**(6428317 Mukesh P)**

***WEEK 2***

***Mockito Hands-On Exercises***

***MANDATORY HANS-ON:***

**Exercise 1: Mocking and Stubbing**

**ExternalAPi.java**

package com.example;

public interface ExternalApi {

String getData();

void update(String input);

void riskyOperation();

}

**Myservice.java**

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

public void updateData(String data) {

api.update(data);

}

public void callMultipleTimes() {

api.getData();

api.getData();

}

public void triggerRiskyOperation() {

api.riskyOperation();

}

} **MyServiceTest.java:**

package com.example.test;

import com.example.\*;

import org.junit.Test;

import static org.junit.Assert.\*;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = mock(ExternalApi.class);

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

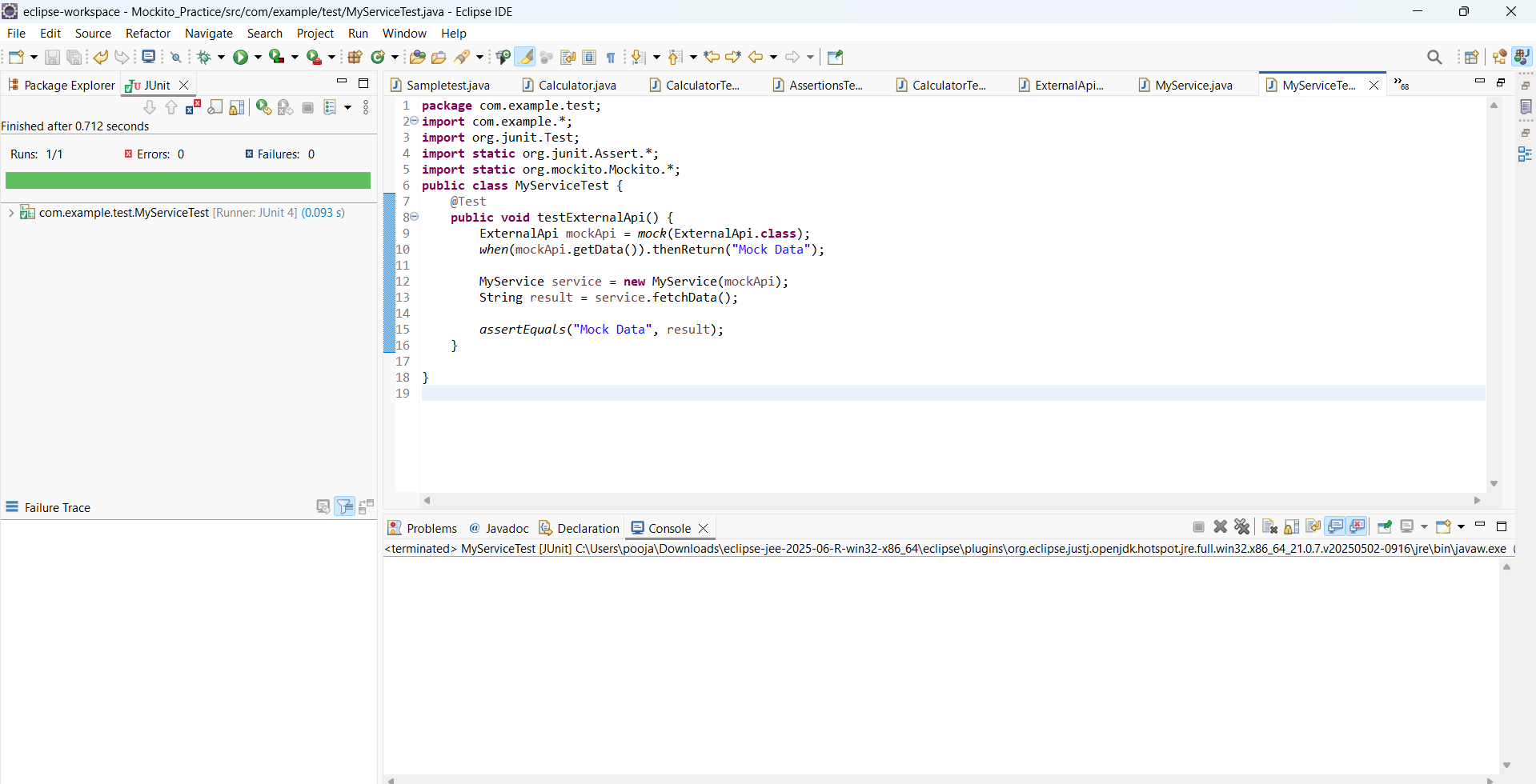
MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}



**Exercise 2: Verifying Interactions:**

@Test

**public** **void** testVerifyInteraction() {

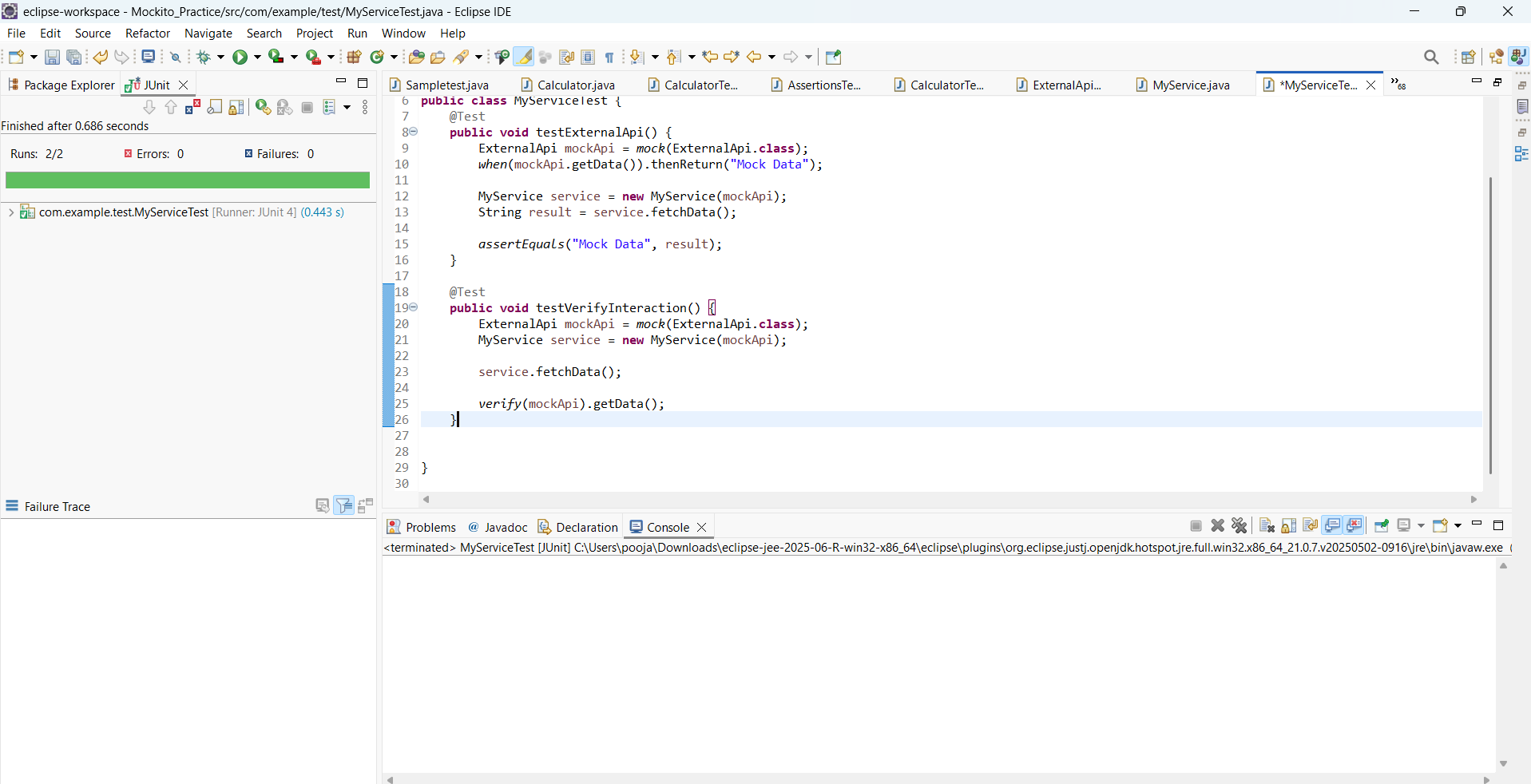
ExternalApi mockApi = *mock*(ExternalApi.**class**);

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

service.fetchData();

*verify*(mockApi).getData();

}

****

***Additional Hands-on:***

**Exercise 3: Argument Matching**

@Test

**public** **void** testArgumentMatching() {

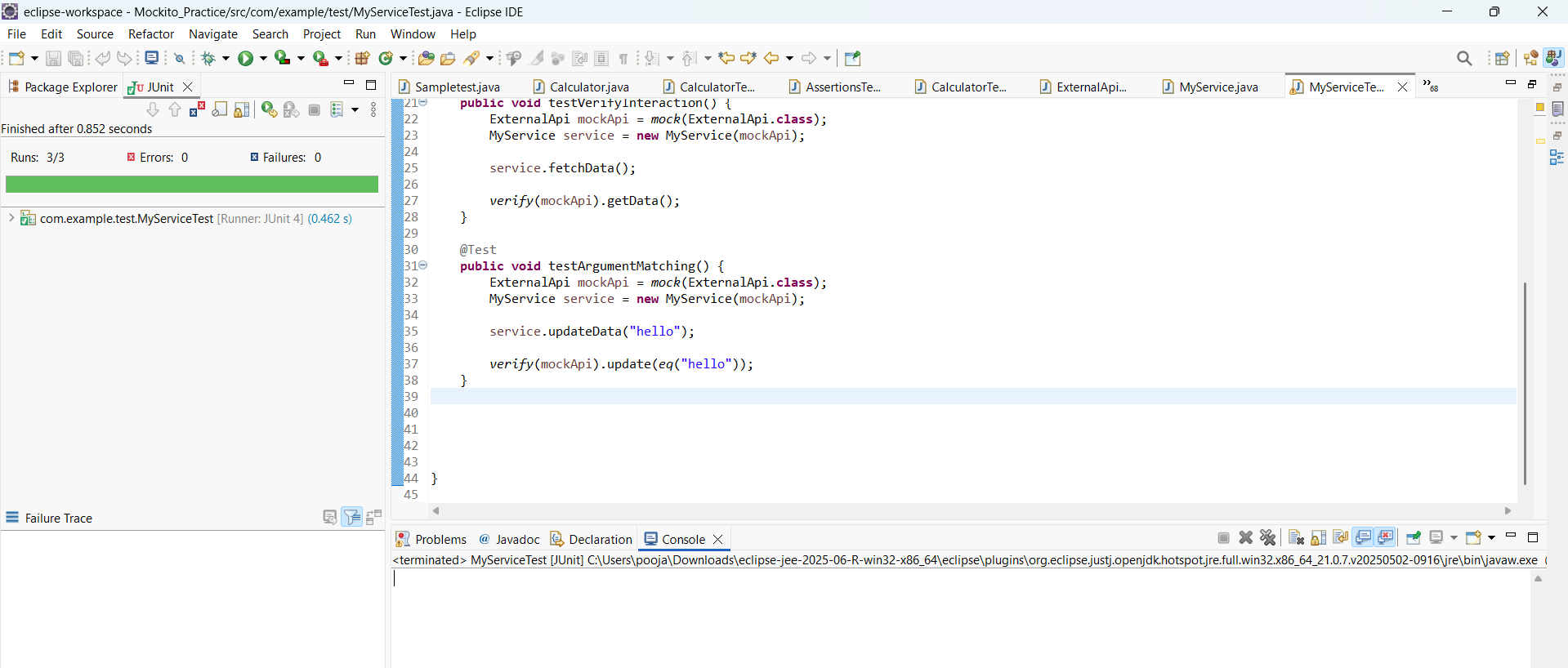
ExternalApi mockApi = *mock*(ExternalApi.**class**);

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

service.updateData("hello");

*verify*(mockApi).update(*eq*("hello"));

}



**Exercise 4: Handling Void Methods Scenario**

@Test

**public** **void** testVoidMethod() {

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

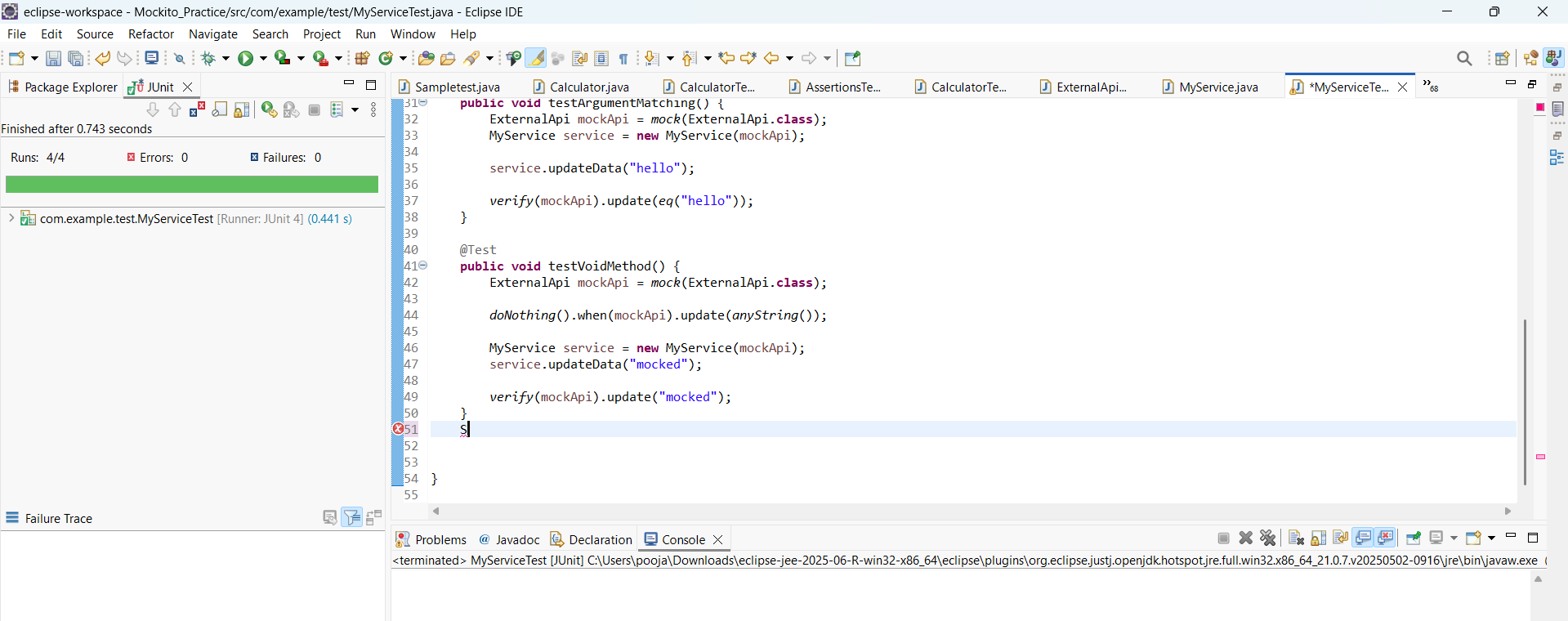
*doNothing*().when(mockApi).update(*anyString*());

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

service.updateData("mocked");

*verify*(mockApi).update("mocked");

}



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

@Test

**public** **void** testMultipleReturns() {

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

*when*(mockApi.getData())

.thenReturn("First Call")

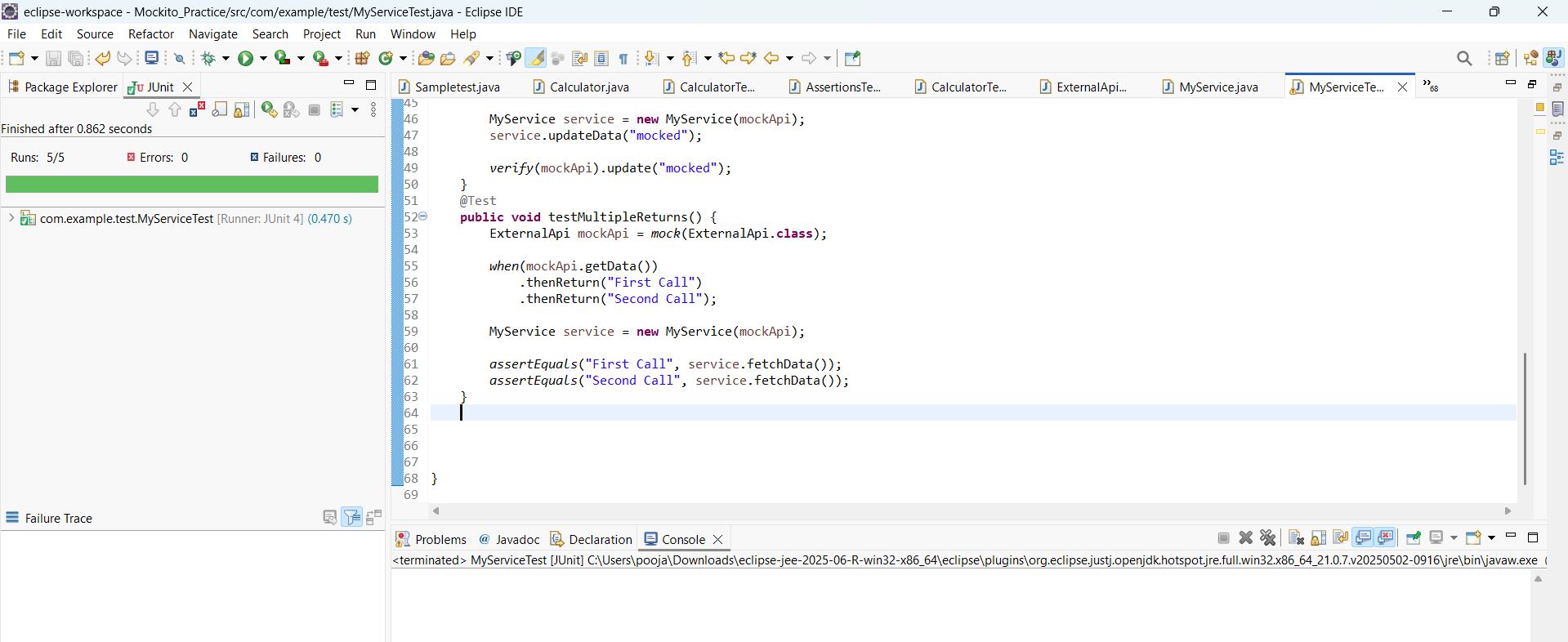
.thenReturn("Second Call");

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

*assertEquals*("First Call", service.fetchData());

*assertEquals*("Second Call", service.fetchData());

}



**Exercise 6: Verifying Interaction Order Scenario**

@Test

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

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

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

service.fetchData();

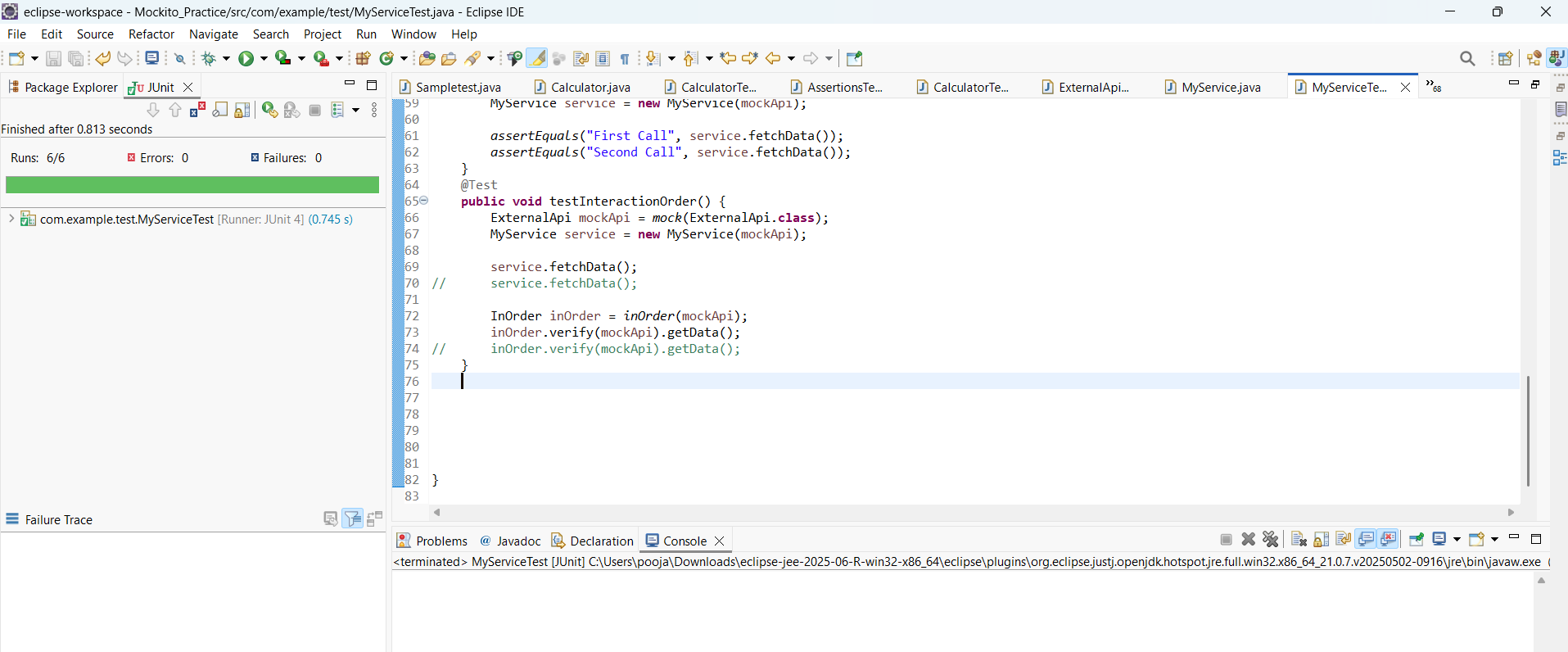
service.fetchData();

InOrder inOrder = *inOrder*(mockApi);

inOrder.verify(mockApi).getData();

inOrder.verify(mockApi).getData();

}



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

@Test(expected = RuntimeException.**class**)

**public** **void** testVoidThrowsException() {

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

*doThrow*(**new** RuntimeException("Error"))

.when(mockApi)

.riskyOperation();

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

service.triggerRiskyOperation();

}

