**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.

**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);

}

}

**Program:**

ExternalApi.java

package com.example;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example;

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**MyServiceTest.java**

package com.example;

import org.junit.Test;

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

import static org.mockito.Mockito.\*;

public class MyServiceTest {

*@Test*

public void testExternalApi() {

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

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

MyService service = new MyService(mockApi);

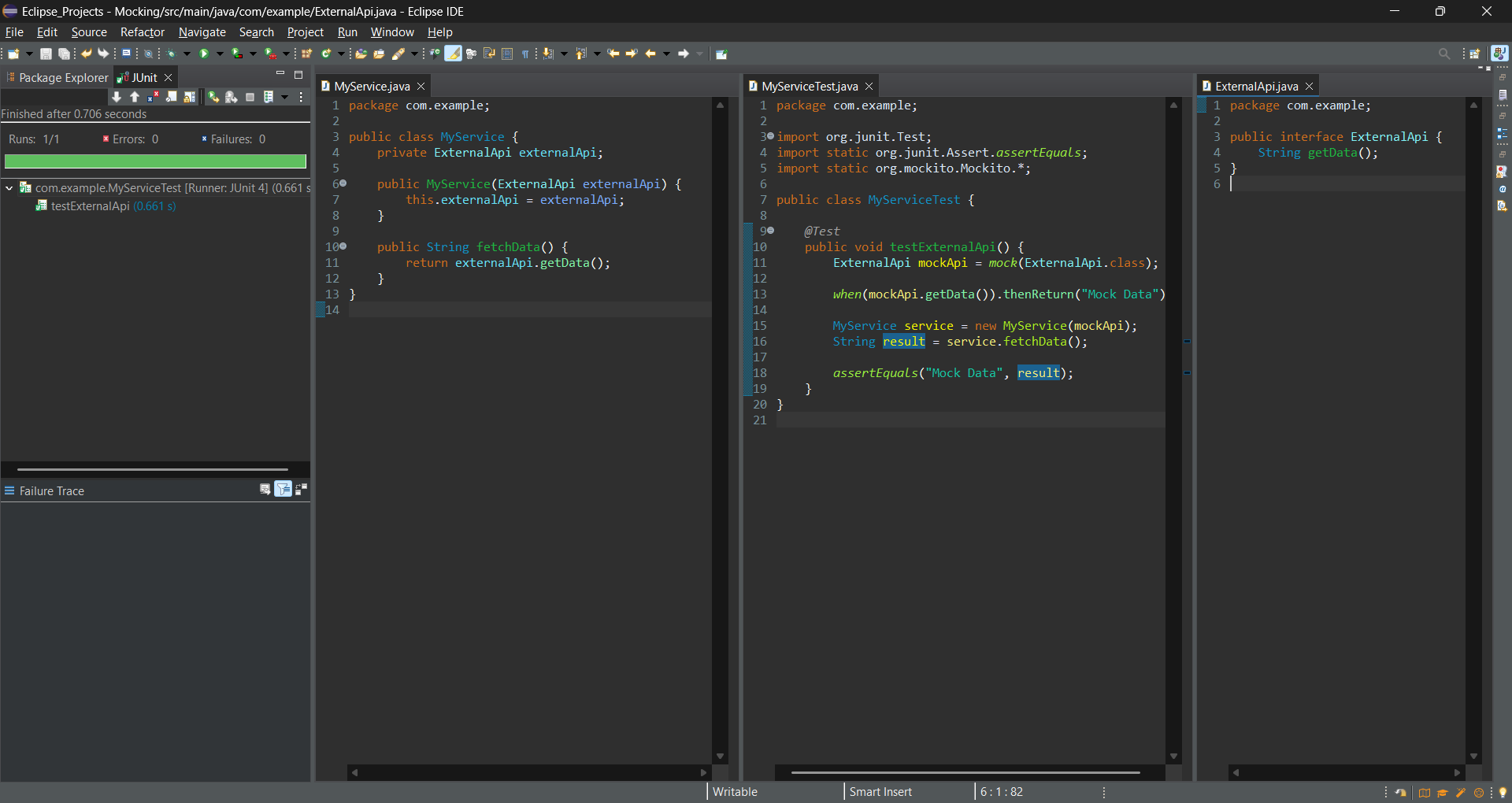
String result = service.fetchData();

*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.

**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();

}

}

**Program:**

ExternalApi.java

package com.example;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example;

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**MyServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

*@Test*

public void testVerifyInteraction() {

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

MyService service = new MyService(mockApi);

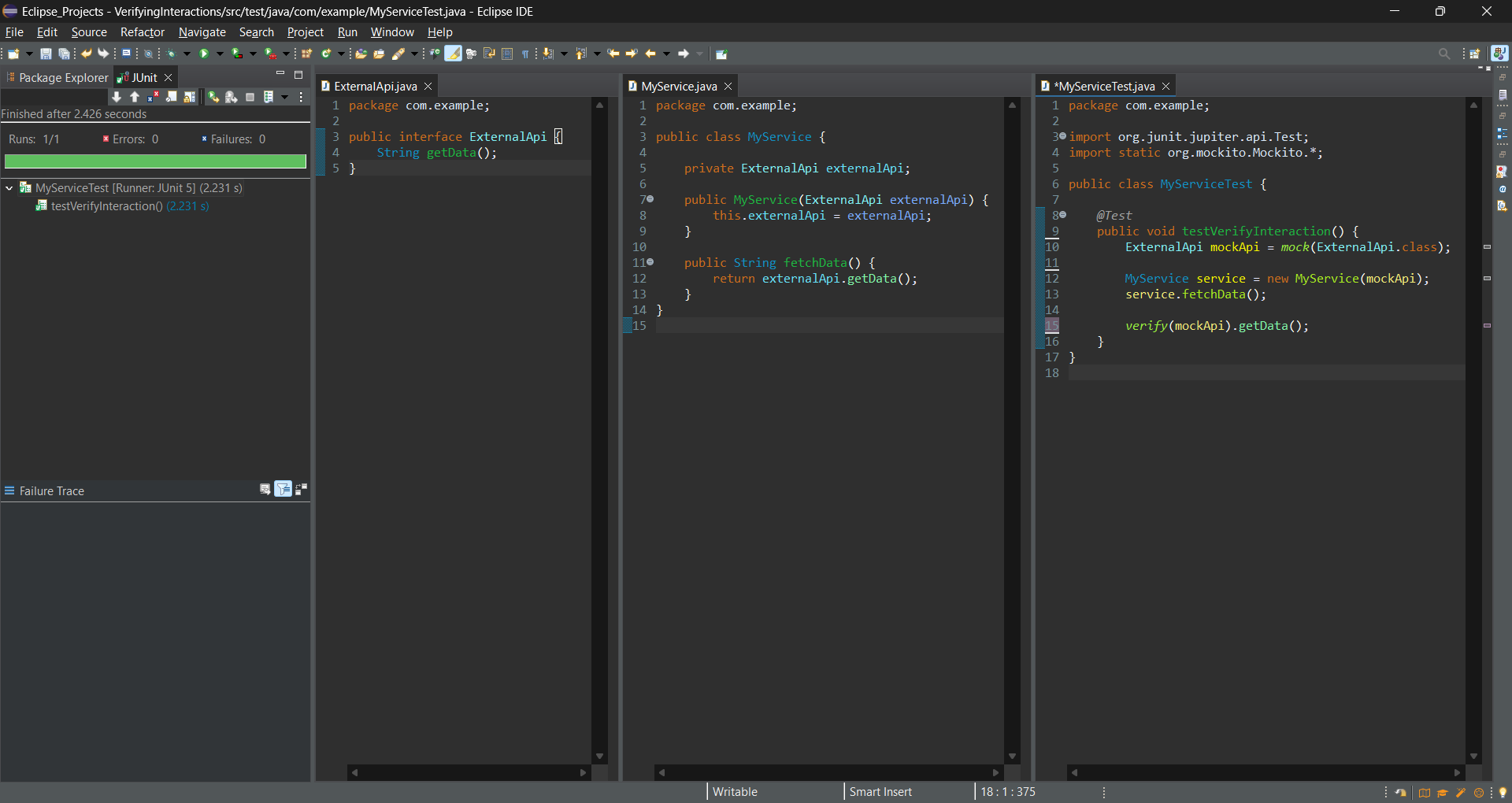
service.fetchData();

*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.

**Program:**

**Messenger.java**

package com.example;

public interface Messenger {

void send(String message);

}

**NotificationService.java**

package com.example;

public class NotificationService {

private Messenger messenger;

public NotificationService(Messenger messenger) {

this.messenger = messenger;

}

public void notifyUser(String userMessage) {

messenger.send(userMessage);

}

}

**NotificationServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

public class NotificationServiceTest {

*@Test*

public void testArgumentMatching() {

Messenger mockMessenger = *mock*(Messenger.class);

NotificationService service = new NotificationService(mockMessenger);

service.notifyUser("Hello, Lohesh!");

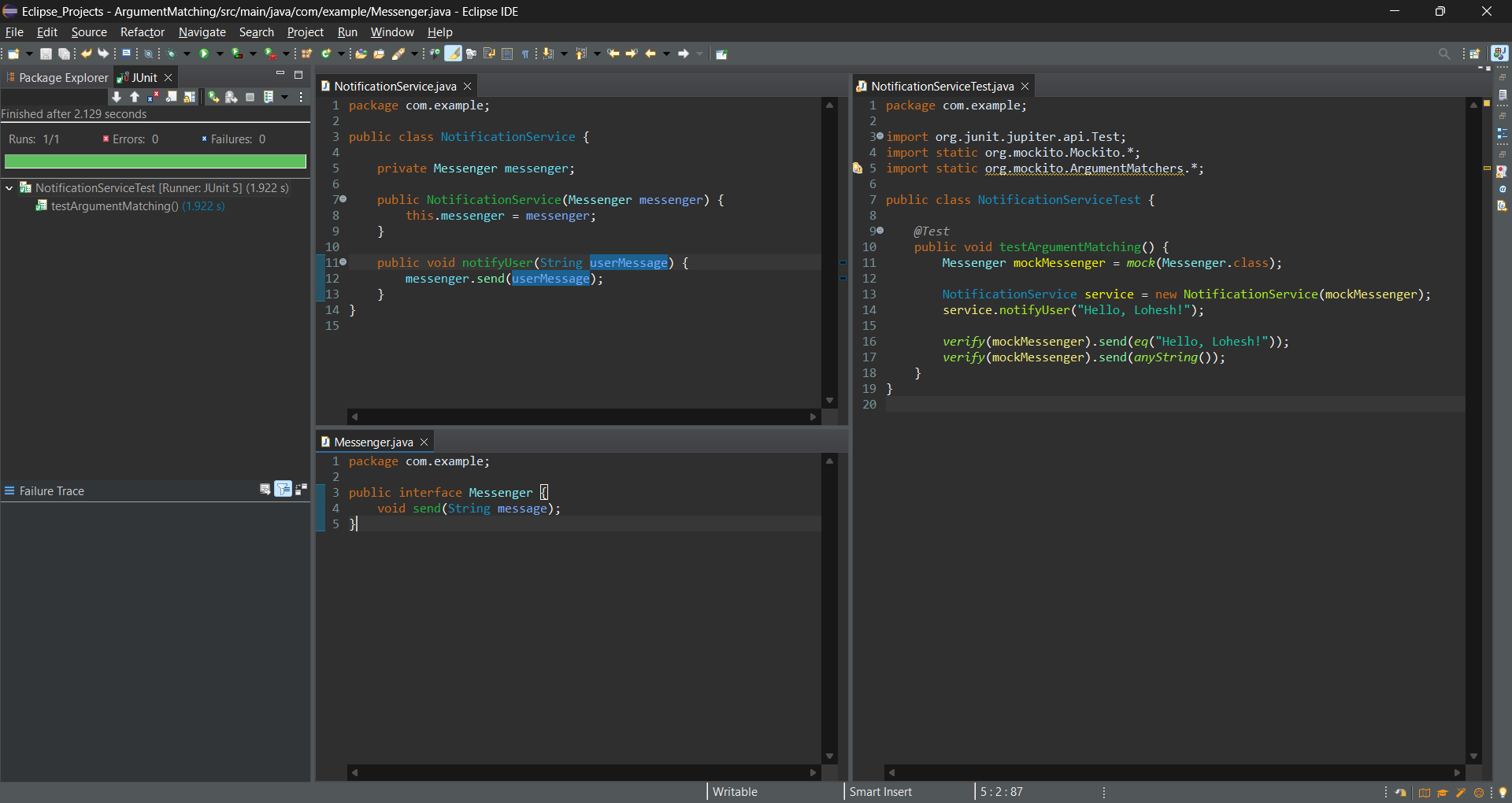
*verify*(mockMessenger).send(*eq*("Hello, Lohesh!"));

*verify*(mockMessenger).send(*anyString*());

}

}

**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 com.example;

public interface Logger {

void log(String message);

}

**AuditService.java**

package com.example;

public class AuditService {

private Logger logger;

public AuditService(Logger logger) {

this.logger = logger;

}

public void auditAction(String action) {

logger.log("Action performed: " + action);

}

}

**AuditServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

public class AuditServiceTest {

*@Test*

public void testVoidMethodInteraction() {

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

*doNothing*().when(mockLogger).log(*anyString*());

AuditService service = new AuditService(mockLogger);

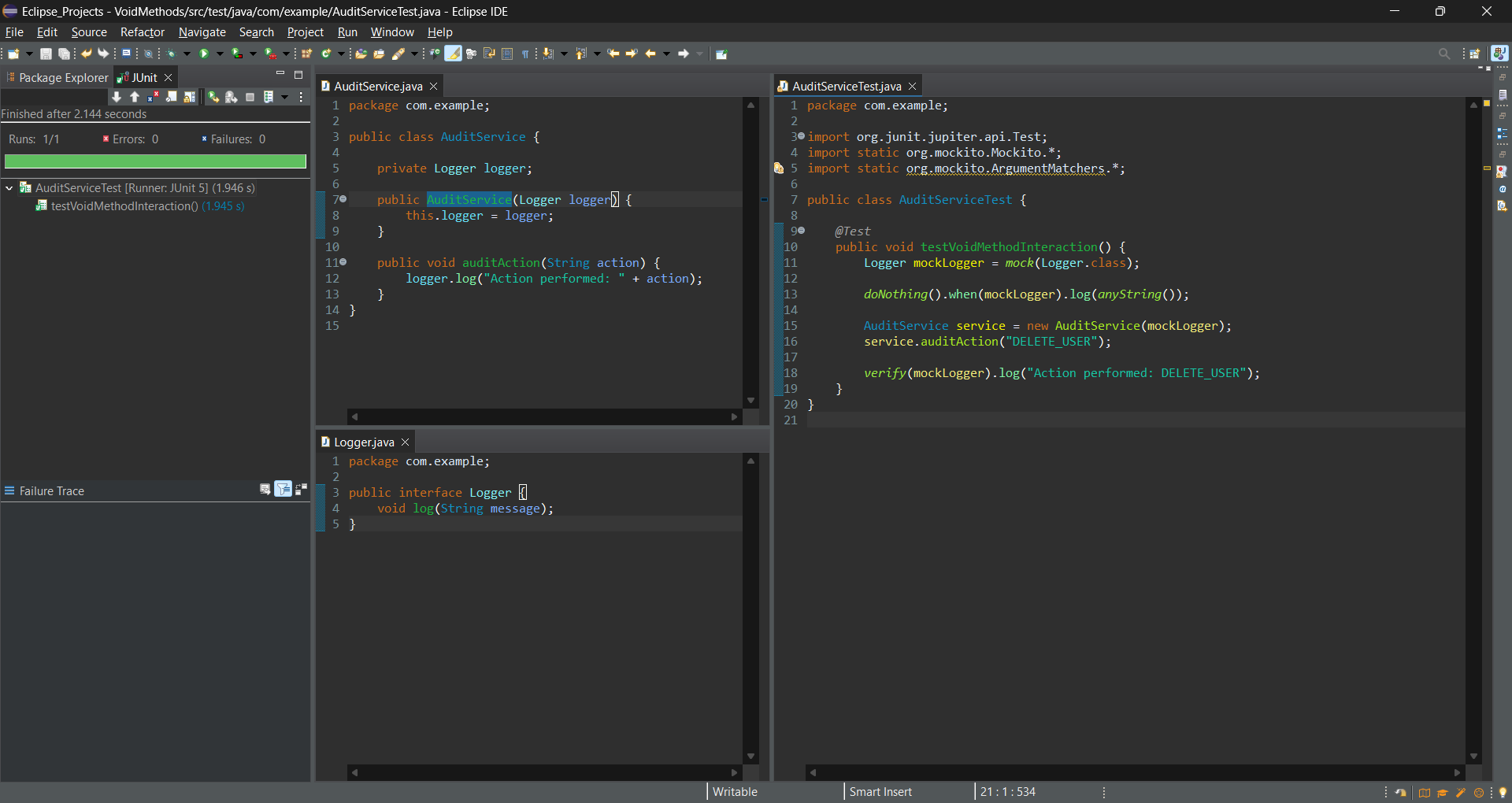
service.auditAction("DELETE\_USER");

*verify*(mockLogger).log("Action performed: DELETE\_USER");

}

}

**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.

**Steps:**

1. Create a mock object for the external API.

2. Stub the methods to return different values on consecutive calls.

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

**Program:**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

}

**PollingService.java**

package com.example;

public class PollingService {

private ExternalApi externalApi;

public PollingService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String pollTwiceAndCombine() {

String first = externalApi.getData();

String second = externalApi.getData();

return first + " & " + second;

}

}

**PollingServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class PollingServiceTest {

*@Test*

public void testMultipleReturns() {

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

*when*(mockApi.getData())

.thenReturn("First Call")

.thenReturn("Second Call");

PollingService service = new PollingService(mockApi);

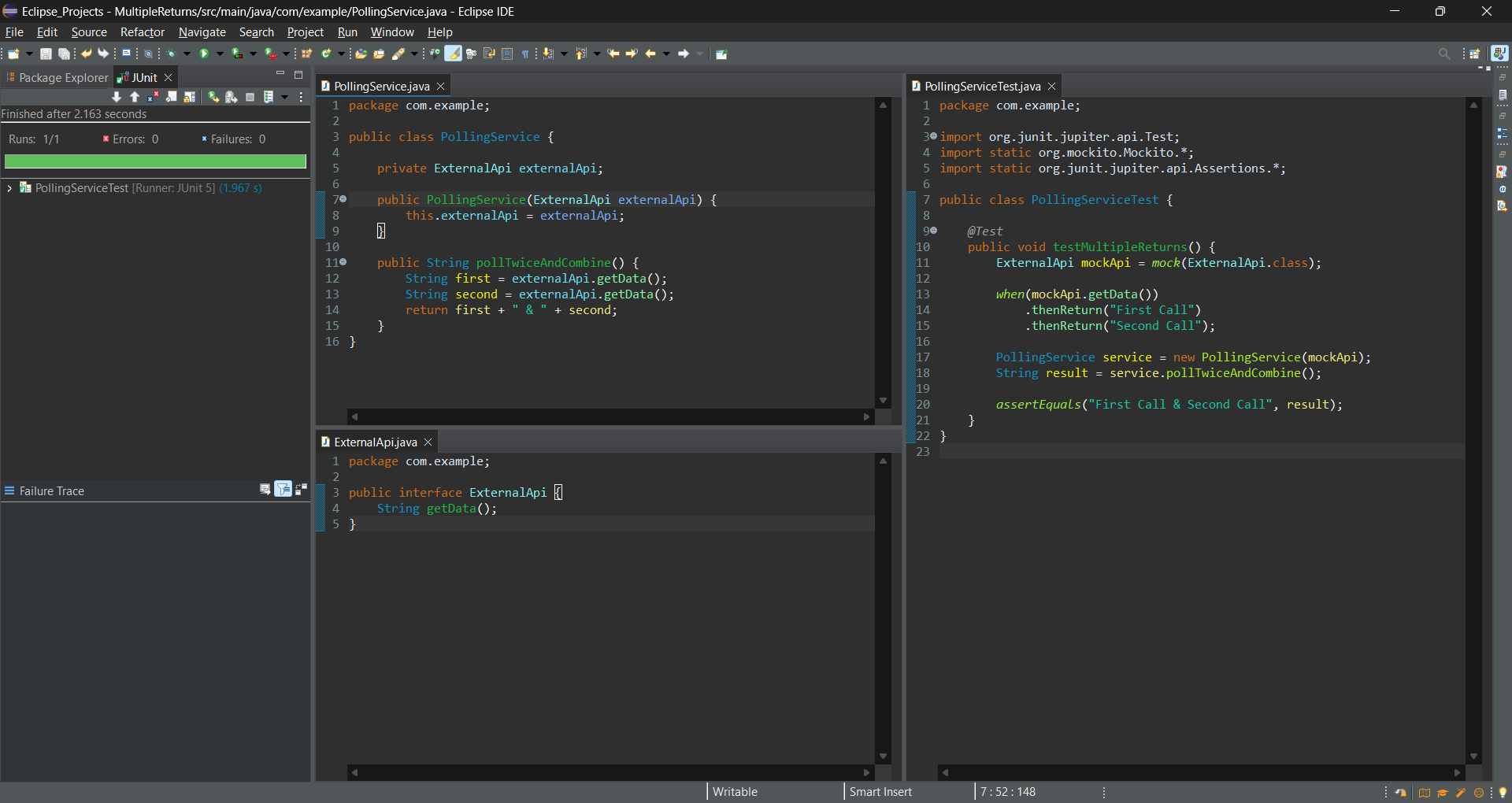
String result = service.pollTwiceAndCombine();

*assertEquals*("First Call & Second Call", result);

}

}

**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.

**Program:**

**Processor.java**

package com.example;

public interface Processor {

void start();

void process();

void finish();

}

**WorkFlowService.java**

package com.example;

public class WorkFlowService {

private Processor processor;

public WorkFlowService(Processor processor) {

this.processor = processor;

}

public void executeWorkflow() {

processor.start();

processor.process();

processor.finish();

}

}

**WorkFlowServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

import static org.mockito.Mockito.\*;

public class WorkFlowServiceTest {

*@Test*

public void testMethodCallOrder() {

// Step 1: Create mock

Processor mockProcessor = *mock*(Processor.class);

// Step 2: Use the mock in the service

WorkFlowService service = new WorkFlowService(mockProcessor);

service.executeWorkflow();

// Step 3: Verify the order of interactions

InOrder inOrder = *inOrder*(mockProcessor);

inOrder.verify(mockProcessor).start();

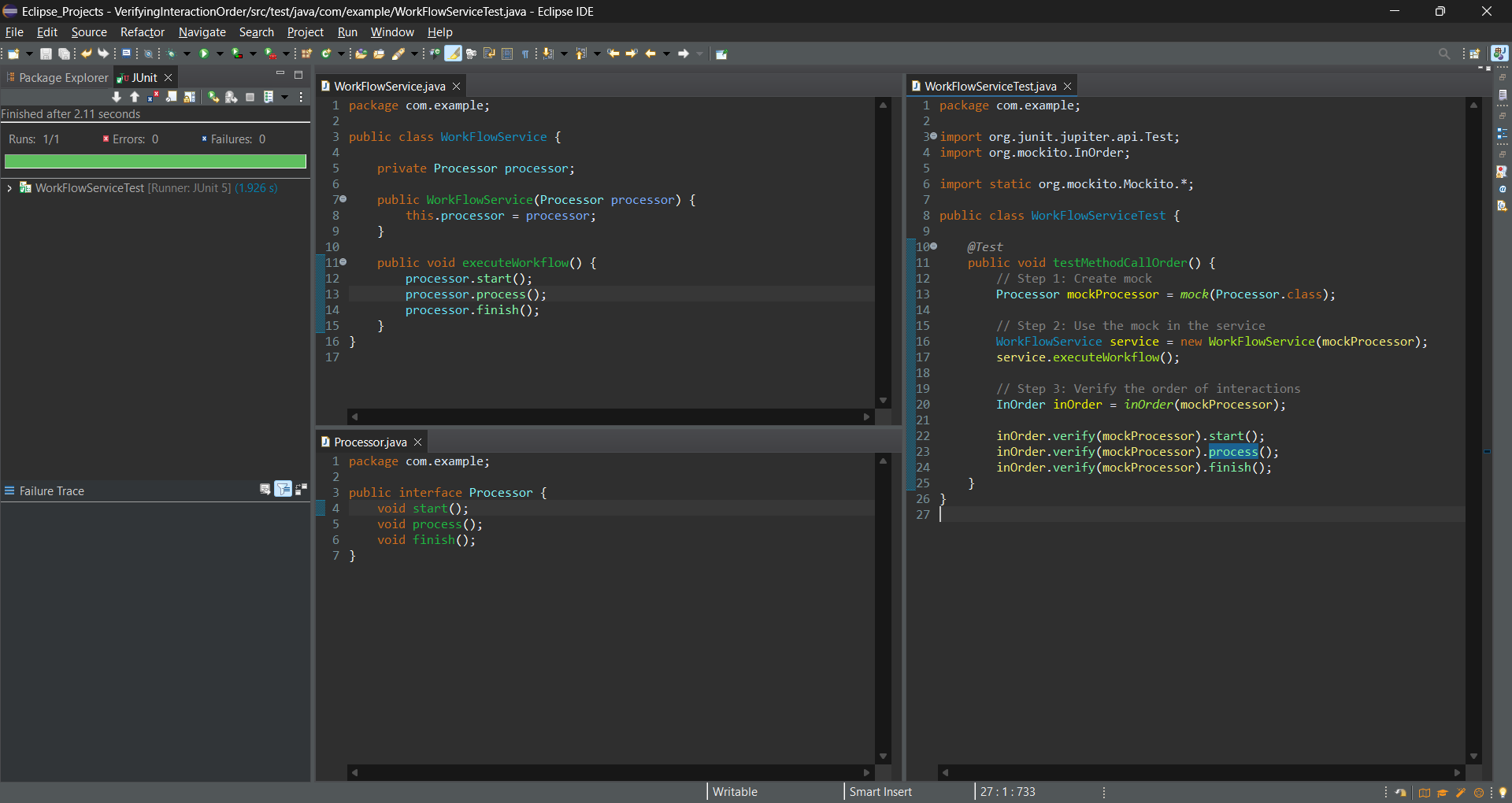
inOrder.verify(mockProcessor).process();

inOrder.verify(mockProcessor).finish();

}

}

**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.

**Program:**

**Notifier.java**

package com.example;

public interface Notifier {

void send(String message);

}

**AlertService.java**

package com.example;

public class AlertService {

private final Notifier notifier;

public AlertService(Notifier notifier) {

this.notifier = notifier;

}

public void triggerAlert(String msg) {

notifier.send(msg);

}

}

**AlertServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class AlertServiceTest {

*@Test*

public void testVoidMethodThrowsException() {

Notifier mockNotifier = *mock*(Notifier.class);

*doThrow*(new RuntimeException("Send failed"))

.when(mockNotifier).send("ERROR");

AlertService service = new AlertService(mockNotifier);

RuntimeException thrown = *assertThrows*(

RuntimeException.class,

() -> service.triggerAlert("ERROR")

);

*assertEquals*("Send failed", thrown.getMessage());

*verify*(mockNotifier).send("ERROR");

}

}

**Output:**

