JUNIT TESTING EXERCISE

EXERCISE 1 :JUnitProject – Basic Java Unit Testing with Junit(MANDATORY)

**Calculator.java**

package com.sharu.junit;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorTest.java**

package com.sharu.junit;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class CalculatorTest {

@Test

public void testAdd() {

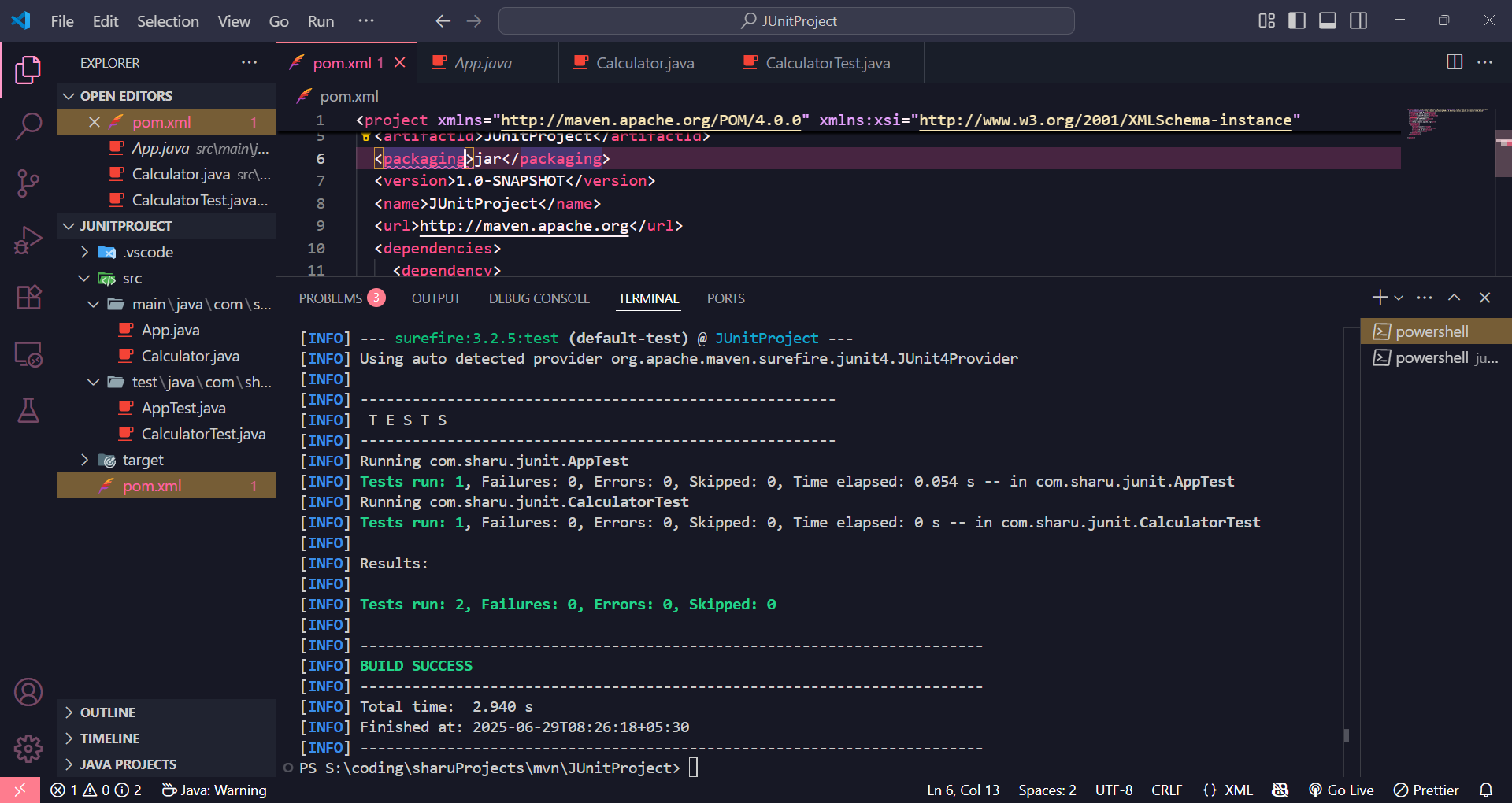
Calculator calc = new Calculator();

assertEquals(5, calc.add(2, 3));

}

}

**Output :**

****

EXERCISE 2 :JUnit Testing Setup for Simple Calculator Class

**Calculator.java**

package com.sharu.junit;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

**CalculatorTest.java**

package com.sharu.junit;

import org.junit.jupiter.api.Test;

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

public class CalculatorTest {

@Test

void testAdd() {

Calculator calc = new Calculator();

assertEquals(5, calc.add(2, 3));

}

@Test

void testSubtract() {

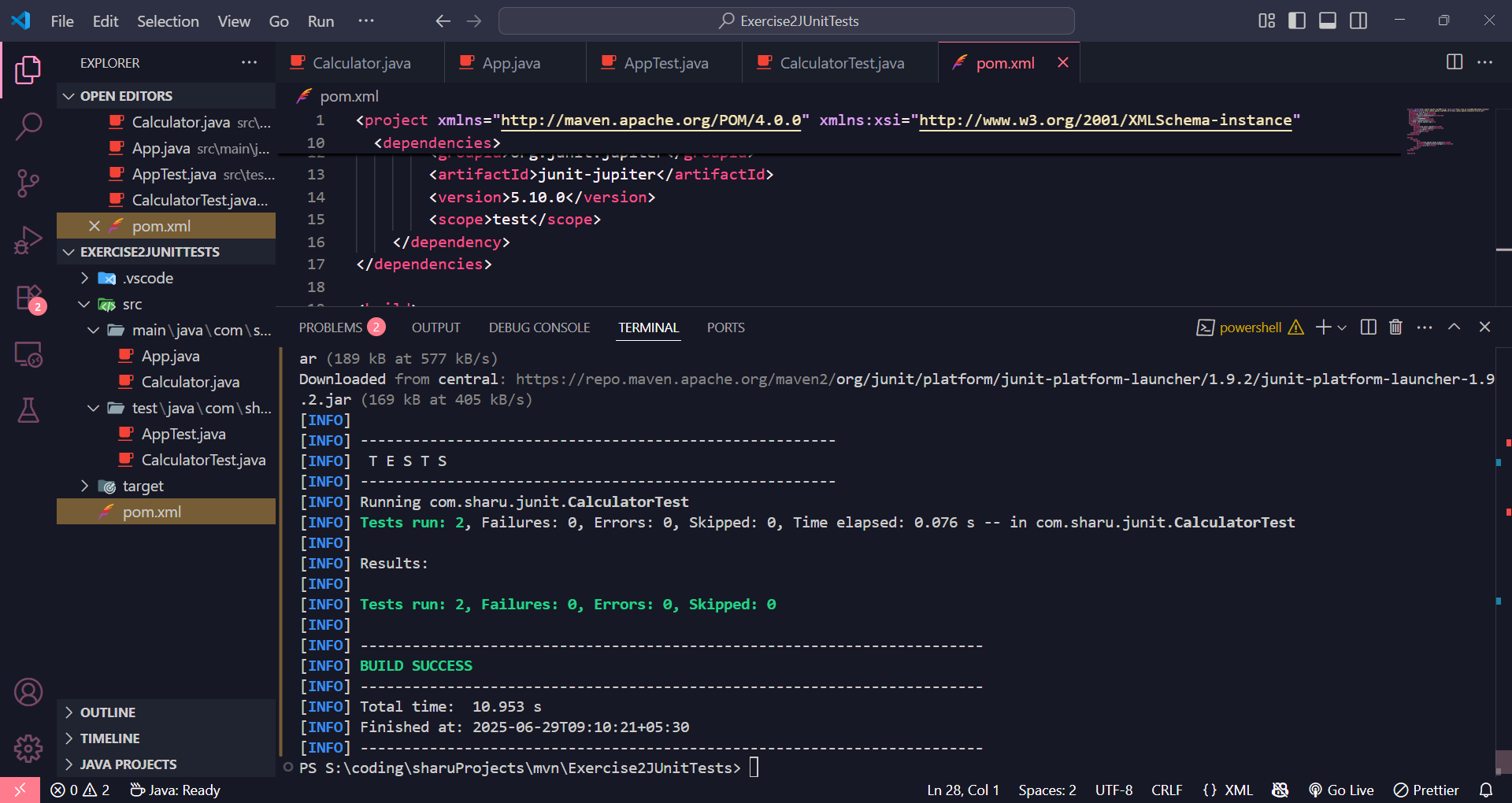
Calculator calc = new Calculator();

assertEquals(1, calc.subtract(3, 2));

}

}

**Output:**

****

EXERCISE 3 : JUnit Assertions Demonstration(MANDATORY)

**AssertionsTest.java**

package com.sharu.junit;

import org.junit.jupiter.api.Test;

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

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals(5, 2 + 3);

assertTrue(5 > 3);

assertFalse(5 < 3);

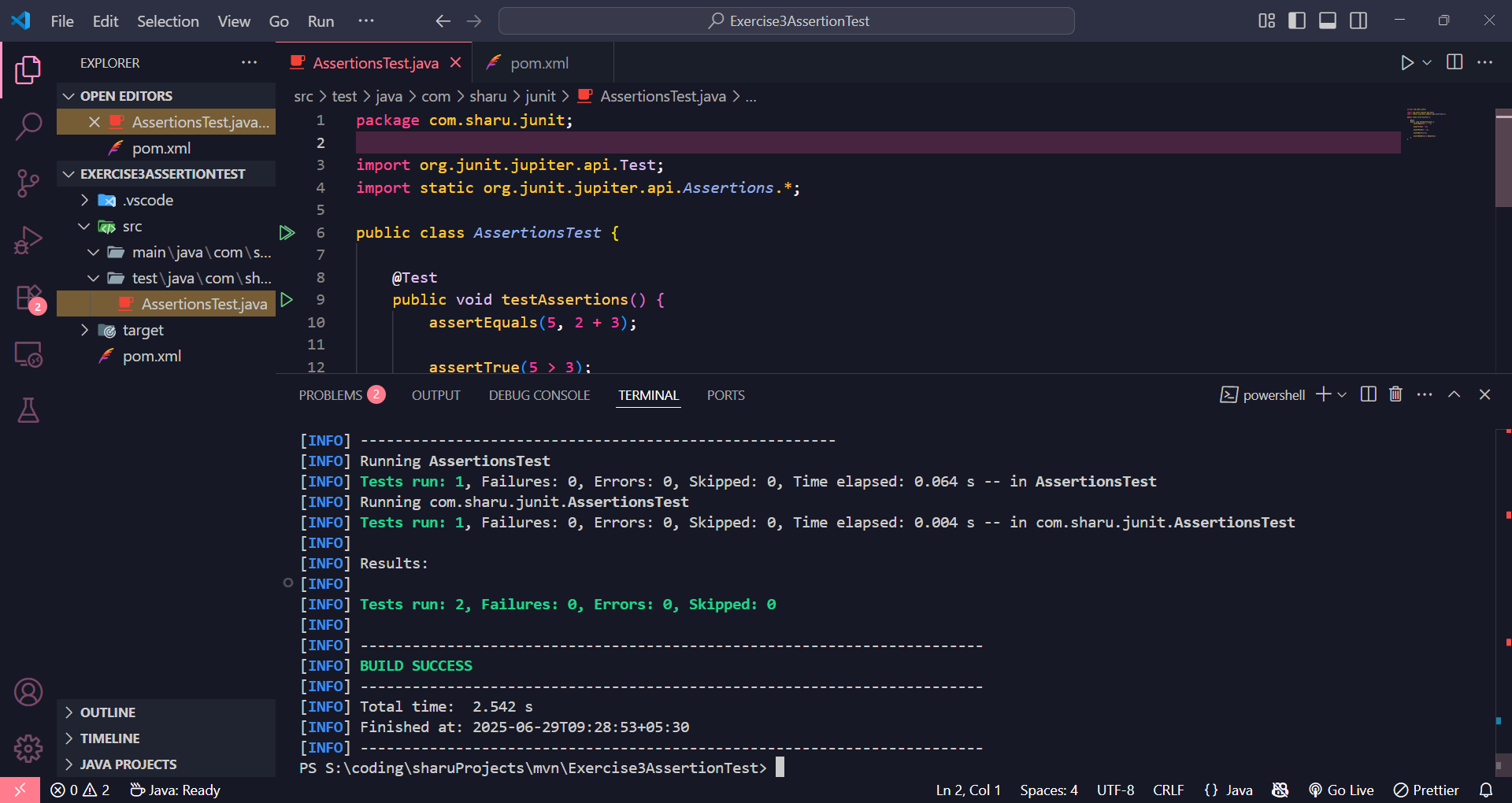
assertNull(null);

assertNotNull(new Object());

}

}

**Output:**

****

EXERCISE 4 : JUnit Testing with Arrange-Act-Assert (AAA) Pattern and Test Fixtures(MANDATORY)

**CalculatorTest.java**

package com.sharu.junit;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.Test;

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

public class CalculatorTest {

private Calculator calc;

@BeforeEach

public void setUp() {

calc = new Calculator();

}

@AfterEach

public void tearDown() {

calc = null;

}

@Test

public void testAddition() {

int result = calc.add(2, 3);

assertEquals(5, result);

}

@Test

public void testSubtraction() {

int result = calc.subtract(5, 3);

assertEquals(2, result);

}

}

**Calculator.java**

package com.sharu.junit;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

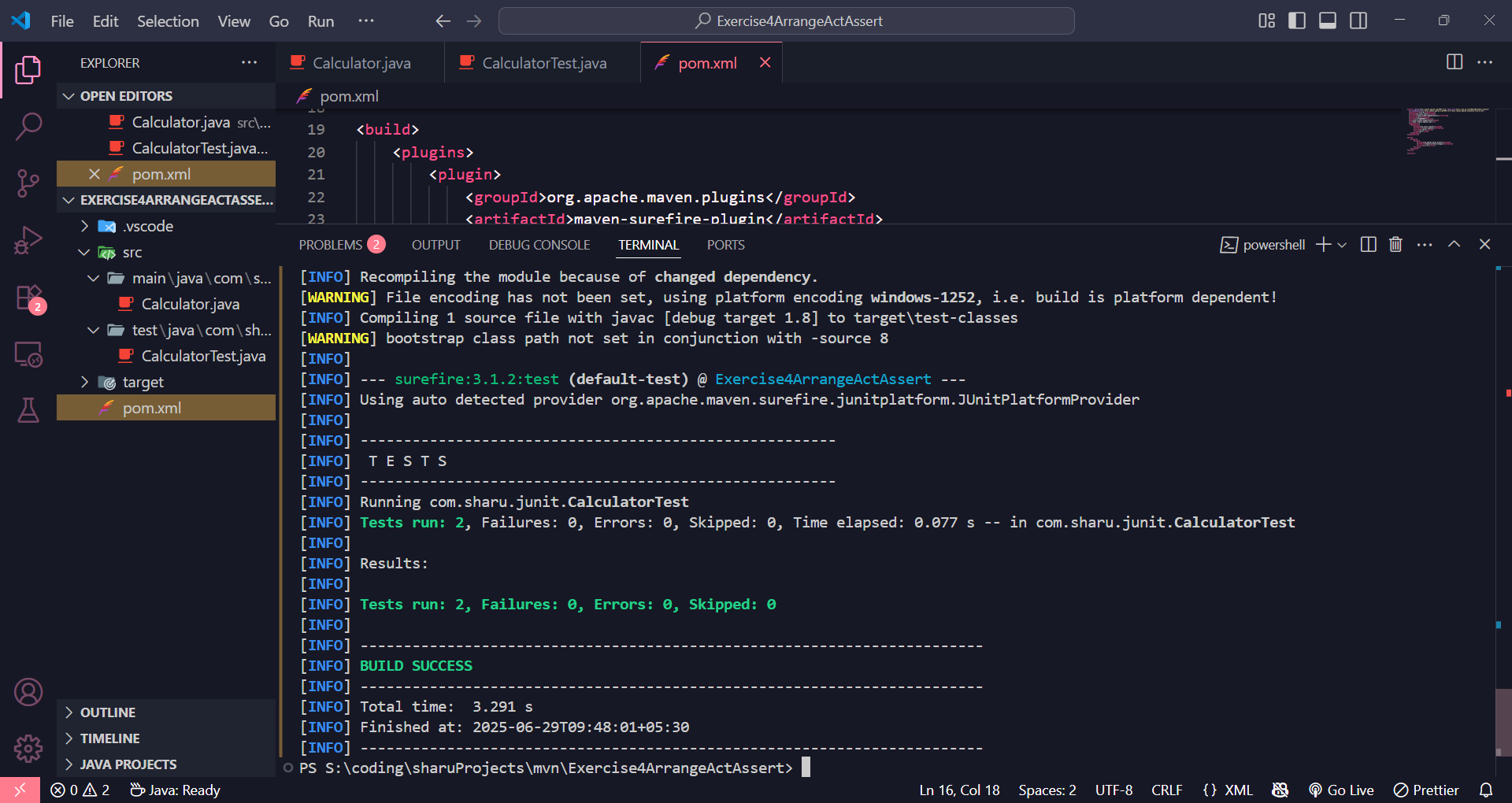
public int subtract(int a, int b) {

return a - b;

}

}

**Output:**



JUNIT ADVANCED EXERCISE

EXERCISE 1 : Advanced JUnit Testing Parameterized Tests for Even Number Checker

**EvenChecker.java**

package com.sharu.junit;

public class EvenChecker {

public boolean isEven(int number) {

return number % 2 == 0;

}

}

**EvenCheckerTest.java**

package com.sharu.junit;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.ValueSource;

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

public class EvenCheckerTest {

private final EvenChecker checker = new EvenChecker();

@ParameterizedTest

@ValueSource(ints = {2, 4, 6, 8, 10, 0, -2, -4})

public void testIsEven\_withEvenNumbers(int number) {

assertTrue(checker.isEven(number), number + " should be even");

}

@ParameterizedTest

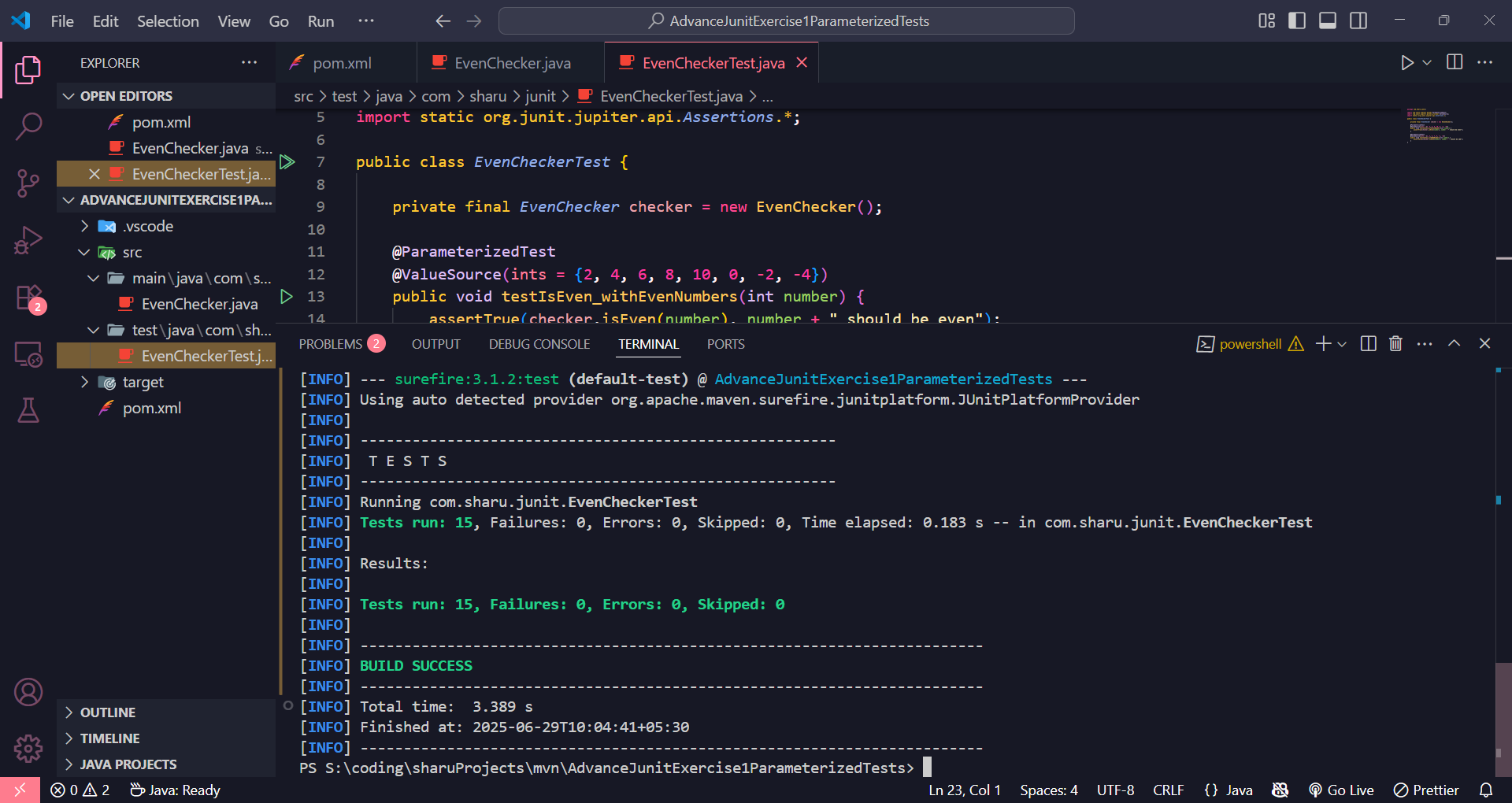
@ValueSource(ints = {1, 3, 5, 7, 9, -1, -3})

public void testIsEven\_withOddNumbers(int number) {

assertFalse(checker.isEven(number), number + " should be odd");

}

}

**Output**

EXERCISE 2 : JUnit Test Suites and Grouping Test Classes

**Calculator.java**

package com.sharu.junit;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

**File: CalculatorAddTest.java**

package com.sharu.junit;

import org.junit.jupiter.api.Test;

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

public class CalculatorAddTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

assertEquals(5, calc.add(2, 3));

}

}

**CalculatorSubtractTest.java**

package com.sharu.junit;

import org.junit.jupiter.api.Test;

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

public class CalculatorSubtractTest {

@Test

public void testSubtract() {

Calculator calc = new Calculator();

assertEquals(2, calc.subtract(5, 3));

}

}

**AllTests.java**

package com.sharu.junit;

import org.junit.platform.suite.api.SelectClasses;

import org.junit.platform.suite.api.Suite;

@Suite

@SelectClasses({

CalculatorAddTest.class,

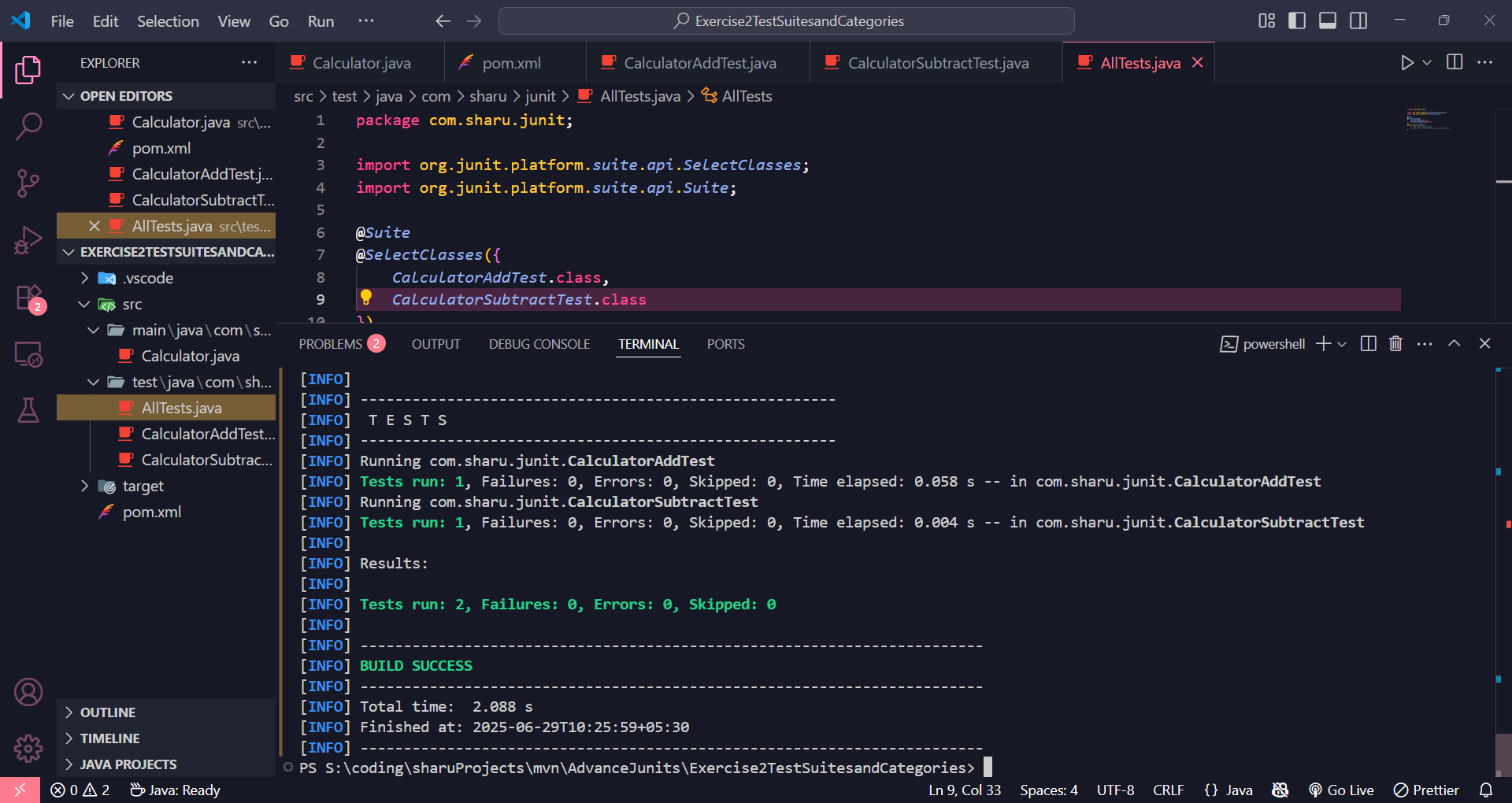
CalculatorSubtractTest.class

})

public class AllTests {

}

**Output**



EXERCISE 3 : JUnit Test Execution Order Control using @TestMethodOrder

**OrderedTests.java**

package com.sharu.junit;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.Order;

import org.junit.jupiter.api.TestMethodOrder;

import org.junit.jupiter.api.MethodOrderer.OrderAnnotation;

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

@TestMethodOrder(OrderAnnotation.class)

public class OrderedTests {

@Test

@Order(3)

public void testC() {

System.out.println("Running testC");

assertTrue(true);

}

@Test

@Order(1)

public void testA() {

System.out.println("Running testA");

assertEquals(2 + 2, 4);

}

@Test

@Order(2)

public void testB() {

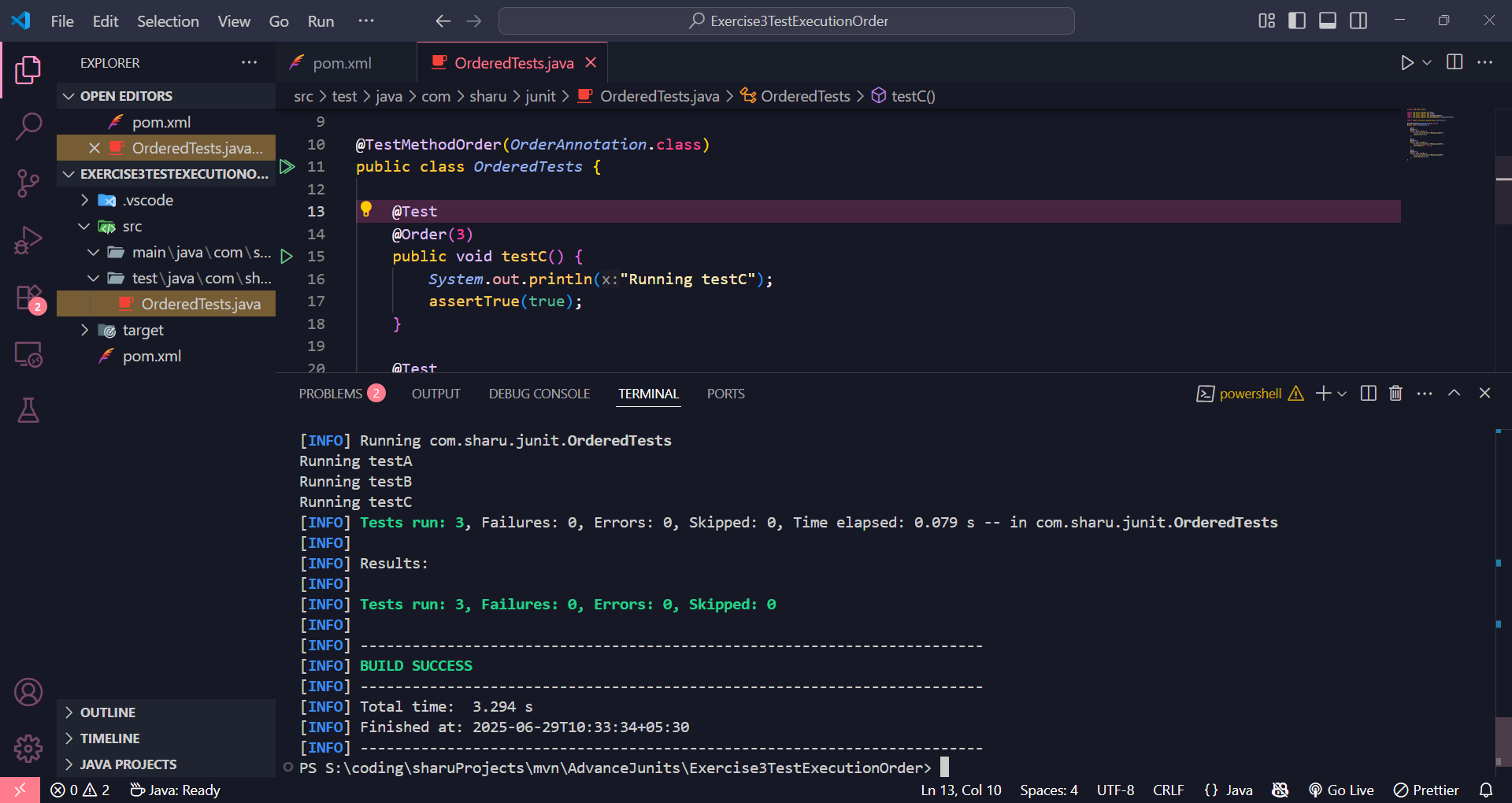
System.out.println("Running testB");

assertFalse(false);

}

}

**Output**



EXERCISE 4 : JUnit Exception Testing with assertThrows

**ExceptionThrower.java**

package com.sharu.junit;

public class ExceptionThrower {

public void throwException() {

throw new IllegalArgumentException("This is an illegal argument");

}

}

**ExceptionThrowerTest.java**

package com.sharu.junit;

import org.junit.jupiter.api.Test;

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

public class ExceptionThrowerTest {

@Test

public void testThrowException() {

ExceptionThrower thrower = new ExceptionThrower();

Exception exception = assertThrows(IllegalArgumentException.class, () -> {

thrower.throwException();

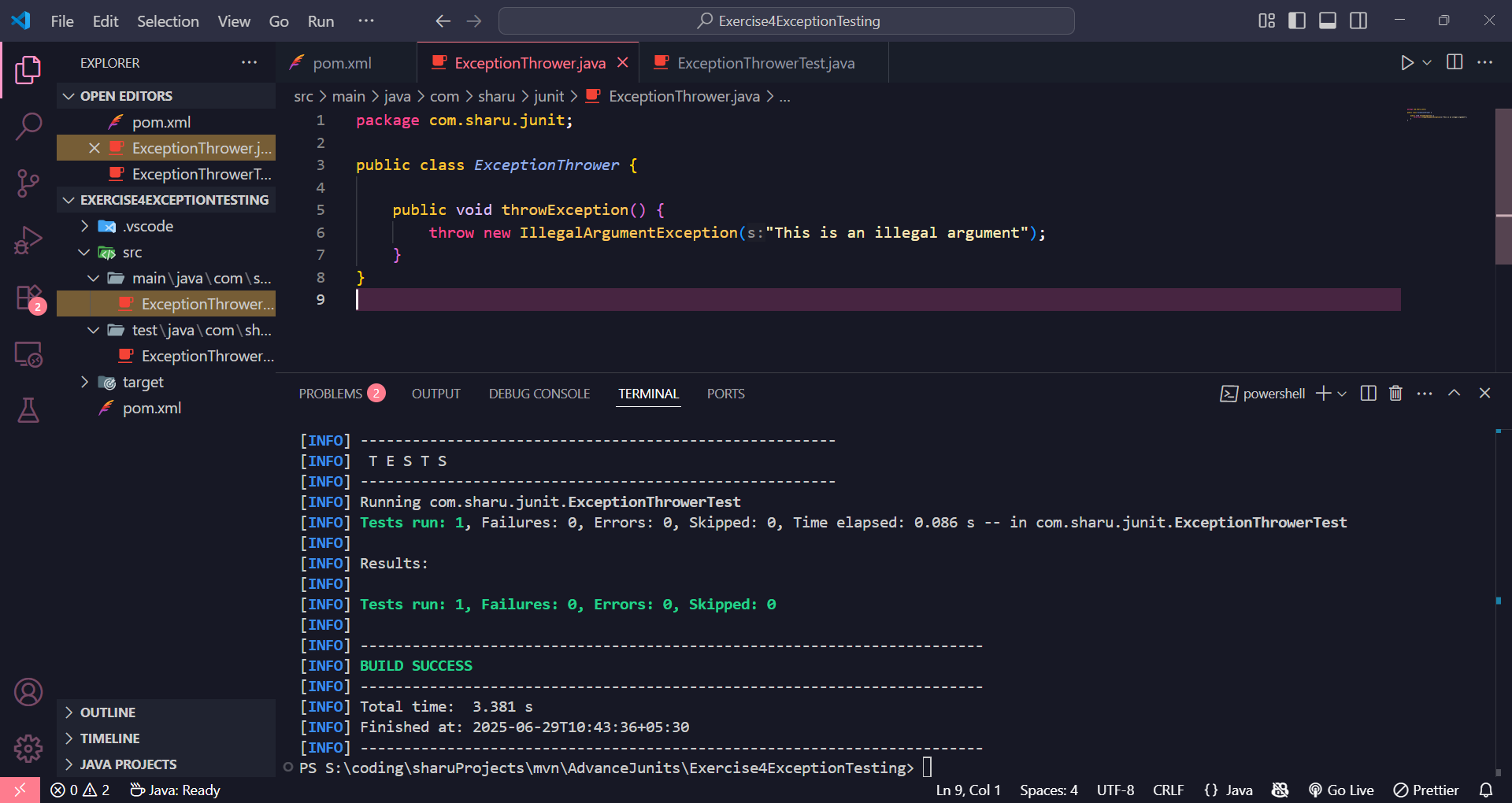
});

assertEquals("This is an illegal argument", exception.getMessage());

}

}

**Output**



EXERCISE 5 : JUnit Timeout and Performance Testing with assertTimeout

**PerformanceTester.java**

package com.sharu.junit;

public class PerformanceTester {

public void performTask() {

try {

Thread.sleep(300);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

**PerformanceTesterTest.java**

package com.sharu.junit;

import org.junit.jupiter.api.Test;

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

import java.time.Duration;

public class PerformanceTesterTest {

@Test

public void testPerformTaskCompletesInTime() {

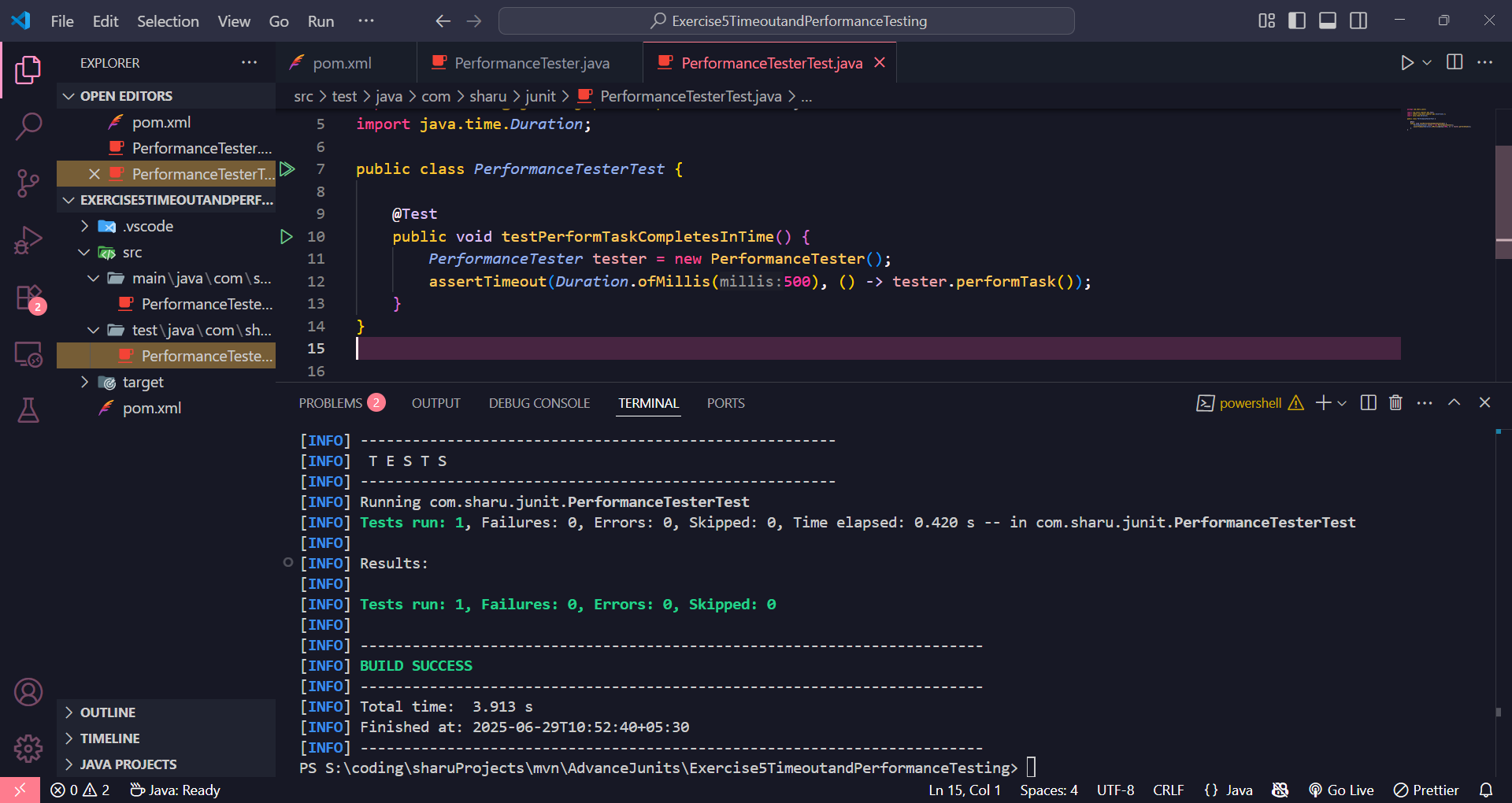
PerformanceTester tester = new PerformanceTester();

assertTimeout(Duration.ofMillis(500), () -> tester.performTask());

}

}

**Output**



MOKITO EXERCISE

EXERCISE 1 : JUnit Mocking and Stubbing with Mockito(MANDATORY)

**ExternalApi.java**

package com.sharu.mocktest;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.sharu.mocktest;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

package com.sharu.mocktest;

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 testExternalApi() {

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

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

MyService service = new MyService(mockApi);

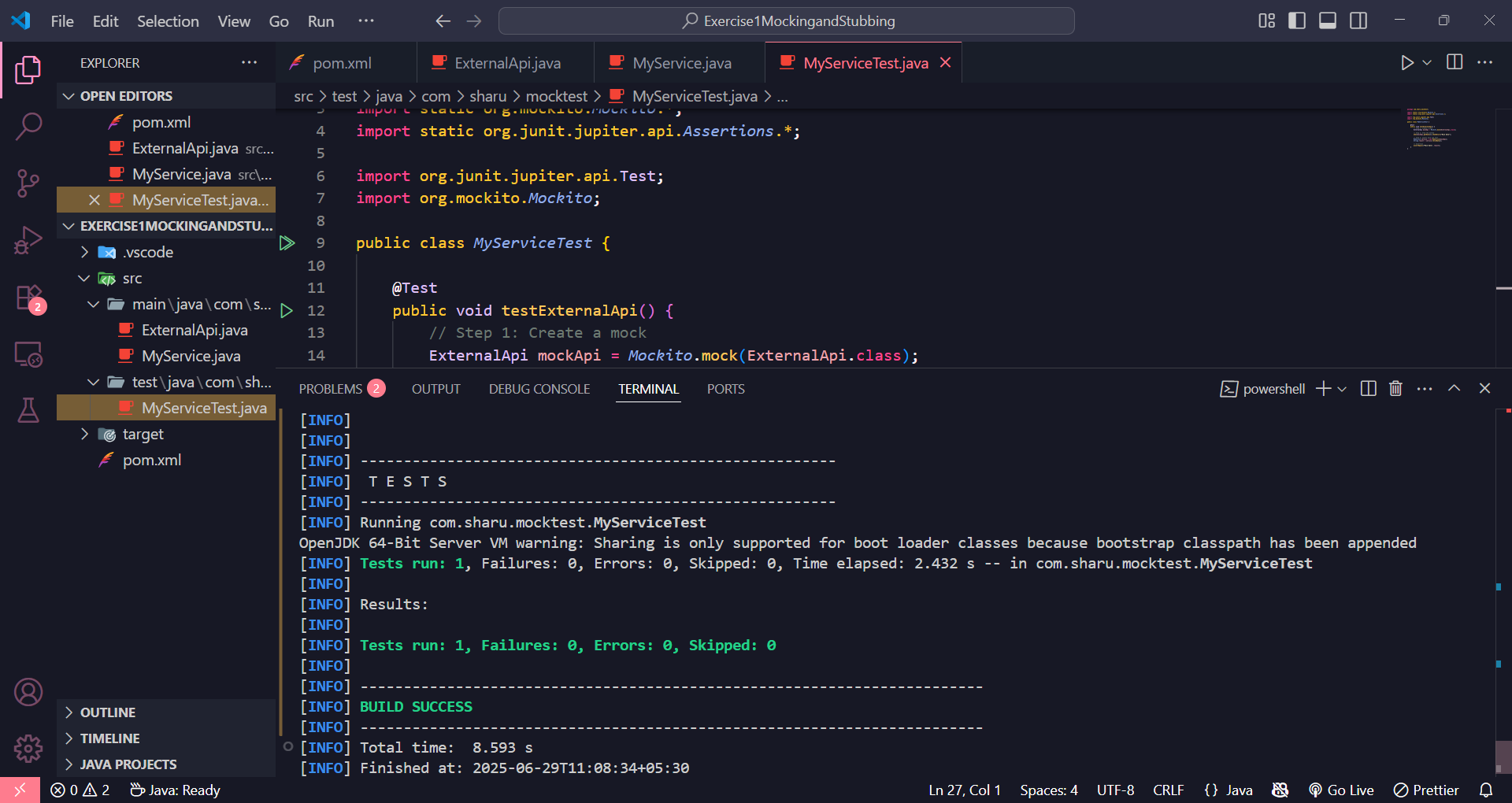
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**Output**



EXERCISE 2 : JUnit Interaction Verification with Mockito(MANDATORY)

**ExternalApi.java**

package com.sharu.mocktest;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.sharu.mocktest;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

package com.sharu.mocktest;

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);

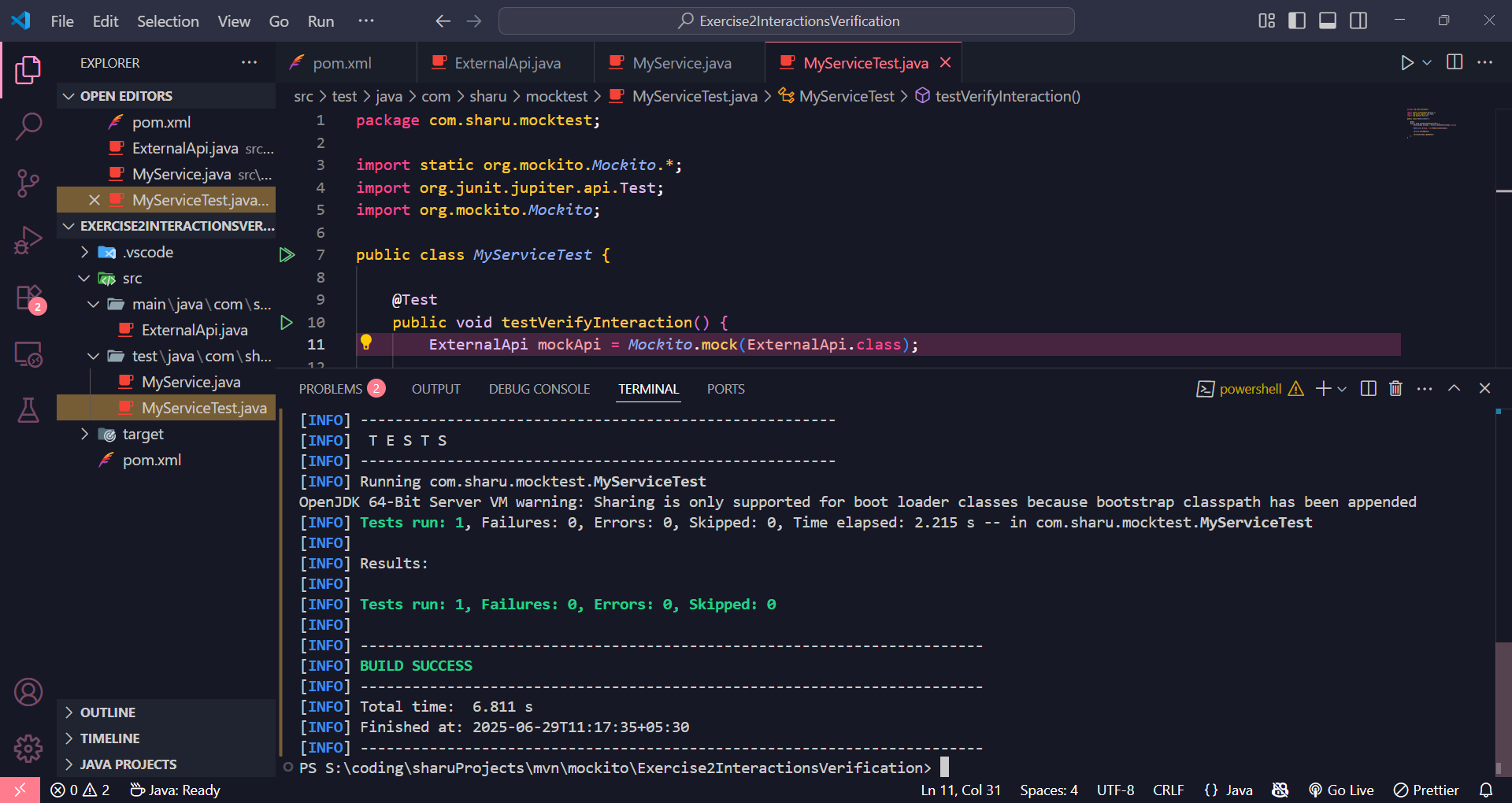
service.fetchData();

verify(mockApi).getData();

}

}

**Output**



EXERCISE 3 : JUnit Argument Matching with Mockito

**ExternalApi.java**

package com.sharu.mocktest;

public interface ExternalApi {

void sendMessage(String userId, String message);

}

**MyService.java**

package com.sharu.mocktest;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void notifyUser(String userId) {

String message = "Hello " + userId;

api.sendMessage(userId, message);

}

}

**MyServiceTest.java**

package com.sharu.mocktest;

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testArgumentMatching() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

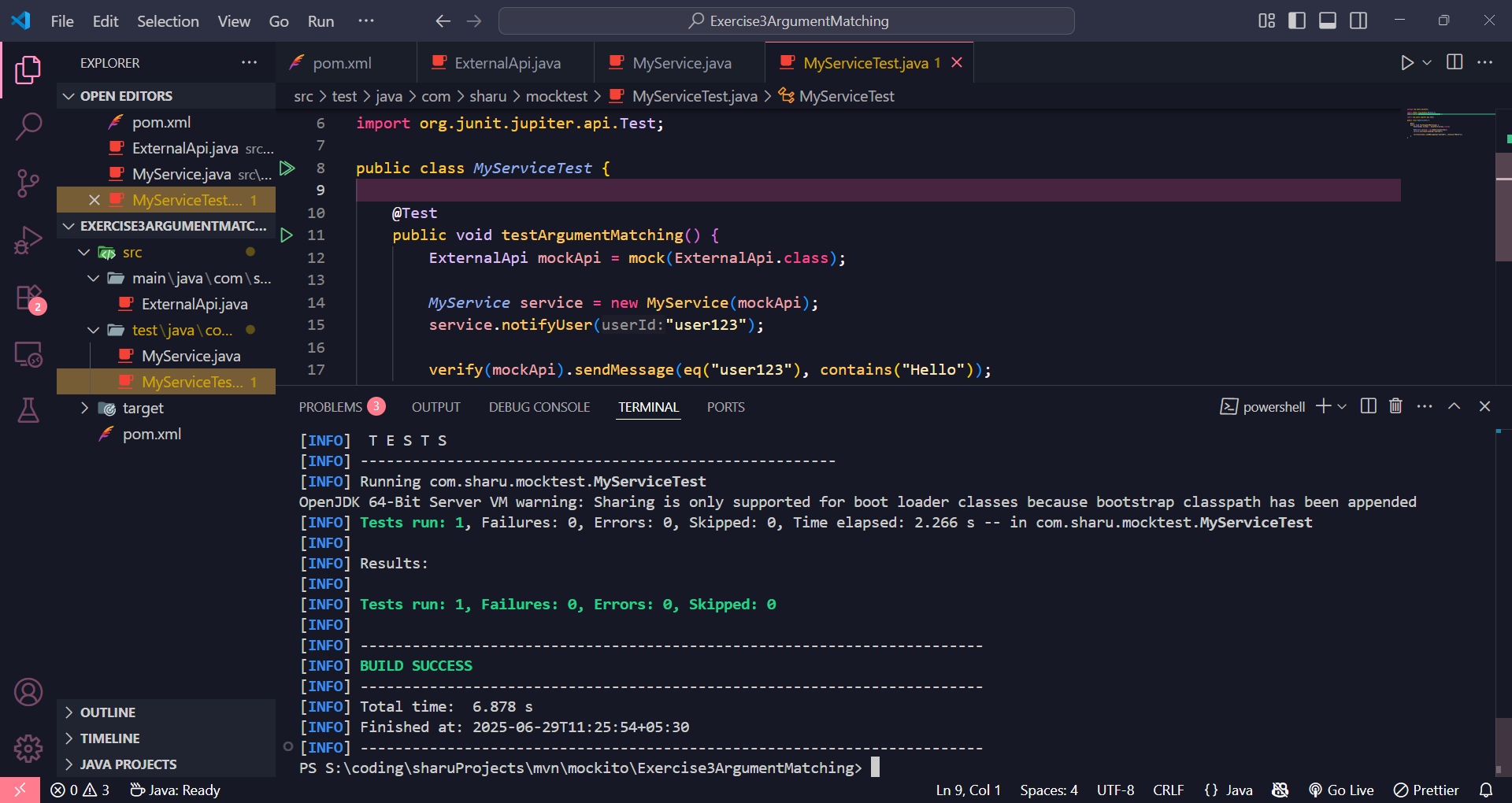
service.notifyUser("user123");

verify(mockApi).sendMessage(eq("user123"), contains("Hello"));

}

}

**Output**



EXERCISE 4 : JUnit Testing and Verification of Void Methods with Mockito

**ExternalApi.java**

package com.sharu.mocktest;

public interface ExternalApi {

void logEvent(String event);

}

**MyService.java**

package com.sharu.mocktest;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void performAction() {

api.logEvent("Action performed");

}

}

**MyServiceTest.java**

package com.sharu.mocktest;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testVoidMethodInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

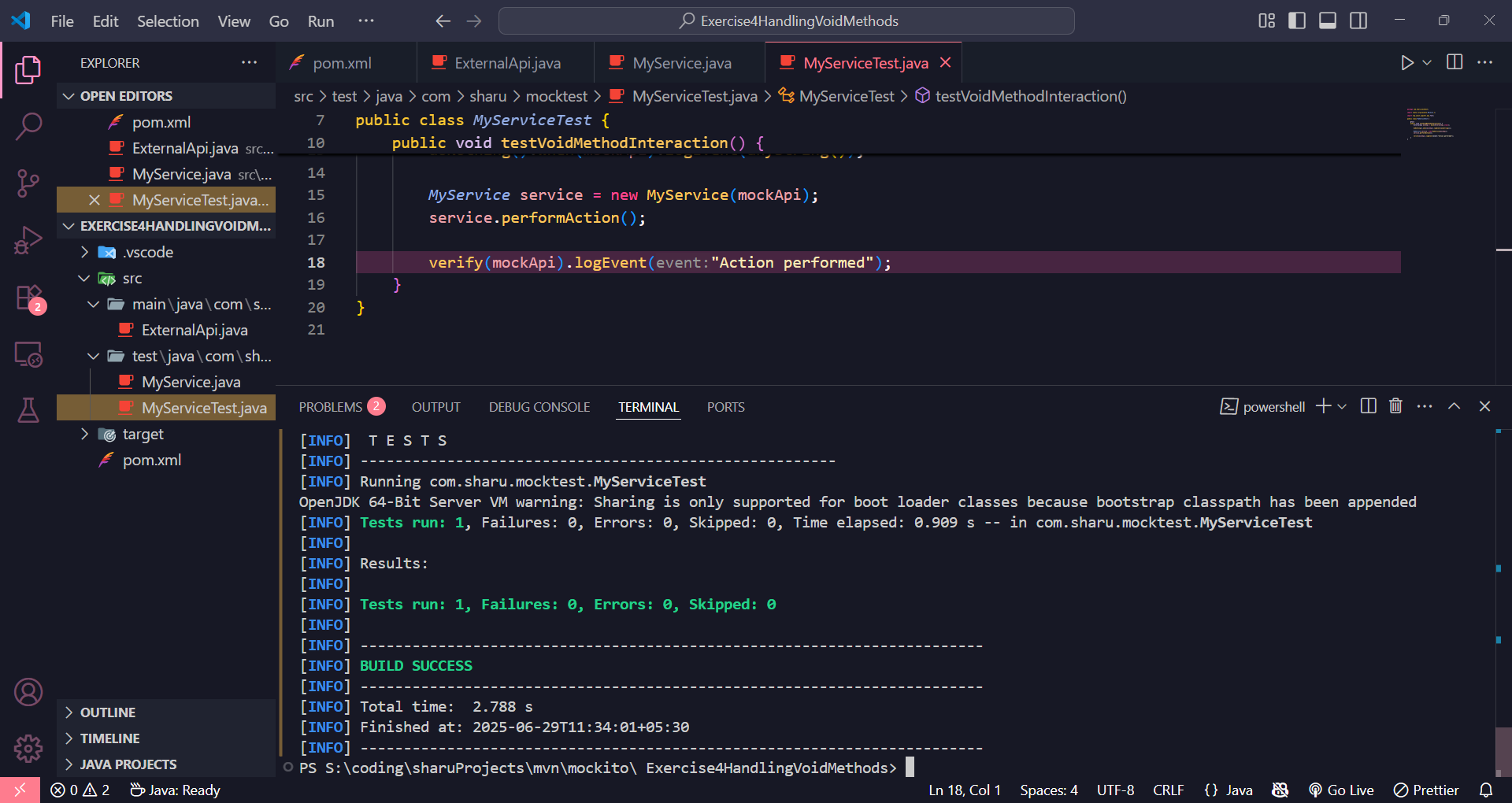
service.performAction();

verify(mockApi).logEvent("Action performed");

}

}

**Output**



EXERCISE 5 : JUnit Mocking and Stubbing with Multiple Returns using Mockito

**ExternalApi.java**

package com.sharu.mocktest;

public interface ExternalApi {

String getStatus();

}

**MyService.java**

package com.sharu.mocktest;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String checkStatusMultipleTimes() {

String first = api.getStatus();

String second = api.getStatus();

return first + " -> " + second;

}

}

**MyServiceTest.java**

package com.sharu.mocktest;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testMultipleReturnValues() {

ExternalApi mockApi = mock(ExternalApi.class);

when(mockApi.getStatus())

.thenReturn("Pending")

.thenReturn("Completed");

MyService service = new MyService(mockApi);

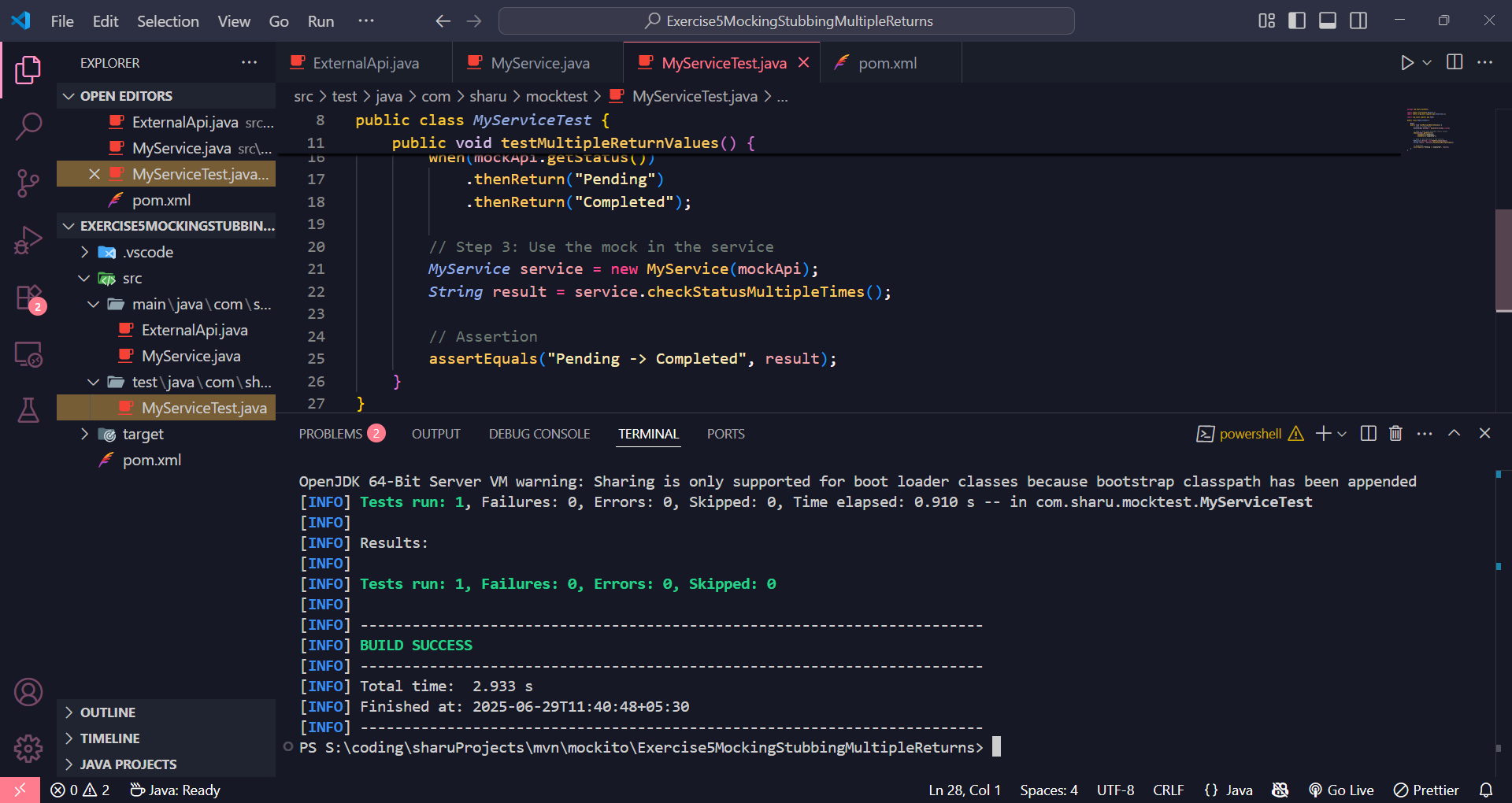
String result = service.checkStatusMultipleTimes();

assertEquals("Pending -> Completed", result);

}

}

**Output**



EXERCISE 6 : JUnit Verifying Method Call Order with Mockito

**ExternalApi.java**

package com.sharu.mocktest;

public interface ExternalApi {

void connect();

void fetch();

void disconnect();

}

**MyService.java**

package com.sharu.mocktest;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void process() {

api.connect();

api.fetch();

api.disconnect();

}

}

**MyServiceTest.java**

package com.sharu.mocktest;

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.process();

InOrder inOrder = inOrder(mockApi);

inOrder.verify(mockApi).connect();

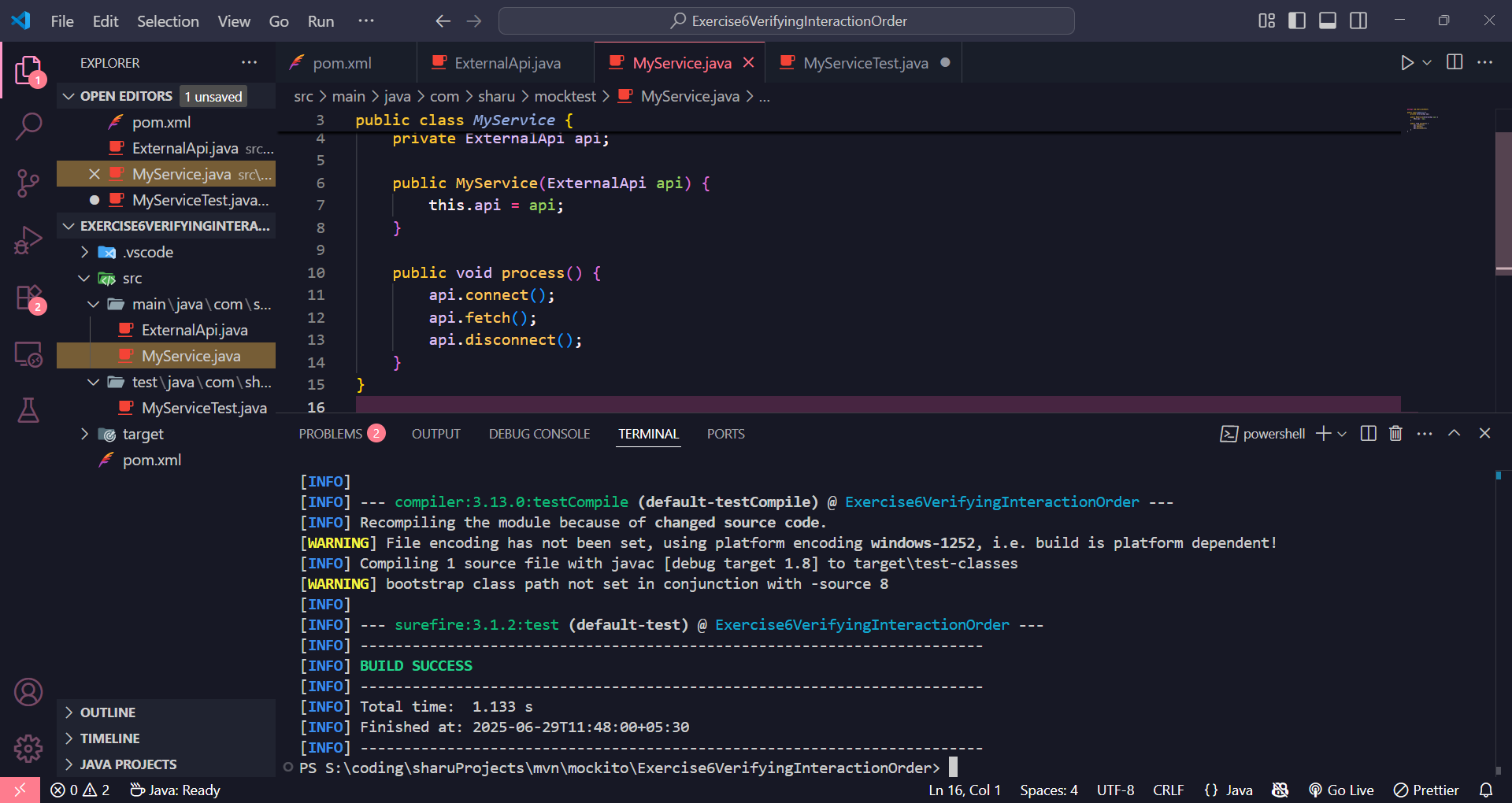
inOrder.verify(mockApi).fetch();

inOrder.verify(mockApi).disconnect();

}

}

**Output**



EXERCISE 7 : JUnit Testing Void Methods That Throw Exceptions with Mockito

**ExternalApi.java**

package com.sharu.mocktest;

public interface ExternalApi {

void deleteResource(String resourceId);

}

**MyService.java**

package com.sharu.mocktest;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void remove(String id) {

api.deleteResource(id);

}

}

**MyServiceTest.java**

package com.sharu.mocktest;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testVoidMethodThrowsException() {

ExternalApi mockApi = mock(ExternalApi.class);

doThrow(new RuntimeException("Resource not found"))

.when(mockApi).deleteResource("invalid-id");

MyService service = new MyService(mockApi);

RuntimeException thrown = assertThrows(

RuntimeException.class,

() -> service.remove("invalid-id")

);

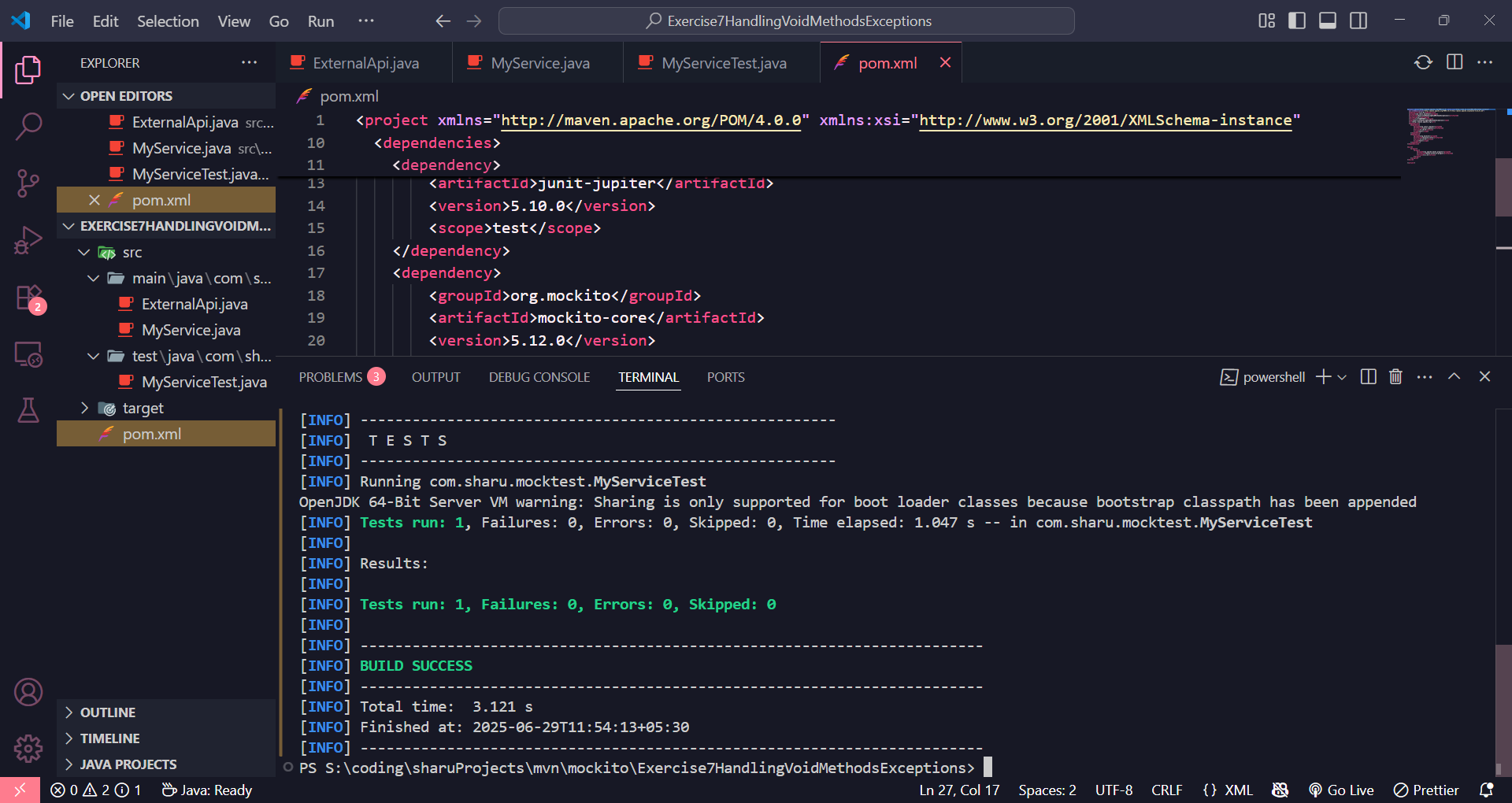
assertEquals("Resource not found", thrown.getMessage());

verify(mockApi).deleteResource("invalid-id");

}

}

**Output**



MOCKITO ADVANCED EXERCISE

EXERCISE 1 : JUnit Testing with Mocked Repositories using Mockito

**Repository.java**

package com.sharu.mocktest;

public interface Repository {

String getData();

}

**Service.java**

package com.sharu.mocktest;

public class Service {

private Repository repository;

public Service(Repository repository) {

this.repository = repository;

}

public String processData() {

return "Processed " + repository.getData();

}

}

**ServiceTest.java**

package com.sharu.mocktest;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

public class ServiceTest {

@Test

public void testServiceWithMockRepository() {

Repository mockRepository = mock(Repository.class);

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

Service service = new Service(mockRepository);

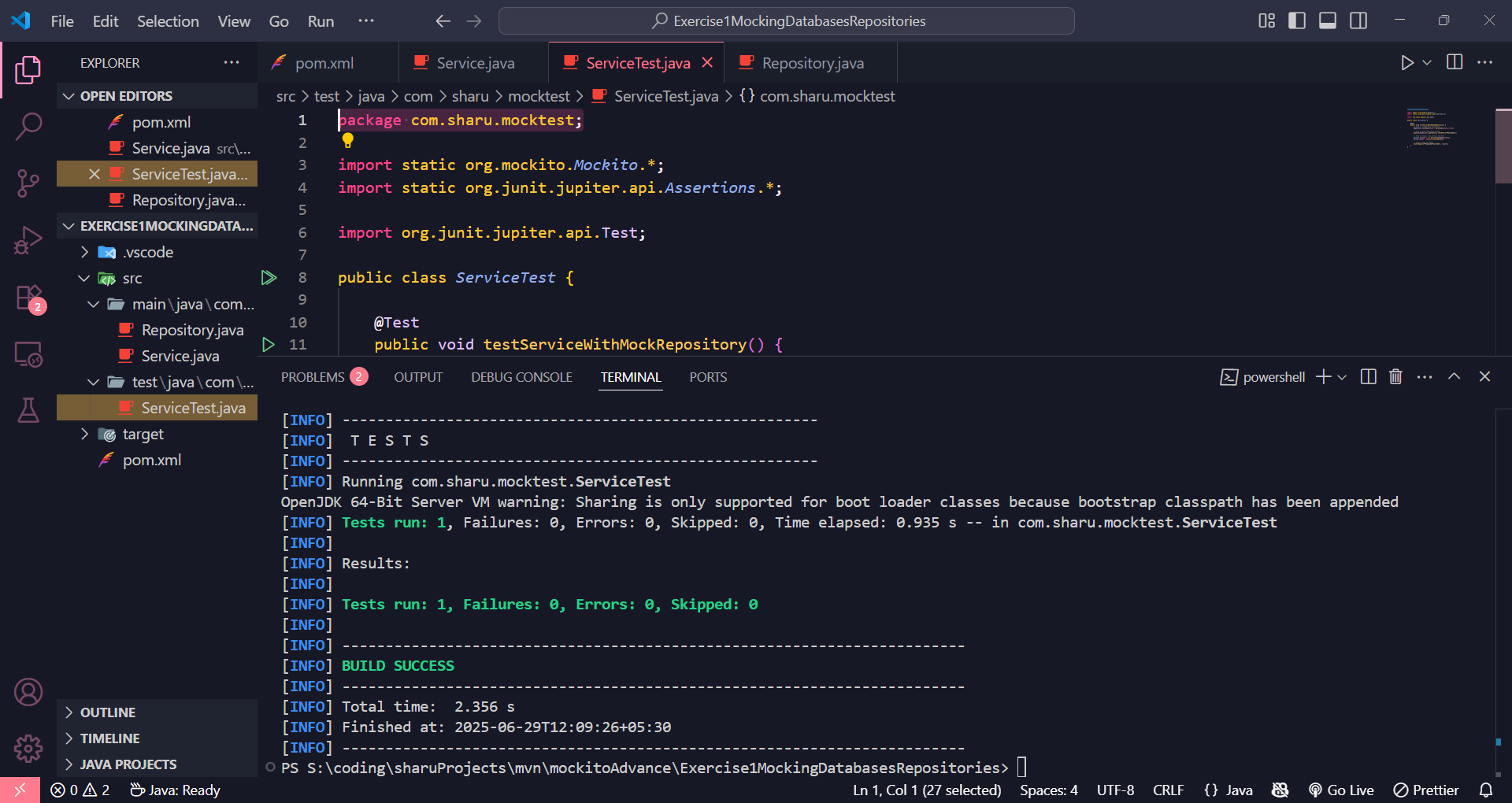
String result = service.processData();

assertEquals("Processed Mock Data", result);

}

}

**Output**



EXERCISE 2 : JUnit Testing with Mocked REST Clients using Mockito

**RestClient.java**

package com.sharu.mockrest;

public interface RestClient {

String getResponse();

}

**ApiService.java**

package com.sharu.mockrest;

public class ApiService {

private RestClient restClient;

public ApiService(RestClient restClient) {

this.restClient = restClient;

}

public String fetchData() {

return "Fetched " + restClient.getResponse();

}

}

**ApiServiceTest.java**

package com.sharu.mockrest;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

public class ApiServiceTest {

@Test

public void testServiceWithMockRestClient() {

RestClient mockRestClient = mock(RestClient.class);

when(mockRestClient.getResponse()).thenReturn("Mock Response");

ApiService apiService = new ApiService(mockRestClient);

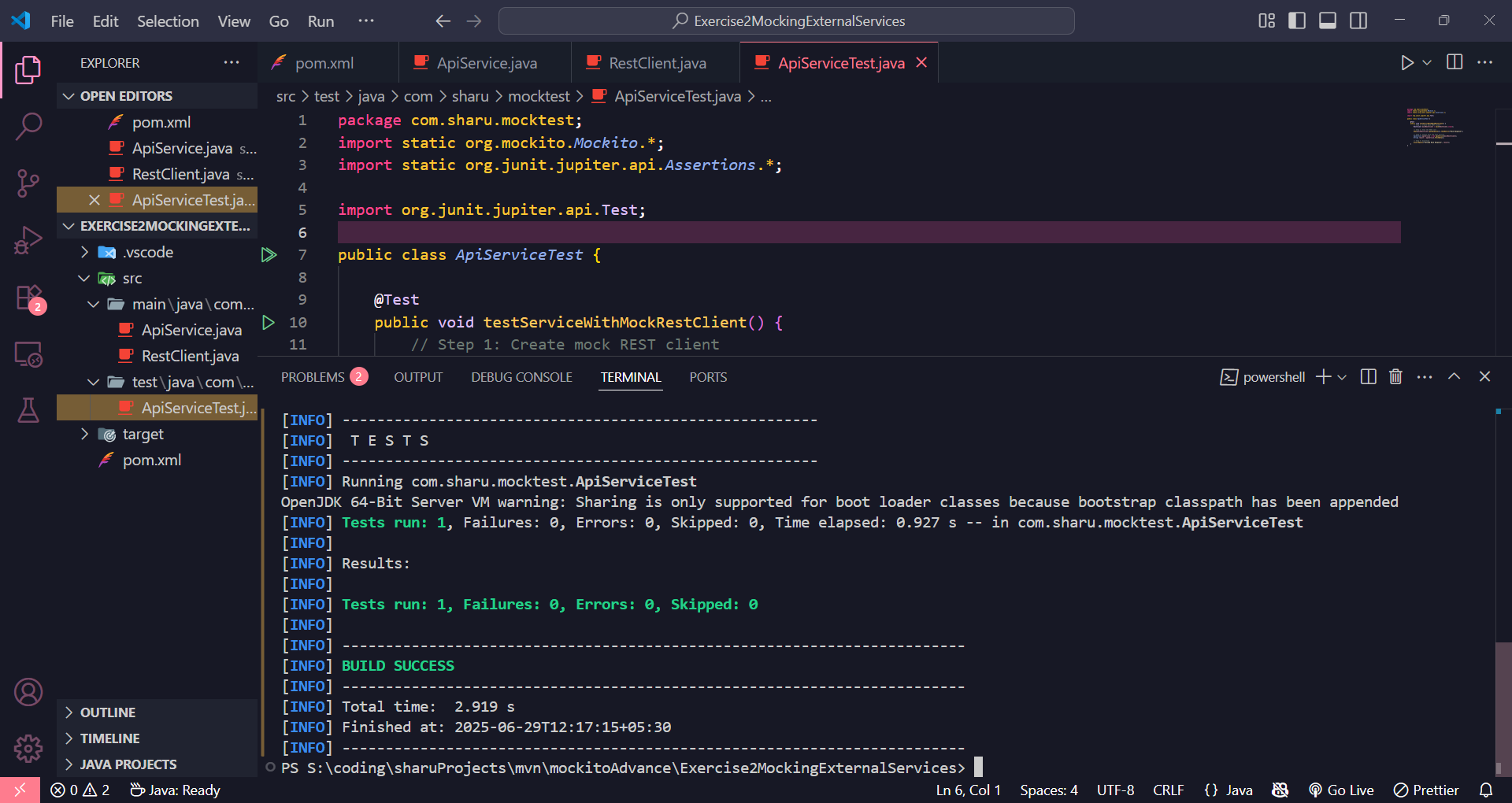
String result = apiService.fetchData();

assertEquals("Fetched Mock Response", result);

}

}

**Output**



EXERCISE 3 : JUnit Testing File I/O Services with Mocked Readers and Writers Using Mockito

**FileReader.java**

package com.sharu.mockfile;

public interface FileReader {

String read();

}

**FileWriter.java**

package com.sharu.mockfile;

public interface FileWriter {

void write(String content);

}

**FileService.java**

package com.sharu.mockfile;

public class FileService {

private FileReader fileReader;

private FileWriter fileWriter;

public FileService(FileReader fileReader, FileWriter fileWriter) {

this.fileReader = fileReader;

this.fileWriter = fileWriter;

}

public String processFile() {

String content = fileReader.read();

fileWriter.write(content);

return "Processed " + content;

}

}

**FileServiceTest.java**

package com.sharu.mockfile;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

public class FileServiceTest {

@Test

public void testServiceWithMockFileIO() {

FileReader mockFileReader = mock(FileReader.class);

FileWriter mockFileWriter = mock(FileWriter.class);

when(mockFileReader.read()).thenReturn("Mock File Content");

FileService fileService = new FileService(mockFileReader, mockFileWriter);

String result = fileService.processFile();

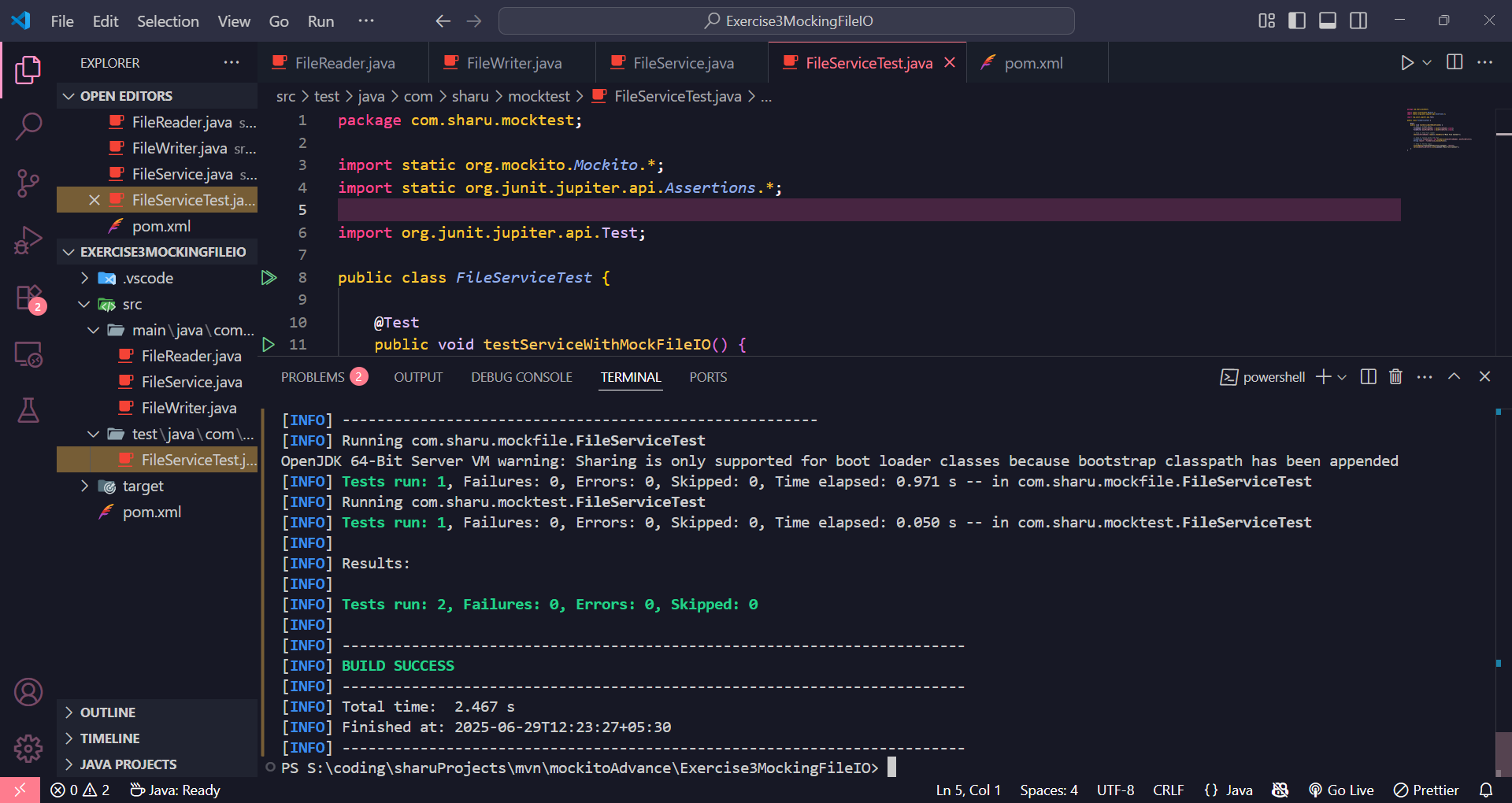
assertEquals("Processed Mock File Content", result);

verify(mockFileWriter).write("Mock File Content");

}

}

**Output**



EXERCISE 4 : JUnit Testing Network Services with Mocked Network Clients Using Mockito

**NetworkClient.java**

package com.sharu.mocknetwork;

public interface NetworkClient {

String connect();

}

**NetworkService.java**

package com.sharu.mocknetwork;

public class NetworkService {

private NetworkClient networkClient;

public NetworkService(NetworkClient networkClient) {

this.networkClient = networkClient;

}

public String connectToServer() {

return "Connected to " + networkClient.connect();

}

}

**NetworkServiceTest.java**

package com.sharu.mocknetwork;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

public class NetworkServiceTest {

@Test

public void testServiceWithMockNetworkClient() {

NetworkClient mockNetworkClient = mock(NetworkClient.class);

when(mockNetworkClient.connect()).thenReturn("Mock Connection");

NetworkService networkService = new NetworkService(mockNetworkClient);

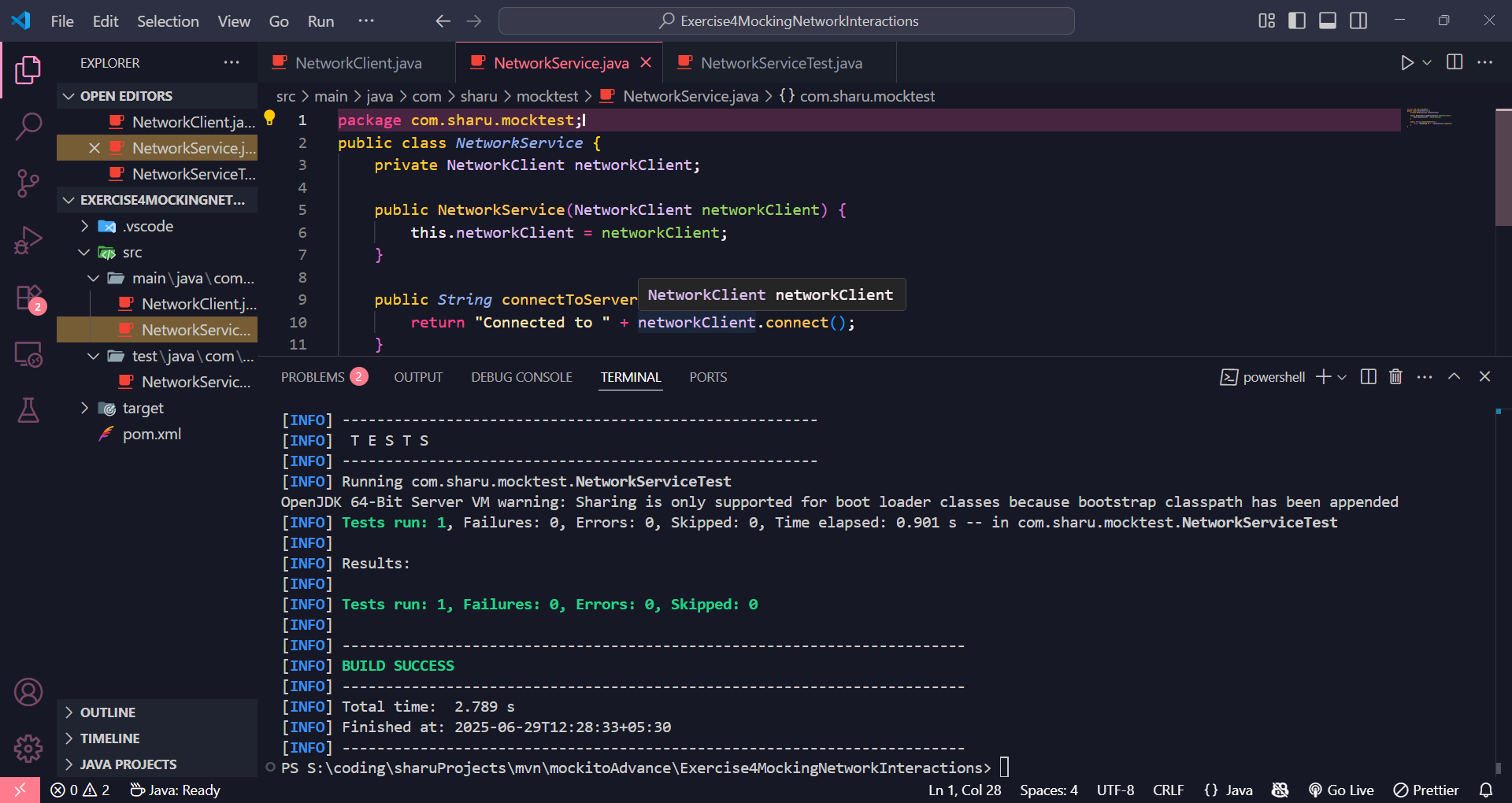
String result = networkService.connectToServer();

assertEquals("Connected to Mock Connection", result);

}

}

**Output**



EXERCISE 5 : JUnit Testing Multiple Return Values with Mockito

**Repository.java**

package com.sharu.multimock;

public interface Repository {

String getData();

}

**Service.java**

package com.sharu.multimock;

public class Service {

private Repository repository;

public Service(Repository repository) {

this.repository = repository;

}

public String processData() {

return "Processed " + repository.getData();

}

}

**MultiReturnServiceTest.java**

package com.sharu.multimock;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

public class MultiReturnServiceTest {

@Test

public void testServiceWithMultipleReturnValues() {

Repository mockRepository = mock(Repository.class);

when(mockRepository.getData())

.thenReturn("First Mock Data")

.thenReturn("Second Mock Data");

Service service = new Service(mockRepository);

String firstResult = service.processData();

String secondResult = service.processData();

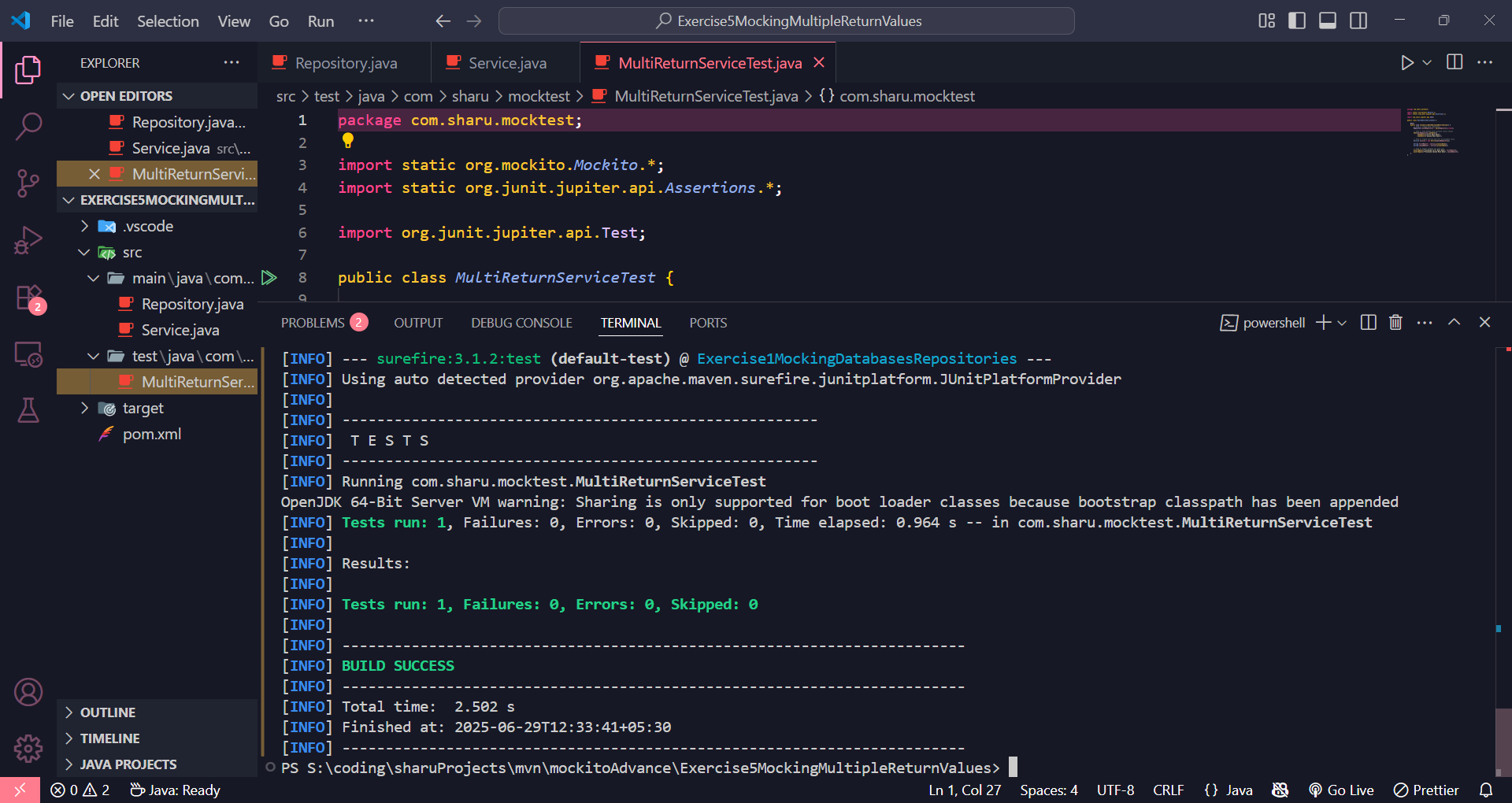
assertEquals("Processed First Mock Data", firstResult);

assertEquals("Processed Second Mock Data", secondResult);

}

}

**Output**



JUNIT SPRING TEST EXERCISE

EXERCISE 1 : JUnit Basic Unit Test for CalculatorService Add Method

**CalculatorService.java**

package com.sharu.junit;

import org.springframework.stereotype.Service;

@Service

public class CalculatorService {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorServiceTest.java**

package com.sharu.junit;

import org.junit.jupiter.api.Test;

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

public class CalculatorServiceTest {

private final CalculatorService calculatorService = new CalculatorService();

@Test

public void testAdd\_TwoPositiveNumbers() {

assertEquals(8, calculatorService.add(5, 3));

}

@Test

public void testAdd\_PositiveAndNegativeNumber() {

assertEquals(3, calculatorService.add(5, -2));

}

@Test

public void testAdd\_TwoNegativeNumbers() {

assertEquals(-10, calculatorService.add(-4, -6));

}

@Test

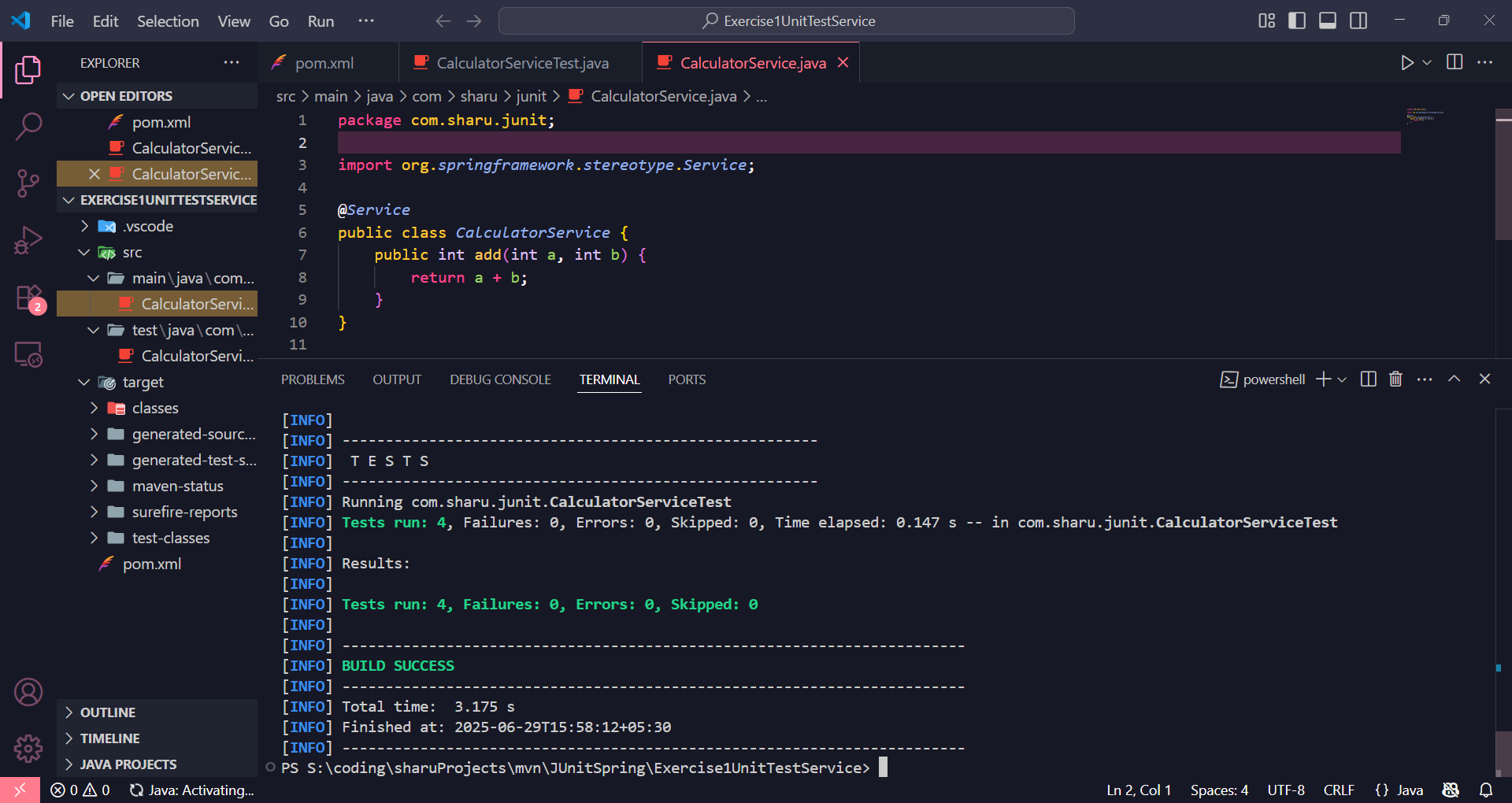
public void testAdd\_WithZero() {

assertEquals(7, calculatorService.add(0, 7));

}

}

**Output**



EXERCISE 2 : Mocking a Repository in a Service Test with JUnit and Mockito

**User.java**

package com.sharu.junit.model;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class User {

@Id

private Long id;

private String name;

public User() {

}

public User(Long id, String name) {

this.id = id;

this.name = name;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

**UserRepository.java**

package com.sharu.junit.repository;

import com.sharu.junit.model.User;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserService.java**

package com.sharu.junit.service;

import com.sharu.junit.model.User;

import com.sharu.junit.repository.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public User getUserById(Long id) {

return userRepository.findById(id).orElse(null);

}

}

**UserServiceTest.java**

package com.sharu.junit.service;

import com.sharu.junit.model.User;

import com.sharu.junit.repository.UserRepository;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.junit.jupiter.MockitoExtension;

import java.util.Optional;

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

import static org.mockito.Mockito.\*;

@ExtendWith(MockitoExtension.class)

public class UserServiceTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

@Test

public void testGetUserById\_UserExists() {

User user = new User(1L, "Sharu");

when(userRepository.findById(1L)).thenReturn(Optional.of(user));

User result = userService.getUserById(1L);

assertNotNull(result);

assertEquals("Sharu", result.getName());

verify(userRepository, times(1)).findById(1L);

}

@Test

public void testGetUserById\_UserDoesNotExist() {

when(userRepository.findById(2L)).thenReturn(Optional.empty());

User result = userService.getUserById(2L);

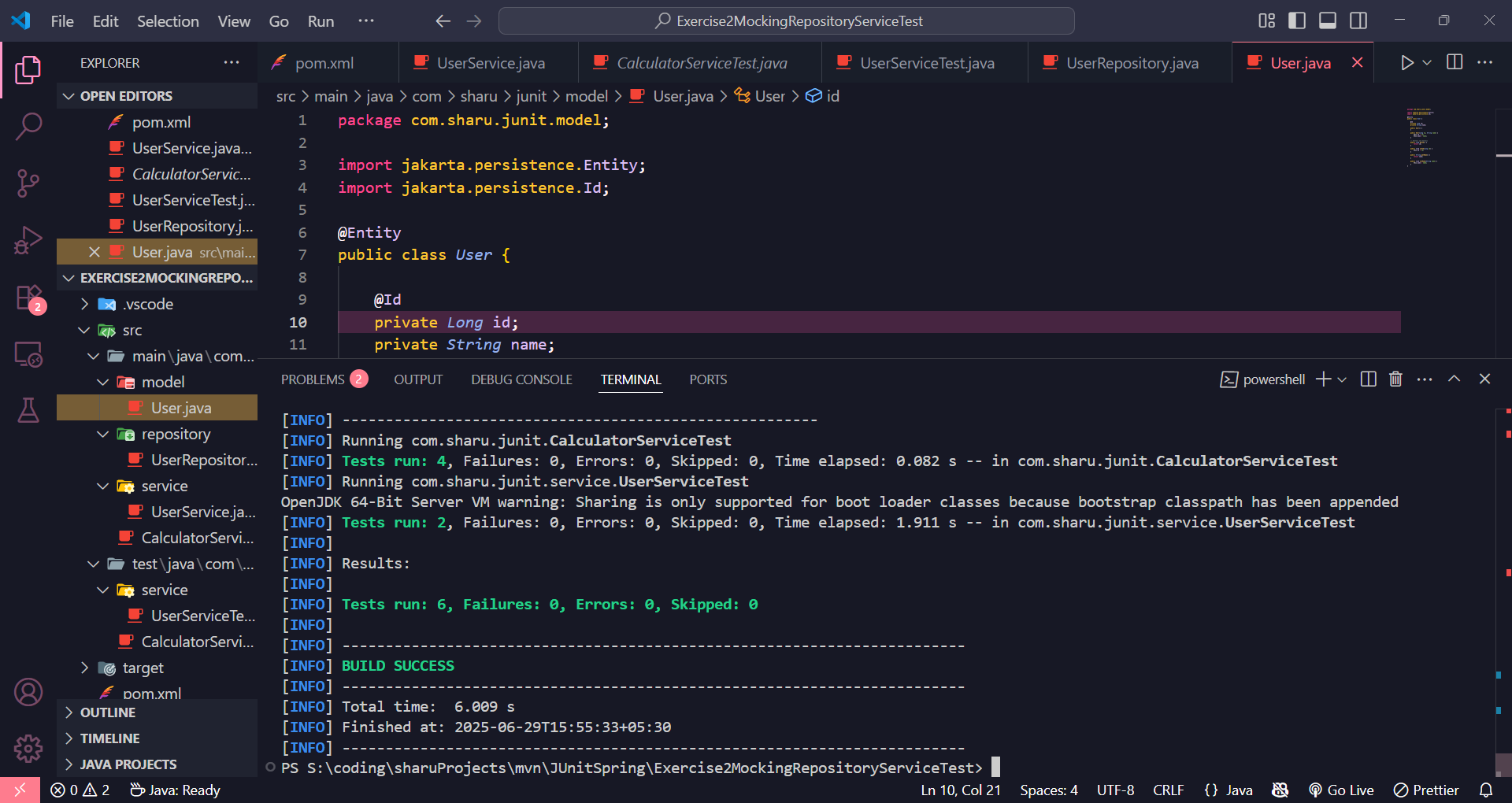
assertNull(result);

verify(userRepository, times(1)).findById(2L);

}

}

**Output**



EXERCISE 3 : Mocking a Repository in a Service Test with JUnit and Mockito

**User.java**

package com.sharu.junit.model;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class User {

@Id

private Long id;

private String name;

public User() {

}

public User(Long id, String name) {

this.id = id;

this.name = name;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

**UserRepository.java**

package com.sharu.junit.repository;

import com.sharu.junit.model.User;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserService.java**

package com.sharu.junit.service;

import com.sharu.junit.model.User;

import com.sharu.junit.repository.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public User getUserById(Long id) {

return userRepository.findById(id).orElse(null);

}

}

**UserServiceTest.java**

package com.sharu.junit.service;

import com.sharu.junit.model.User;

import com.sharu.junit.repository.UserRepository;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.junit.jupiter.MockitoExtension;

import java.util.Optional;

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

import static org.mockito.Mockito.\*;

@ExtendWith(MockitoExtension.class)

public class UserServiceTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

@Test

public void testGetUserById\_UserExists() {

User user = new User(1L, "Sharu");

when(userRepository.findById(1L)).thenReturn(Optional.of(user));

User result = userService.getUserById(1L);

assertNotNull(result);

assertEquals("Sharu", result.getName());

verify(userRepository, times(1)).findById(1L);

}

@Test

public void testGetUserById\_UserDoesNotExist() {

when(userRepository.findById(2L)).thenReturn(Optional.empty());

User result = userService.getUserById(2L);

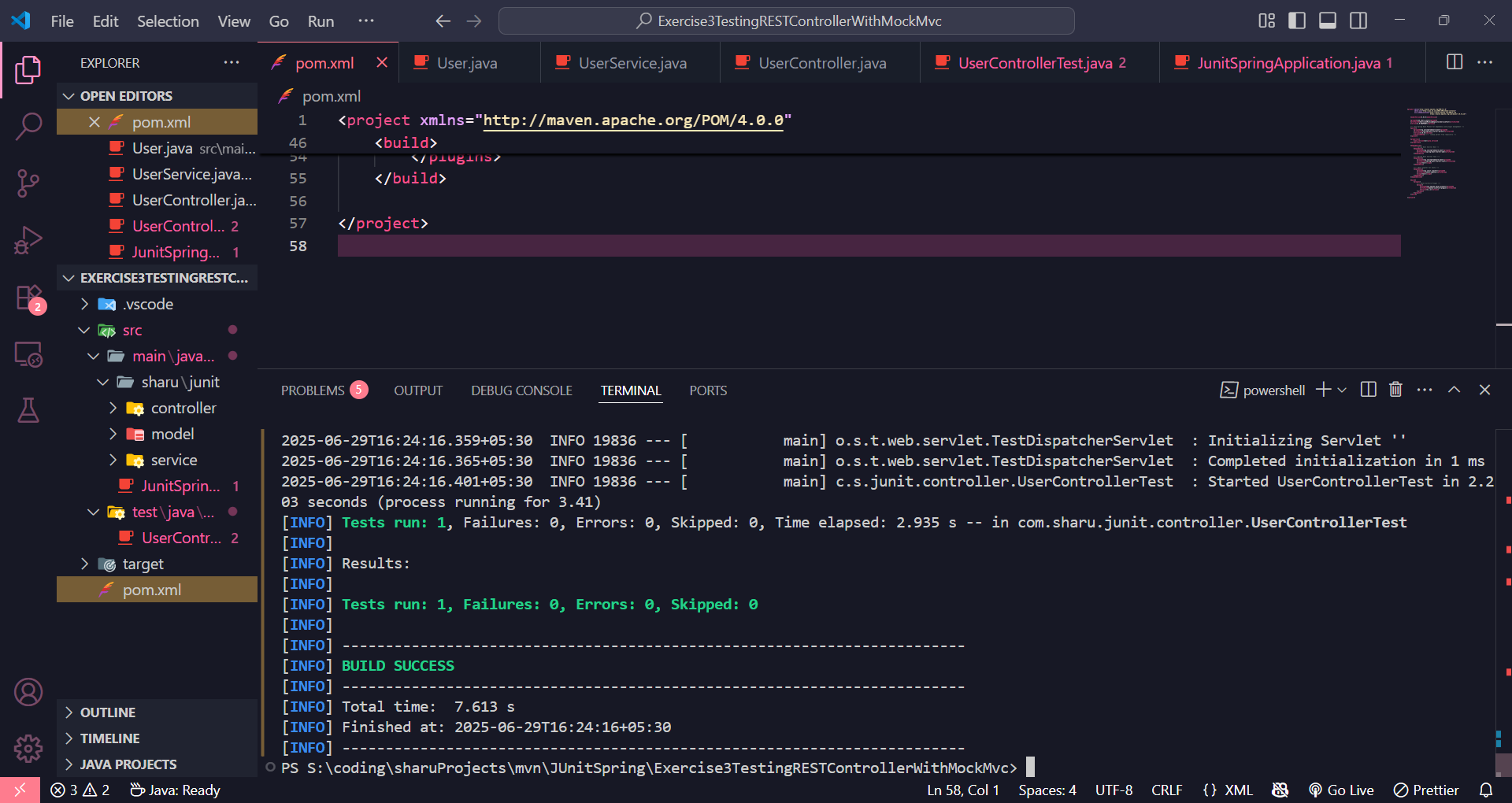
assertNull(result);

verify(userRepository, times(1)).findById(2L);

}

}

**Output**



EXERCISE 4 : Integration Testing Full Flow from Controller to Database with Spring Boot

**JunitSpringApplication.java**

package com.sharu.junit;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class JunitSpringApplication {

public static void main(String[] args) {

SpringApplication.run(JunitSpringApplication.class, args);

}

}

**User.java**

package com.sharu.junit.model;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class User {

@Id

private Long id;

private String name;

public User() {}

public User(Long id, String name) {

this.id = id;

this.name = name;

}

}

**UserRepository.java**

package com.sharu.junit.repository;

import com.sharu.junit.model.User;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserService.java**

package com.sharu.junit.service;

import com.sharu.junit.model.User;

import com.sharu.junit.repository.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.Optional;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public Optional<User> getUserById(Long id) {

return userRepository.findById(id);

}

}

**UserController.java**

package com.sharu.junit.controller;

import com.sharu.junit.model.User;

import com.sharu.junit.service.UserService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/users")

public class UserController {

@Autowired

private UserService userService;

@GetMapping("/{id}")

public ResponseEntity<User> getUser(@PathVariable Long id) {

return userService.getUserById(id)

.map(ResponseEntity::ok)

.orElse(ResponseEntity.notFound().build());

}

}

**UserIntegrationTest.java**

package com.sharu.junit;

import com.sharu.junit.model.User;

import com.sharu.junit.repository.UserRepository;

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.http.MediaType;

import org.springframework.test.web.servlet.MockMvc;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@SpringBootTest

@AutoConfigureMockMvc

public class UserIntegrationTest {

@Autowired

private MockMvc mockMvc;

@Autowired

private UserRepository userRepository;

@Test

public void testGetUserIntegration() throws Exception {

User user = new User(1L, "Sharu");

userRepository.save(user);

mockMvc.perform(get("/users/1")

.accept(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Sharu"));

}

}

EXERCISE 5: Testing POST Endpoint in Spring Boot Controller

**User Entity (User.java)**

package com.sharu.junit.model;

import jakarta.persistence.\*;

@Entity

@Table(name = "\"user\"") // Escape reserved keyword 'user'

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

}

**User Controller (UserController.java)**

package com.sharu.junit.controller;

import com.sharu.junit.model.User;

import com.sharu.junit.service.UserService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/users")

public class UserController {

@Autowired

private UserService userService;

@PostMapping

public ResponseEntity<User> createUser(@RequestBody User user) {

User savedUser = userService.saveUser(user);

return ResponseEntity.ok(savedUser);

}

}

**User Service (UserService.java)**

package com.sharu.junit.service;

import com.sharu.junit.model.User;

import com.sharu.junit.repository.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public User saveUser(User user) {

return userRepository.save(user);

}

}

**User Repository (UserRepository.java)**

package com.sharu.junit.repository;

import com.sharu.junit.model.User;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

}

**Integration Test (UserIntegrationTest.java)**

package com.sharu.junit;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.sharu.junit.model.User;

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.boot.test.web.server.LocalServerPort;

import org.springframework.http.MediaType;

import org.springframework.test.web.reactive.server.WebTestClient;

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)

public class UserIntegrationTest {

@LocalServerPort

private int port;

@Autowired

private WebTestClient webTestClient;

@Autowired

private ObjectMapper objectMapper;

@Test

void testCreateUser() throws Exception {

User user = new User();

user.setName("John Doe");

webTestClient.post()

.uri("/users")

.contentType(MediaType.APPLICATION\_JSON)

.bodyValue(objectMapper.writeValueAsString(user))

.exchange()

.expectStatus().isOk()

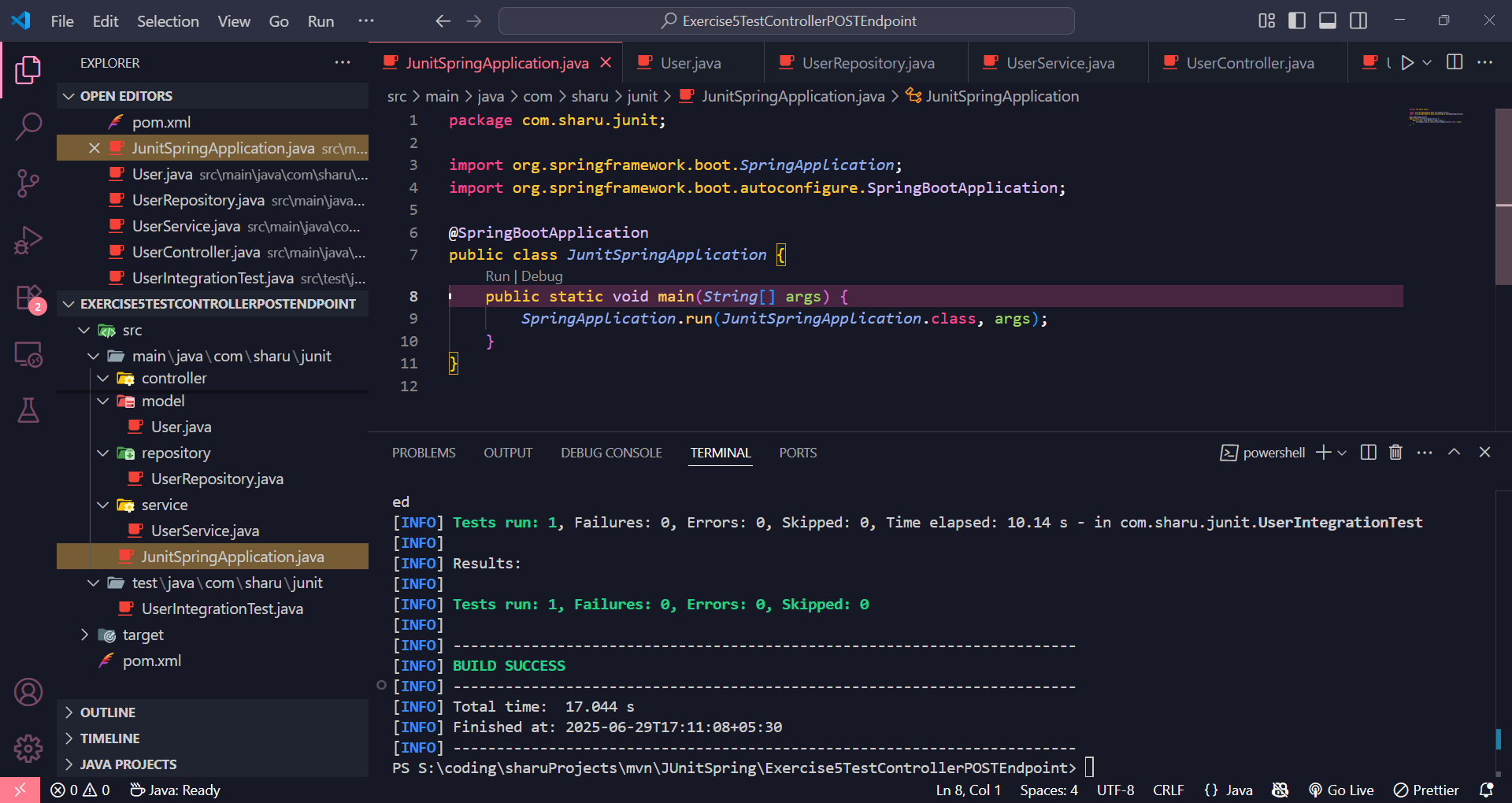
.expectBody()

.jsonPath("$.name").isEqualTo("John Doe");

}

}

**Output**



EXERCISE 6 : JUnit Testing for Service Exception Handling in Spring

**File: User.java**

package com.sharu.service;

public class User {

private Long id;

private String name;

public User(Long id, String name) {

this.id = id;

this.name = name;

}

public Long getId() { return id; }

public String getName() { return name; }

}

**UserRepository.java**

package com.sharu.service;

import java.util.Optional;

public interface UserRepository {

Optional<User> findById(Long id);

}

**UserNotFoundException.java**

package com.sharu.service;

public class UserNotFoundException extends RuntimeException {

public UserNotFoundException(String message) {

super(message);

}

}

**UserService.java**

package com.sharu.service;

import java.util.Optional;

public class UserService {

private final UserRepository userRepository;

public UserService(UserRepository userRepository) {

this.userRepository = userRepository;

}

public User findUserById(Long id) {

return userRepository.findById(id)

.orElseThrow(() -> new UserNotFoundException("User not found with id: " + id));

}

}

**UserServiceTest.java**

package com.sharu.service;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import java.util.Optional;

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

import static org.mockito.Mockito.\*;

public class UserServiceTest {

@Test

void testFindUserById\_UserNotFound\_ThrowsException() {

UserRepository mockRepository = Mockito.mock(UserRepository.class);

Long userId = 1L;

when(mockRepository.findById(userId)).thenReturn(Optional.empty());

UserService userService = new UserService(mockRepository);

UserNotFoundException exception = assertThrows(UserNotFoundException.class, () -> {

userService.findUserById(userId);

});

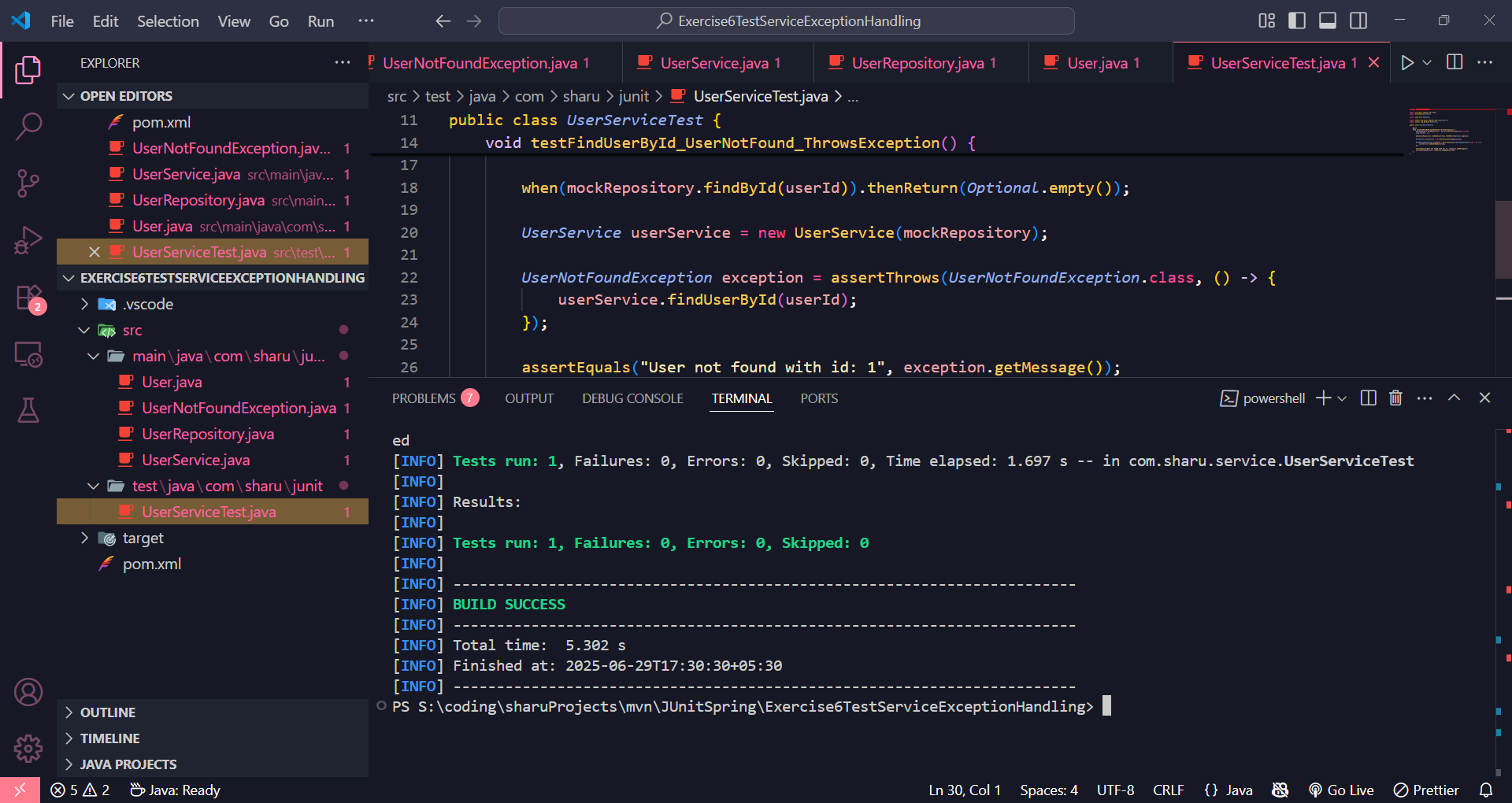
assertEquals("User not found with id: 1", exception.getMessage());

verify(mockRepository, times(1)).findById(userId);

}

}

**Output**



EXERCISE 7 : JUnit Testing for Custom Repository Query in Spring Data JPA

**User.java**

package com.sharu.repository;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

public User() {}

public User(String name) {

this.name = name;

}

public Long getId() { return id; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

}

**UserRepository.java**

package com.sharu.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface UserRepository extends JpaRepository<User, Long> {

List<User> findByName(String name);

}

**TestApplication.java**

package com.sharu.repository;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class TestApplication {

}

**UserRepositoryTest.java**

package com.sharu.repository;

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.context.SpringBootTest;

import java.util.List;

import static org.assertj.core.api.Assertions.assertThat;

@SpringBootTest(classes = TestApplication.class)

public class UserRepositoryTest {

@Autowired

private UserRepository userRepository;

@Test

void testFindByName\_ReturnsMatchingUsers() {

User user1 = new User("Alice");

User user2 = new User("Bob");

User user3 = new User("Alice");

userRepository.save(user1);

userRepository.save(user2);

userRepository.save(user3);

List<User> usersNamedAlice = userRepository.findByName("Alice");

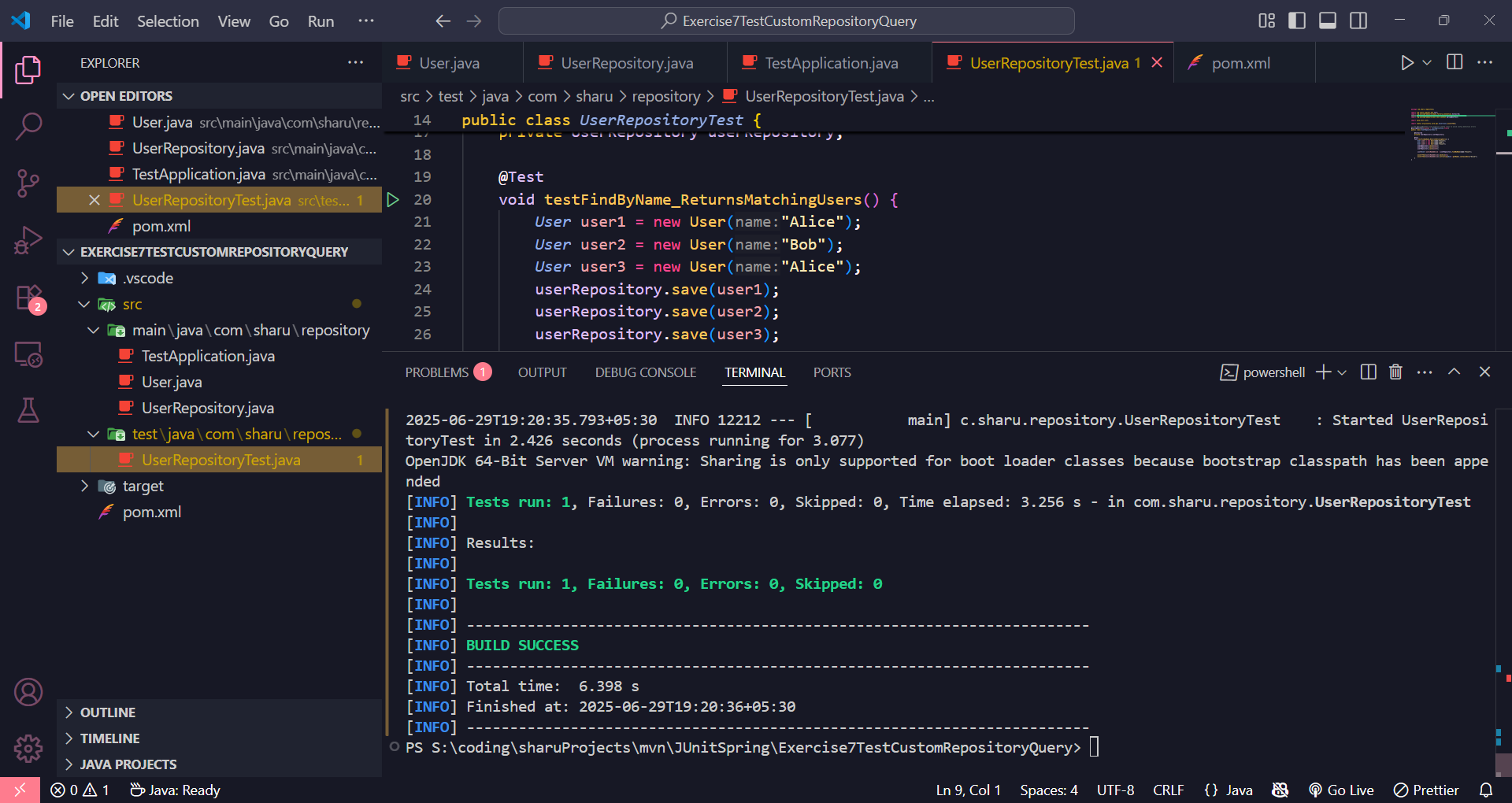
assertThat(usersNamedAlice).hasSize(2);

assertThat(usersNamedAlice).extracting(User::getName).containsOnly("Alice");

}

}

**Output**



EXERCISE 8 : JUnit Testing for Controller Exception Handling in Spring Boot

**TestApplication.java**

package com.sharu.junit;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class TestApplication { }

**User.java**

package com.sharu.junit.model;

public class User {

private Long id;

private String name;

public User(Long id, String name) {

this.id = id;

this.name = name;

}

public Long getId() { return id; }

public String getName() { return name; }

}

**UserController.java**

package com.sharu.junit.controller;

import com.sharu.junit.model.User;

import org.springframework.web.bind.annotation.\*;

import java.util.NoSuchElementException;

@RestController

@RequestMapping("/users")

public class UserController {

@GetMapping("/{id}")

public User getUserById(@PathVariable Long id) {

if (id == 1L) {

return new User(1L, "Alice");

} else {

throw new NoSuchElementException("User not found");

}

}

}

**GlobalExceptionHandler.java**

package com.sharu.junit.exception;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.NoSuchElementException;

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(NoSuchElementException.class)

public ResponseEntity<String> handleNotFound(NoSuchElementException ex) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body("User not found");

}

}

**UserControllerTest.java**

package com.sharu.junit.controller;

import com.sharu.junit.controller.UserController;

import com.sharu.junit.exception.GlobalExceptionHandler;

import com.sharu.junit.TestApplication;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.context.annotation.Import;

import org.springframework.test.context.ContextConfiguration;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.test.web.servlet.MockMvc;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@WebMvcTest(UserController.class)

@Import(GlobalExceptionHandler.class)

@ContextConfiguration(classes = TestApplication.class)

public class UserControllerTest {

@Autowired

private MockMvc mockMvc;

@Test

void whenUserExists\_thenReturn200() throws Exception {

mockMvc.perform(get("/users/1"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Alice"));

}

@Test

void whenUserNotFound\_thenReturn404() throws Exception {

mockMvc.perform(get("/users/2"))

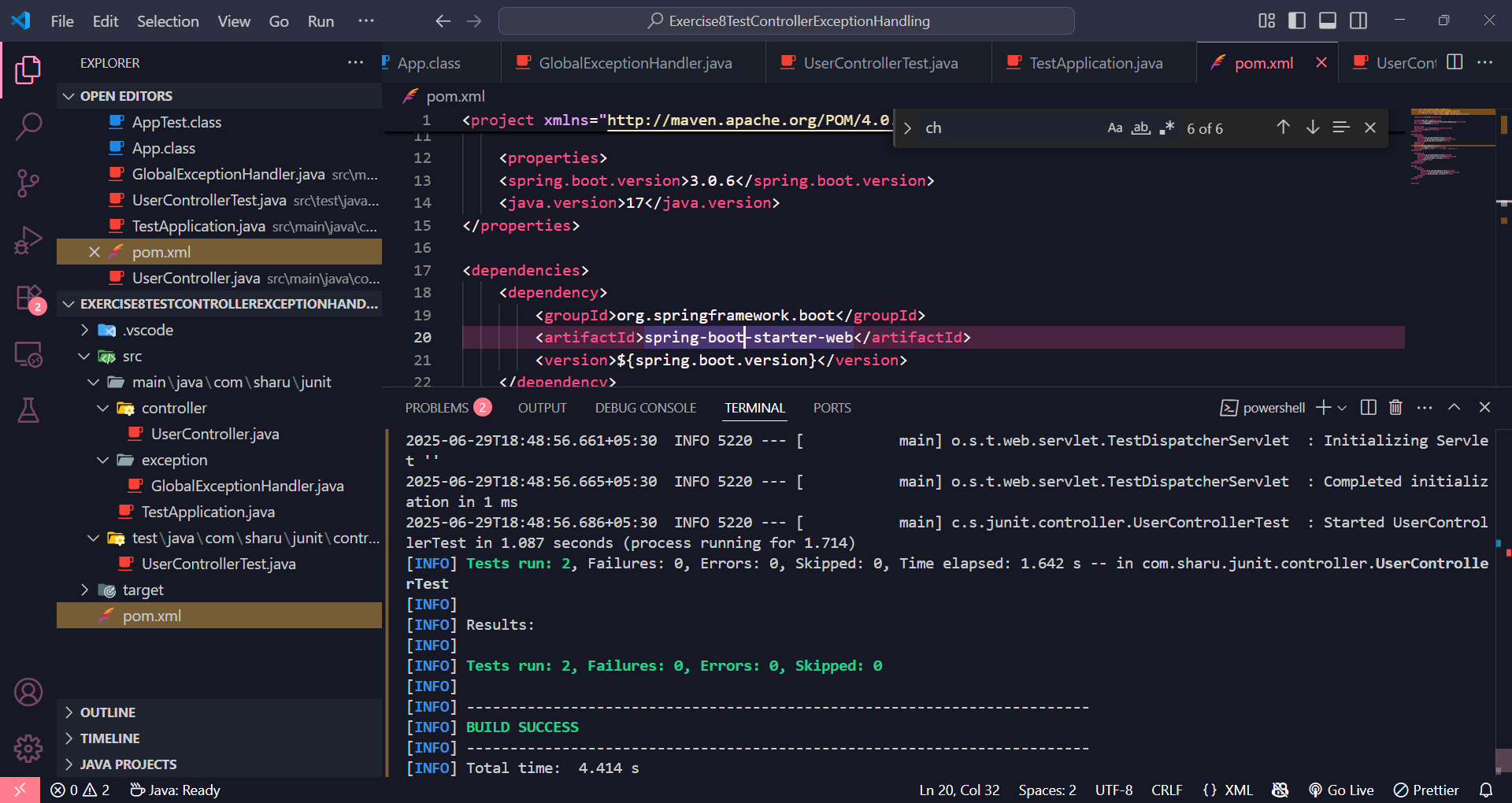
.andExpect(status().isNotFound())

.andExpect(content().string("User not found"));

}

}

**Output**



EXERCISE 9 : JUnit Parameterized Testing for Palindrome Utility

**StringUtils.java**

package com.sharu.util;

public class StringUtils {

public static boolean isPalindrome(String input) {

if (input == null) return false;

String sanitized = input.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();

return new StringBuilder(sanitized).reverse().toString().equals(sanitized);

}

}

**StringUtilsTest.java**

package com.sharu.util;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.CsvSource;

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

public class StringUtilsTest {

@ParameterizedTest

@CsvSource({

"madam, true",

"racecar, true",

"hello, false",

"'Was it a car or a cat I saw', true",

"'No lemon, no melon', true",

"'', true",

"' ', true",

"Java, false"

})

void testIsPalindrome(String input, boolean expected) {

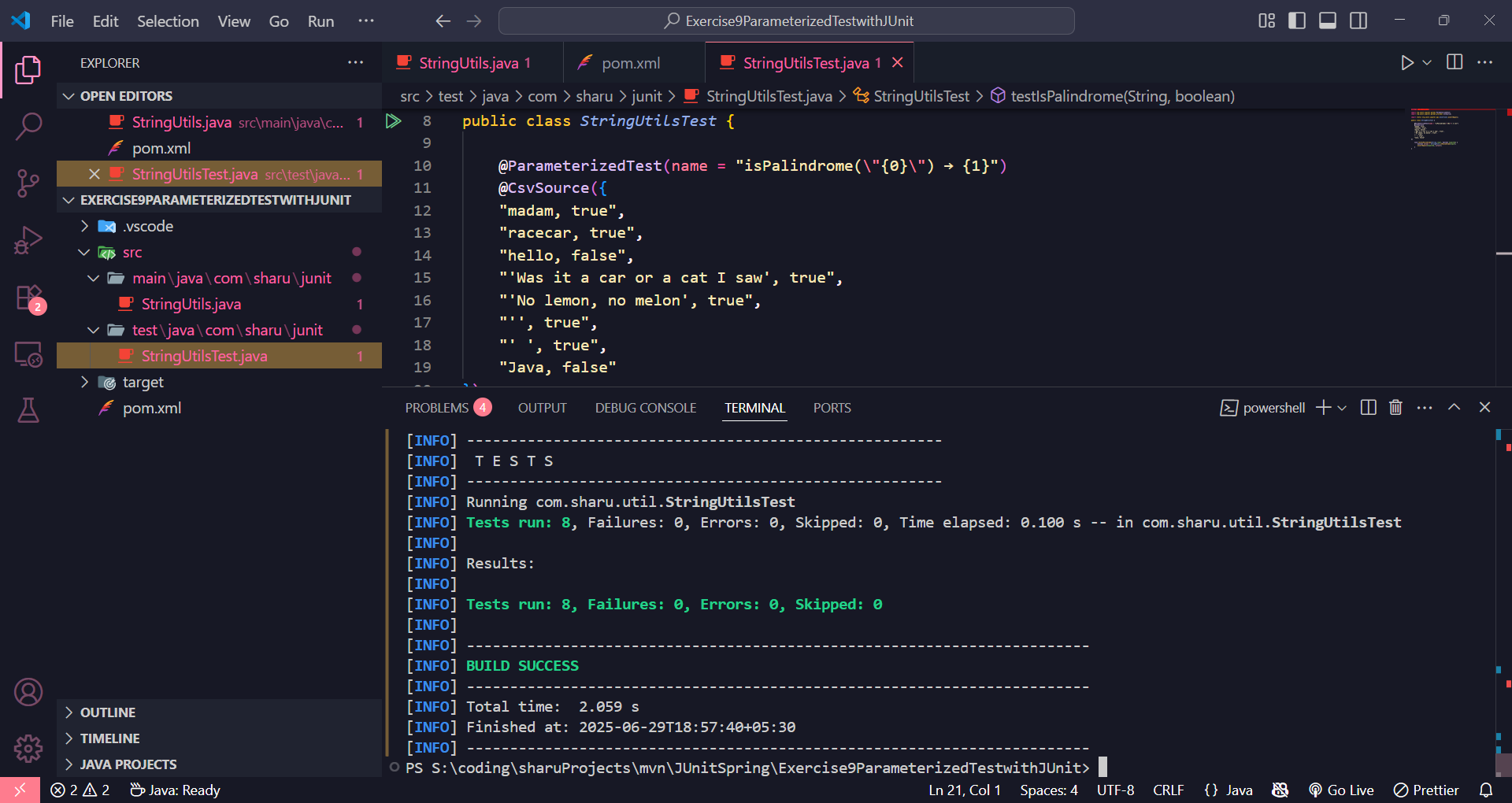
boolean result = StringUtils.isPalindrome(input);

assertEquals(expected, result, "Palindrome test failed for input: " + input);

}

}

**Output**



MOCKITO SPRING EXERCISE

EXERCISE 1: Mocking a Service Dependency in a Controller Test

**1. Main Application Class**

Exercise1MockingServiceDependencyControllerApplication.java

package com.sharu.repository;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class Exercise1MockingServiceDependencyControllerApplication {

public static void main(String[] args) {

SpringApplication.run(Exercise1MockingServiceDependencyControllerApplication.class, args);

}

}

**2. Entity: User.java**

entity/User.java

package com.sharu.repository.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class User {

@Id

private Long id;

private String name;

public User() {}

public User(Long id, String name) {

this.id = id;

this.name = name;

}

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

}

**3. Repository: UserRepository.java**

repository/UserRepository.java

package com.sharu.repository.repository;

import com.sharu.repository.entity.User;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

}

**4. Service: UserService.java**

service/UserService.java

package com.sharu.repository.service;

import com.sharu.repository.entity.User;

import com.sharu.repository.repository.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public User getUserById(Long id) {

return userRepository.findById(id).orElse(null);

}

}

**5. Controller: UserController.java**

controller/UserController.java

package com.sharu.repository.controller;

import com.sharu.repository.entity.User;

import com.sharu.repository.service.UserService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/users")

public class UserController {

@Autowired

private UserService userService;

@GetMapping("/{id}")

public ResponseEntity<User> getUser(@PathVariable Long id) {

User user = userService.getUserById(id);

if (user == null) {

return ResponseEntity.notFound().build();

}

return ResponseEntity.ok(user);

}

}

**6. Test: UserControllerTest.java**

test/controller/UserControllerTest.java

package com.sharu.repository.controller;

import com.sharu.repository.entity.User;

import com.sharu.repository.service.UserService;

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.boot.test.mock.mockito.MockBean;

import org.springframework.http.MediaType;

import org.springframework.test.web.servlet.MockMvc;

import static org.mockito.ArgumentMatchers.anyLong;

import static org.mockito.Mockito.when;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@WebMvcTest(UserController.class)

public class UserControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

public void testGetUser\_ReturnsUser() throws Exception {

User mockUser = new User(1L, "John Doe");

when(userService.getUserById(anyLong())).thenReturn(mockUser);

mockMvc.perform(get("/users/1")

.accept(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.id").value(1L))

.andExpect(jsonPath("$.name").value("John Doe"));

}

@Test

public void testGetUser\_UserNotFound() throws Exception {

when(userService.getUserById(anyLong())).thenReturn(null);

mockMvc.perform(get("/users/99")

.accept(MediaType.APPLICATION\_JSON))

.andExpect(status().isNotFound());

}

}

EXERCISE 2: Mocking a Repository in a Service Test

**User.java**

package com.sharu.repository;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class User {

@Id

private Long id;

private String name;

public User() {}

public User(Long id, String name) {

this.id = id;

this.name = name;

}

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

}

**UserRepository.java**

package com.sharu.repository;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserService.java**

package com.sharu.repository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public User getUserById(Long id) {

return userRepository.findById(id).orElse(null);

}

}

**UserController.java**

package com.sharu.repository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/users")

public class UserController {

@Autowired

private UserService userService;

@GetMapping("/{id}")

public ResponseEntity<User> getUser(@PathVariable Long id) {

User user = userService.getUserById(id);

if (user != null) {

return ResponseEntity.ok(user);

} else {

return ResponseEntity.notFound().build();

}

}

}

**Exercise3Application.java**

package com.sharu.repository;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class Exercise3Application {

public static void main(String[] args) {

SpringApplication.run(Exercise3Application.class, args);

}

}

**UserIntegrationTest.java**

package com.sharu.repository;

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.boot.test.mock.mockito.MockBean;

import org.springframework.test.web.servlet.MockMvc;

import static org.mockito.Mockito.when;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@SpringBootTest(classes = Exercise3Application.class)

@AutoConfigureMockMvc

public class UserIntegrationTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

void testGetUser\_ReturnsUser() throws Exception {

User mockUser = new User(1L, "Sharu");

when(userService.getUserById(1L)).thenReturn(mockUser);

mockMvc.perform(get("/users/1"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.id").value(1))

.andExpect(jsonPath("$.name").value("Sharu"));

}

@Test

void testGetUser\_ReturnsNotFound() throws Exception {

when(userService.getUserById(2L)).thenReturn(null);

mockMvc.perform(get("/users/2"))

.andExpect(status().isNotFound());

}

}

SLF4J LOGGING EXERCISE

EXERCISE 1 : SLF4J Logging: Error and Warning Level Demonstration(MANDATORY)

**LoggingExample.java**

package com.sharu.logging;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

private static final Logger logger = LoggerFactory.getLogger(LoggingExample.class);

public static void main(String[] args) {

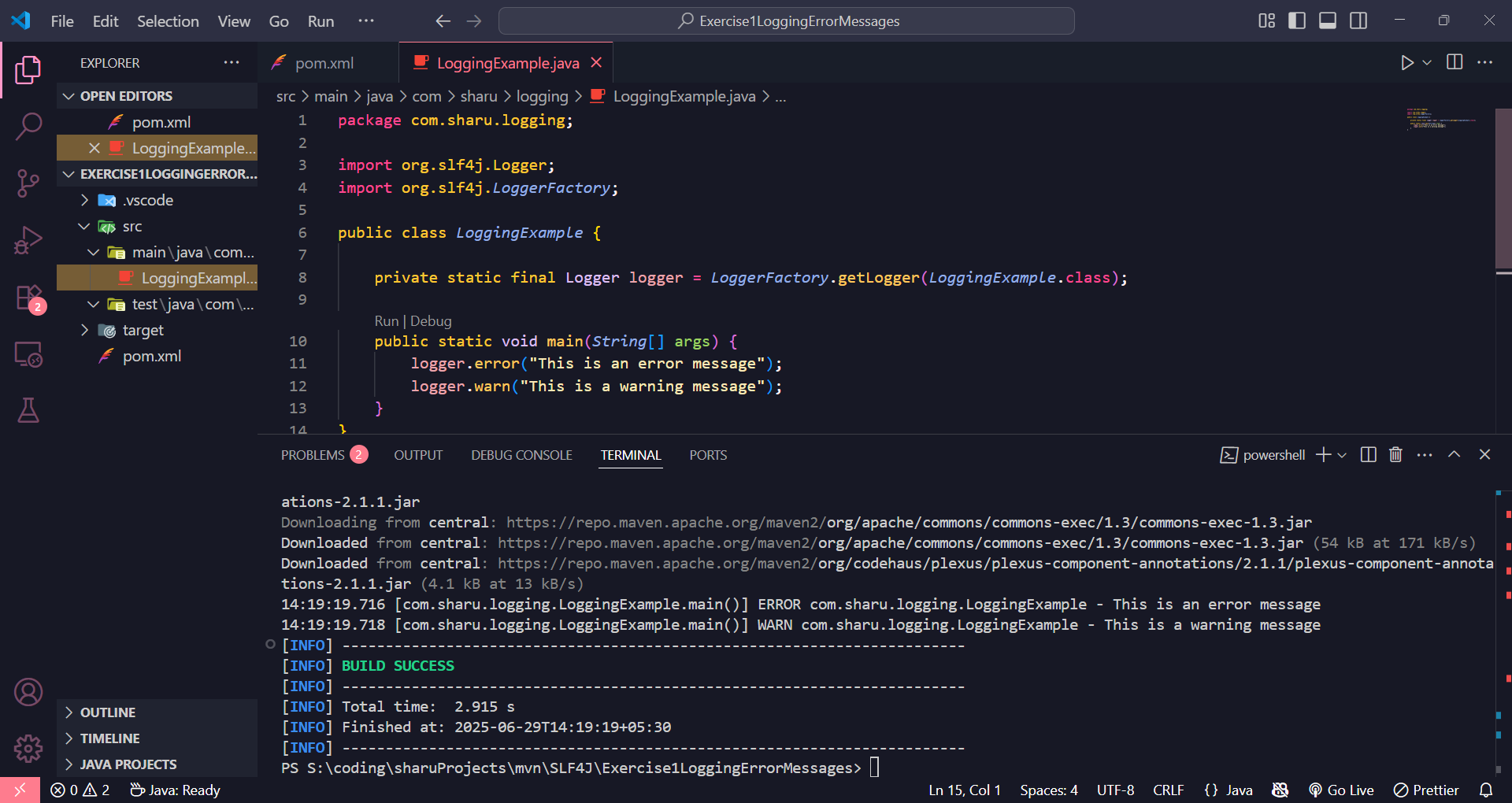
logger.error("This is an error message");

logger.warn("This is a warning message");

}

}

**Output**



EXERCISE 2 : SLF4J Logging: Parameterized Logging Demonstration

**ParameterizedLoggingExample.java**

package com.sharu.logging;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class ParameterizedLoggingExample {

private static final Logger logger = LoggerFactory.getLogger(ParameterizedLoggingExample.class);

public static void main(String[] args) {

String username = "Sharu";

int loginAttempts = 3;

boolean accountLocked = true;

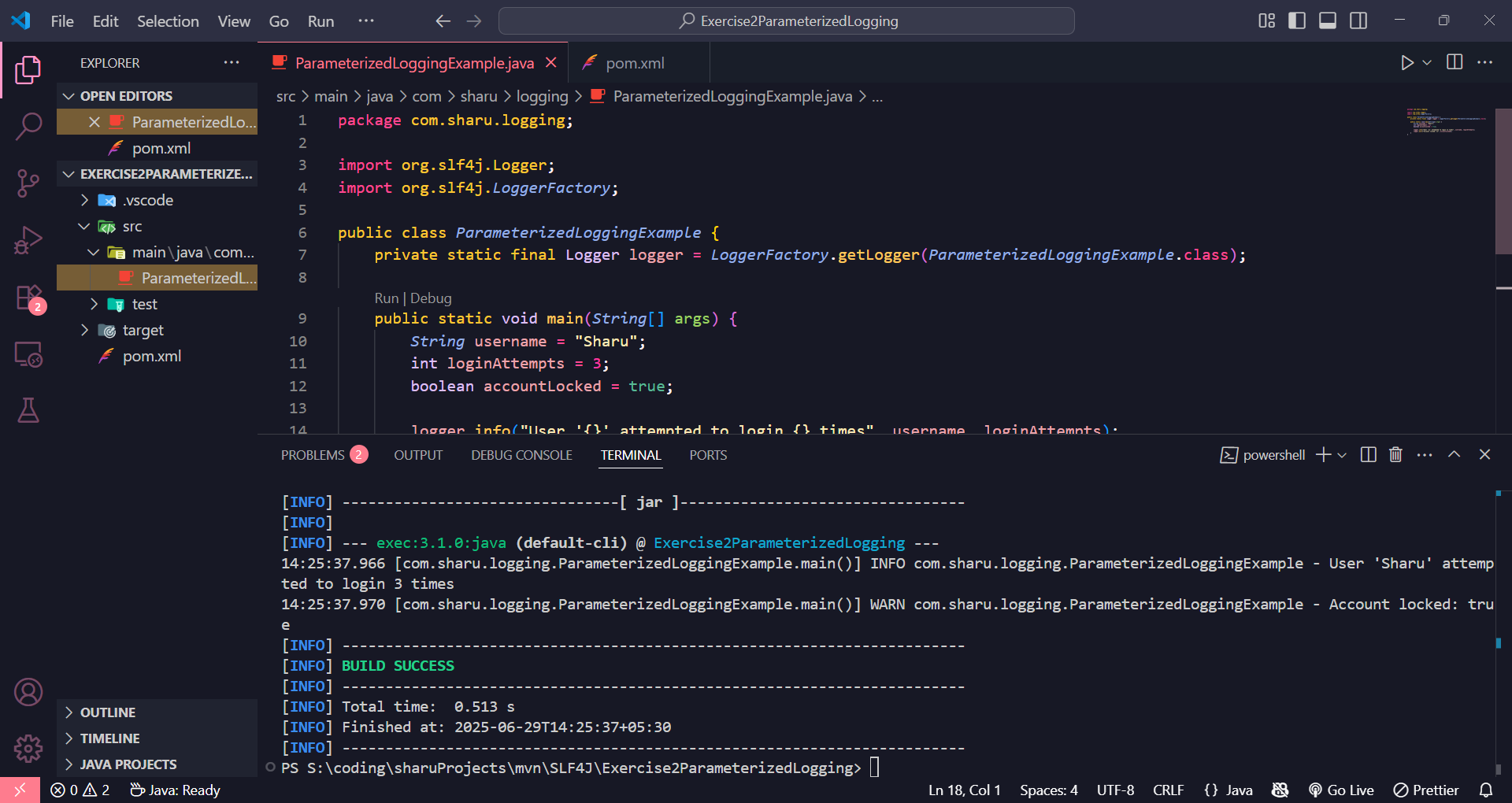
logger.info("User '{}' attempted to login {} times", username, loginAttempts);

logger.warn("Account locked: {}", accountLocked);

}

}

**Output**



EXERCISE 3 : SLF4J Logging: Demonstration of Multiple Appenders

**MultipleAppenderExample.java**

package com.sharu.logging;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class MultipleAppenderExample {

private static final Logger logger = LoggerFactory.getLogger(MultipleAppenderExample.class);

public static void main(String[] args) {

logger.debug("This is a DEBUG level message");

logger.info("This is an INFO level message");

logger.warn("This is a WARN level message");

logger.error("This is an ERROR level message");

}

}

**Output**

