Week 2

Module 4 – Test-driven development and Logging framework

**JUnit Testing Exercises**

Exercise 1:

**Setting Up JUnit**

**Scenario:**

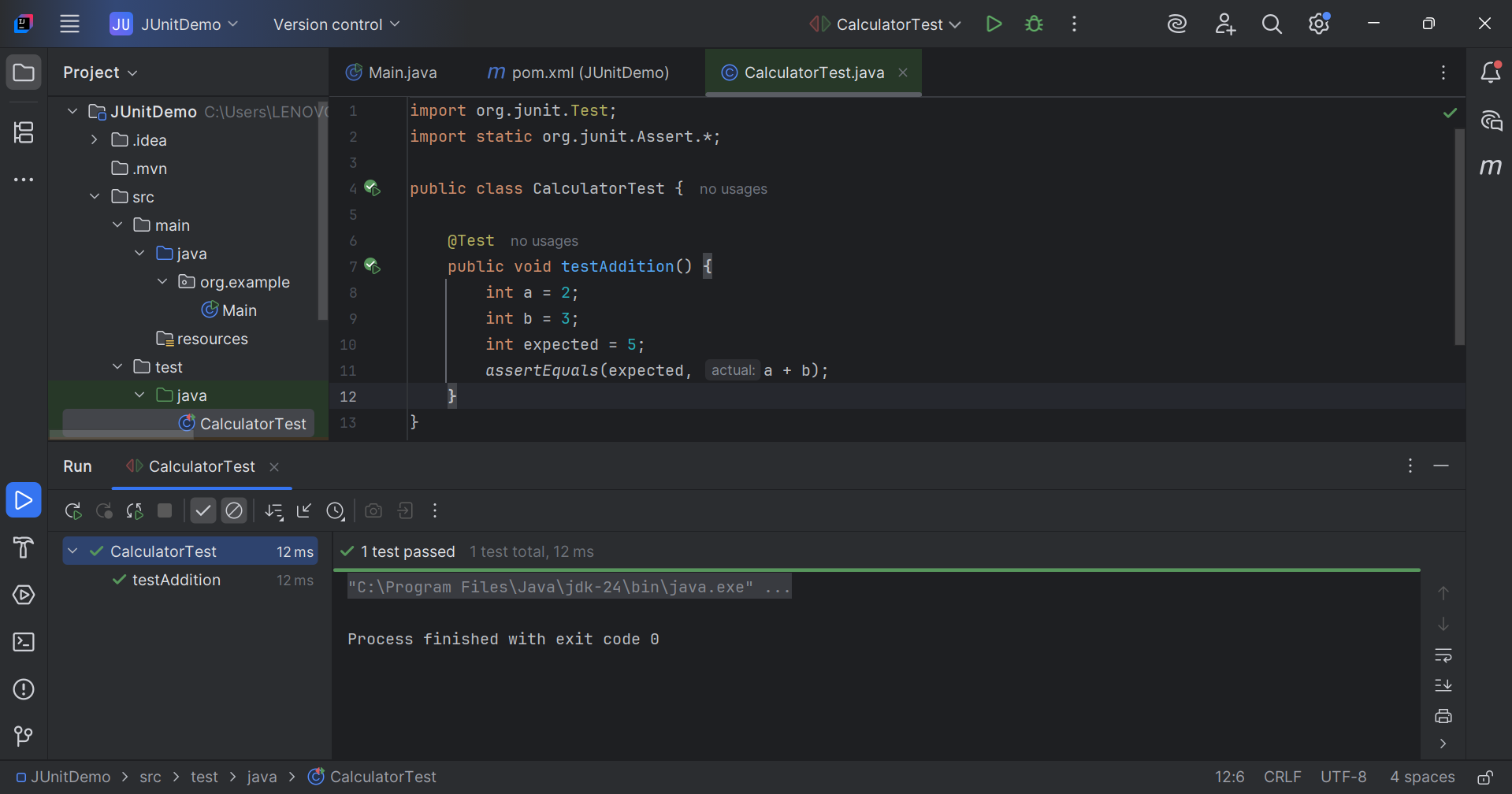
You need to set up JUnit in your Java project to start writing unit tests.

**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.



Exercise 2:

Writing Basic JUnit Tests

**Scenario:** You need to write basic JUnit tests for a simple Java class.

**Steps:**

1. Create a new Java class with some methods to test.

2. Write JUnit tests for these methods.

CODE:

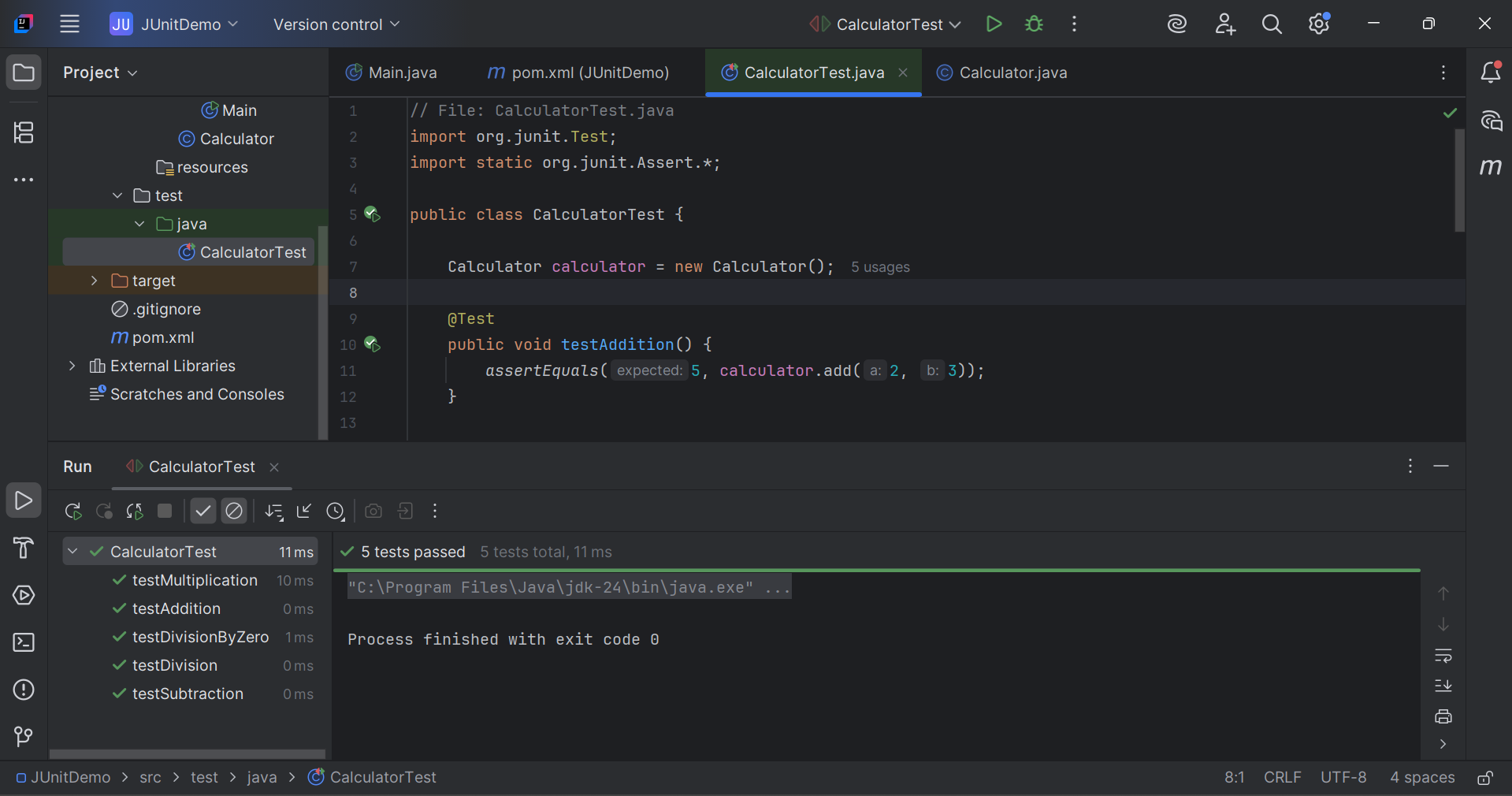
Calculator.java

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("Cannot divide by zero");  
 }  
 return a / b;  
 }  
}

CalculatorTest.java

// File: CalculatorTest.java  
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*(2, calculator.subtract(5, 3));  
 }  
  
 @Test  
 public void testMultiplication() {  
 *assertEquals*(6, calculator.multiply(2, 3));  
 }  
  
 @Test  
 public void testDivision() {  
 *assertEquals*(2, calculator.divide(6, 3));  
 }  
  
 @Test(expected = IllegalArgumentException.class)  
 public void testDivisionByZero() {  
 calculator.divide(5, 0);  
 }  
}

Output:



Exercise 3:

Assertions in JUnit

**Scenario:**

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

**Steps:**

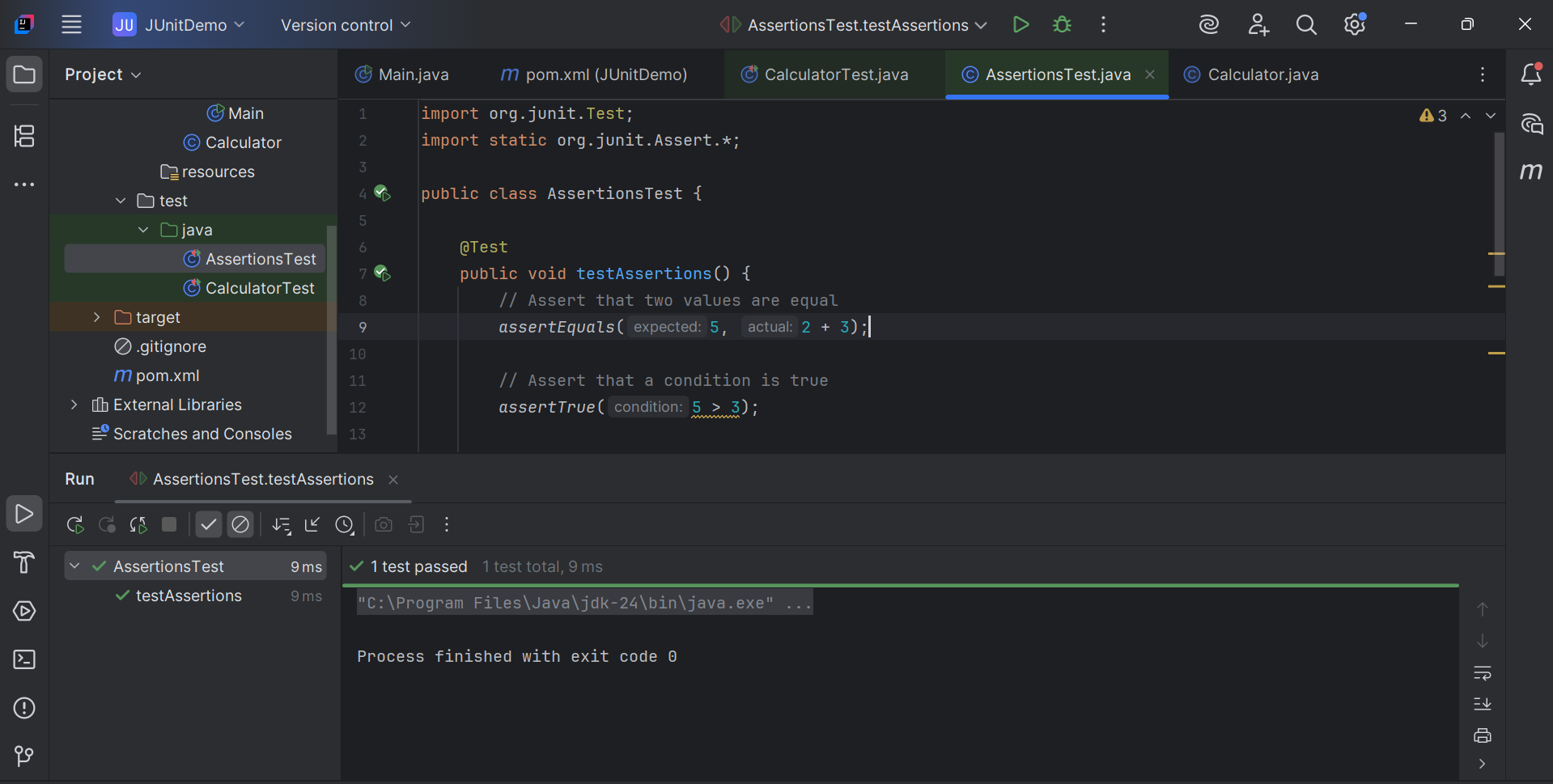
1. Write tests using various JUnit assertions.

CODE:

AssertionsTest.java

import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class AssertionsTest {  
  
 @Test  
 public void testAssertions() {  
 // Assert that two values are equal  
 *assertEquals*(5, 2 + 3);  
  
 // Assert that a condition is true  
 *assertTrue*(5 > 3);  
  
 // Assert that a condition is false  
 *assertFalse*(5 < 3);  
  
 // Assert that an object is null  
 *assertNull*(null);  
  
 // Assert that an object is not 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.

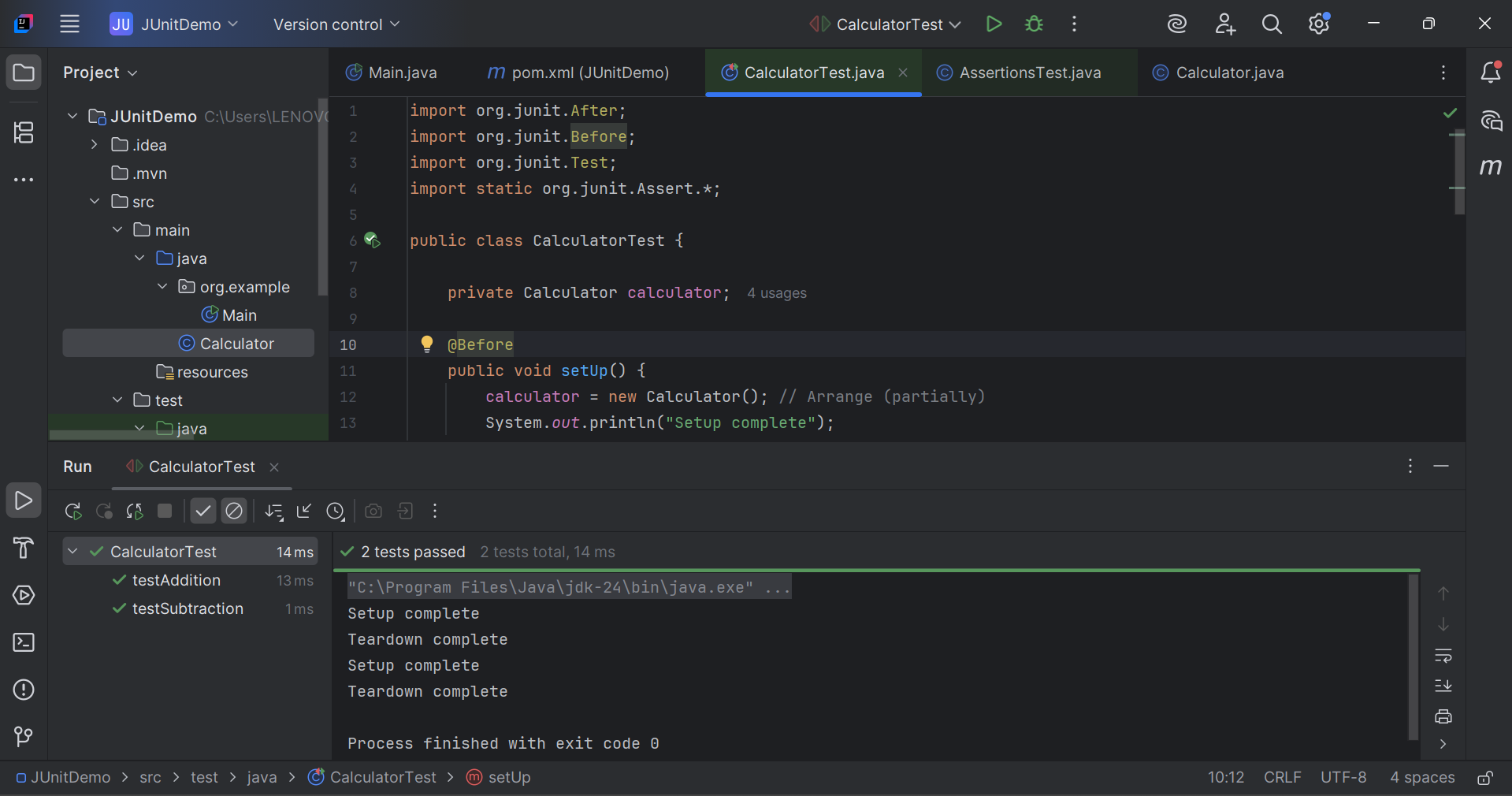
Using the same Calculator class to implement this

CODE:

CalculatorTest.java

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(); // Arrange (partially)  
 System.*out*.println("Setup complete");  
 }  
   
 @After  
 public void tearDown() {  
 calculator = null;  
 System.*out*.println("Teardown complete");  
 }  
   
 @Test  
 public void testAddition() {  
 // Arrange – done in setUp()  
  
 // Act  
 int result = calculator.add(2, 3);  
  
 // Assert  
 *assertEquals*(5, result);  
 }  
   
 @Test  
 public void testSubtraction() {  
 // Act  
 int result = calculator.subtract(10, 4);  
  
 // Assert  
 *assertEquals*(6, result);  
 }  
}

Output:



**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.

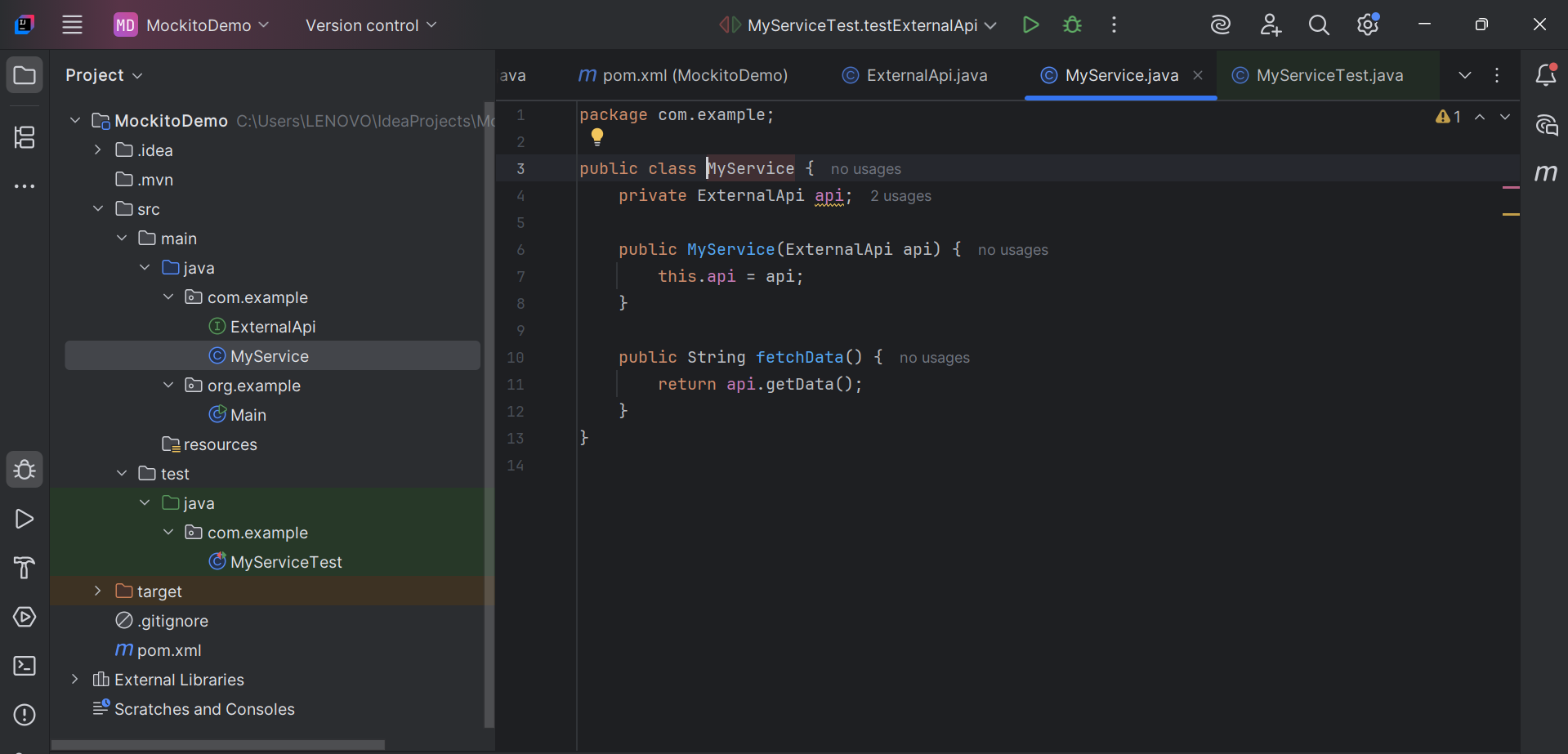
3. Write a test case that uses the mock object.

CODE:

MyServiceTest.java

package com.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() {  
 // Step 1: Create mock of ExternalApi  
 ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);  
  
 // Step 2: Stub getData() to return "Mock Data"  
 *when*(mockApi.getData()).thenReturn("Mock Data");  
  
 // Step 3: Inject mock into MyService  
 MyService service = new MyService(mockApi);  
  
 // Step 4: Call the method and verify result  
 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.

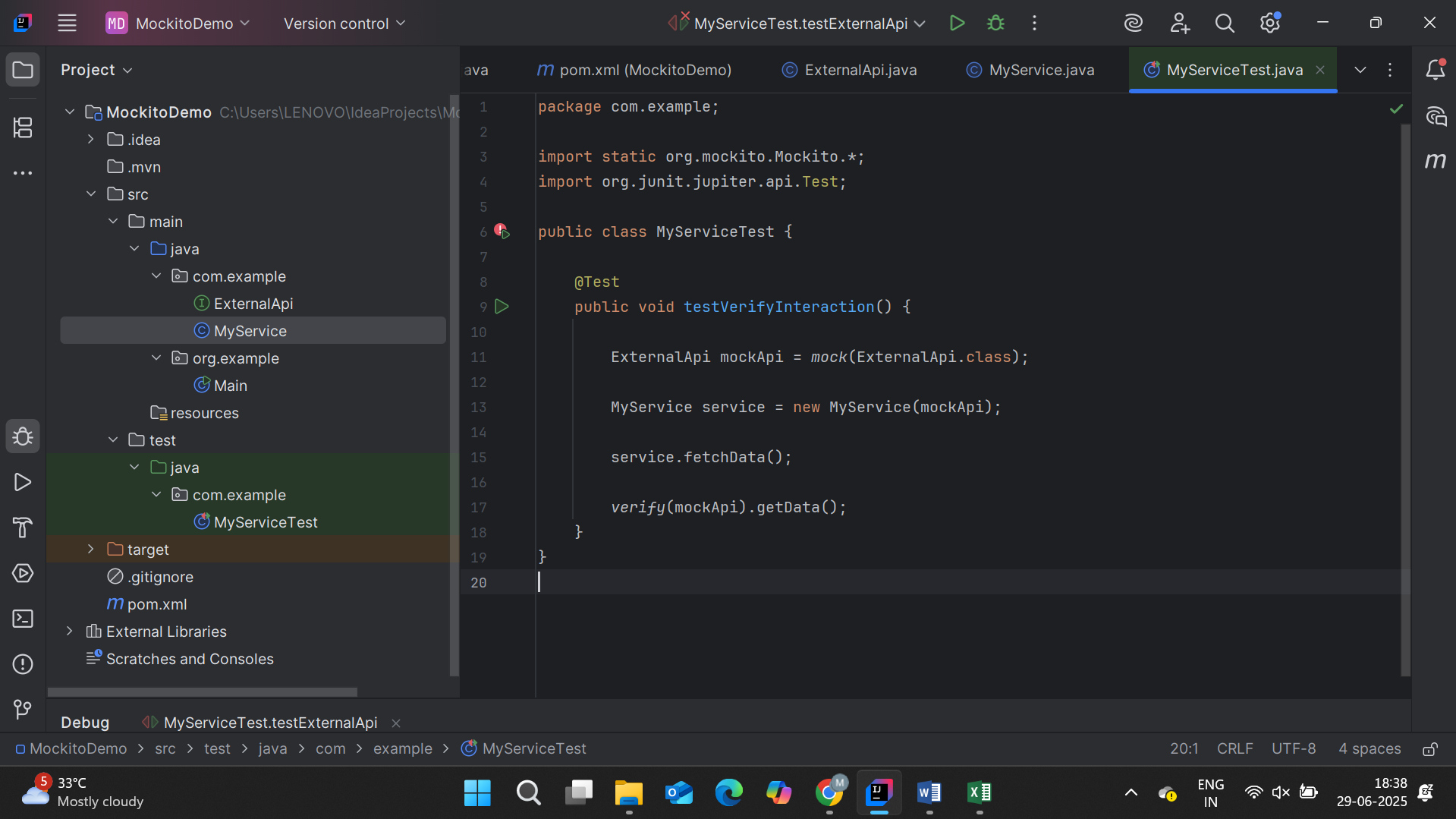
3. Verify the interaction.

CODE:

MyServiceTest.java

package com.example;  
  
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();  
 }  
}

Output:



Logging using SLF4J

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:**

Step 1: Add SLF4J and Logback Dependencies in pom.xml

Step 2: Create a Java Class for Logging

LoggingExample.java

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) {  
 // Logging error and warning messages  
 *logger*.error("This is an error message");  
 *logger*.warn("This is a warning message");  
 }  
}

Step 3: Run the Application

Output:

