**Mockito Hands-On Exercises Assignment**

**Exercise 1: Mocking and Stubbing**

**Question:**

You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.

**Steps Required:**

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

interface ExternalApi {

String getData();

}

class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

public class Exercised1Test{

@Test

public void testExternalApi() {

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

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

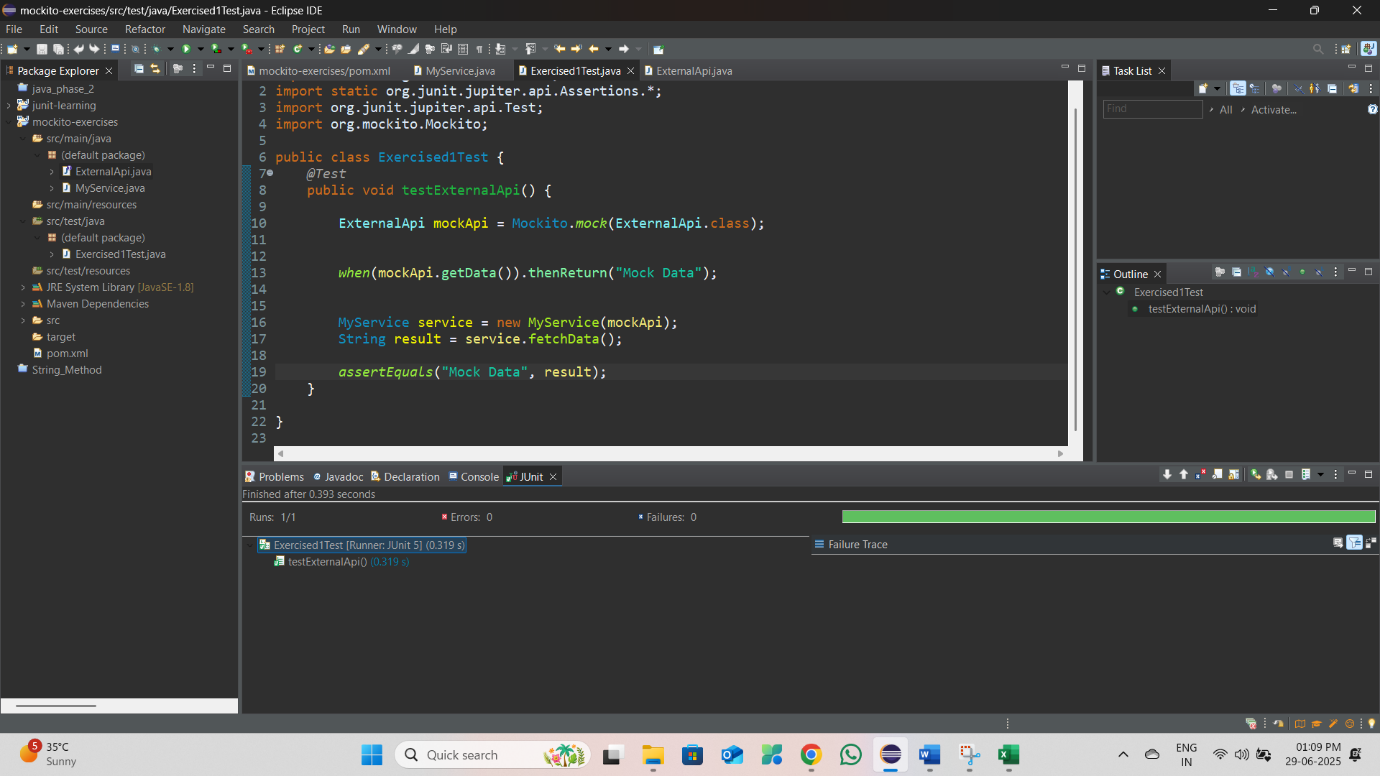
MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**Output:  
  
**

Test passed ✓

Result: "Mock Data"

The mock object successfully returned the stubbed value.

**Exercise 2: Verifying Interactions**

**Question:**

You need to ensure that a method is called with specific arguments and verify the interaction occurred.

**Steps Required:**

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;

public class Exercised2Test{

@Test

public void testVerifyInteraction() {

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

MyService service = new MyService(mockApi);

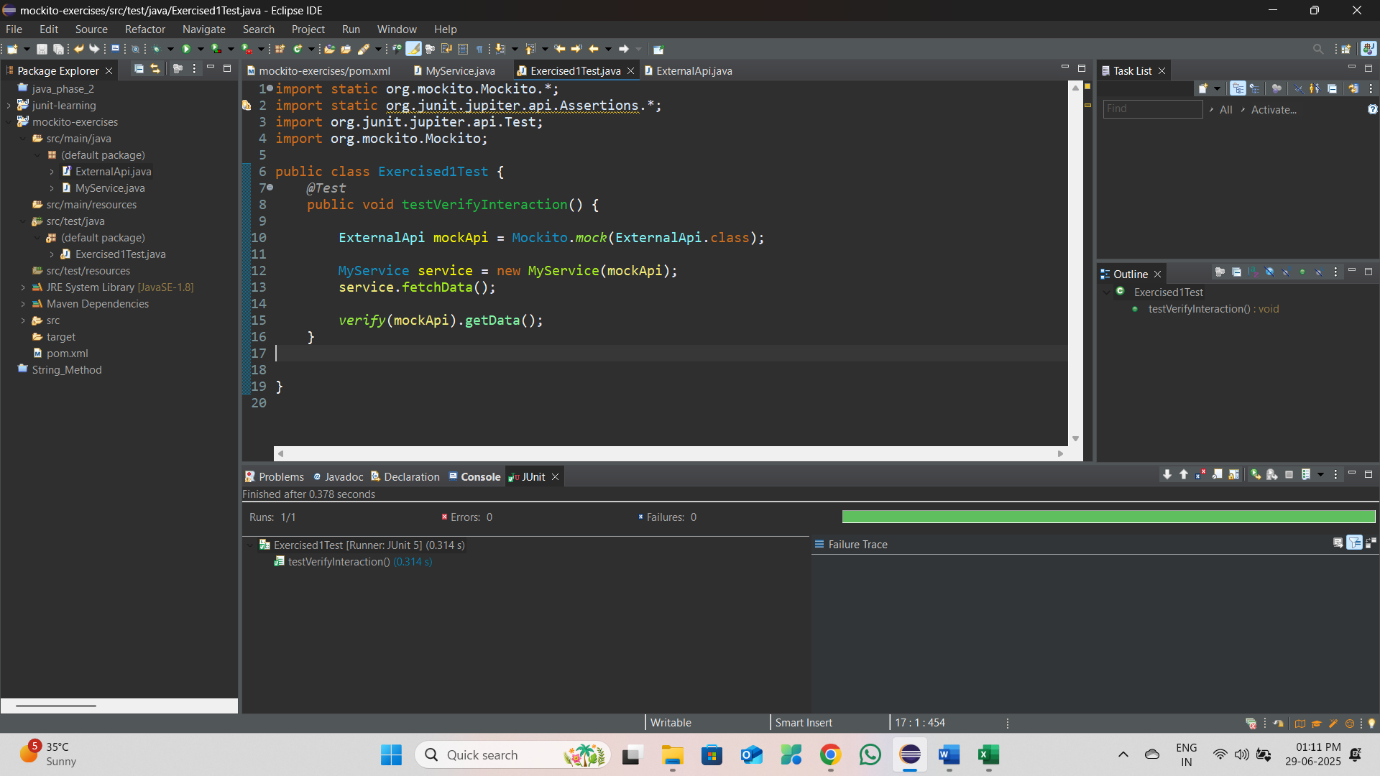
service.fetchData();

verify(mockApi).getData();

}

}

**Output:**

****

Test passed ✓

Verification successful: mockApi.getData() was called exactly once.

**Exercise 3: Argument Matching**

**Question:**

You need to verify that a method is called with specific arguments using argument matchers.

**Steps Required:**

1. Create a mock object
2. Call the method with specific arguments
3. Use argument matchers to verify the interaction

**Solution Code:**

import static org.mockito.ArgumentMatchers.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

interface ExternalApi3 {

String getData();

String getDataById(int id);

void saveData(String data, int priority);

}

class MyService3 {

private ExternalApi3 externalApi;

public MyService3(ExternalApi3 externalApi) {

this.externalApi = externalApi;

}

public String fetchDataById(int id) {

return externalApi.getDataById(id);

}

public void saveImportantData(String data) {

externalApi.saveData(data, 1);

}

}

public class Exercised3Test{

@Test

public void testArgumentMatching() {

ExternalApi3 mockApi = Mockito.mock(ExternalApi3.class);

when(mockApi.getDataById(anyInt())).thenReturn("Data for any ID");

MyService3 service = new MyService3(mockApi);

service.fetchDataById(123);

service.saveImportantData("Important Data");

verify(mockApi).getDataById(eq(123));

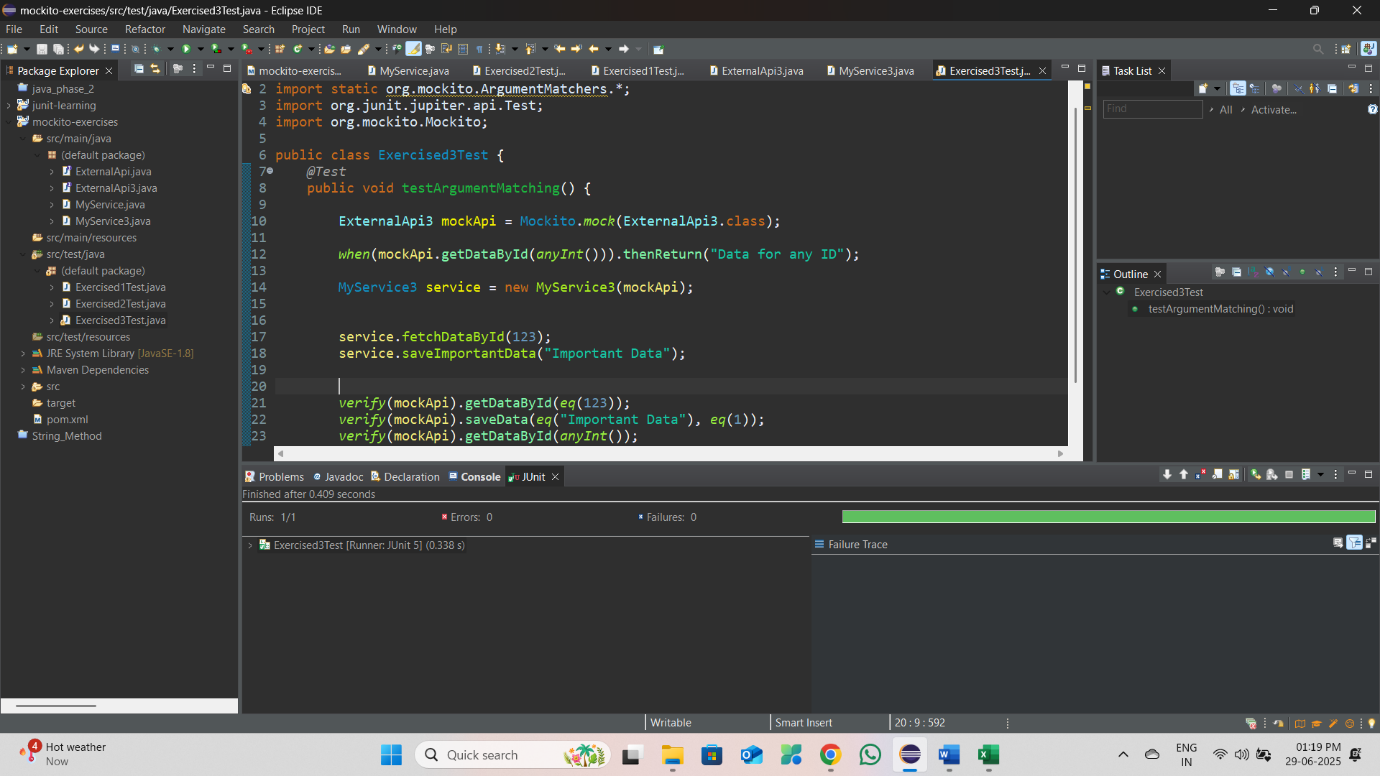
verify(mockApi).saveData(eq("Important Data"), eq(1));

verify(mockApi).getDataById(anyInt());

}

}

**Output:**

****

Test passed ✓

Argument matching verification successful:

- getDataById(123) was called with exact argument

- saveData("Important Data", 1) was called with exact arguments

- getDataById(anyInt()) matched the call with any integer argument

**Exercise 4: Handling Void Methods**

**Question:**

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

**Steps Required:**

1. Create a mock object
2. Stub the void method
3. Verify the interaction

**Solution Code:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

interface ExternalApi4 {

String getData();

void deleteData(int id);

void logActivity(String message);

}

class MyService4 {

private ExternalApi4 externalApi;

public MyService4(ExternalApi4 externalApi) {

this.externalApi = externalApi;

}

public void removeData(int id) {

externalApi.logActivity("Deleting data with ID: " + id);

externalApi.deleteData(id);

}

}

public class Exercised4Test{

@Test

public void testVoidMethod() {

ExternalApi4 mockApi = Mockito.mock(ExternalApi4.class);

doNothing().when(mockApi).deleteData(anyInt());

doNothing().when(mockApi).logActivity(anyString());

MyService4 service = new MyService4(mockApi);

service.removeData(456);

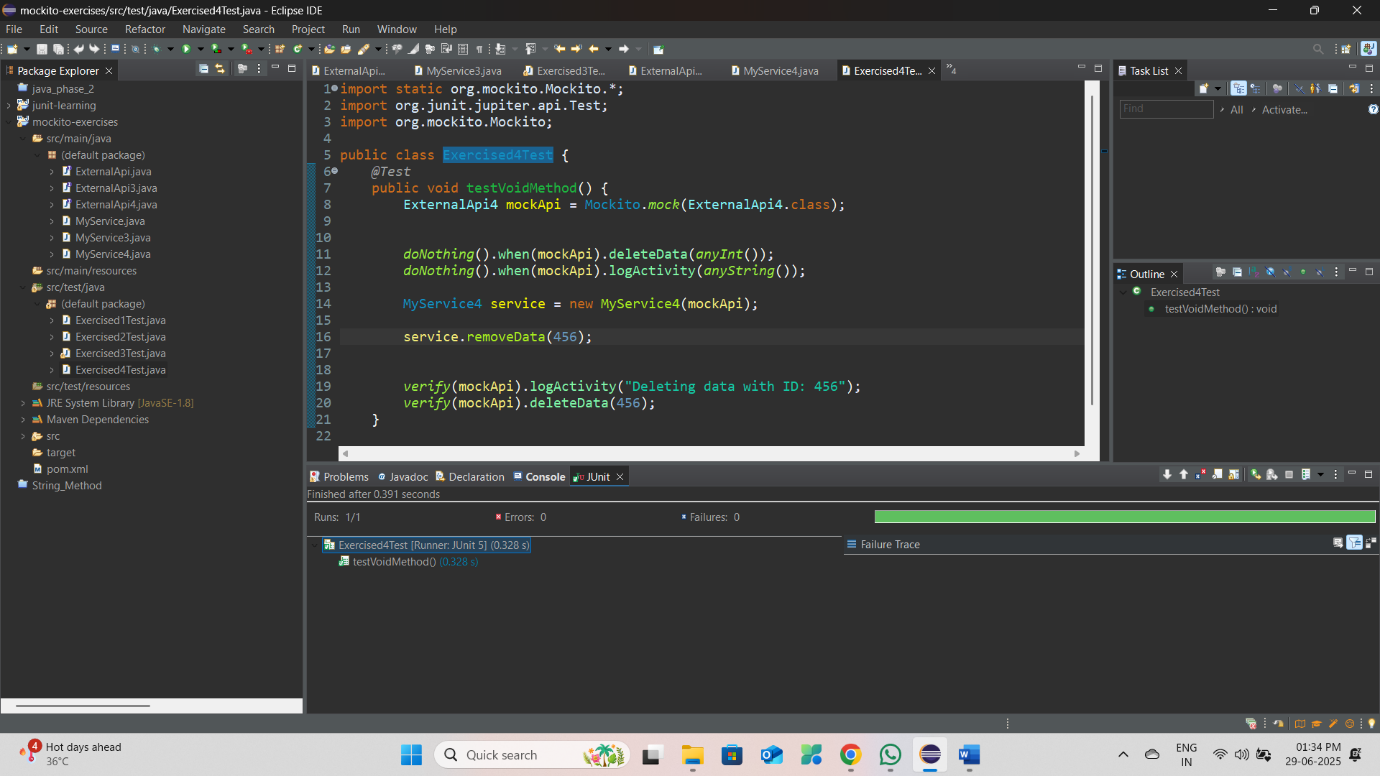
verify(mockApi).logActivity("Deleting data with ID: 456");

verify(mockApi).deleteData(456);

}

}

**Expected Output:**

****

Test passed ✓

Void method verification successful:

- logActivity("Deleting data with ID: 456") was called

- deleteData(456) was called

Both void methods were invoked as expected.

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

**Question:**

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

**Steps Required:**

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

**Solution Code:**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class Exercised5Test{

@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 result1 = service.fetchData();

String result2 = service.fetchData();

String result3 = service.fetchData();

assertEquals("First Call", result1);

assertEquals("Second Call", result2);

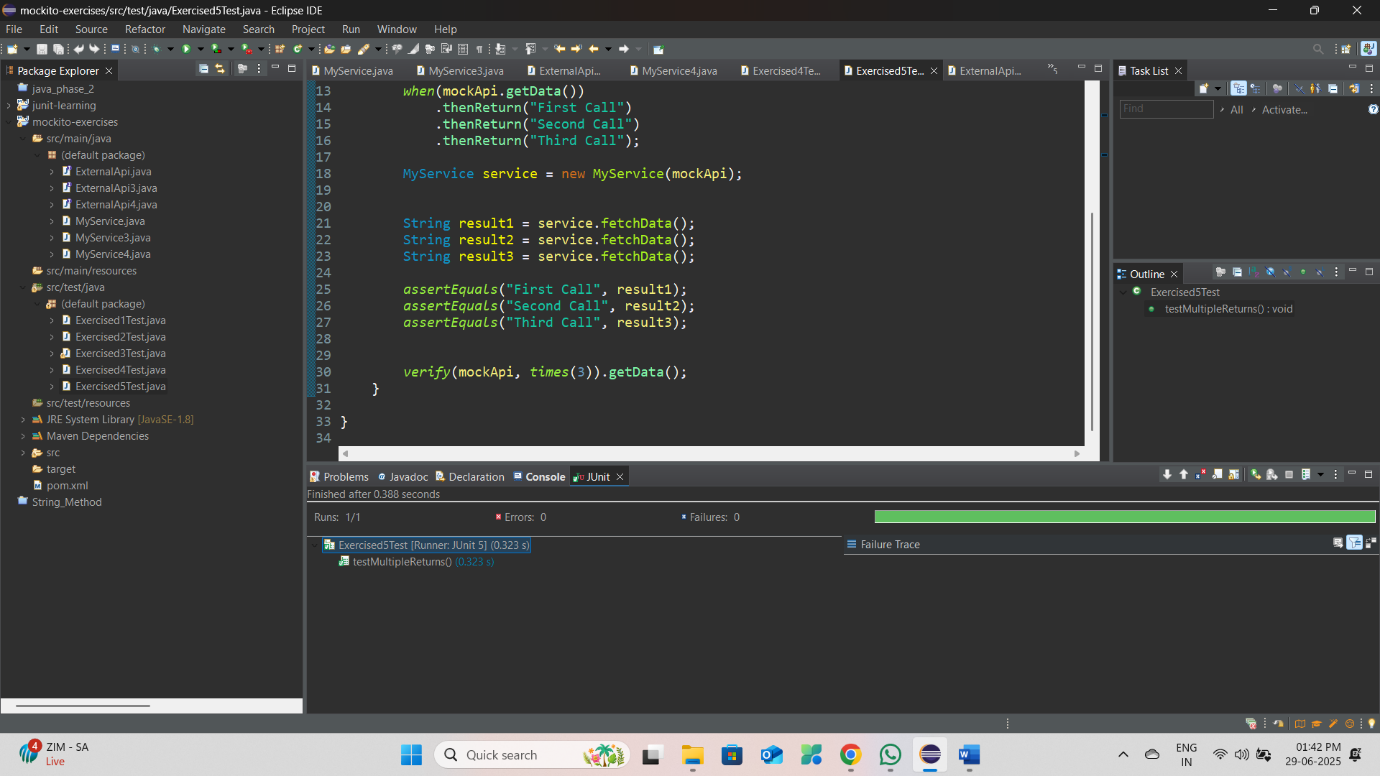
assertEquals("Third Call", result3);

verify(mockApi, times(3)).getData();

}

}

**Expected Output:**

****

Test passed ✓

Multiple returns verification successful:

- First call returned: "First Call"

- Second call returned: "Second Call"

- Third call returned: "Third Call"

- getData() was called exactly 3 times

**Exercise 6: Verifying Interaction Order**

**Question:**

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

**Steps Required:**

1. Create a mock object
2. Call the methods in a specific order
3. Verify the interaction order

**Solution Code:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

import org.mockito.Mockito;

class MyService6 {

private ExternalApi4 externalApi;

public MyService6(ExternalApi4 externalApi) {

this.externalApi = externalApi;

}

public void processData() {

externalApi.logActivity("Starting data processing");

String data = externalApi.getData();

// paste this in ExternalApi4 file void saveData(String data, int priority);

externalApi.saveData(data, 1);

externalApi.logActivity("Data processing completed");

}

}

public class Exercised6Test {

@Test

public void testInteractionOrder() {

ExternalApi4 mockApi = Mockito.mock(ExternalApi4.class);

when(mockApi.getData()).thenReturn("Processed Data");

MyService6 service = new MyService6mockApi);

service.processData();

InOrder inOrder = inOrder(mockApi);

inOrder.verify(mockApi).logActivity("Starting data processing");

inOrder.verify(mockApi).getData();

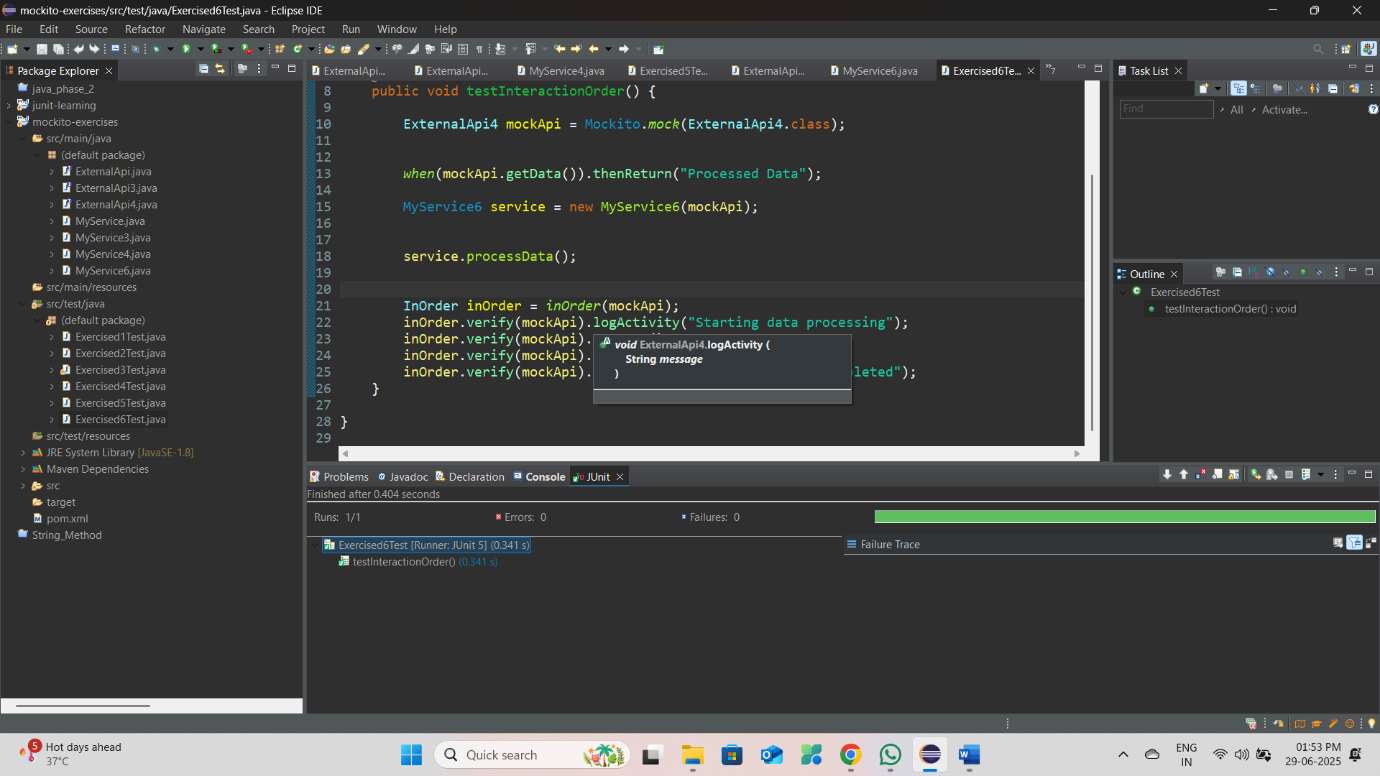
inOrder.verify(mockApi).saveData("Processed Data", 1);

inOrder.verify(mockApi).logActivity("Data processing completed");

}

}

**Expected Output:**

****

Test passed ✓

Interaction order verification successful:

Methods were called in the correct order:

1. logActivity("Starting data processing")

2. getData()

3. saveData("Processed Data", 1)

4. logActivity("Data processing completed")

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

**Question:**

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

**Steps Required:**

1. Create a mock object
2. Stub the void method to throw an exception
3. Verify the interaction and exception handling

**Solution Code:**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

class DataProcessingException extends Exception {

public DataProcessingException(String message) {

super(message);

}

}

// Updated Service class with exception handling

class MyService7 {

private ExternalApi externalApi;

public MyService7(ExternalApi4 externalApi) {

this.externalApi = externalApi;

}

public boolean deleteDataSafely(int id) {

try {

externalApi.deleteData(id);

return true;

} catch (Exception e) {

externalApi.logActivity("Error deleting data: " + e.getMessage());

return false;

}

}

}

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class Exercised7Test {

*@Test*

public void testVoidMethodWithException() {

ExternalApi4 mockApi = Mockito.*mock*(ExternalApi4.class);

*doThrow*(new RuntimeException("Database connection failed"))

.when(mockApi).deleteData(789);

MyService7 service = new MyService7(mockApi);

boolean result = service.deleteDataSafely(789);

*assertFalse*(result);

*verify*(mockApi).deleteData(789);

*verify*(mockApi).logActivity("Error deleting data: Database connection failed");

}

*@Test*

public void testVoidMethodWithoutException() {

ExternalApi4 mockApi = Mockito.*mock*(ExternalApi4.class);

*doNothing*().when(mockApi).deleteData(789);

MyService7 service = new MyService7(mockApi);

boolean result = service.deleteDataSafely(789);

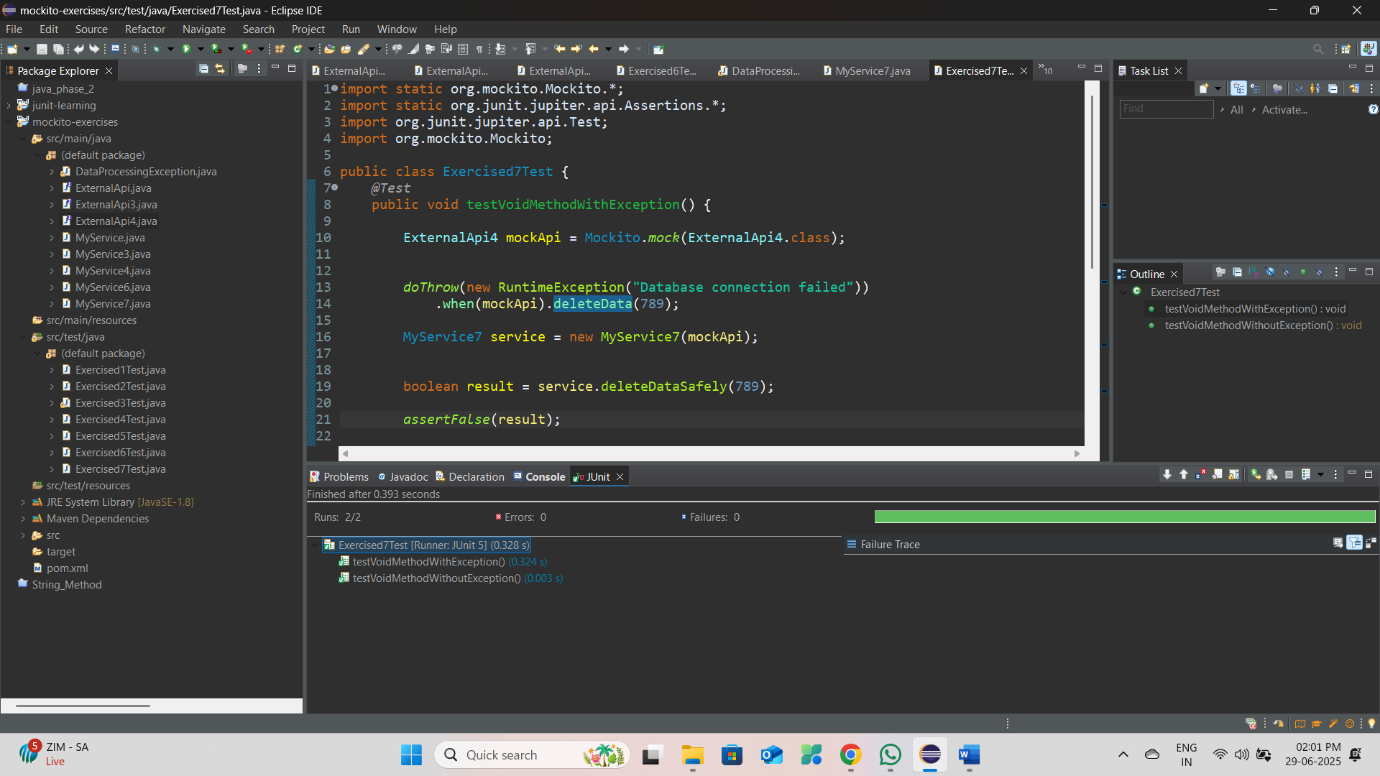
*assertTrue*(result);

*verify*(mockApi).deleteData(789);

*verify*(mockApi, *never*()).logActivity(*anyString*());

}

}

**Expected Output:  
  
**

Test 1 passed ✓

Exception handling verification successful:

- deleteData(789) threw RuntimeException as expected

- Exception was caught and handled properly

- logActivity was called with error message

- Method returned false indicating failure

Test 2 passed ✓

Normal execution verification successful:

- deleteData(789) executed without exception

- No error logging occurred

- Method returned true indicating success