**WEEK 2 – TDD USING JUNIT5 AND MOCKITO**

**EXERCISE 1: Setting Up JUnit**

**Step 1:** Create a Maven Project in Eclipse.

**Step 2:** Create a file Calculator.java

package junitdemo;

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;

}

public int square(int a) {

return a \* a;

}

}

**Step 3:** Create a test file CalculatorTest.java

package junitdemo;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

Calculator calc = new Calculator();

*@Test*

public void testAdditionPositive() {

*assertEquals*(10, calc.add(6, 4));

}

*@Test*

public void testAdditionNegative() {

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

}

*@Test*

public void testSubtraction() {

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

}

*@Test*

public void testSubtractionNegativeResult() {

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

}

*@Test*

public void testMultiplication() {

*assertEquals*(15, calc.multiply(3, 5));

}

*@Test*

public void testMultiplicationWithZero() {

*assertEquals*(0, calc.multiply(0, 100));

}

*@Test*

public void testDivision() {

*assertEquals*(4, calc.divide(8, 2));

}

*@Test*(expected = IllegalArgumentException.class)

public void testDivisionByZero() {

calc.divide(10, 0);

}

*@Test*

public void testSquarePositive() {

*assertEquals*(49, calc.square(7));

}

*@Test*

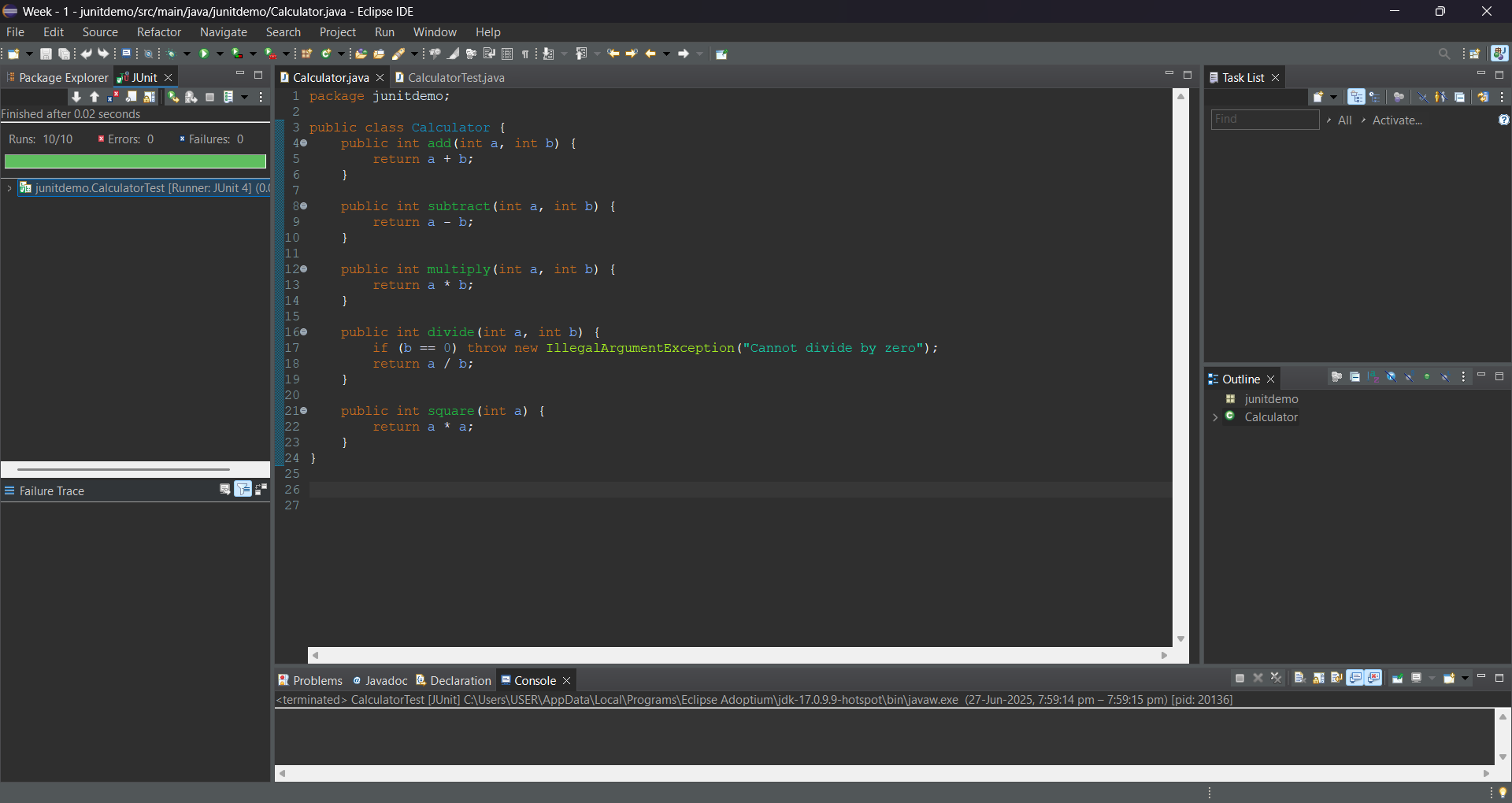
public void testSquareNegative() {

*assertEquals*(25, calc.square(-5));

}

}

**OUTPUT:**



**EXERCISE 3: ASSERTIONS IN JUnit**

**Step 1 :** Create anew Maven Project in Eclipse.

**Step 2:** Create a class SampleAssertions.java

package com.assertion;

public class SampleAssertions {

public int getSum(int a, int b) {

return a + b;

}

public boolean isPositive(int number) {

return number > 0;

}

public boolean isNegative(int number) {

return number < 0;

}

public String getNullString() {

return null;

}

public String getNotNullString() {

return "Hello";

}

}

Step 3: Create a test file AssertionsTest.java

package com.assertion;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

SampleAssertions sa = new SampleAssertions();

*@Test*

public void testSum() {

*assertEquals*(15, sa.getSum(10, 5));

}

*@Test*

public void testIsPositive() {

*assertTrue*(sa.isPositive(7));

}

*@Test*

public void testIsNegative() {

*assertFalse*(sa.isPositive(-1));

}

*@Test*

public void testNullString() {

*assertNull*(sa.getNullString());

}

*@Test*

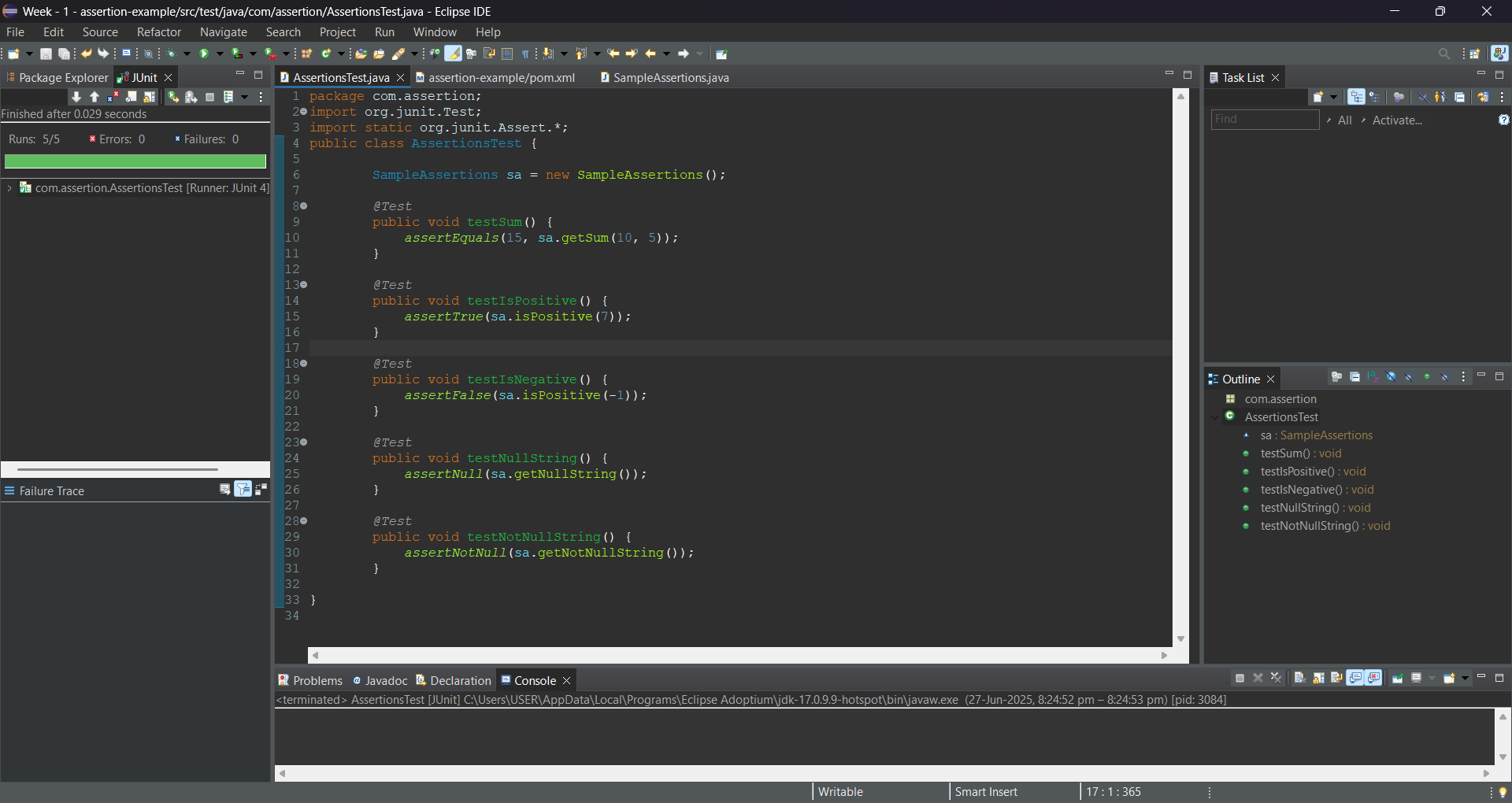
public void testNotNullString() {

*assertNotNull*(sa.getNotNullString());

}

}

**OUTPUT:**



**EXERCISE 4: ARRANGE-ACT-ASSERT (AAA) PATTERN WITH SETUP AND TEARDOWN** **IN** **JUNIT**

**Step 1: Create a Java Maven Project.**

**Step 2: Create a Main class – Calculator.java**

package junitexercise;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

}

**Step 3:** Create a Test file – CalculatorTest.java

package junitexercise;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

Calculator calc;

*@Before*

public void setUp() {

System.***out***.println("Setting up Calculator...");

calc = new Calculator();

}

*@After*

public void tearDown() {

System.***out***.println("Tearing down Calculator...\n");

calc = null;

}

*@Test*

public void testAdd() {

// Arrange

int a = 5, b = 3;

// Act

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

// Assert

*assertEquals*(8, result);

}

*@Test*

public void testMultiply() {

// Arrange

int x = 4, y = 2;

// Act

int result = calc.multiply(x, y);

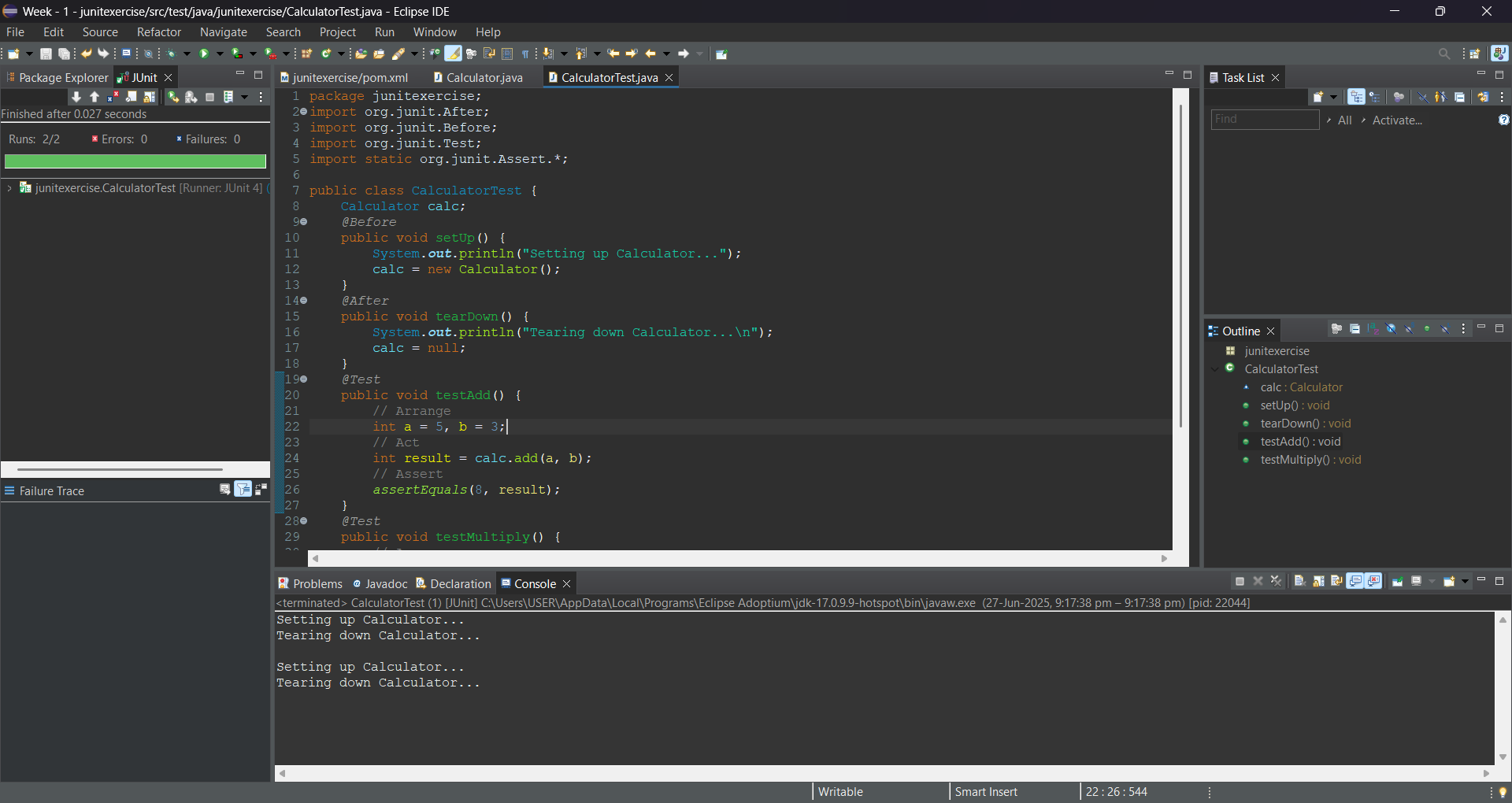
// Assert

*assertEquals*(8, result);

}

}

**OUTPUT:**



**MOCKITO EXERCISES**

**EXERCISE 1: MOCKING AND STUBBING**

**Step 1:** Create a java maven project in eclipse.

**Step 2:** Update the pom.xml file and update the Maven project.

**Step 3:** Create a ExternalApi.java in src/main/java

package com.example;

public interface ExternalApi {

String getData();

}

**Step 4:** Create a Service.java file in src/main/java

package com.example;

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**Step 5:** Create a ServiceTest.java file in src/test/java

package com.example;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import static org.mockito.Mockito.\*;

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

public class MyServiceTest {

*@Test*

public void testExternalApi() {

// Step 1: Create mock

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

// Step 2: Stub method

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

// Step 3: Use mock in service

MyService service = new MyService(mockApi);

String result = service.fetchData();

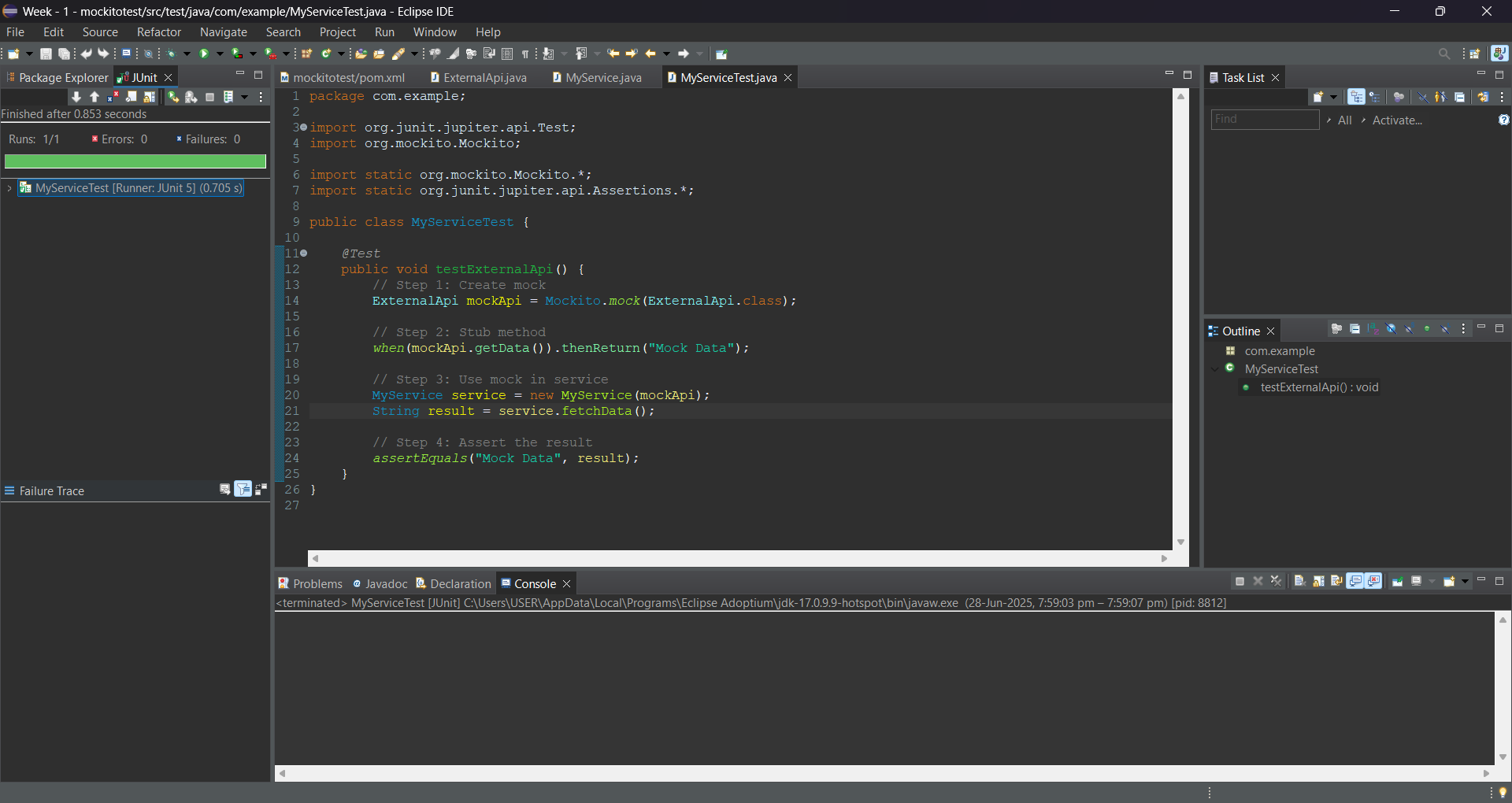
// Step 4: Assert the result

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

}

}

**Output:**



**EXERCISE 2: VERIFYING INTERACTIONS**

**Step 1:** Create a java maven project in eclipse.

**Step 2:** Update the pom.xml file and update the Maven project.

**Step 3:** Create a ExternalApi.java in src/main/java

package com.example;

public interface ExternalApi {

String getData();

}

**Step 4:** Create a Service.java file in src/main/java

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**Step 5:** Create a ServiceTest.java file in src/test/java

package com.example;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

*@Test*

public void testVerifyInteraction() {

// Step 1: Mock the ExternalApi

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

// Step 2: Pass mock to service

MyService service = new MyService(mockApi);

service.fetchData();

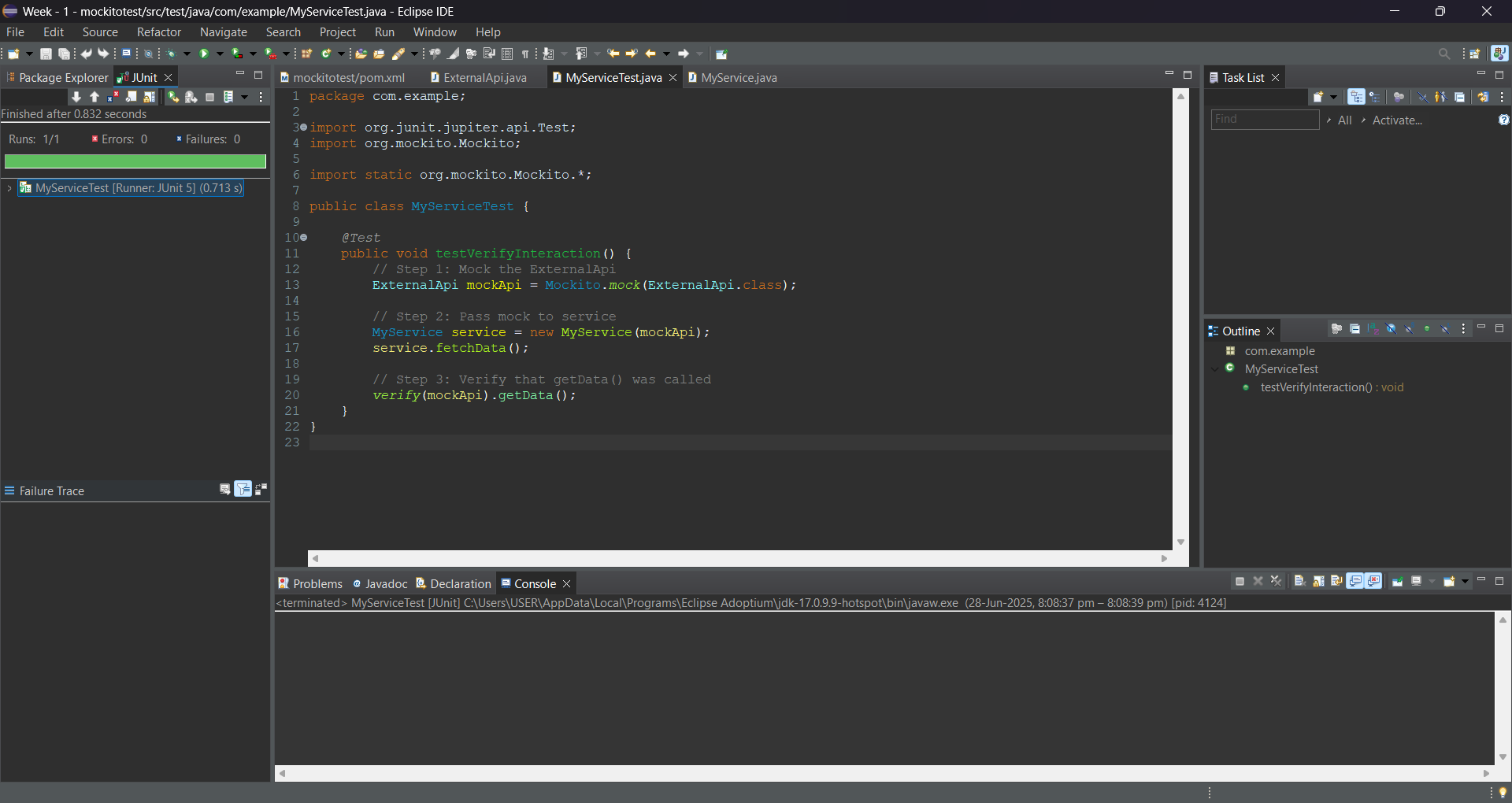
// Step 3: Verify that getData() was called

*verify*(mockApi).getData();

}

}

**OUTPUT:**

****