**MOCKITO EXERCISES**

**1.Mocking And Stubbing   
  
ExternalApi.java**

public interface ExternalApi {

String getData();

}  
  
**MyService.java**

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**MyServiceTest.java**

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = mock(ExternalApi.class);

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

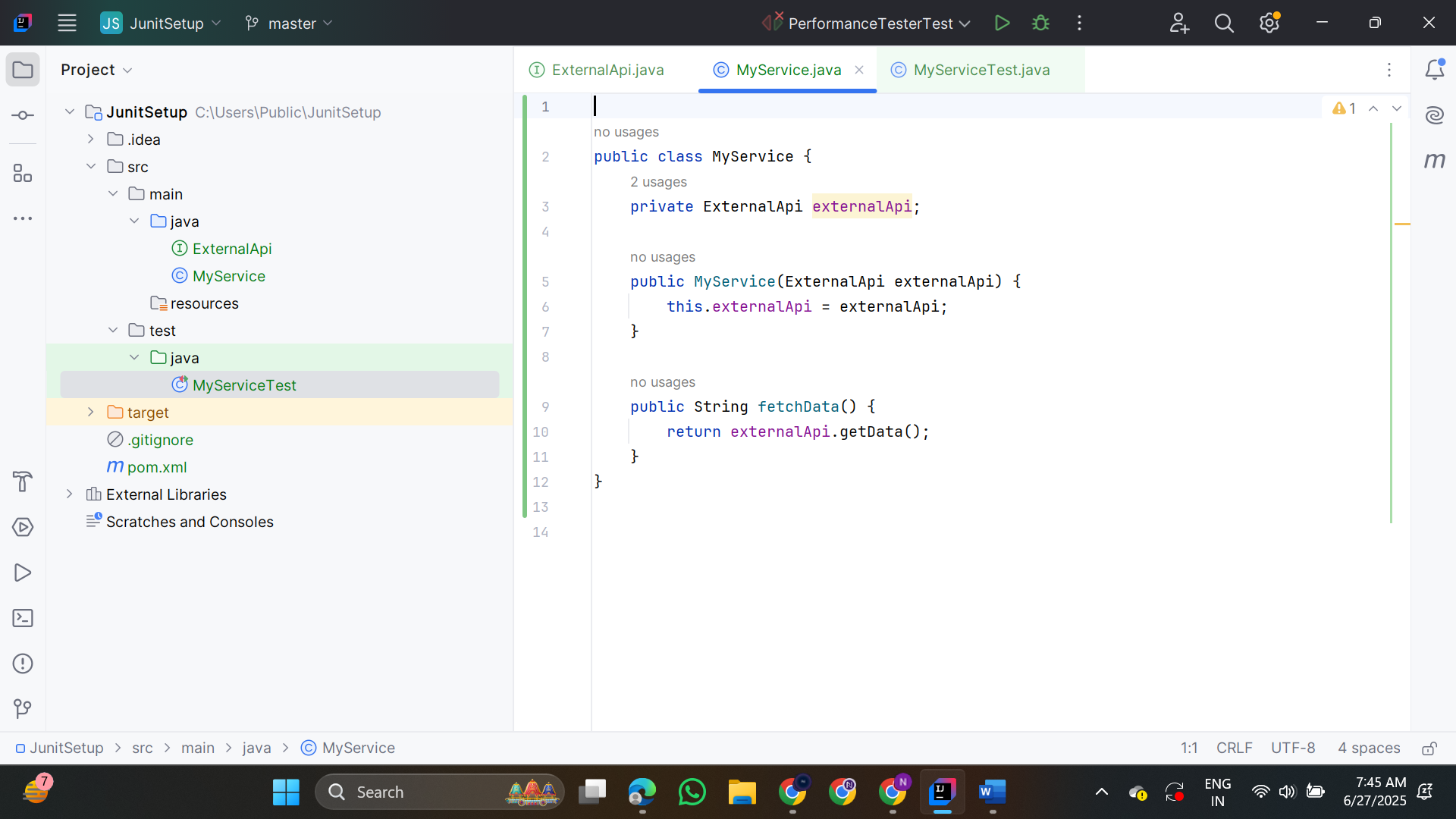
MyService service = new MyService(mockApi);

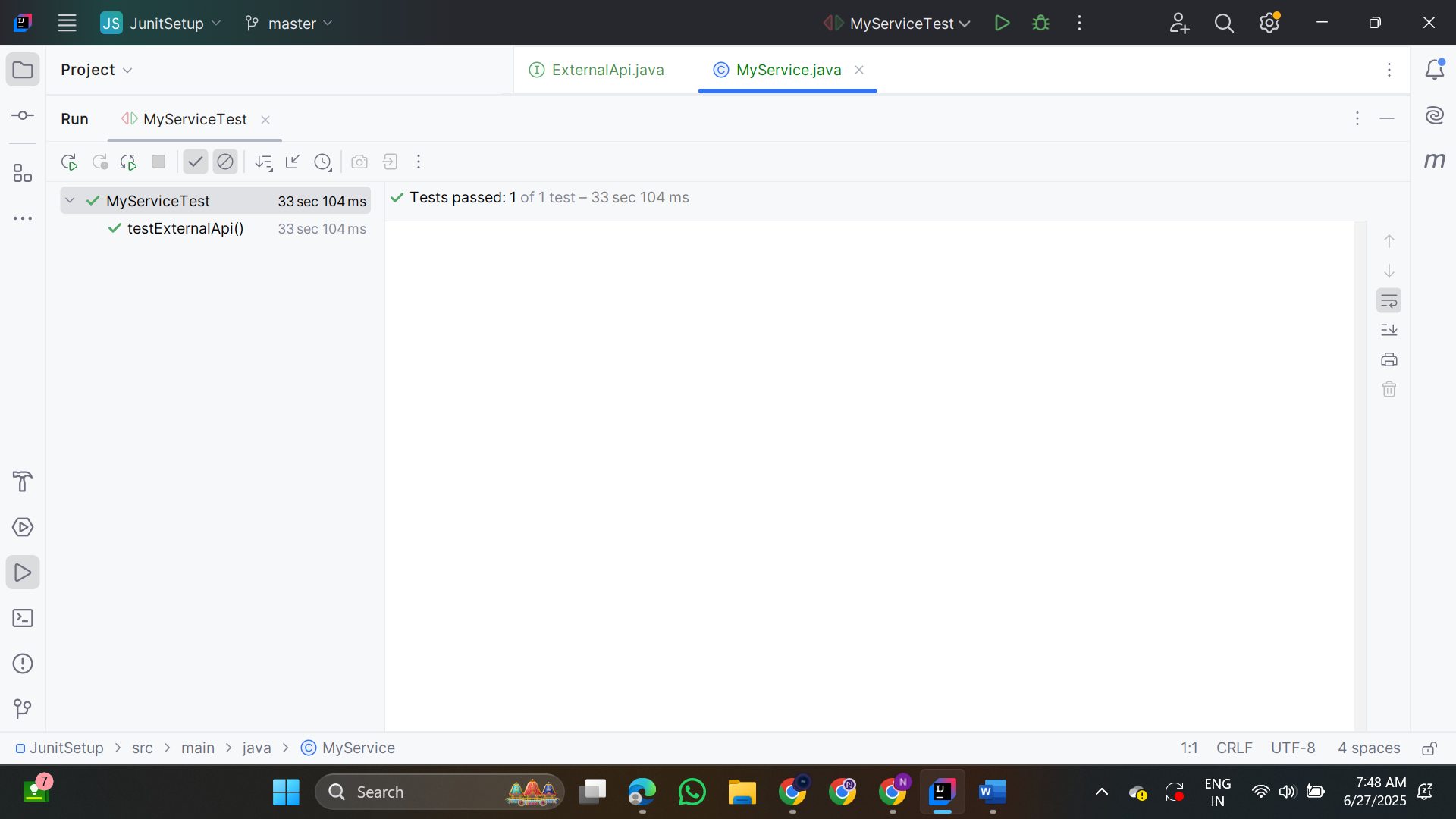
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}





**2.Verifying Interactions**

* Ensure that a method is called with specific arguments
* Mockito's verify method
* Check that MyService calls ExternalApi’s getData() method

**ExternalApi.java**

public interface ExternalApi {

String getData();

}  
  
**MyService.java**

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.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);

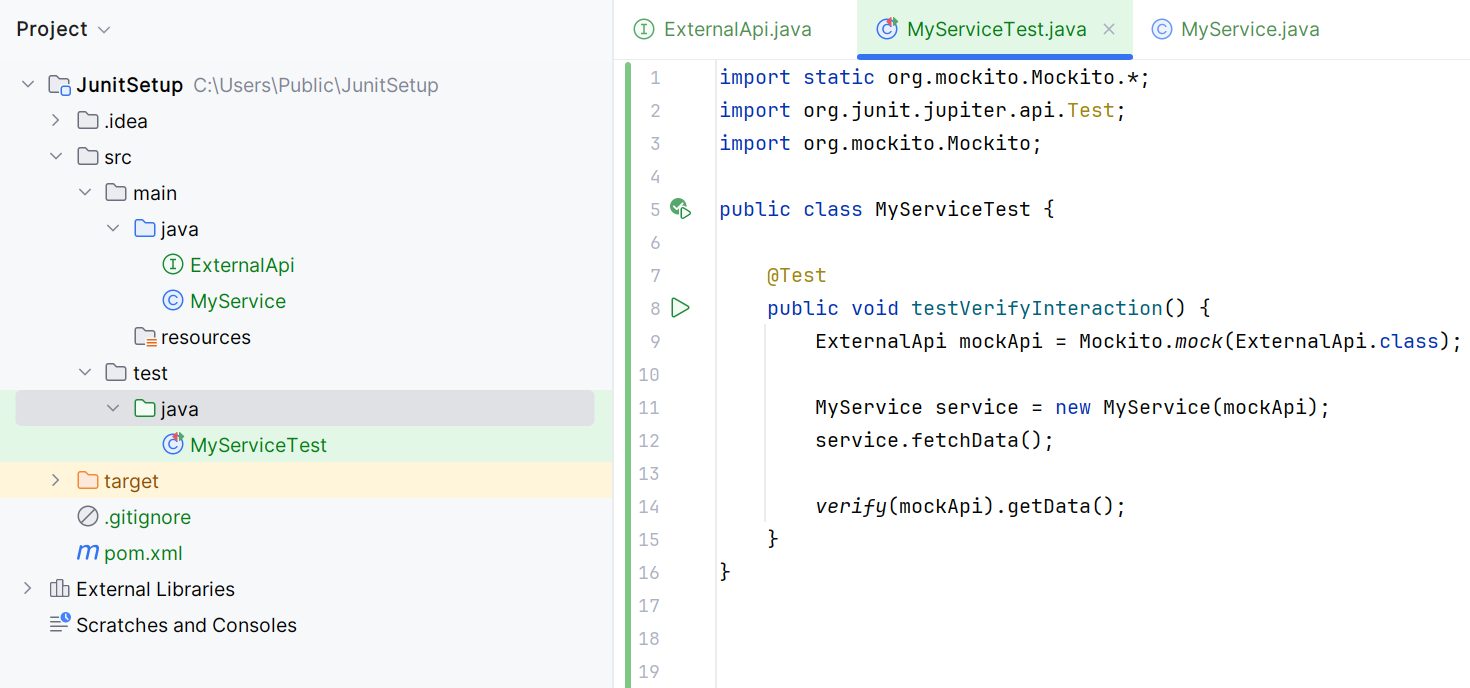
MyService service = new MyService(mockApi);

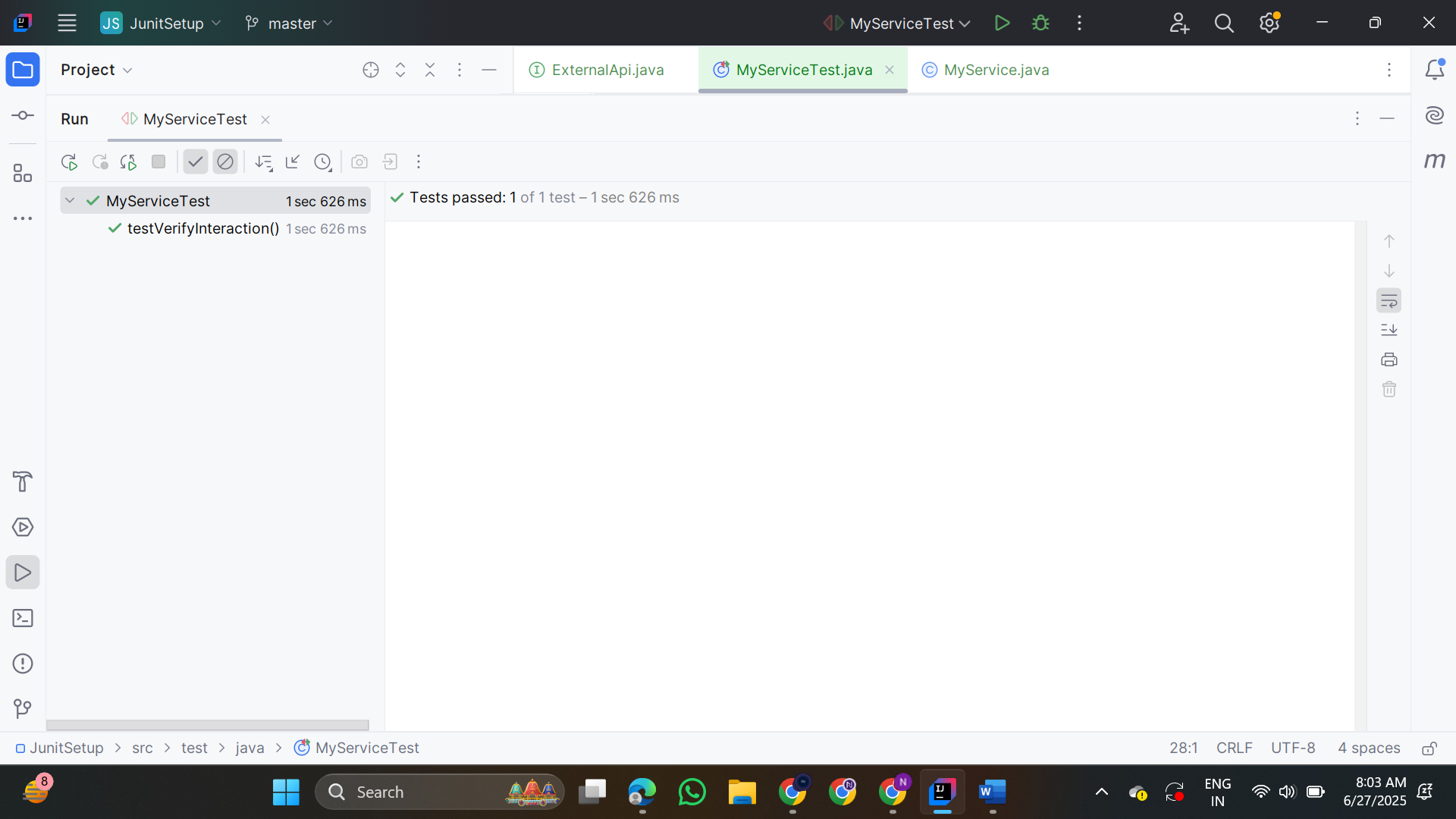
service.fetchData();

verify(mockApi).getData();

}

}





**3.Argument Matching**

**ExternalApi.java**

public interface ExternalApi {

void sendMessage(String user, String message);

}

**MyService.java**

public class MyService {

private final ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public void notifyUser(String user) {

externalApi.sendMessage(user, "Welcome!");

}

}

**MyServiceTest.java**

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testArgumentMatching() {

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

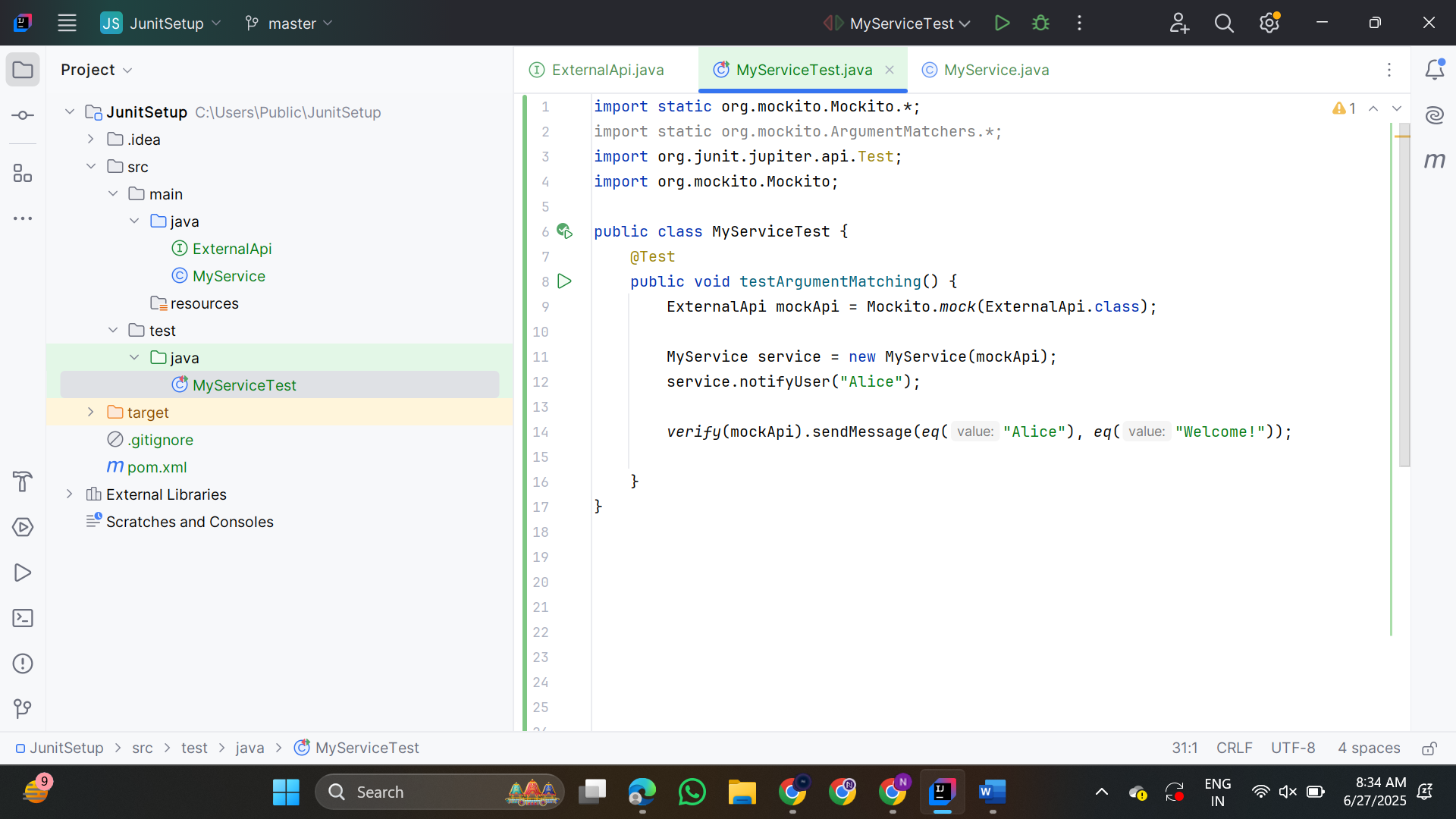
MyService service = new MyService(mockApi);

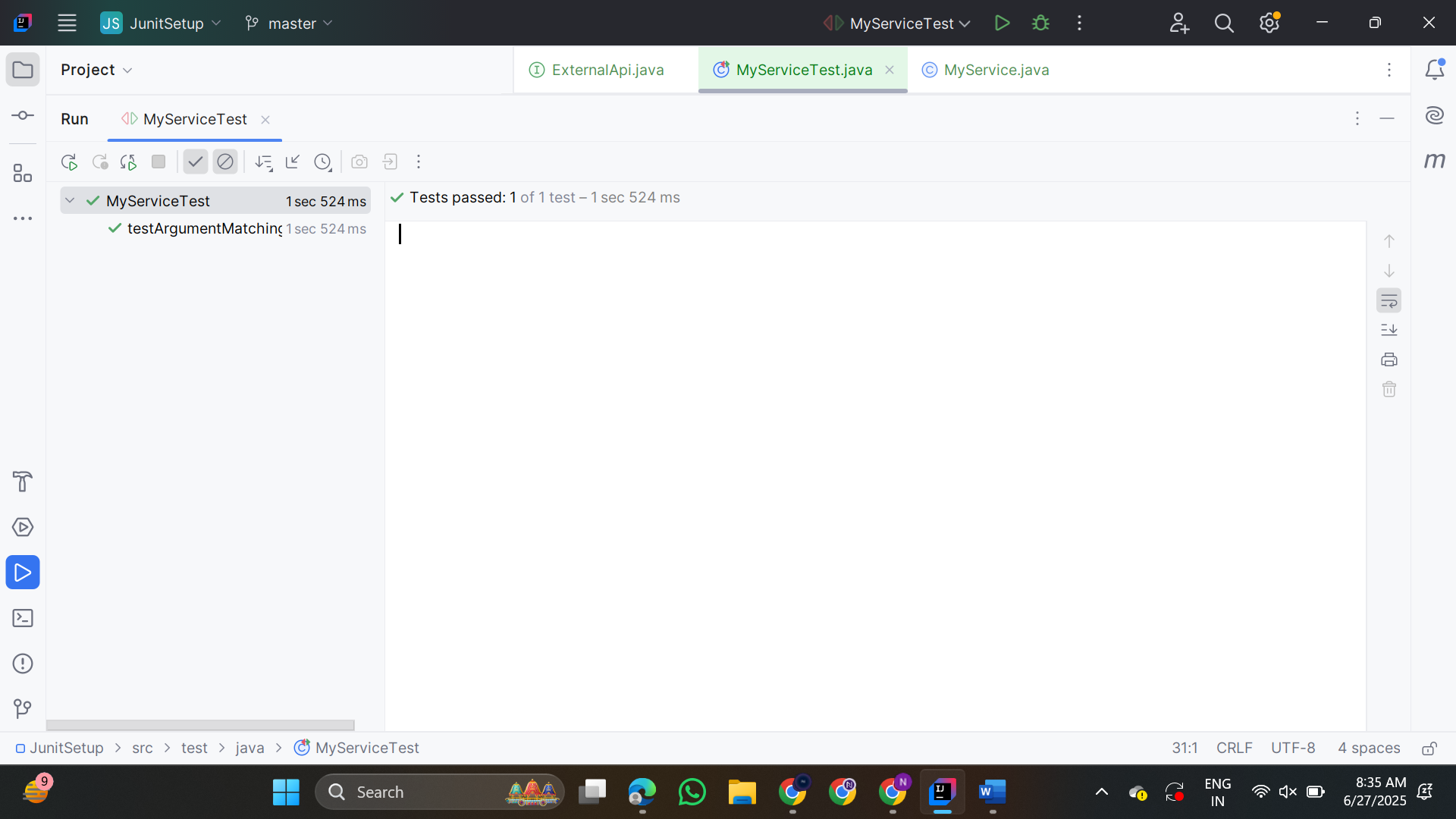
service.notifyUser("Alice");

verify(mockApi).sendMessage(eq("Alice"), eq("Welcome!"));

}

}





**4.Handling Void Methods**

**LoggerService.java**

public interface LoggerService {

void log(String message);

}

**MyProcessor.java**

public class MyProcessor {

private final LoggerService loggerService;

public MyProcessor(LoggerService loggerService) {

this.loggerService = loggerService;

}

public void process() {

// perform some logic

loggerService.log("Processing completed");

}

}

**MyProcessorTest.java**

import static org.mockito.Mockito.\*;

import static org.mockito.Mockito.doNothing;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyProcessorTest {

@Test

public void testVoidMethodInteraction() {

LoggerService mockLogger = Mockito.mock(LoggerService.class);

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

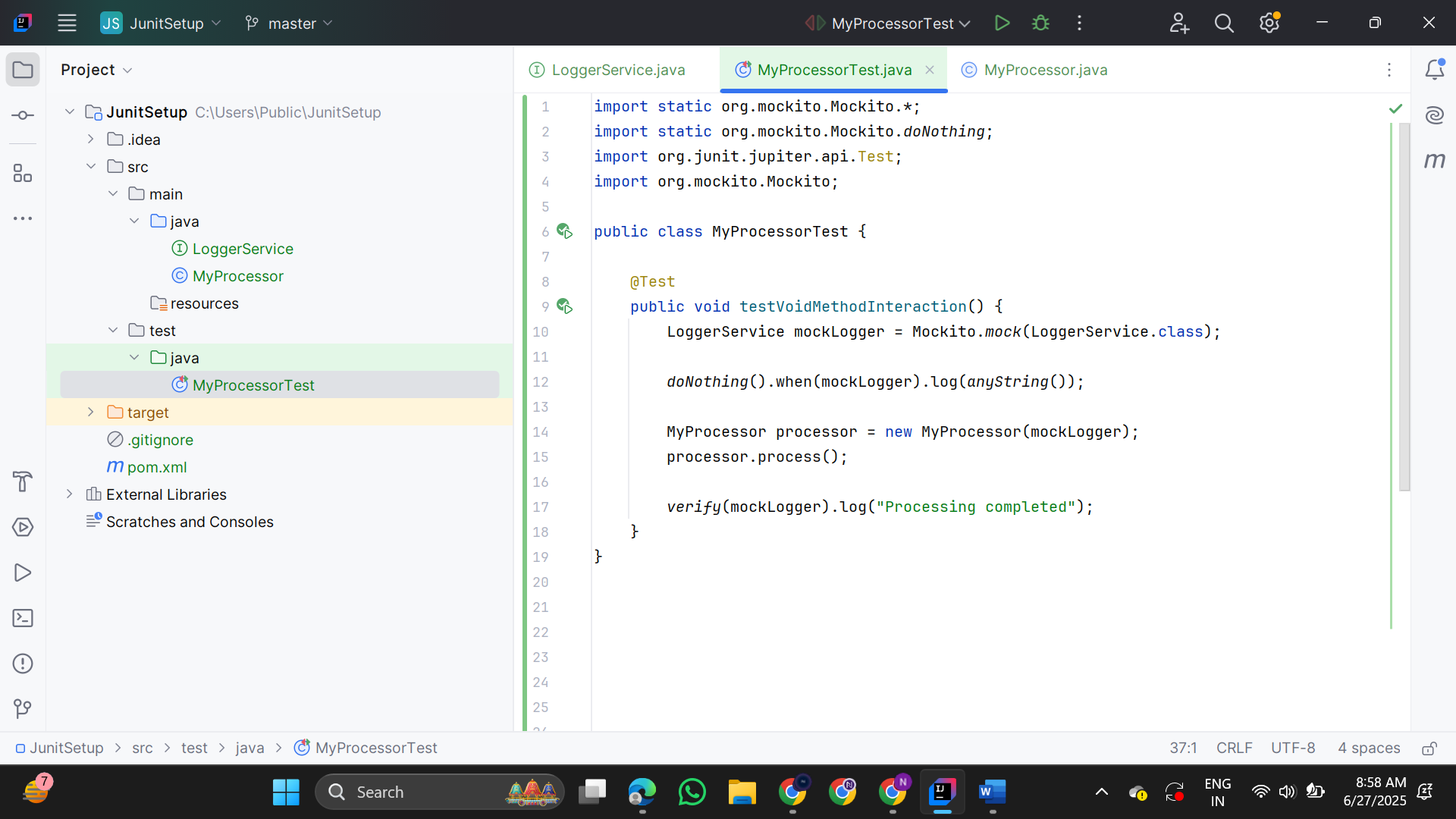
MyProcessor processor = new MyProcessor(mockLogger);

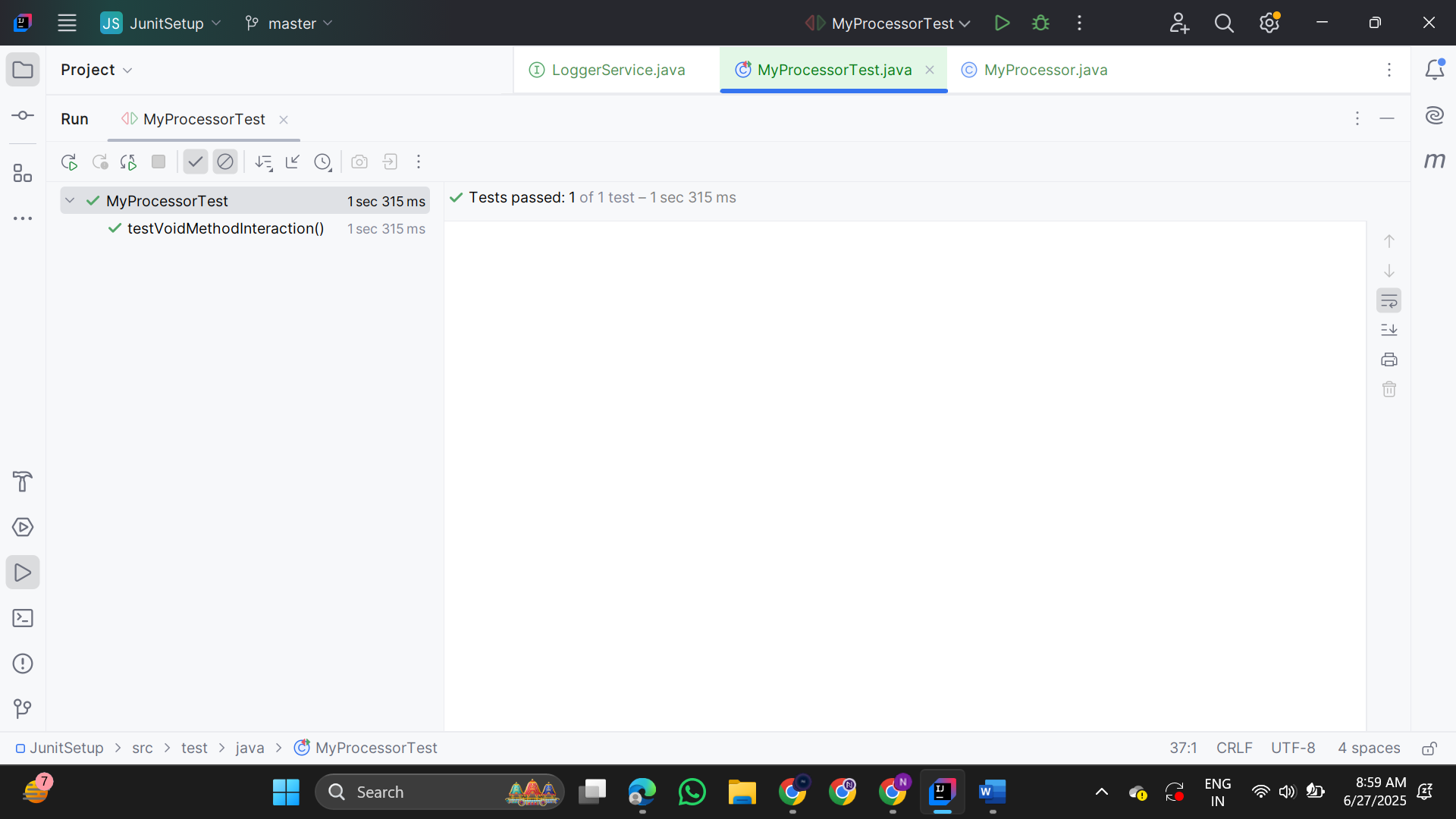
processor.process();

verify(mockLogger).log("Processing completed");

}

}





**5.Mocking And Stubbing With Multiple Returns**

**ExternalApi.java**

public interface ExternalApi {

String getData();

}

**MyService.java**

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String[] fetchDataMultipleTimes() {

return new String[] {

api.getData(),

api.getData(),

api.getData()

};

}

}

**MyServiceTest.java**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testMultipleReturnsFromApi() {

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

when(mockApi.getData())

.thenReturn("First Call")

.thenReturn("Second Call")

.thenReturn("Third Call");

MyService service = new MyService(mockApi);

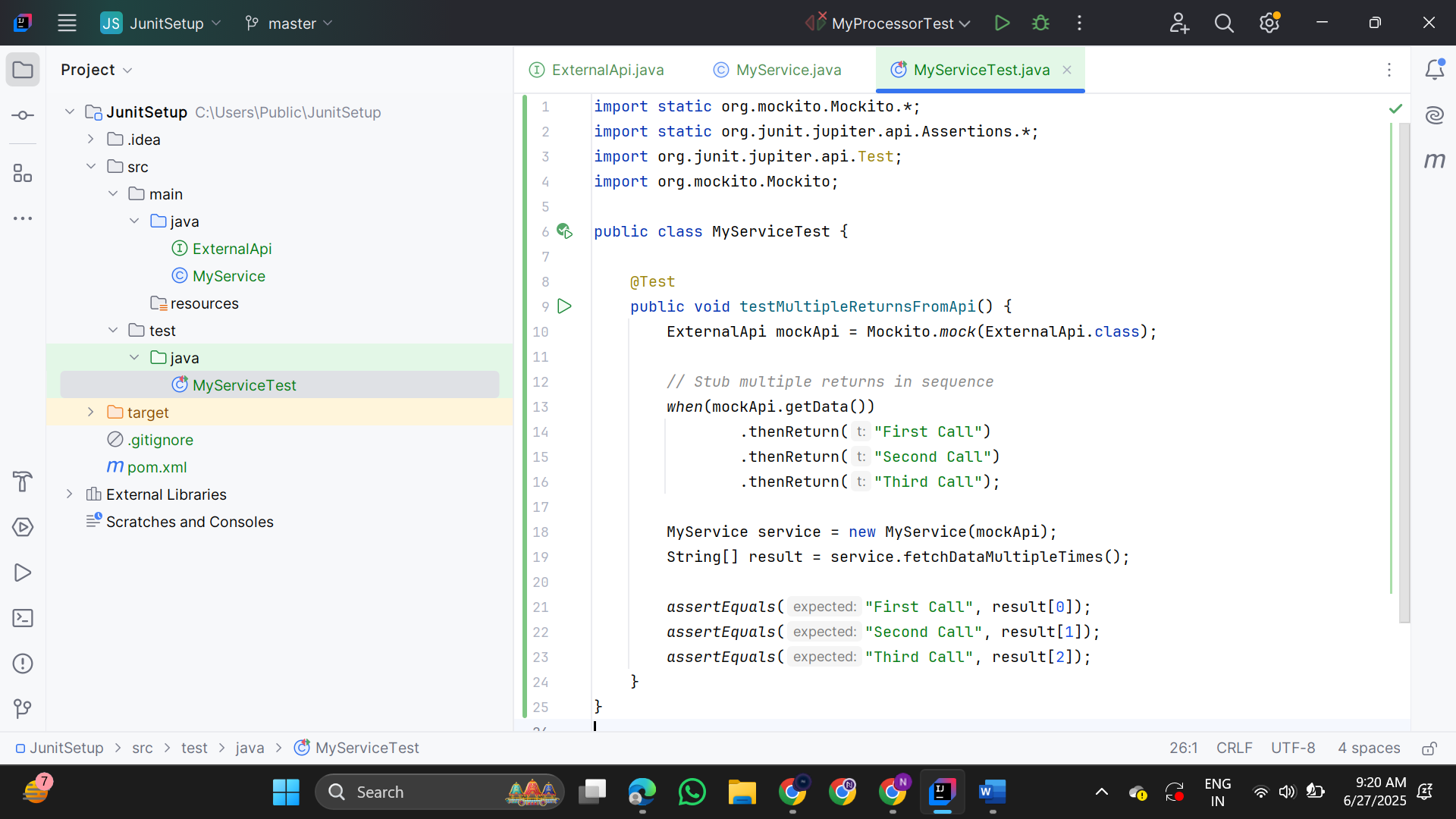
String[] result = service.fetchDataMultipleTimes();

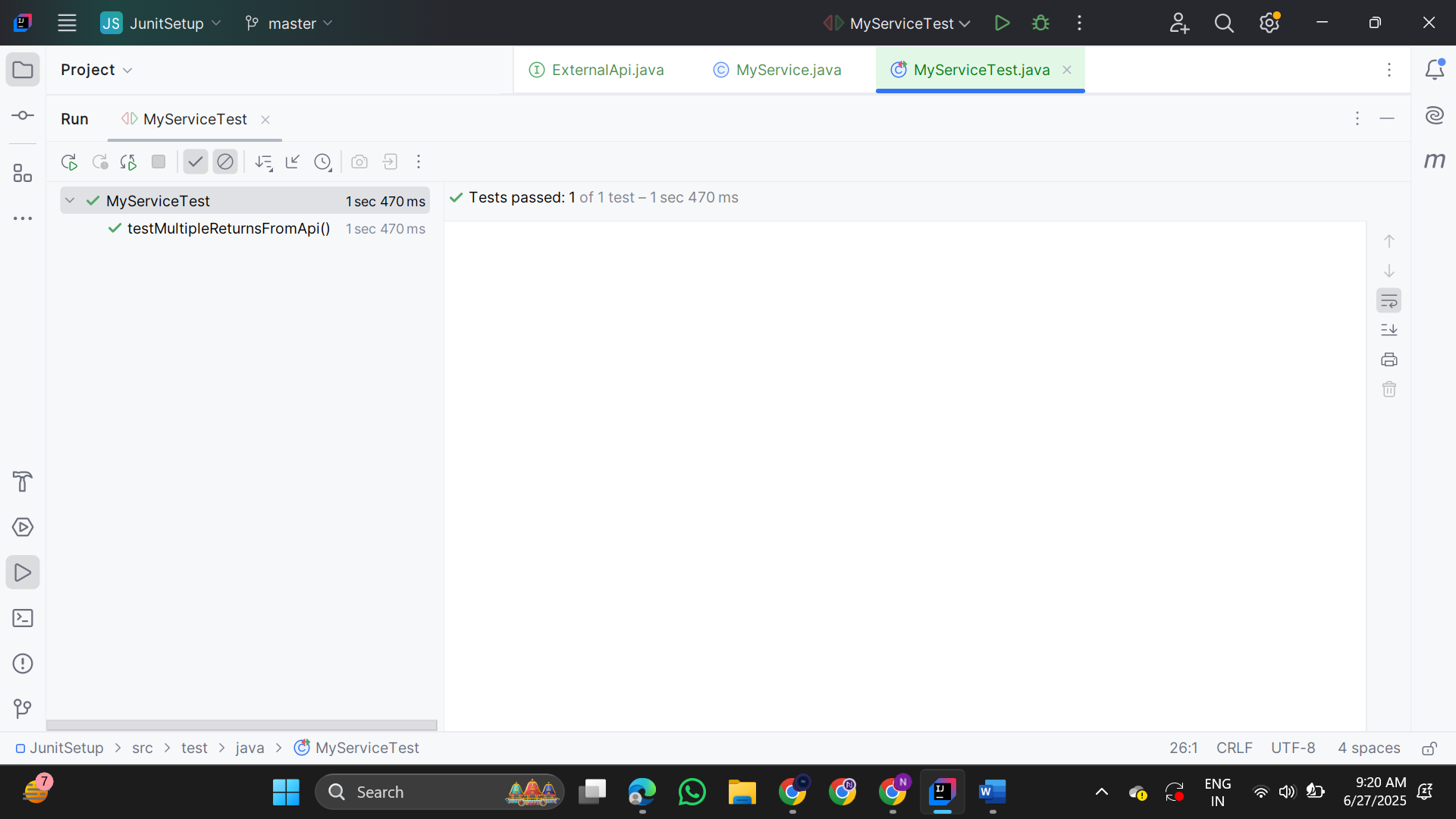
assertEquals("First Call", result[0]);

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

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

}}





**6.Verifying Interaction Order**

**ExternalApi.java**

public interface ExternalApi {

void start();

void process();

void finish();

}

**MyService.java**

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void execute() {

api.start();

api.process();

api.finish();

}

}

**MyServiceTest.java**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

public class MyServiceTest {

@Test

public void testMethodCallOrder() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.execute();

InOrder inOrder = inOrder(mockApi);

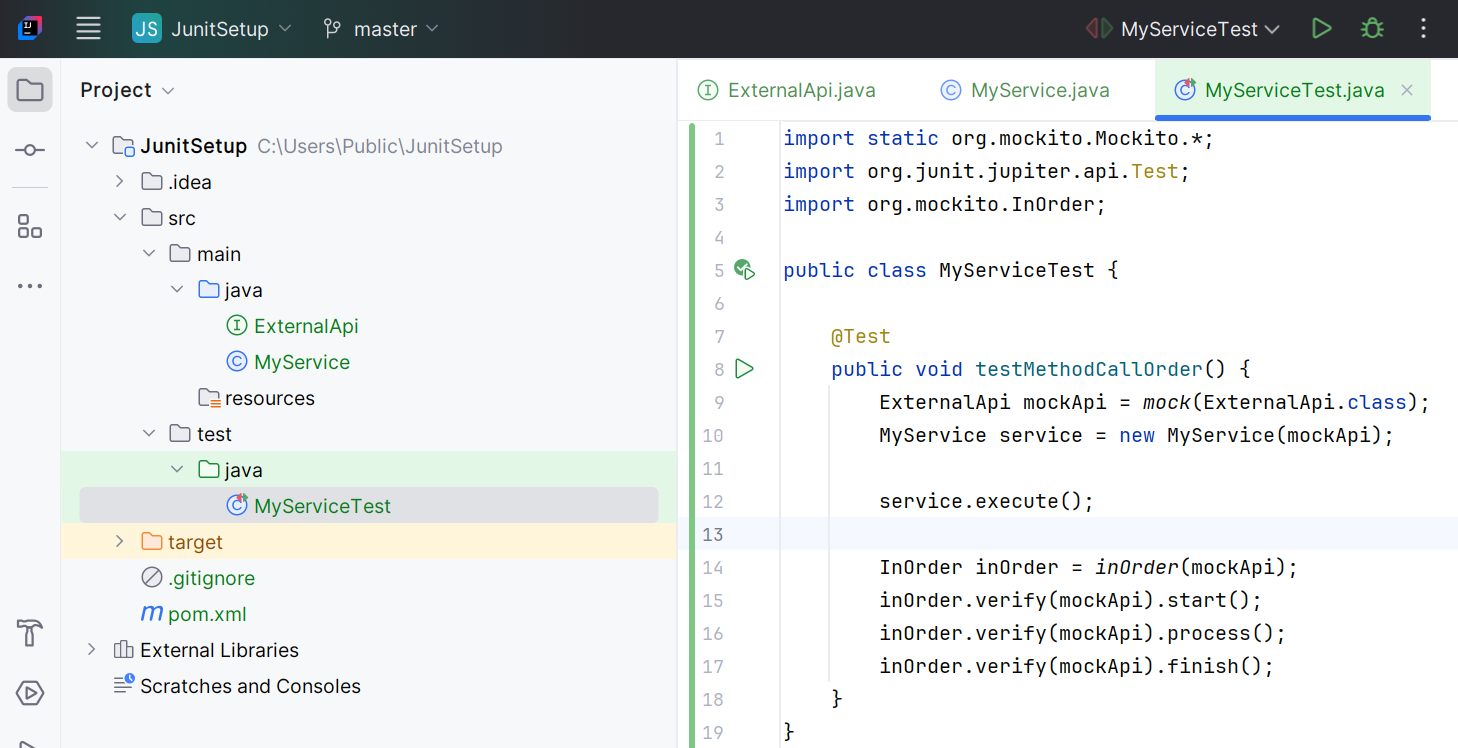
inOrder.verify(mockApi).start();

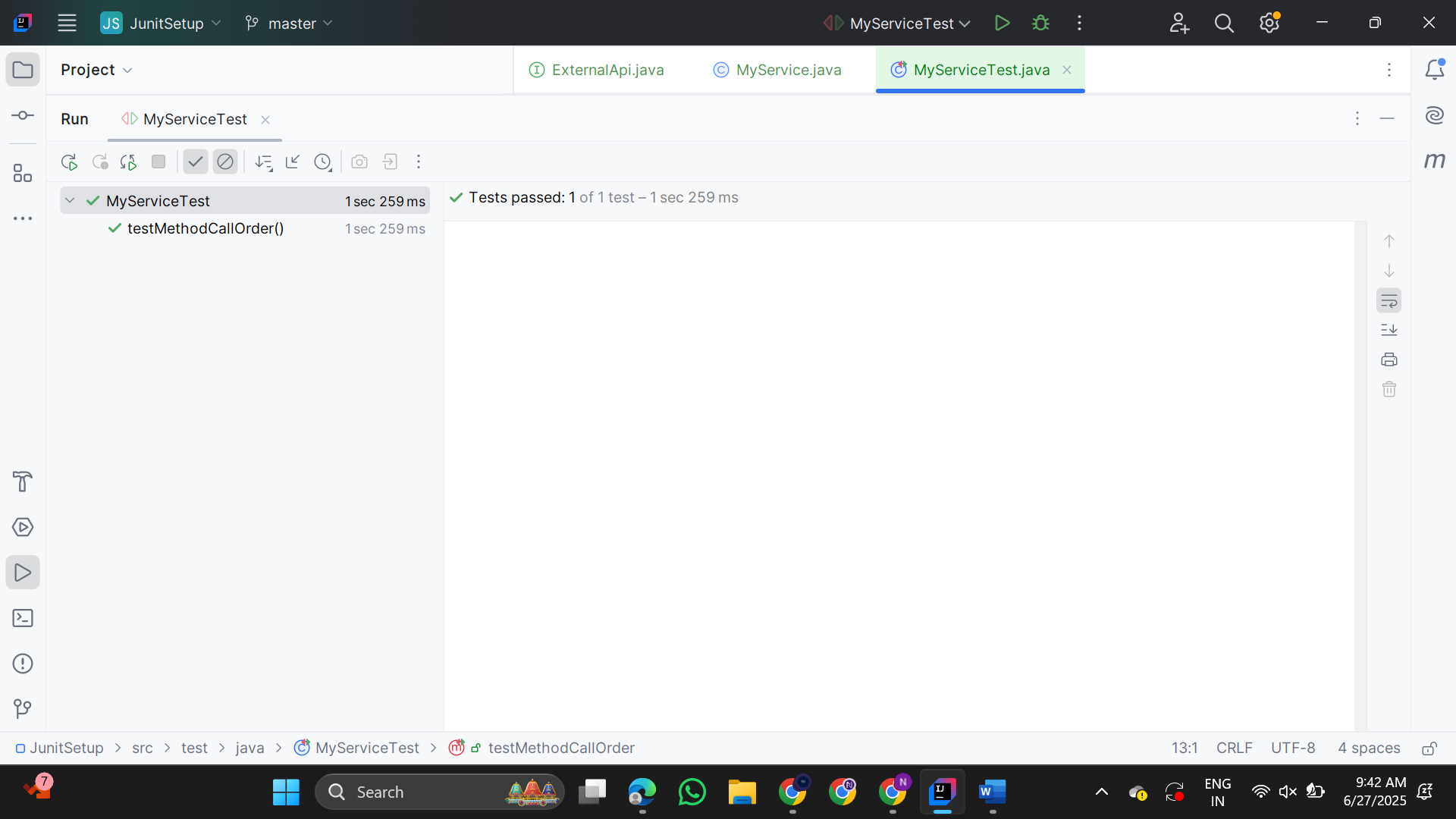
inOrder.verify(mockApi).process();

inOrder.verify(mockApi).finish();

}

}





**7.Handling Void Methods With Exceptions**

**ExternalApi.java**

public interface ExternalApi {

void deleteResource();

}

**MyService.java**

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void performDeletion() {

api.deleteResource();

}

}

**MyServiceTest.java**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVoidMethodThrowsException() {

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

doThrow(new RuntimeException("Deletion failed")).when(mockApi).deleteResource();

MyService service = new MyService(mockApi);

Exception exception = assertThrows(RuntimeException.class, service::performDeletion);

assertEquals("Deletion failed", exception.getMessage());

verify(mockApi).deleteResource();

}

}

