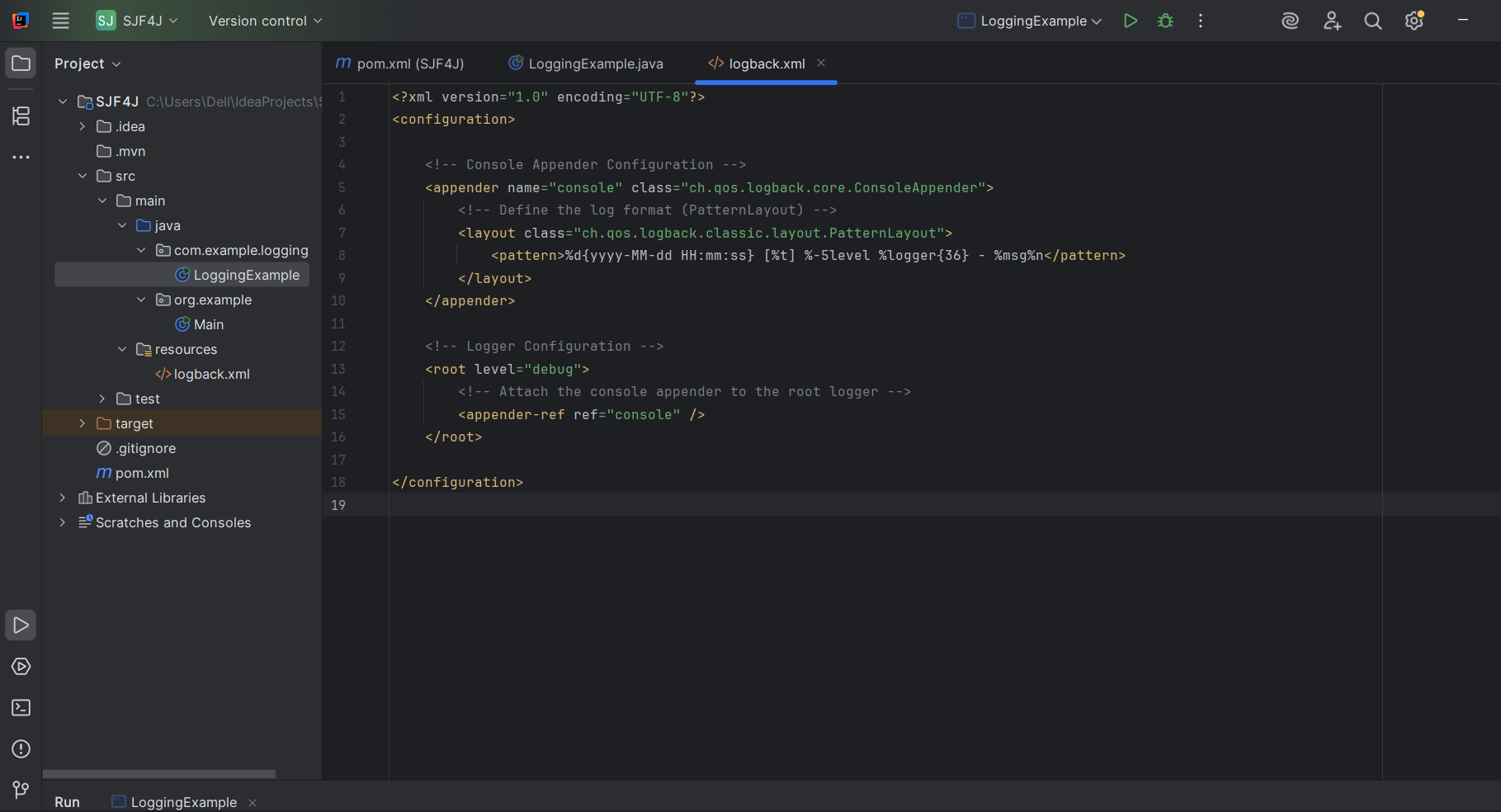
}

}

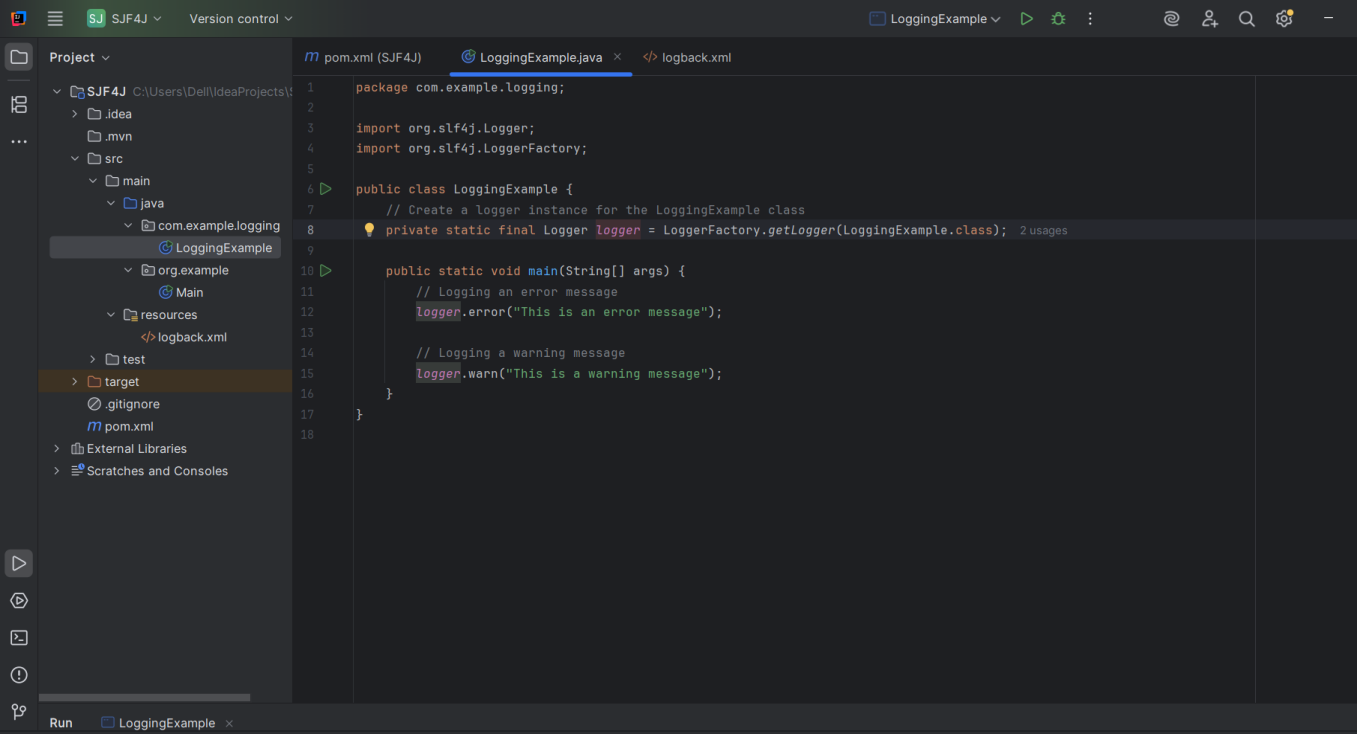
**Answer:**

Step 1: Adding SLF4J and Logback dependencies to`pom.xml` file

****

****

Step 2: Creating a Java class that uses SLF4J for logging



**Output:**

