COGNIZANT

Digital Nurture 4.0

Deep Skilling - Java FSE

WEEK-2 HANDS ON

By Kaviya P

**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.

**ExternalApi.java**

**package** org.sample.my\_first\_maven;

**public** **interface** ExternalApi {

String getData();

}

**MyService.java**

**package** org.sample.my\_first\_maven;

**public** **class** MyService {

**private** ExternalApi externalApi;

**public** MyService(ExternalApi externalApi) {

**this**.externalApi = externalApi;

}

**public** String fetchData() {

**return** externalApi.getData();

}

}

**MyServiceTest.java**

**package** org.sample.my\_first\_maven;

**import** org.sample.my\_first\_maven.ExternalApi;

**import** **static** org.mockito.Mockito.\*;

**import** **static** org.junit.jupiter.api.Assertions.*assertEquals*; //

**import** org.junit.jupiter.api.Test;

**import** org.mockito.Mockito;

**public** **class** MyServiceTest {

@Test

**public** **void** testExternalApi() {

// Step 1: Create a mock of ExternalApi

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

// Step 2: Stub the getData() method

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

// Step 3: Use mock in MyService

MyService service = **new** MyService(mockApi);

String result = service.fetchData();

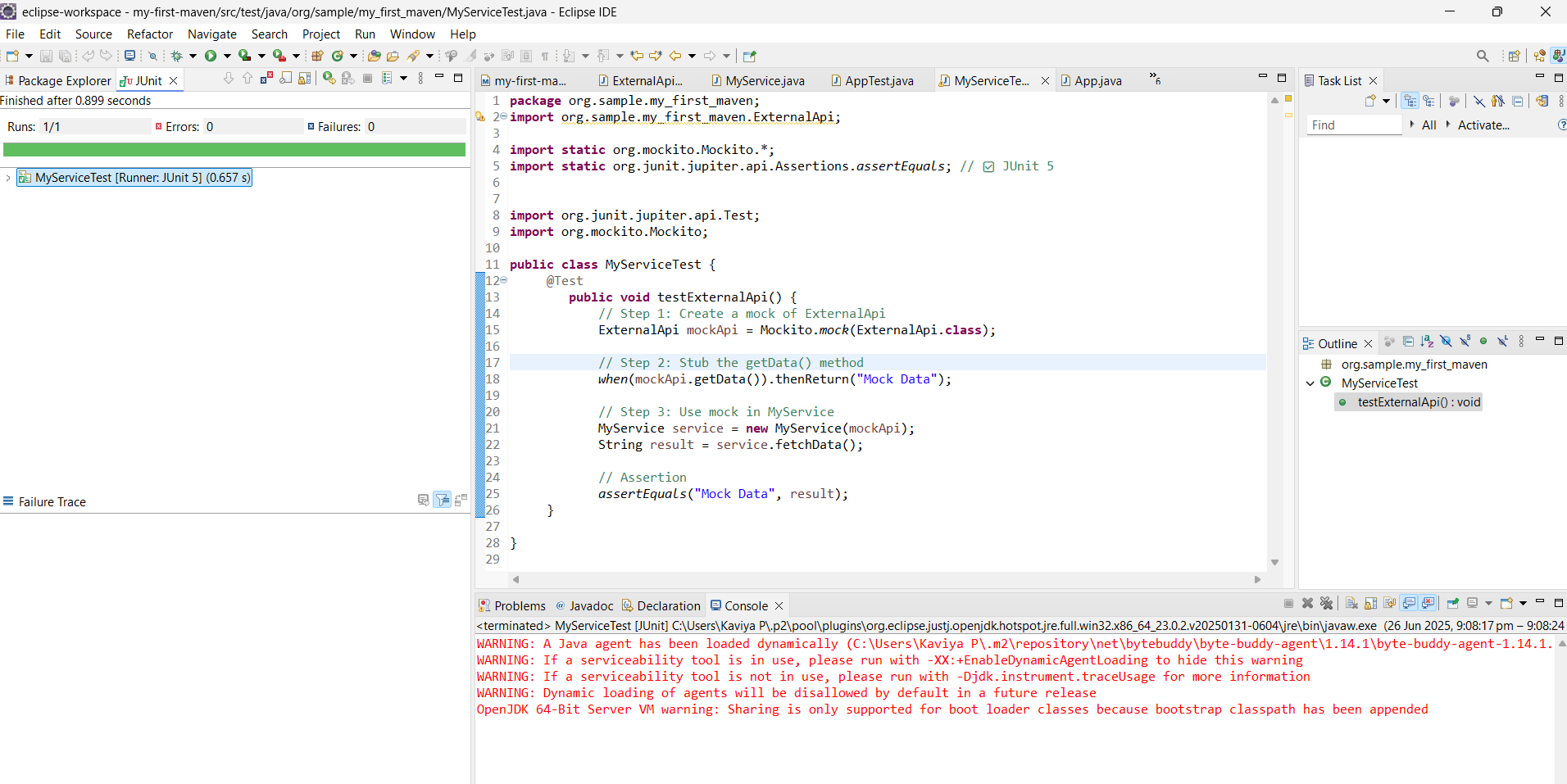
// Assertion

*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.**

package org.sample.my\_first\_maven;

import org.sample.my\_first\_maven.ExternalApi;

import static org.mockito.Mockito.\*;

import static org.junit.jupiter.api.Assertions.assertEquals; //

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);

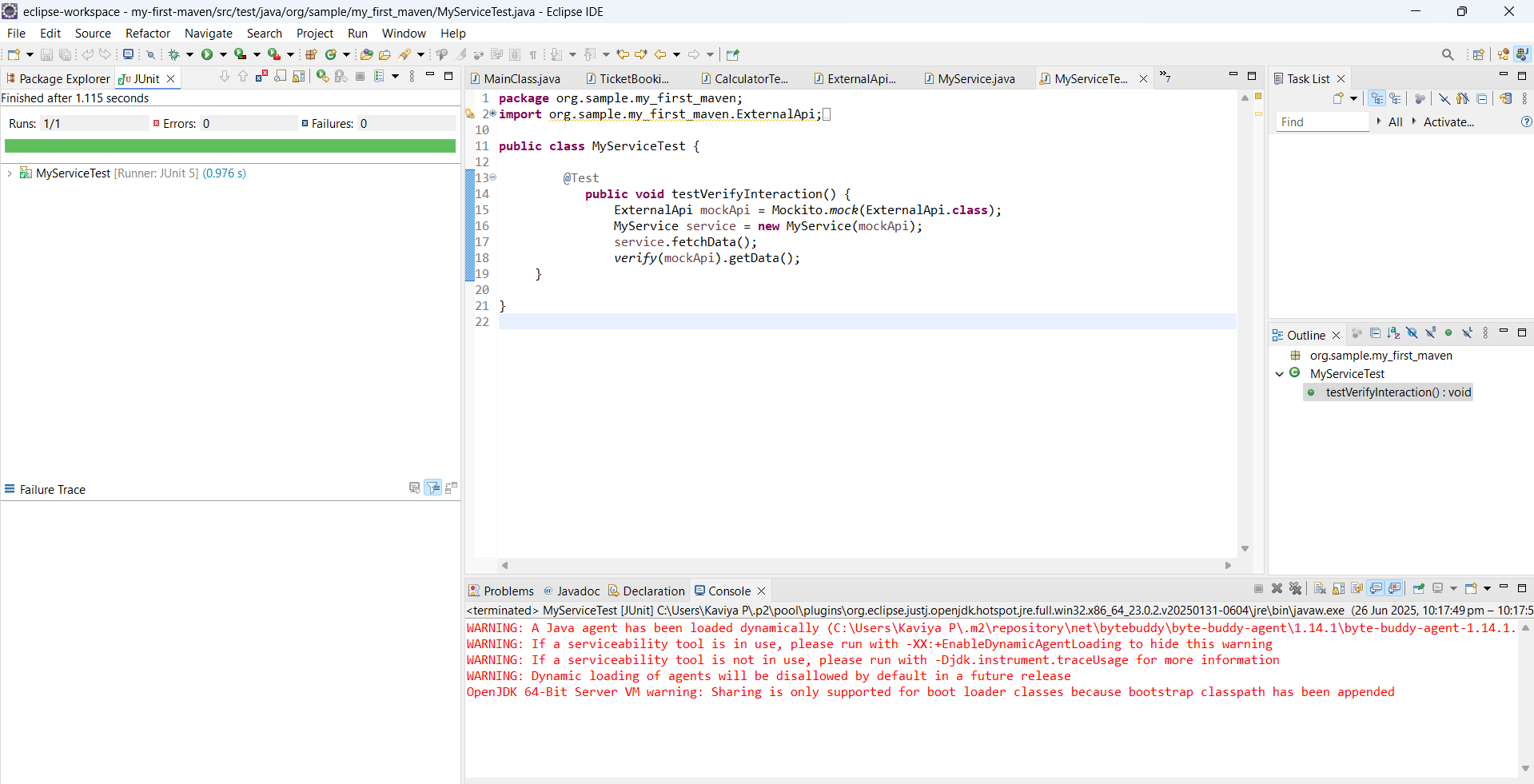
                service.fetchData();

                verify(mockApi).getData();

     }

}

**OUTPUT**



**Exercise 3: Argument Matching**

**Scenario:**

**You need to verify that a method is called with specific arguments.**

**Steps:**

**1. Create a mock object.**

**2. Call the method with specific arguments.**

**3. Use argument matchers to verify the interaction**.

**MyCalculator.java**

**package** org.sample.my\_first\_maven;

**public** **interface** MyCalculator {

**int** add(**int** a, **int** b);

}

**CalculatorService**

**package** org.sample.my\_first\_maven;

**public** **class** CalculatorService {

**private** MyCalculator calculator;

**public** CalculatorService(MyCalculator calculator) {

**this**.calculator = calculator;

}

**public** **int** performAdd(**int** x, **int** y) {

**return** calculator.add(x, y);

}

}

**CalculatorTest.java**

**package** org.sample.my\_first\_maven;

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

**import** **static** org.mockito.ArgumentMatchers.*anyInt*;

**import** **static** org.mockito.ArgumentMatchers.*eq*;

**import** **static** org.mockito.Mockito.*mock*;

**import** **static** org.mockito.Mockito.*verify*;

**import** **static** org.mockito.Mockito.*when*;

**import** org.junit.Before;

**import** org.junit.Test;

**import** org.mockito.Mockito;

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

@Test

**public** **void** testPerformAddWithArgumentMatchers() {

// Step 1: Create mock of MyCalculator

MyCalculator mockCalculator = Mockito.*mock*(MyCalculator.**class**);

// Step 2: Stub the add() method using argument matchers

*when*(mockCalculator.add(*anyInt*(), *eq*(5))).thenReturn(10);

// Step 3: Inject mock into service

CalculatorService service = **new** CalculatorService(mockCalculator);

// Step 4: Call the service method

**int** result = service.performAdd(100, 5);

// Step 5: Assert the result

*assertEquals*(10, result);

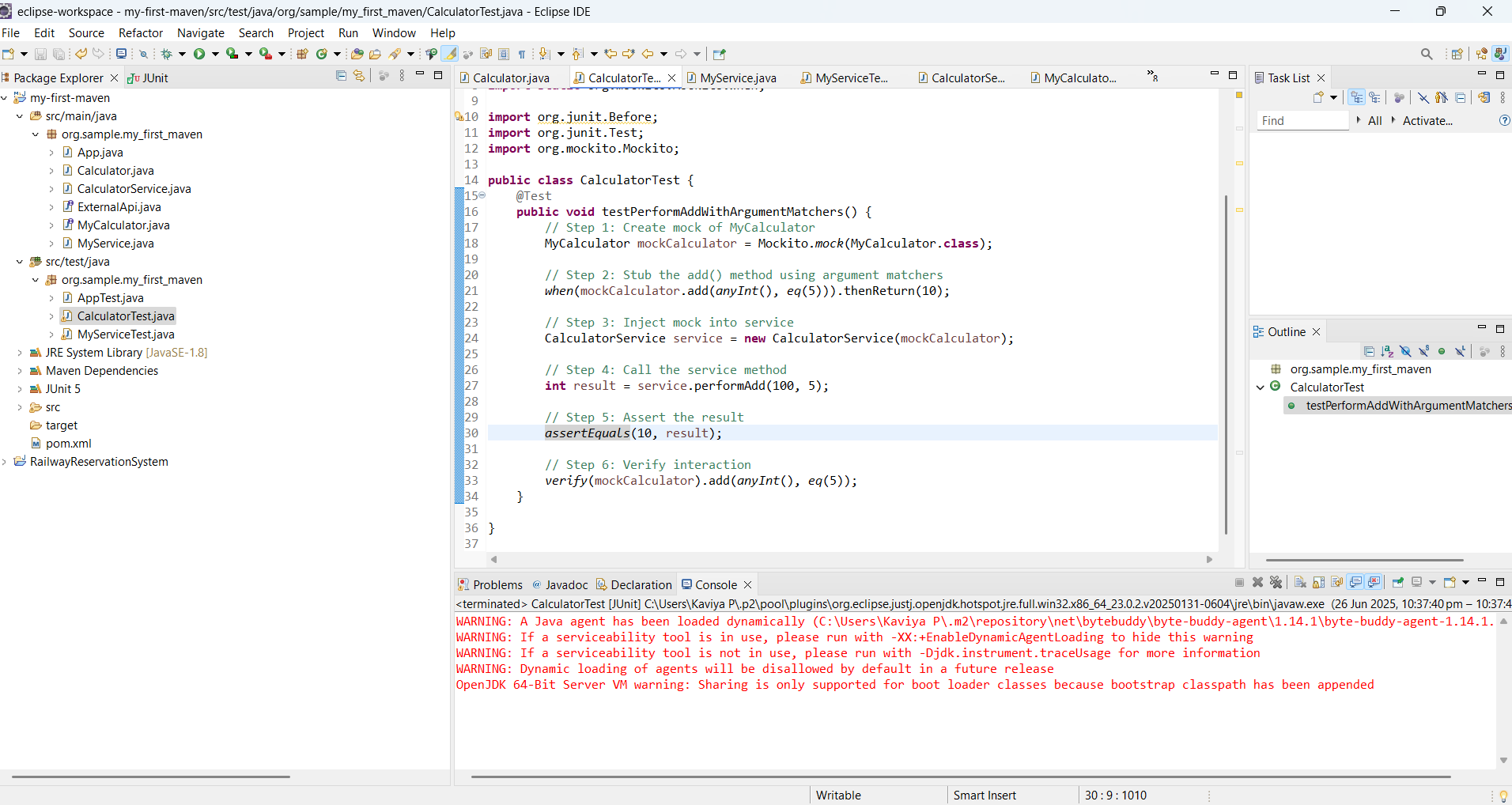
// Step 6: Verify interaction

*verify*(mockCalculator).add(*anyInt*(), *eq*(5));

}

}

**OUTPUT**



Exercise 4: Handling Void Methods

Scenario:

You need to test a void method that performs some action.

Steps:

1. Create a mock object.

2. Stub the void method.

3. Verify the interaction.

**Logger.java**

**package** org.sample.my\_first\_maven;

**public** **interface** Logger {

**void** log(String message);

}

**LogService.java**

**package** org.sample.my\_first\_maven;

**public** **class** LogService {

**private** Logger logger;

**public** LogService(Logger logger) {

**this**.logger = logger;

}

**public** **void** doSomething() {

logger.log("Action performed");

}

}

**LogServiceTest.java**

**package** org.sample.my\_first\_maven;

**import** **static** org.mockito.Mockito.*mock*;

**import** **static** org.mockito.Mockito.*verify*;

**import** org.junit.Test;

**import** org.mockito.Mockito;

**public** **class** LogServiceTest {

@Test

**public** **void** testMethod() {

Logger mockLogger = *mock*(Logger.**class**);

LogService service = **new** LogService(mockLogger);

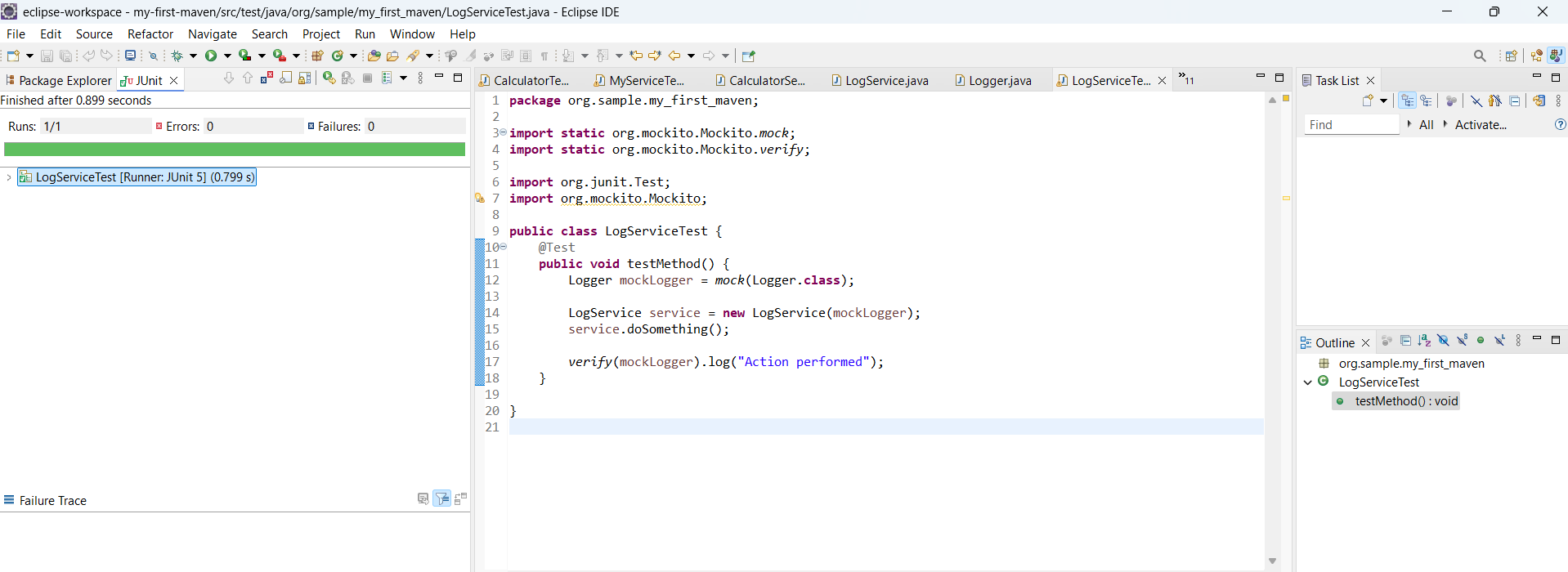
service.doSomething();

*verify*(mockLogger).log("Action performed");

}

}

**OUTPUT**



Exercise 5: Mocking and Stubbing with Multiple Returns

Scenario:

You need to test a service that depends on an external API with multiple return values.

**ExternalApi.java**

**package** org.sample.my\_first\_maven;

**public** **interface** ExternalApi {

String getStatus();

}

**MyServiceTest.java**

**package org.sample.my\_first\_maven;**

**import org.sample.my\_first\_maven.ExternalApi;**

**import static org.mockito.Mockito.\*;**

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

**import org.junit.jupiter.api.Test;**

**import org.mockito.Mockito;**

**public class MyServiceTest {**

**@Test**

**public void testMultipleReturns() {**

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

**when(mockApi.getStatus()).thenReturn("Loading", "Processing", "Done");**

**assertEquals("Loading", mockApi.getStatus());**

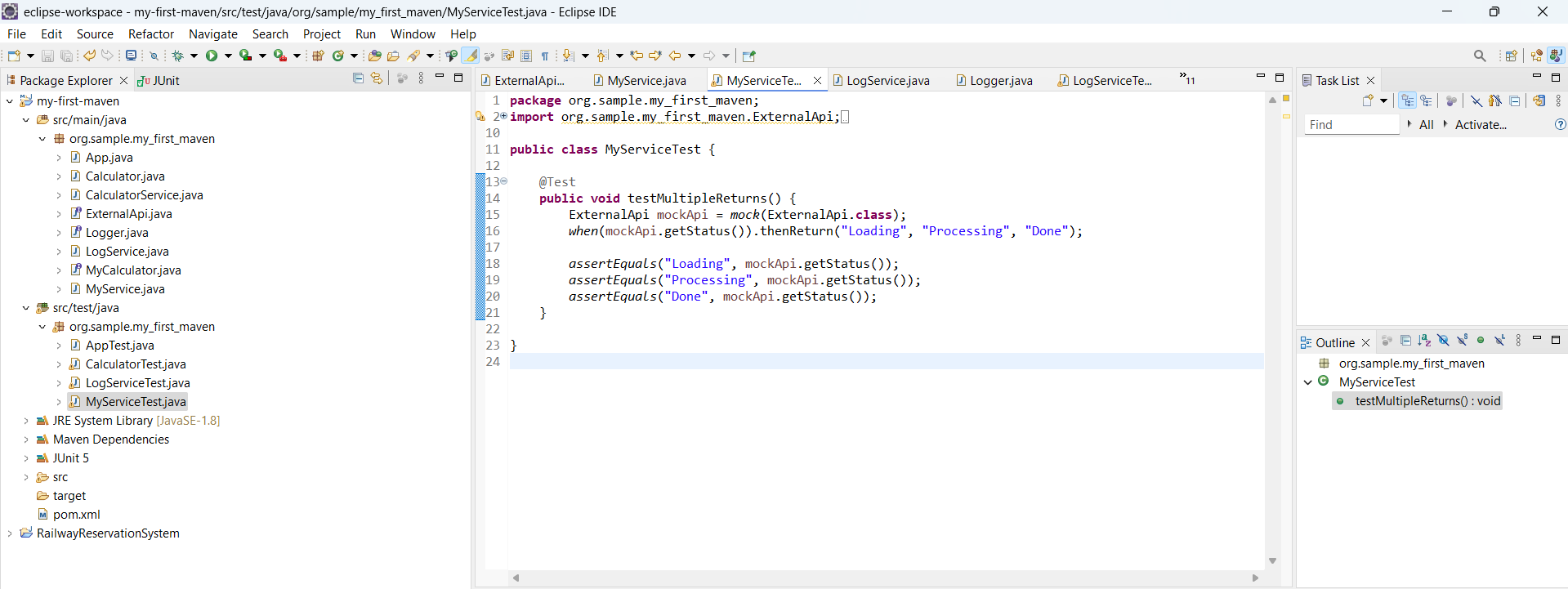
**assertEquals("Processing", mockApi.getStatus());**

**assertEquals("Done", mockApi.getStatus());**

**}**

**}**

**OUTPUT**



Exercise 6: Verifying Interaction Order

Scenario:

You need to ensure that methods are called in a specific order.

Steps:

1. Create a mock object.

2. Call the methods in a specific order.

3. Verify the interaction order.

**Audit.java**

**package** org.sample.my\_first\_maven;

**public** **interface** Audit {

**void** start();

**void** process();

**void** end();

}

**InteractionTest.java**

**package** org.sample.my\_first\_maven;

**import** **static** org.mockito.Mockito.*mock*;

**import** org.junit.Test;

**import** org.mockito.InOrder;

**import** org.mockito.Mockito;

**public** **class** InteractionTest {

@Test

**public** **void** testInteractionOrder() {

Audit mockAudit = *mock*(Audit.**class**);

mockAudit.start();

mockAudit.process();

mockAudit.end();

InOrder inOrder = Mockito.*inOrder*(mockAudit);

inOrder.verify(mockAudit).start();

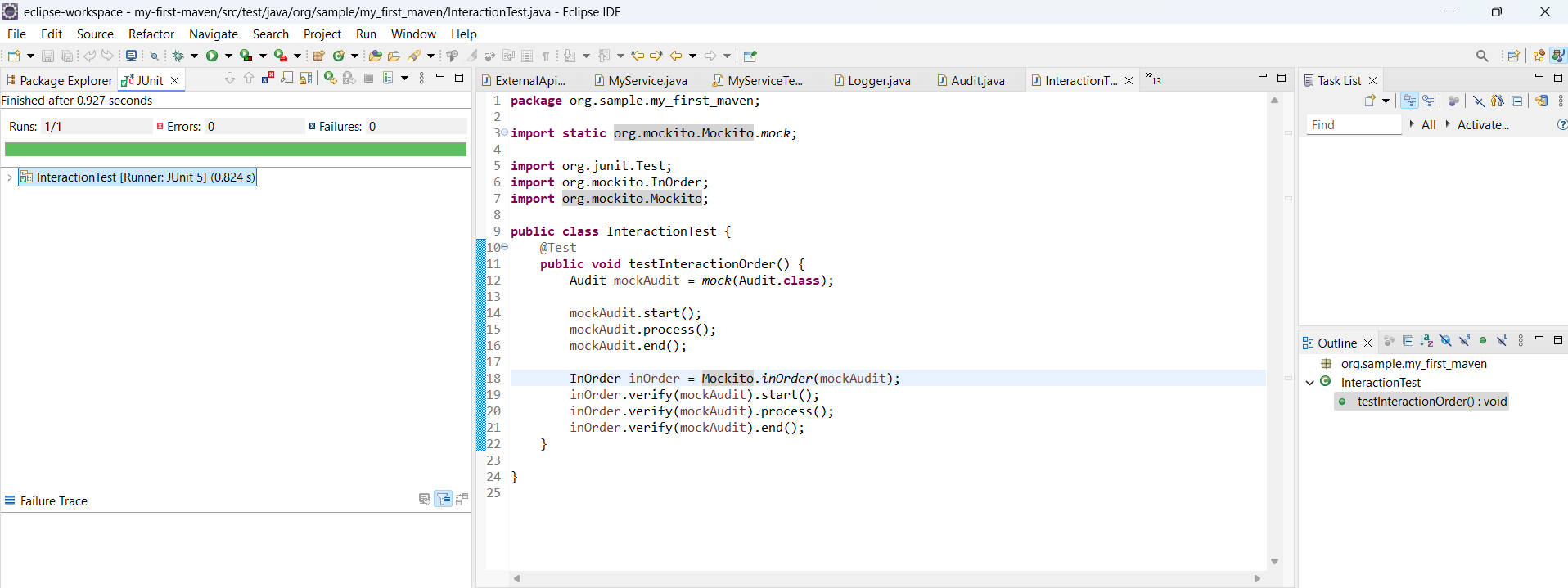
inOrder.verify(mockAudit).process();

inOrder.verify(mockAudit).end();

}

}

**OUTPUT**



Exercise 7: Handling Void Methods with Exceptions

Scenario:

You need to test a void method that throws an exception.

Steps:

1. Create a mock object.

2. Stub the void method to throw an exception.

3. Verify the interaction.

**NotificationService.java**

**package** org.sample.my\_first\_maven;

**public** **interface** NotificationService {

**void** sendEmail(String recipient);

}

**ExceptionVoidTest.java**

**package** org.sample.my\_first\_maven;

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

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

**import** **static** org.mockito.Mockito.*doThrow*;

**import** **static** org.mockito.Mockito.*mock*;

**import** **static** org.mockito.Mockito.*verify*;

**import** org.junit.Test;

**public** **class** ExceptionVoidTest {

@Test

**public** **void** testVoidMethodThrowsException() {

NotificationService mockService = *mock*(NotificationService.**class**);

*doThrow*(**new** RuntimeException("Email failed")).when(mockService).sendEmail("test@example.com");

RuntimeException exception = *assertThrows*(RuntimeException.**class**, () -> {

mockService.sendEmail("test@example.com");

});

*assertEquals*("Email failed", exception.getMessage());

*verify*(mockService).sendEmail("test@example.com");

}

}

**OUTPUT**

