**WEEK-2**

**Mockito Hands-On Exercises**

**Superset ID: 6419740**

**Exercise 1: Mocking and Stubbing**

**Code:**

**ExternalApi.java:**

public interface ExternalApi {

String getData();

}

**MyService.java:**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

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);

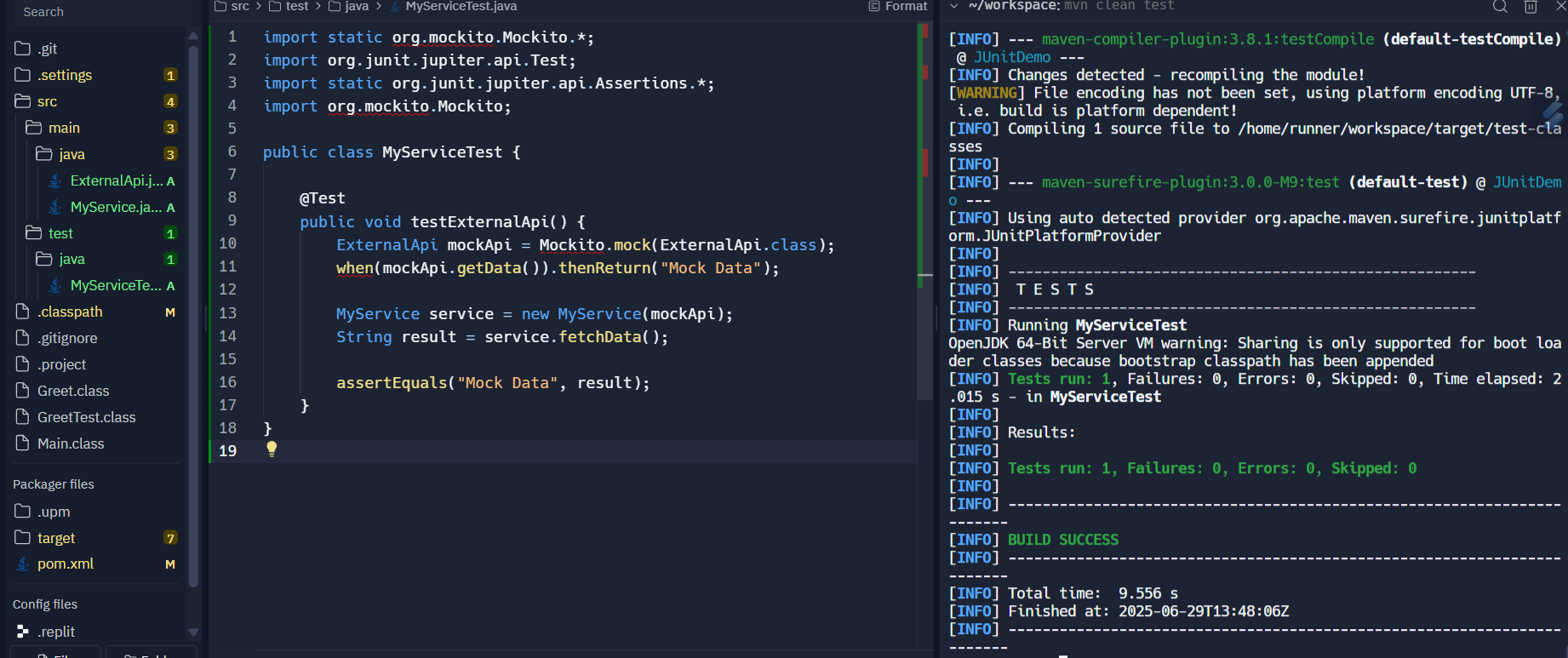
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**Output:**

****

**Exercise 2: Verifying Interactions**

**Code:**

**ExternalApi.java:**

public interface ExternalApi {

String getData();

}

**MyService.java:**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java:**

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);

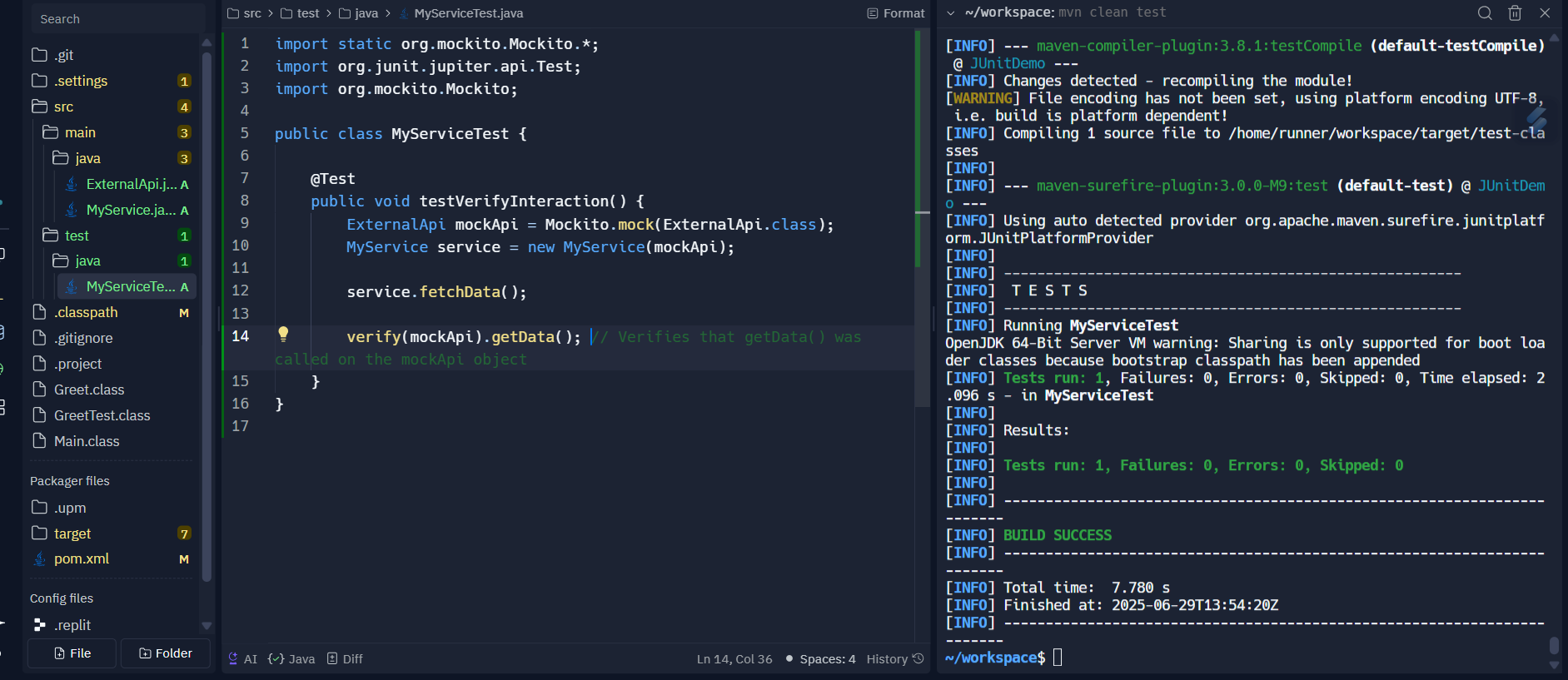
service.fetchData(); // Act

verify(mockApi).getData(); // Assert that getData() was called

}

}

**Output:**

****

**Exercise 3: Argument Matching**

**Code:**

**ExternalApi.java:**

public interface ExternalApi {

void sendMessage(String userId, String message);

}

**Notifier.java:**

public class Notifier {

private ExternalApi api;

public Notifier(ExternalApi api) {

this.api = api;

}

public void notifyUser(String userId) {

api.sendMessage(userId, "Your task is complete");

}

}

**NotifierTest.java:**

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

public class NotifierTest {

@Test

public void testSendMessageCalledWithCorrectArgs() {

ExternalApi mockApi = mock(ExternalApi.class);

Notifier notifier = new Notifier(mockApi);

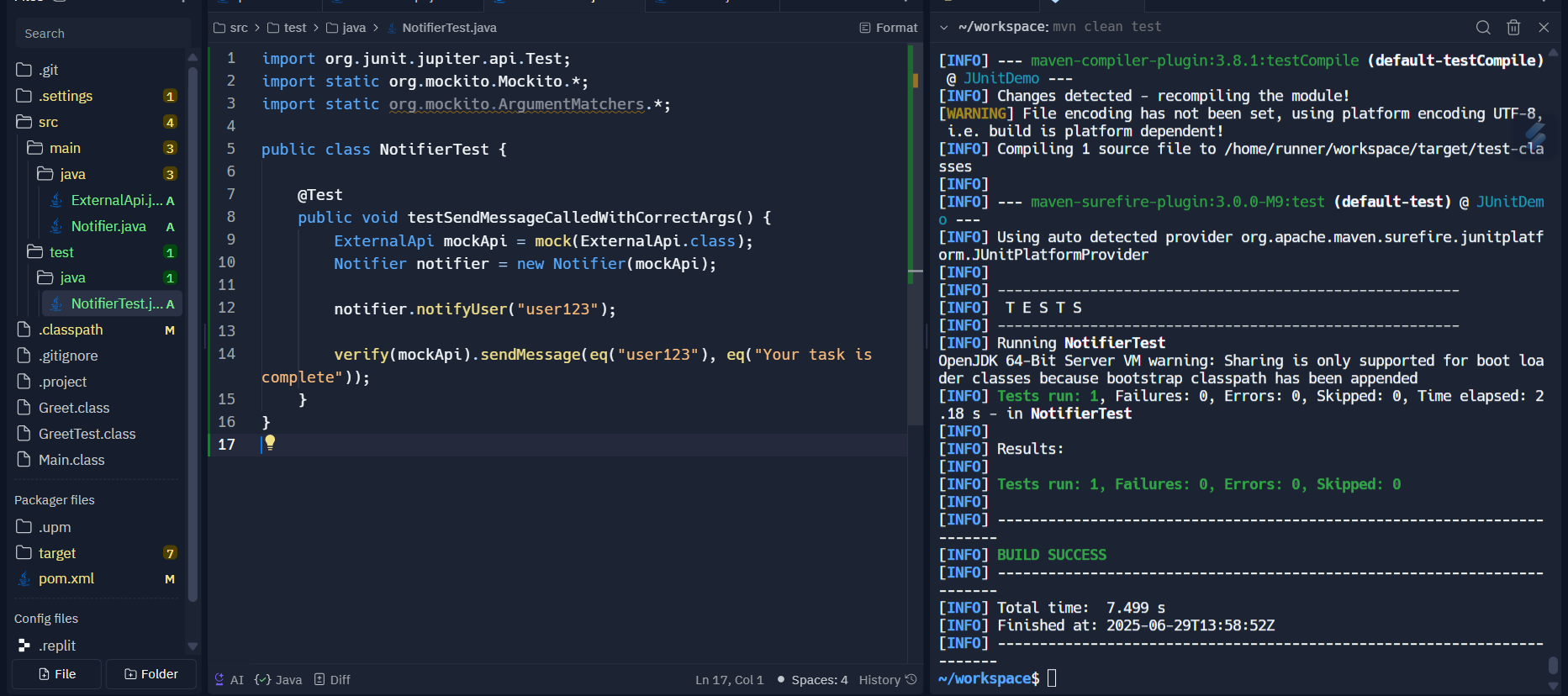
notifier.notifyUser("user123");

verify(mockApi).sendMessage(eq("user123"), eq("Your task is complete"));

}

}

**Output:**

****

**Exercise 4: Handling Void Methods**

**Code:**

**Logger.java:**

public interface Logger {

void log(String message);

}

**Service.java:**

public class Service {

private Logger logger;

public Service(Logger logger) {

this.logger = logger;

}

public void doSomething() {

logger.log("Doing something important");

}

}

**ServiceTest.java:**

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class ServiceTest {

@Test

public void testVoidMethodCalled() {

Logger mockLogger = mock(Logger.class);

Service service = new Service(mockLogger);

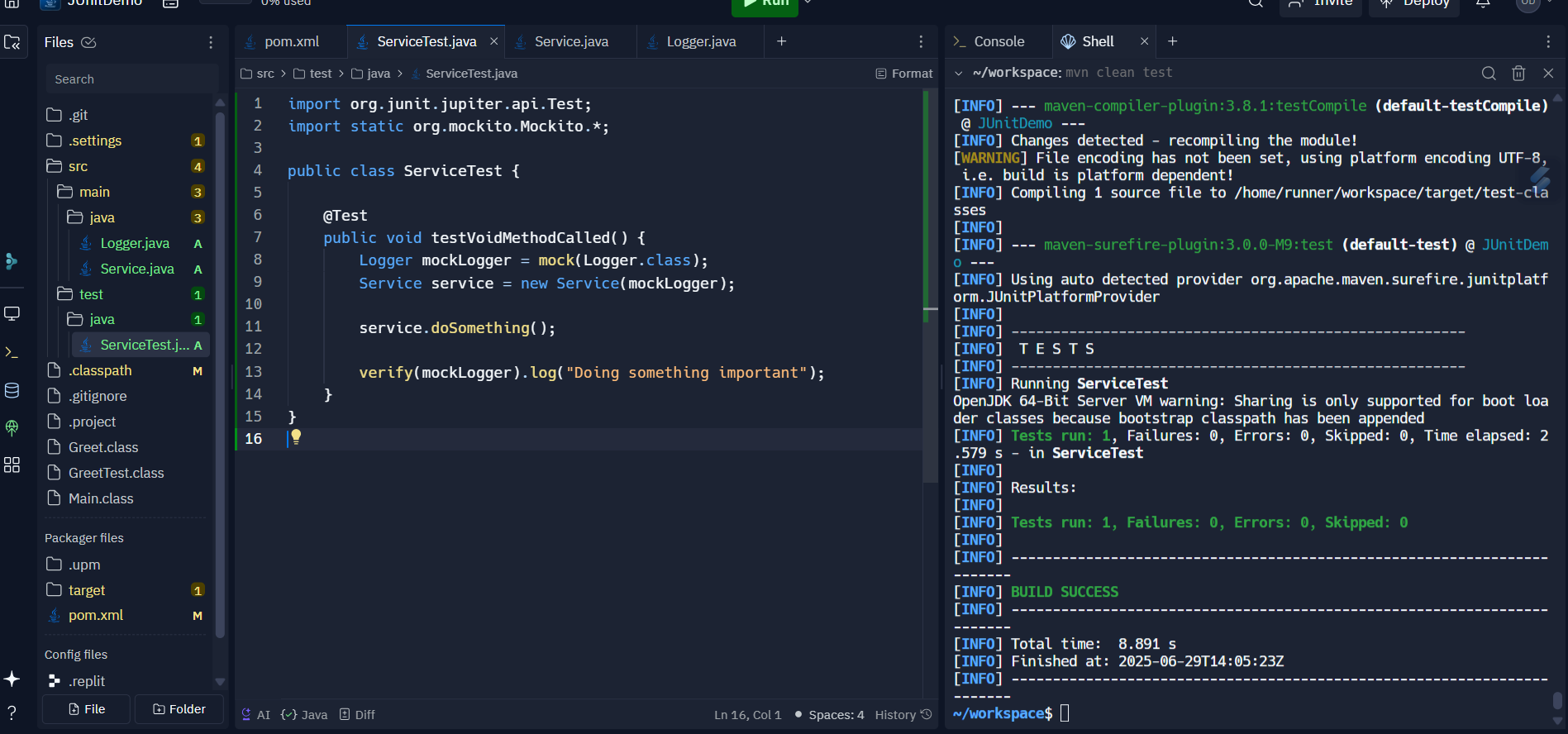
service.doSomething();

verify(mockLogger).log("Doing something important");

}

}

**Output:**

****

**Exercise 5: Mocking and Stubbing with Multiple Returns**

**Code:**

**MyService.java:**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String[] fetchMultipleData() {

return new String[] { api.getData(), api.getData(), api.getData() };

}

}

**ExternalApi.java:**

public interface ExternalApi {

String getData();

}

**MyServiceMultipleReturnsTest.java:**

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

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceMultipleReturnsTest {

@Test

public void testMultipleReturns() {

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

when(mockApi.getData())

.thenReturn("First Call")

.thenReturn("Second Call")

.thenReturn("Third Call");

MyService service = new MyService(mockApi);

String[] results = service.fetchMultipleData();

assertEquals("First Call", results[0]);

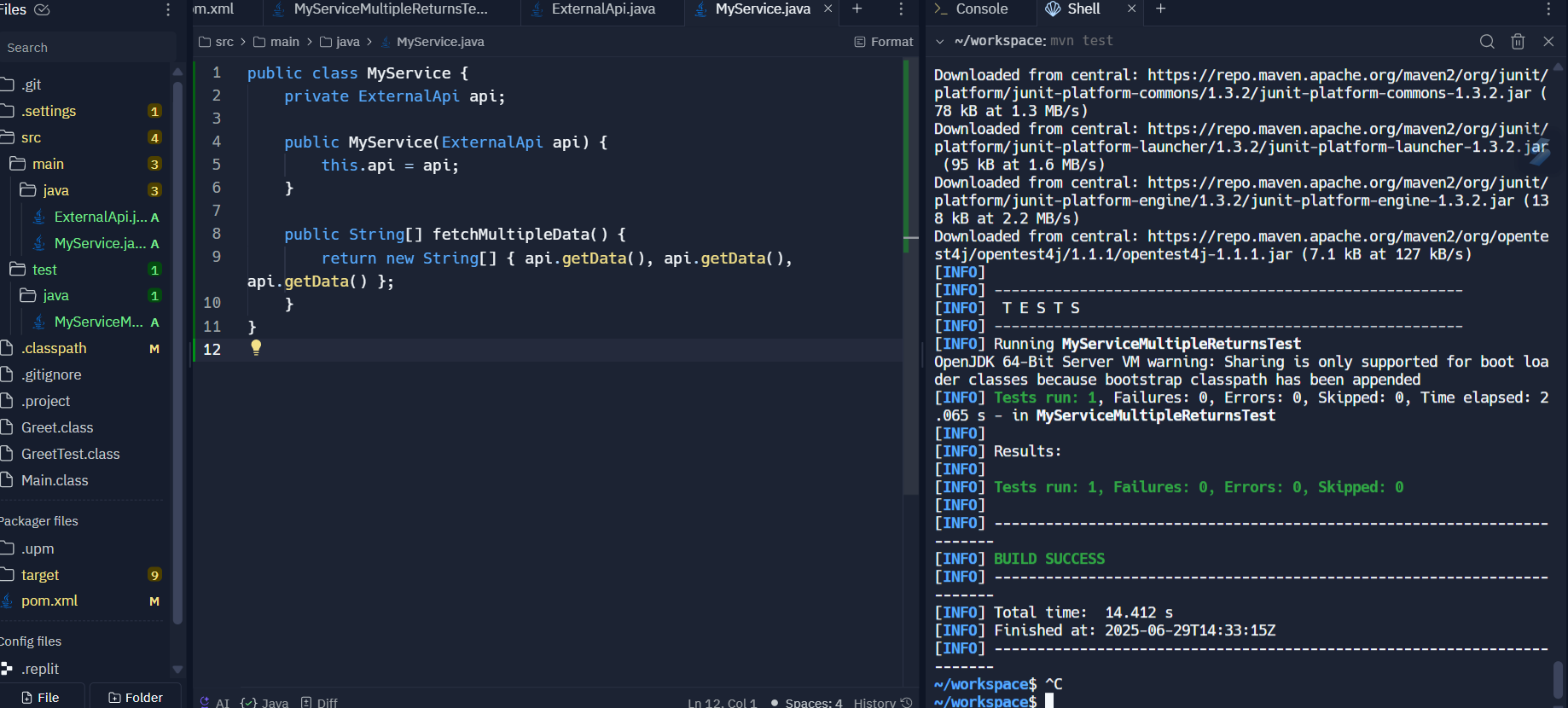
assertEquals("Second Call", results[1]);

assertEquals("Third Call", results[2]);

}

}

**Output:**

****

**Exercise 6: Verifying Interaction Order**

**Code:**

**ExternalApi.java:**

public interface ExternalApi {

void stepOne();

void stepTwo();

void stepThree();

String getData();

}

**MyService.java:**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void performSteps() {

api.stepOne();

api.stepTwo();

api.stepThree();

}

public String fetchMultipleData() {

return api.getData();

}

}

**MyServiceInteractionOrderTest.java:**

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

import static org.mockito.Mockito.\*;

public class MyServiceInteractionOrderTest {

@Test

public void testInteractionOrder() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.performSteps();

InOrder inOrder = inOrder(mockApi);

inOrder.verify(mockApi).stepOne();

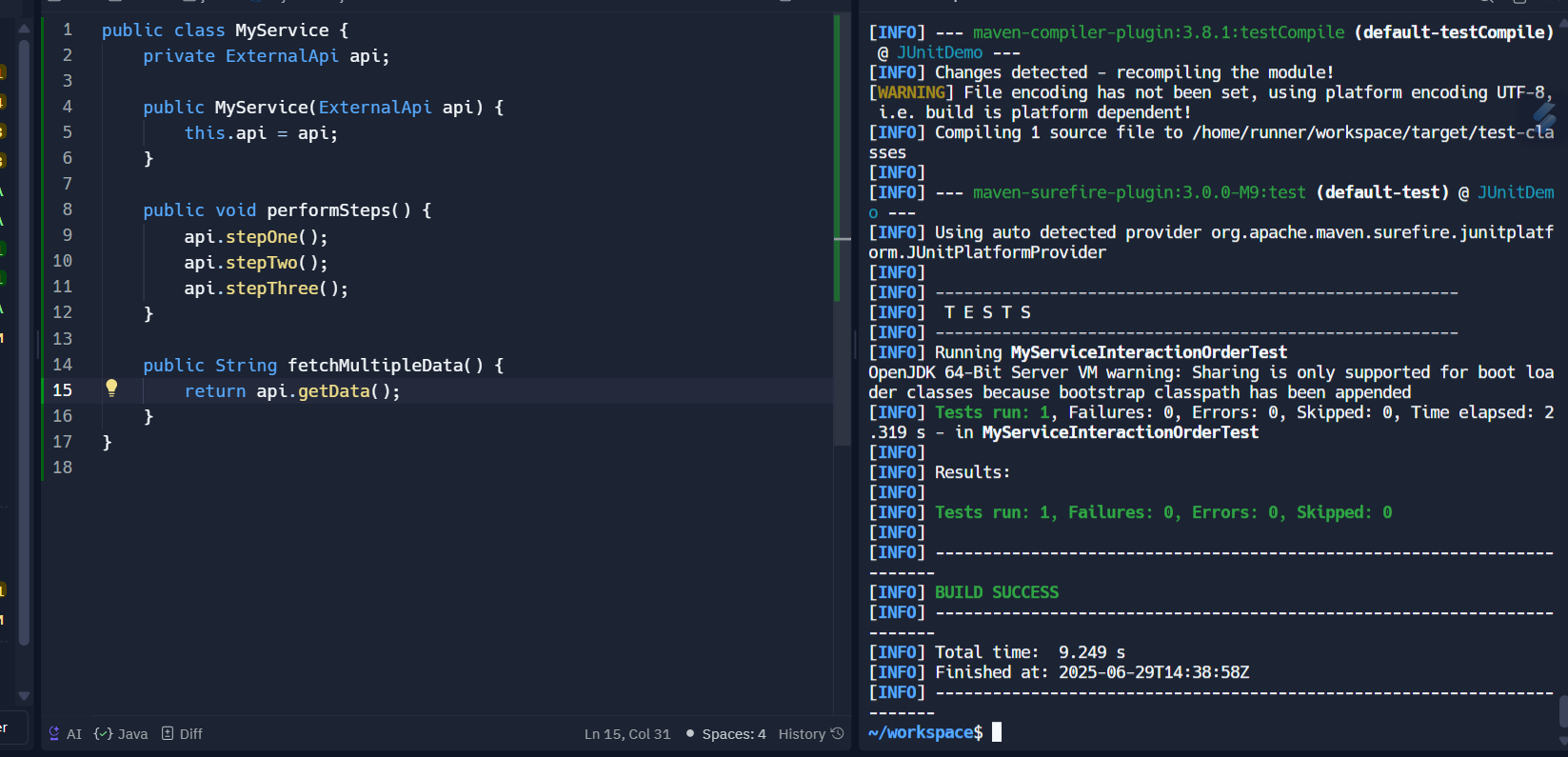
inOrder.verify(mockApi).stepTwo();

inOrder.verify(mockApi).stepThree();

}

}

**Output:**

****

**Exercise 7: Handling Void Methods with Exceptions**

**Code:**

**ExternalApi.java:**

public interface ExternalApi {

void riskyOperation();

}

**MyService.java:**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void handleRisk() {

api.riskyOperation();

}

}

**MyServiceExceptionTest.java:**

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

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceExceptionTest {

@Test

public void testRiskyOperationThrowsException() {

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

doThrow(new RuntimeException("Failure")).when(mockApi).riskyOperation();

MyService service = new MyService(mockApi);

assertThrows(RuntimeException.class, () -> {

service.handleRisk();

});

verify(mockApi).riskyOperation();

}

}

**Output:**

