JUnit Testing Exercises

**Exercise 1:**

**Setting Up JUnit Scenario: You need to set up JUnit in your Java project to start writing unit tests.**

**Code:**

**Testcase.java:**

package com.example.test;

import static org.junit.Assert.\*;

import org.junit.Test;

public class Testcase {

@Test

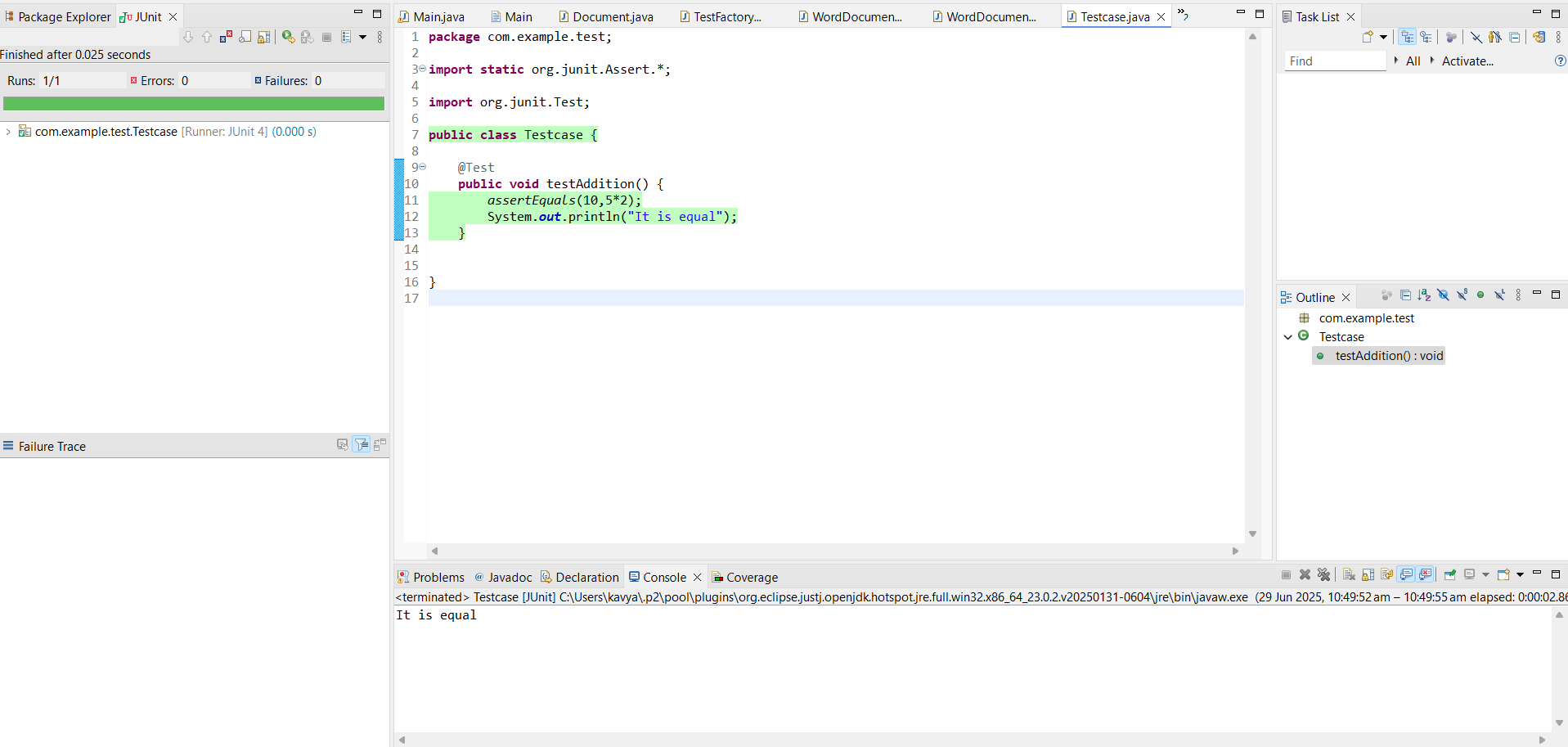
public void testAddition() {

*assertEquals*(10,5\*2);

}

}

**Output:**



**Exercise 3: Assertions in JUnit Scenario:**

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

**Code: Assertionstest.java**

package com.example.test;

import static org.junit.Assert.\*;

import org.junit.Test;

public class Assertionstest {

@Test

public void testAssertions() {

*assertEquals*(25, 5\*5);

System.*out*.println("5\*5 equals 25");

*assertTrue*(10 > 9);

System.*out*.println("10 is greater than 9");

*assertFalse*(25 > 50);

System.*out*.println("25 is not greater than 50");

*assertNull*(null);

System.*out*.println("Value is null");

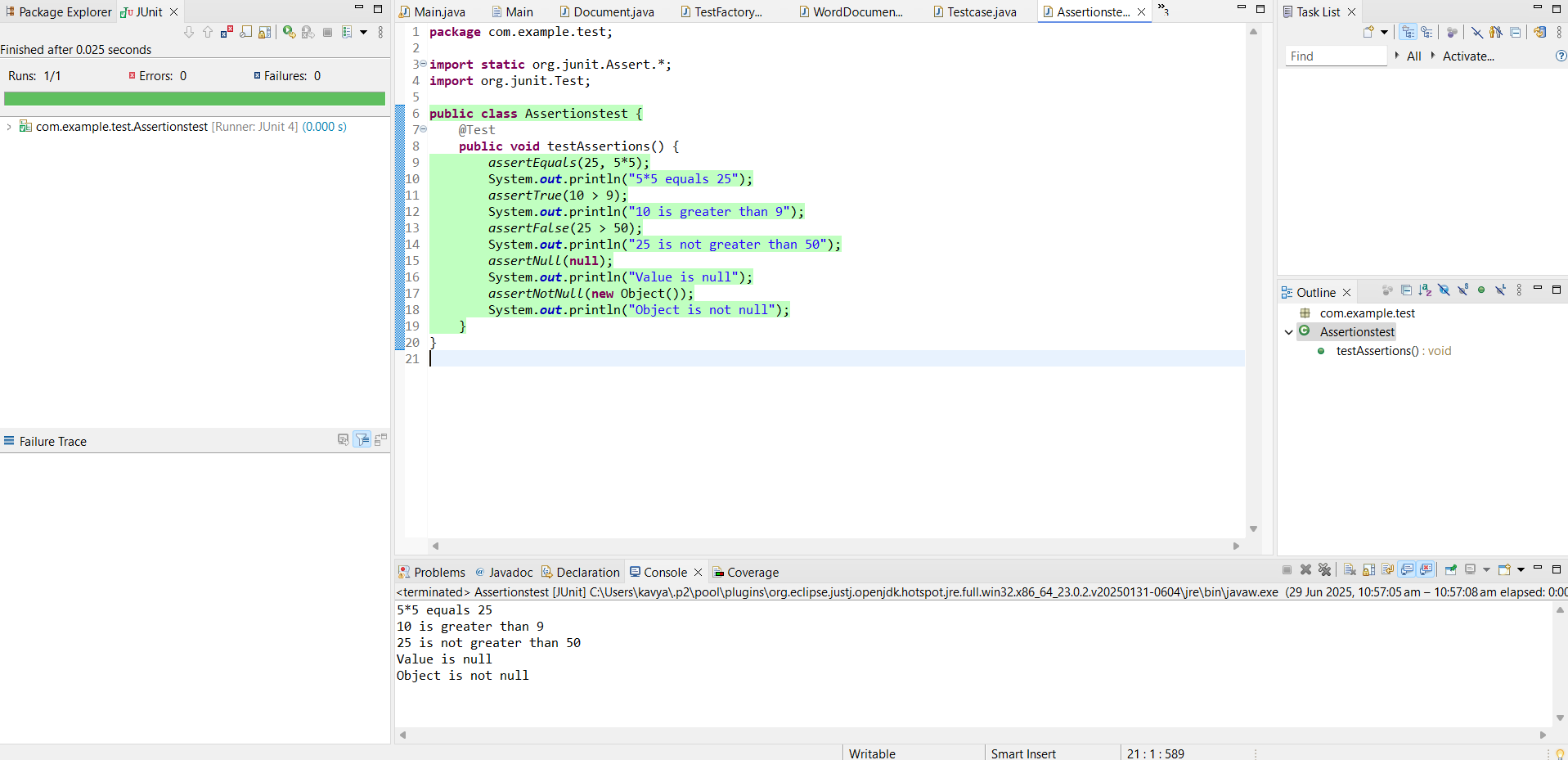
*assertNotNull*(new Object());

System.*out*.println("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:**

**package** com.example.test;

**import** **static** org.junit.Assert.\*;

**import** org.junit.Before;

**import** org.junit.After;

**import** org.junit.Test;

**public** **class** CalculatorTest {

**private** **int** value;

@Before

**public** **void** setUp() {

value = 10;

System.***out***.println("Setup:The value is initialized");

}

@After

**public** **void** tearDown() {

System.***out***.println("Teardown:The test is completed");

}

@Test

**public** **void** testAddition() {

**int** result = value \* 10;

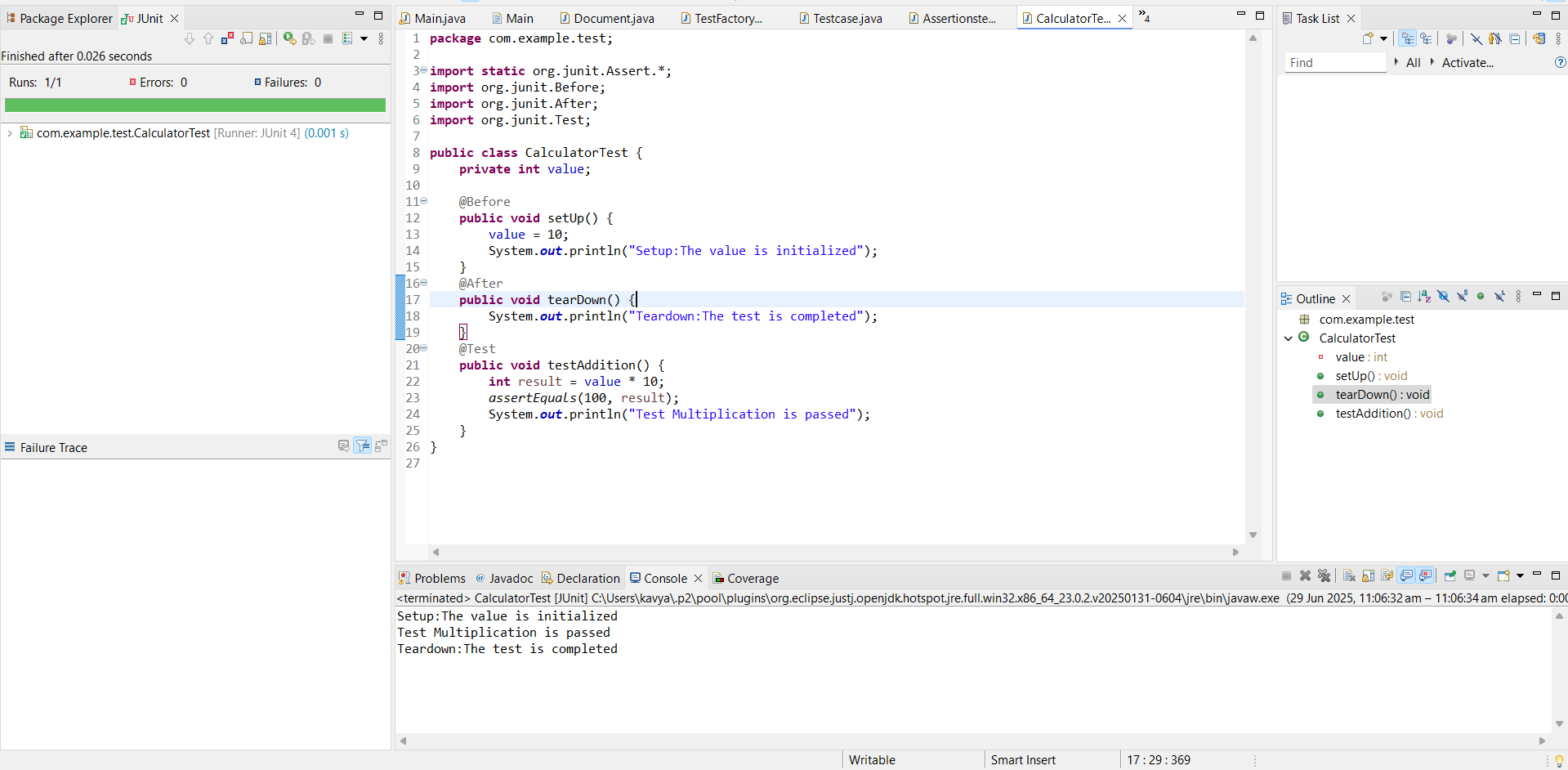
*assertEquals*(100, result);

System.***out***.println("Test Multiplication is passed");

}

}

**Output:**



# 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.**

**Code:**

**ExternalApi.java**

package com.example.mockito;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example.mockito;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

package com.example.mockito.mockitotest;

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

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testExternalApi() {

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

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

System.*out*.println("Created mock for ExternalApi");

MyService service = new MyService(mockApi);

System.*out*.println("Created MyService with mock ExternalApi");

String result = service.fetchData();

System.*out*.println("fetchData() returned: " + result);

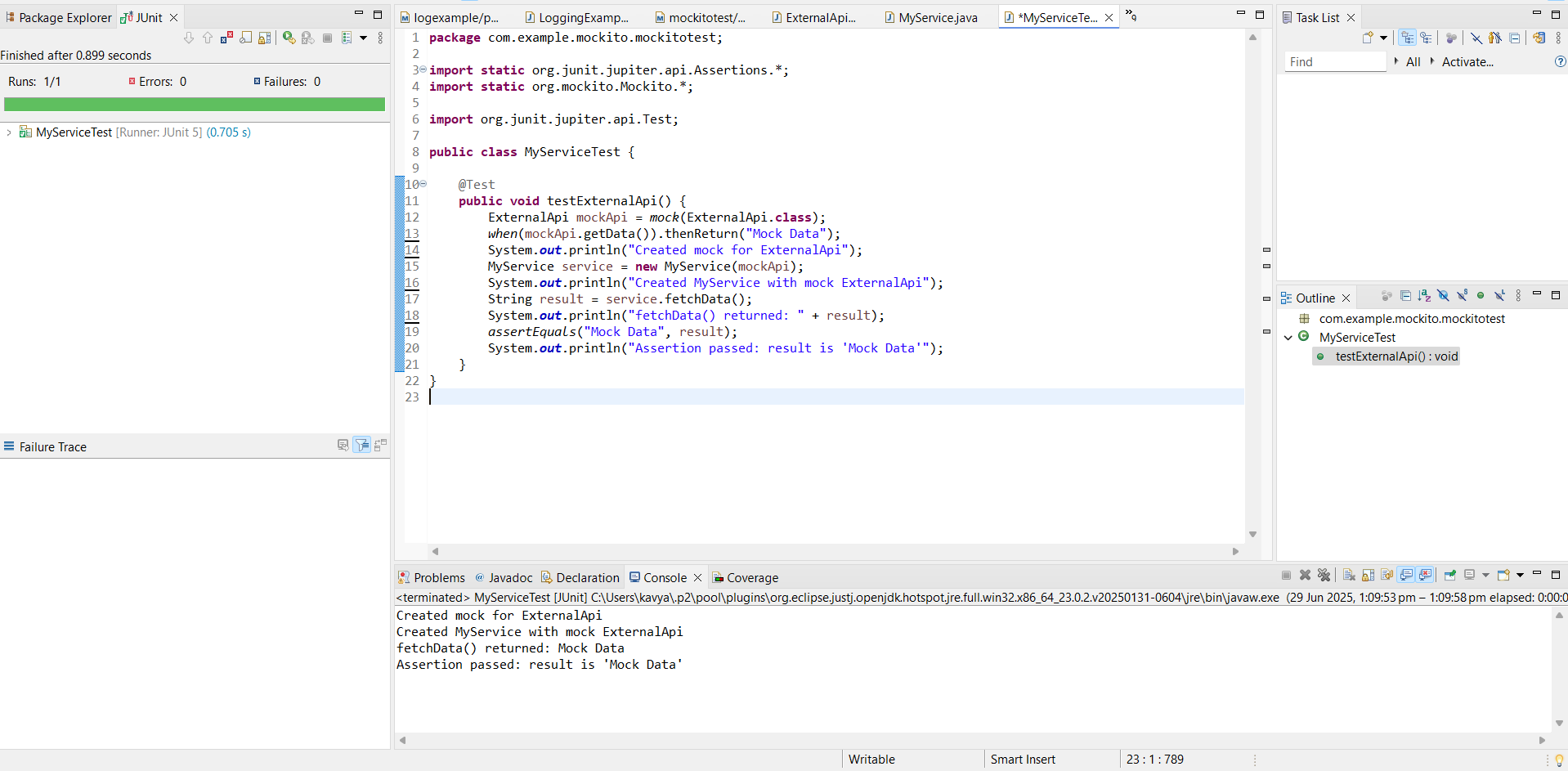
*assertEquals*("Mock Data", result);

System.*out*.println("Assertion passed: result is 'Mock Data'");

}

**}**

**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:**

**ExternalApi.java**

package com.example.mockito;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example.mockito;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

package com.example.mockito.mockitotest;

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();

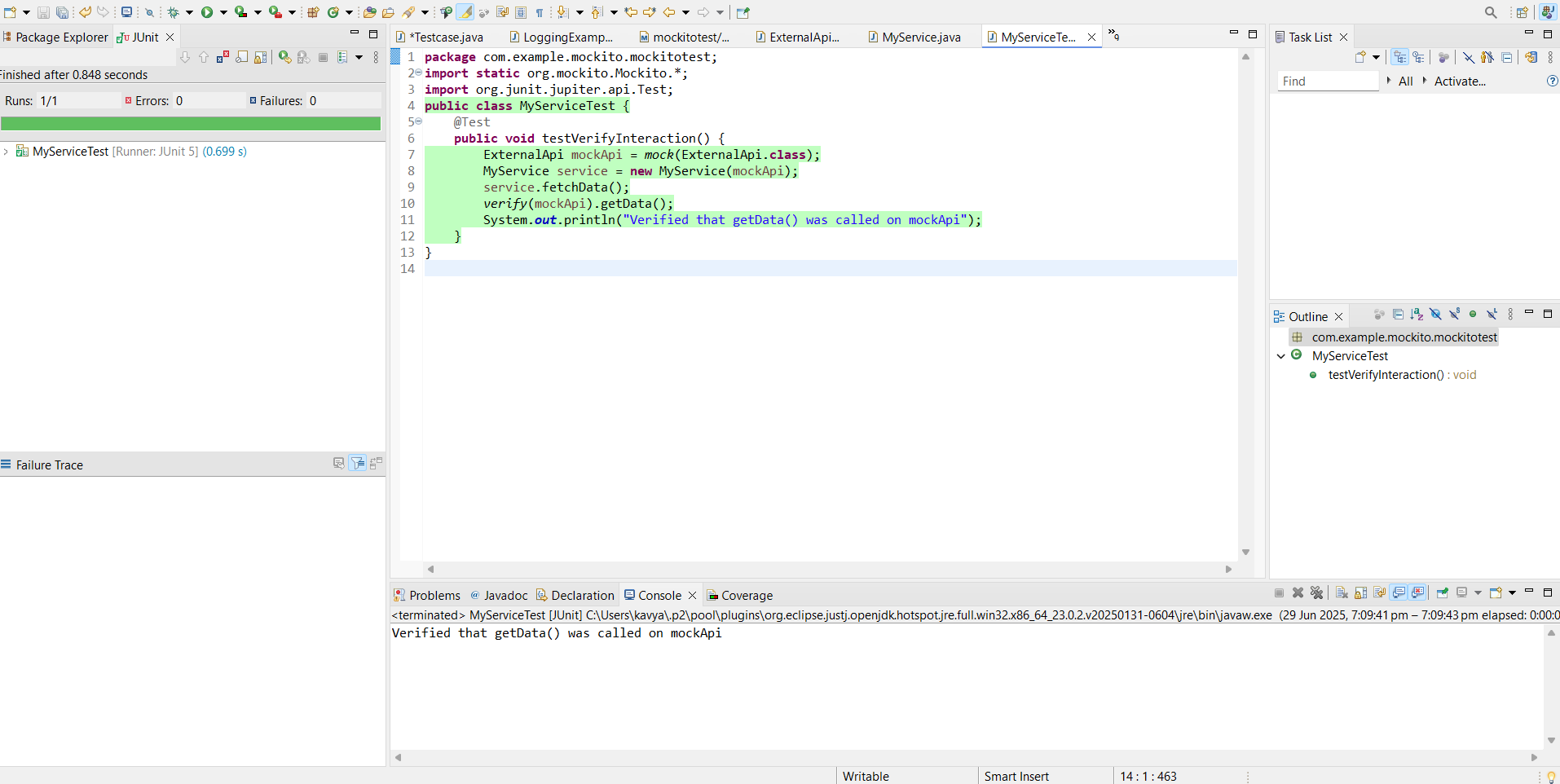
*verify*(mockApi).getData();

System.*out*.println("Verified that getData() was called on mockApi");

}

}

**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.**

**Code:**

package com.example.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");

System.*out*.println("Demonstration completed");

}

}

**Output:**

