**JUnit Basic Testing Exercises**

**Exercise 1: Setting Up JUnit**

**Code:**

**<!--pom.xml-->**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example.junitbasics</groupId>

<artifactId>junit\_basic\_testing</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>17</maven.compiler.source>

<maven.compiler.target>17</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**//ExampleTest.java**

public class ExampleTest {

}

**Exercise 2: Writing Basic JUnit Tests**

**Code:**

**//Calculator.java**

package com.example.junitbasics;

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**

import org.junit.Test;

import static org.junit.Assert.\*;

import com.example.junitbasics.Calculator;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

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

}

@Test

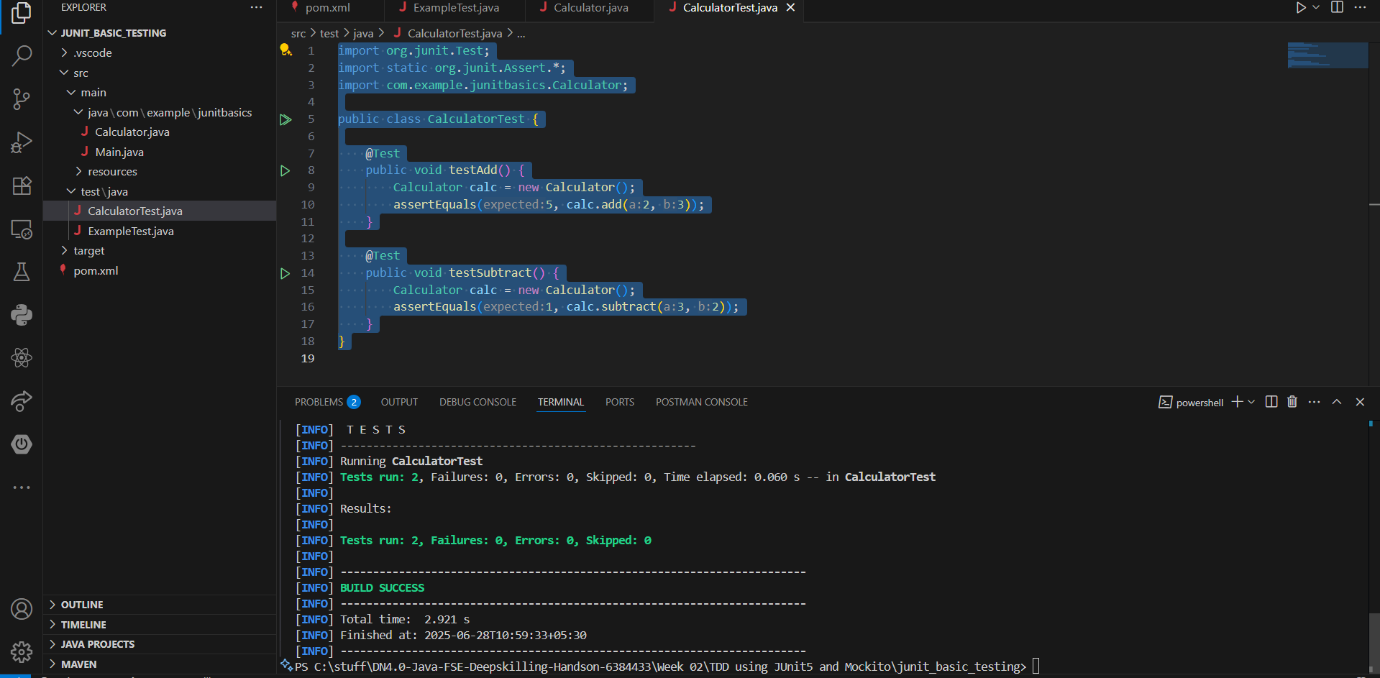
public void testSubtract() {

Calculator calc = new Calculator();

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

}

}

**Output:**

**Exercise 3: Assertions in JUnit**

**Code:**

**//AssertionsTest.java**

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals(5, 2 + 3);

assertTrue(5 > 3);

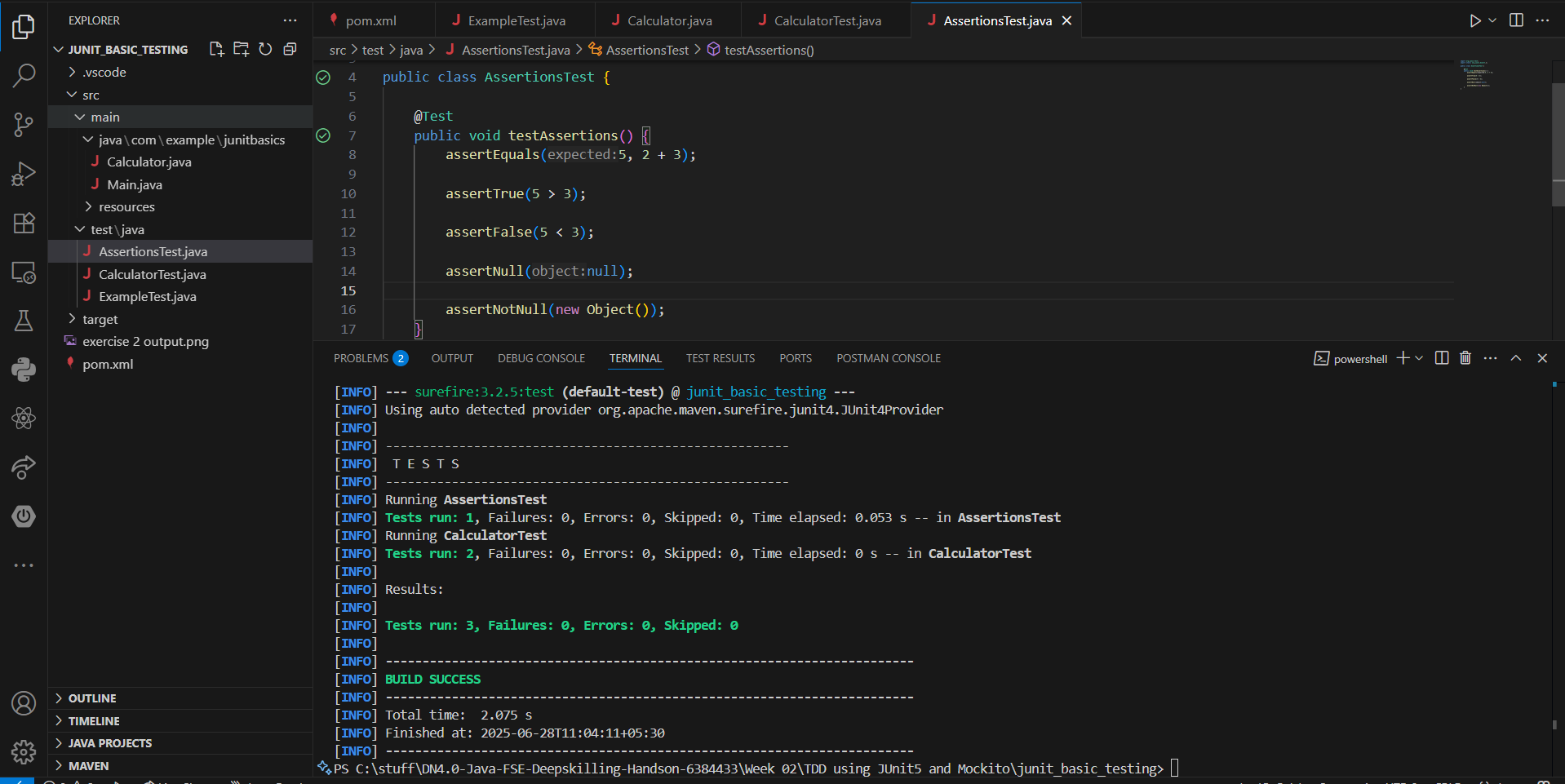
assertFalse(5 < 3);

assertNull(null);

assertNotNull(new Object());

}

}

**Output:**

**Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and**

**Teardown Methods in JUnit**

**Code:**

**//CalculatorFixtureTest.java**

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

import com.example.junitbasics.Calculator;

public class CalculatorFixtureTest {

private Calculator calculator;

@Before

public void setUp() {

calculator = new Calculator();

System.out.println("Setup done");

}

@After

public void tearDown() {

calculator = null;

System.out.println("Teardown done");

}

@Test

public void testAddition() {

int result = calculator.add(10, 5);

assertEquals(15, result);

}

@Test

public void testSubtraction() {

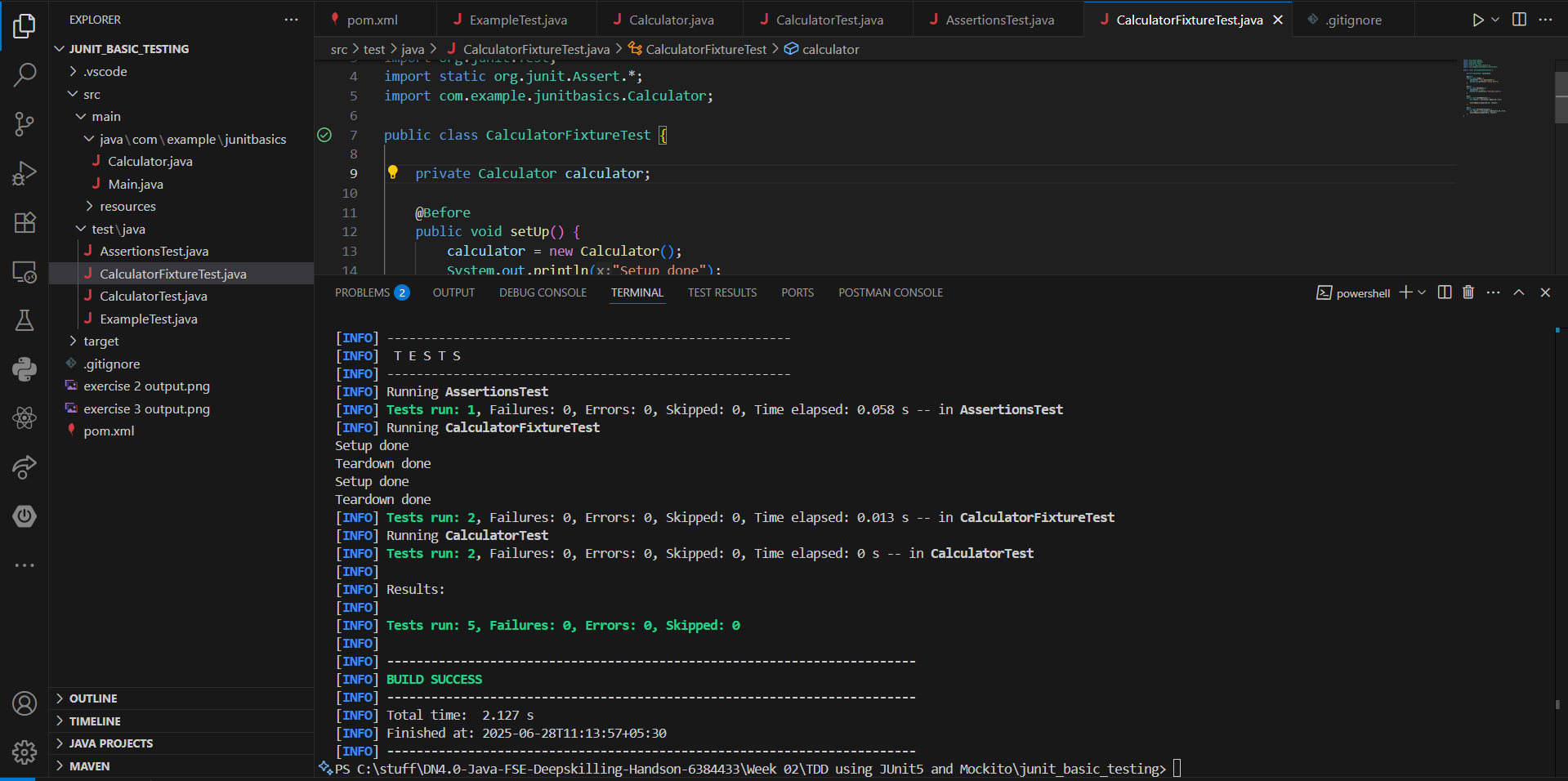
int result = calculator.subtract(10, 5);

assertEquals(5, result);

}

}

**Output:**



**Mockito Exercises**

**Exercise 1: Mocking and Stubbing**

**Code:**

**<!--pom.xml-->**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example.mockito</groupId>

<artifactId>mockito</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>17</maven.compiler.source>

<maven.compiler.target>17</maven.compiler.target>

</properties>

<dependencies>

<!-- JUnit Jupiter API -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>5.10.2</version>

<scope>test</scope>

</dependency>

<!-- JUnit Jupiter Engine -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-engine</artifactId>

<version>5.10.2</version>

<scope>test</scope>

</dependency>

<!-- Mockito Core -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.12.0</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**//ExternalApi.java**

package com.example.mockito;

public interface ExternalApi {

String getData();

}

**//MyService.java**

package com.example.mockito;

public class MyService {

private final ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.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;

import com.example.mockito.ExternalApi;

import com.example.mockito.MyService;

public class MyServiceTest {

@Test

public void testExternalApi() {

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

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

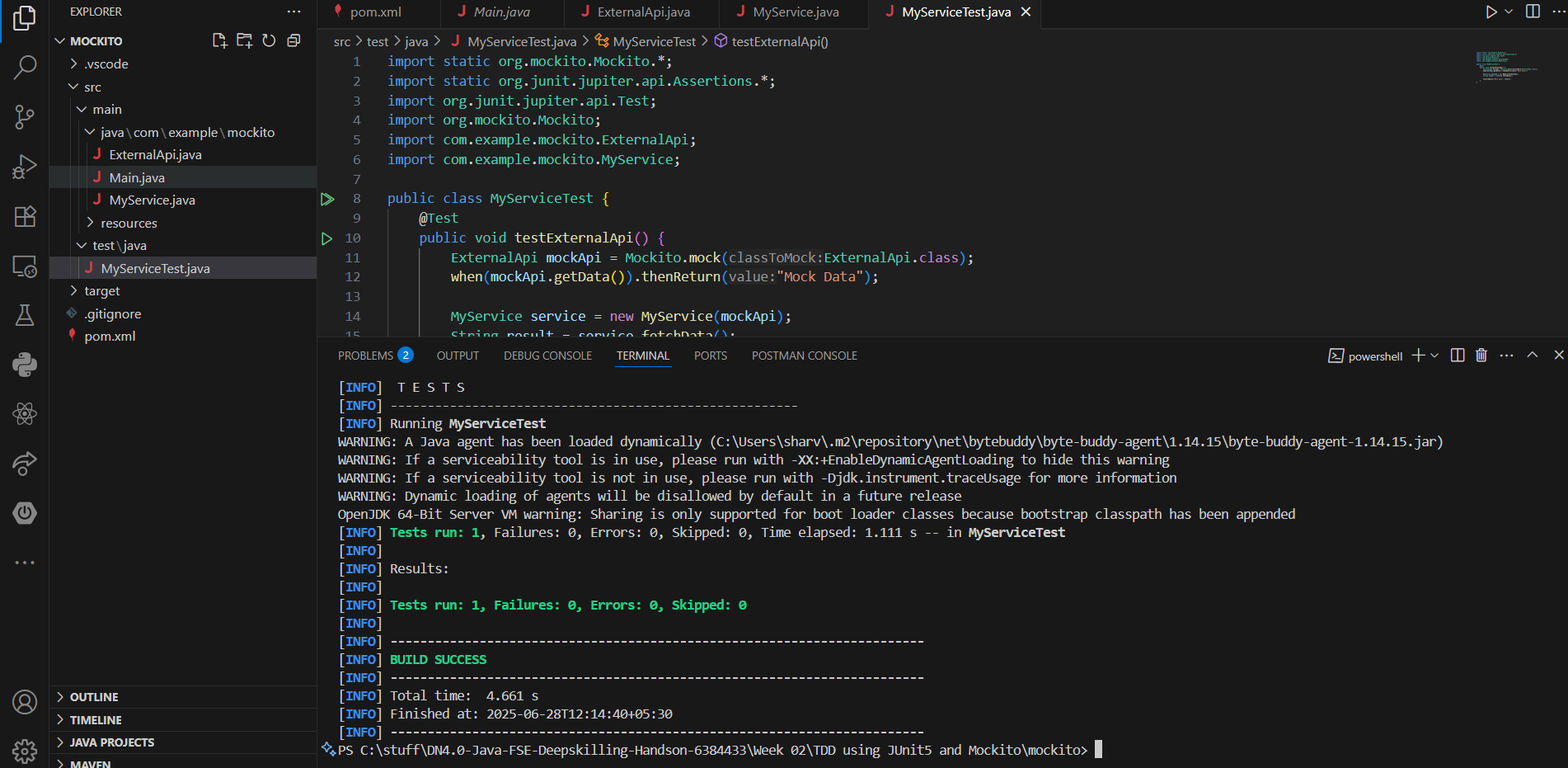
MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**Output:**

**Exercise 2: Verifying Interactions**

**Code:**

**//MyServiceTest.java**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import com.example.mockito.ExternalApi;

import com.example.mockito.MyService;

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);

}

@Test

public void testVerifyInteraction() {

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

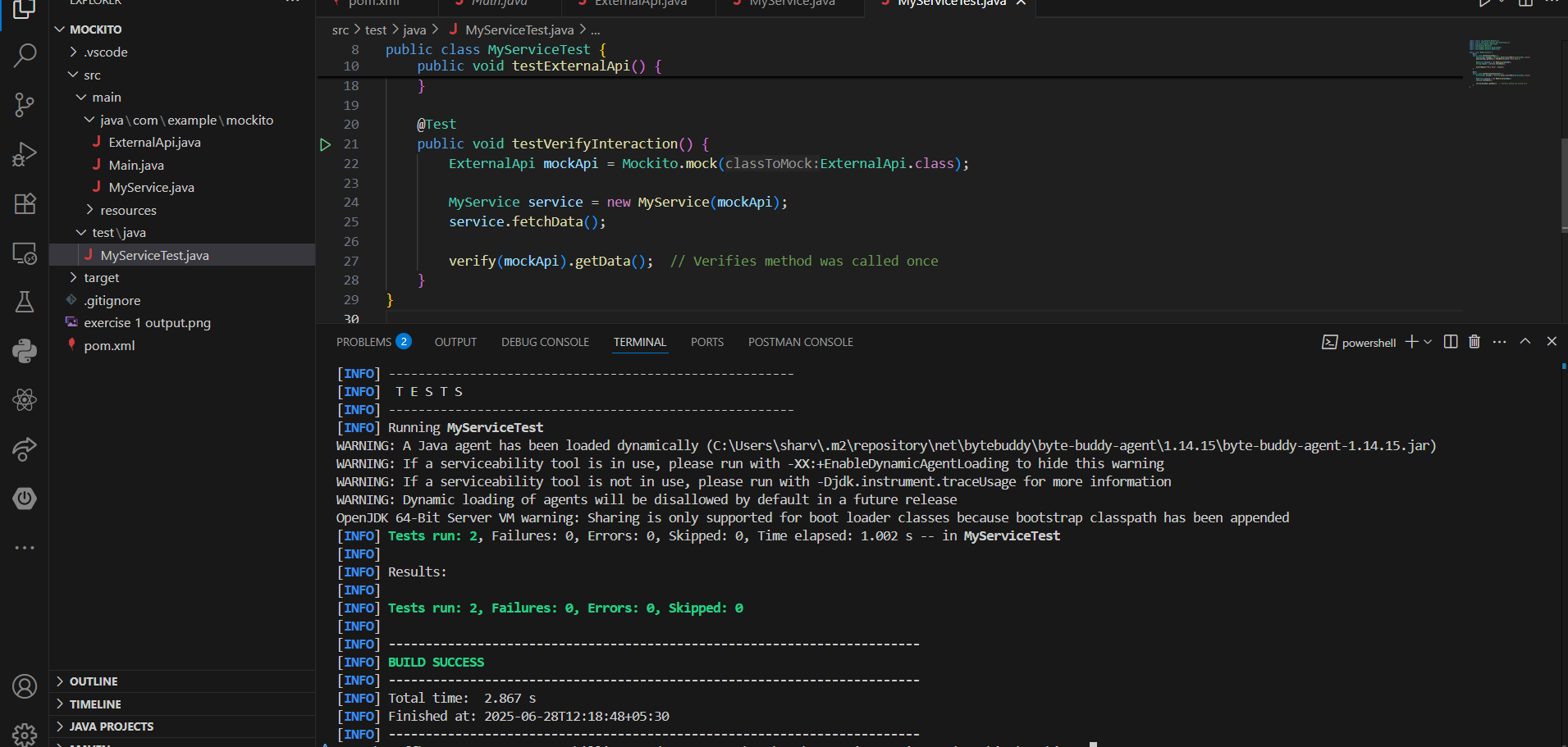
MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

**Output:**

**Exercise 3: Argument Matching**

**Code:**

**//ExternalApi.java**

package com.example.mockito;

public interface ExternalApi {

String getData();

String getData(String id);

}

**//ArgumentMatchingTest.java**

import static org.mockito.Mockito.\*;

import com.example.mockito.ExternalApi;

import org.junit.jupiter.api.Test;

public class ArgumentMatchingTest {

@Test

public void testArgumentMatching() {

ExternalApi mockApi = mock(ExternalApi.class);

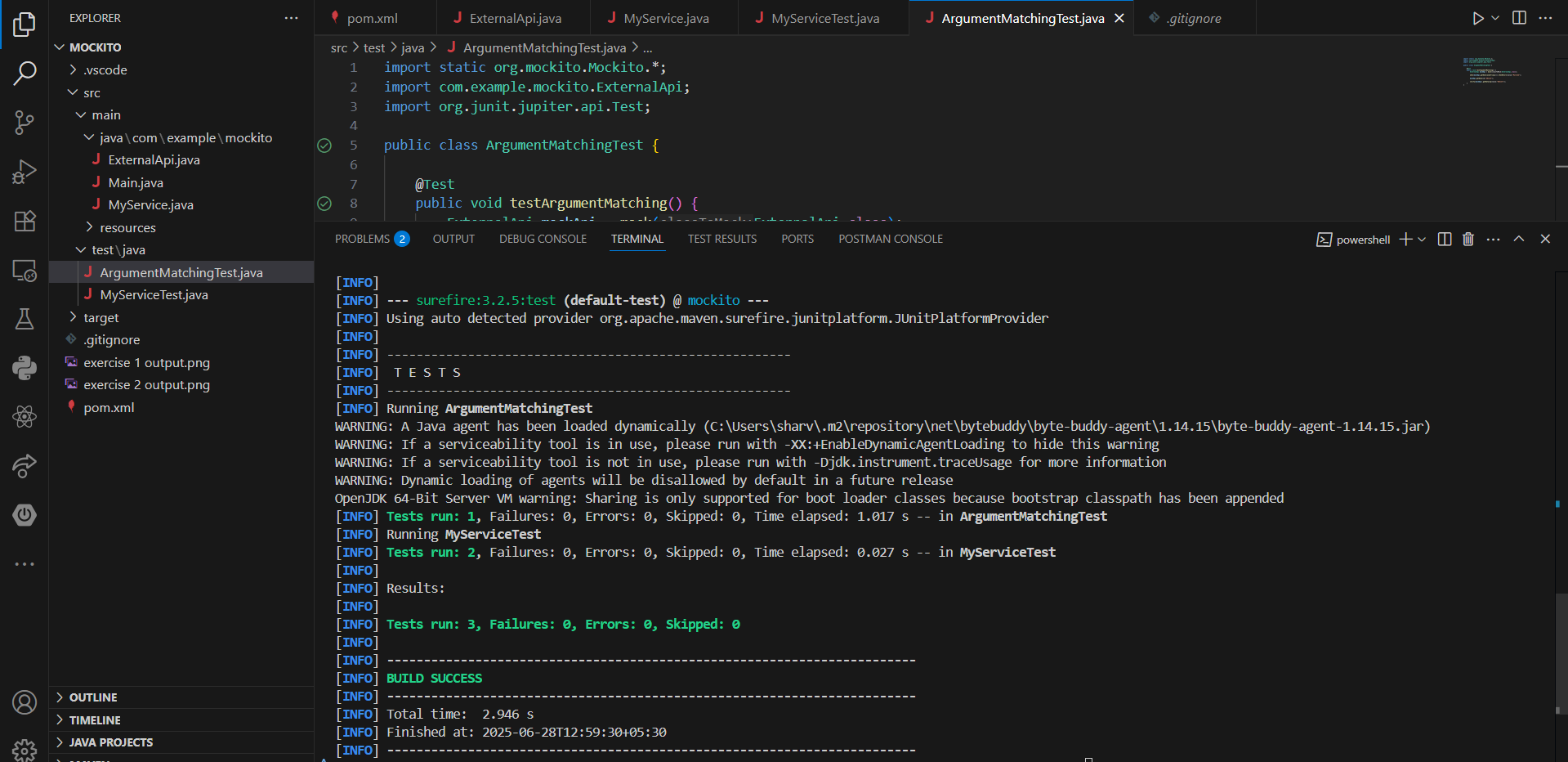
when(mockApi.getData(anyString())).thenReturn("Matched");

mockApi.getData("ABC123");

verify(mockApi).getData(eq("ABC123"));

}

}

**Output:**

**Exercise 4: Handling Void Methods**

**Code:**

**//VoidMethodTest.java**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import com.example.mockito.ExternalApi;

public class VoidMethodTest {

@Test

public void testVoidMethod() {

ExternalApi mockLogger = mock(ExternalApi.class);

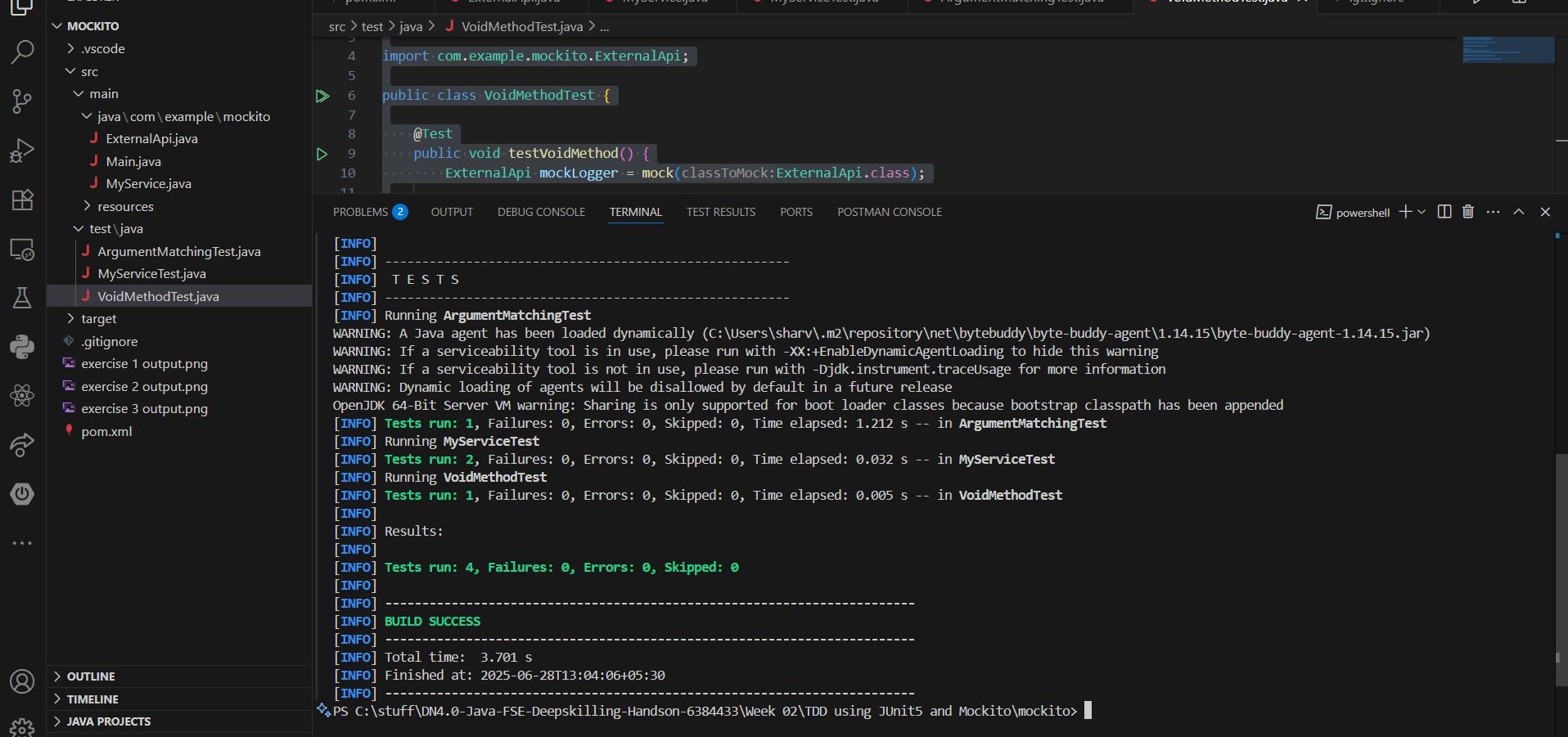
mockLogger.getData("Hello");

verify(mockLogger).getData("Hello");

}

}

**Output:**

****

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

**Code:**

**//** **MultipleReturnsTest.java**

import static org.mockito.Mockito.\*;

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

import com.example.mockito.ExternalApi;

import org.junit.jupiter.api.Test;

public class MultipleReturnsTest {

@Test

public void testMultipleReturns() {

ExternalApi mockApi = mock(ExternalApi.class);

when(mockApi.getData())

.thenReturn("First")

.thenReturn("Second")

.thenReturn("Third");

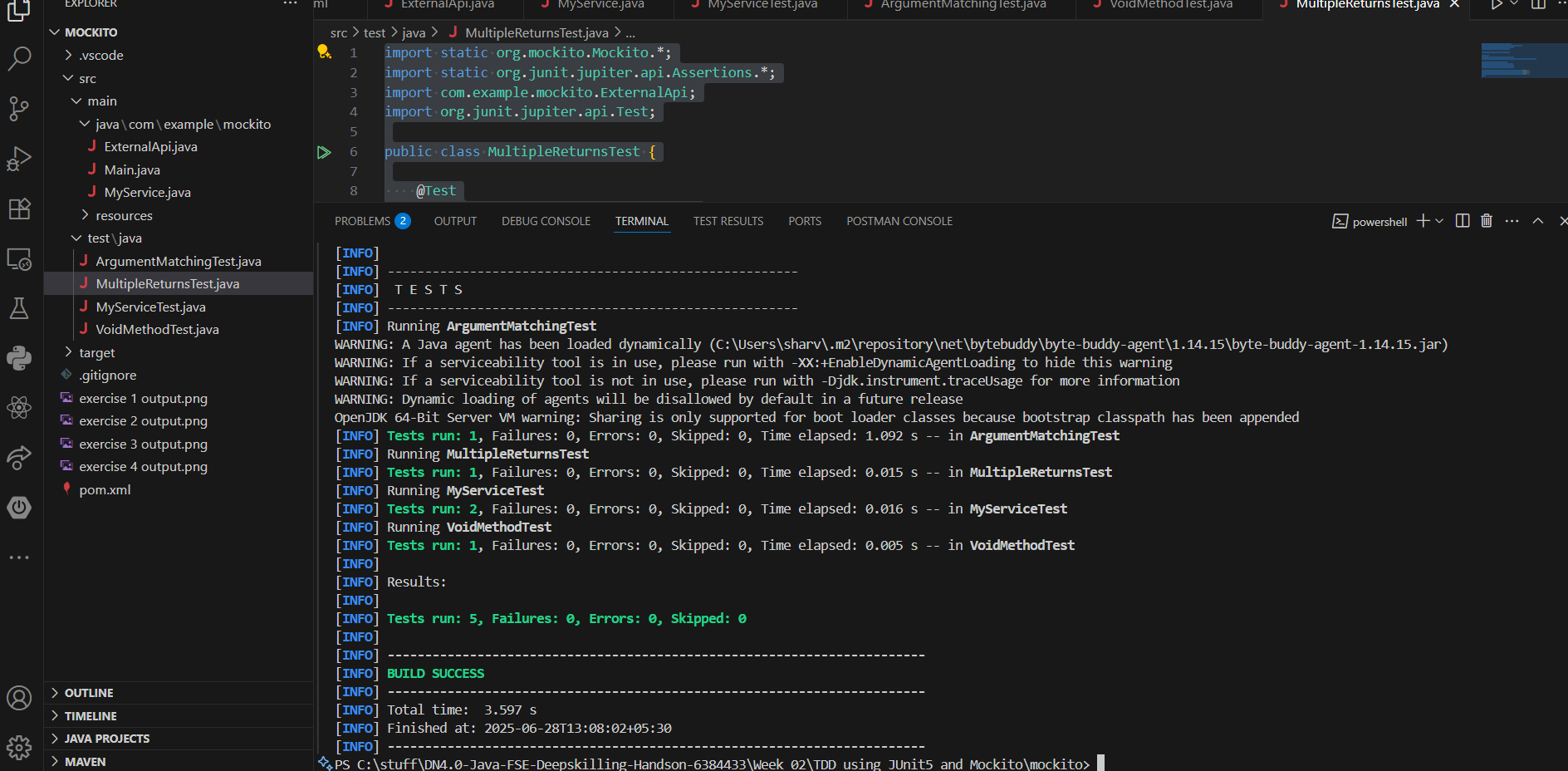
assertEquals("First", mockApi.getData());

assertEquals("Second", mockApi.getData());

assertEquals("Third", mockApi.getData());

}

}

**Output:**

**Exercise 6: Verifying Interaction Order**

**Code:**

**//** **Processor.java**

package com.example.mockito;

public interface Processor {

void start();

void process();

void finish();

}

**//** **InteractionOrderTest.java**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

import com.example.mockito.Processor;

public class InteractionOrderTest {

@Test

public void testInteractionOrder() {

Processor mockProcessor = mock(Processor.class);

mockProcessor.start();

mockProcessor.process();

mockProcessor.finish();

InOrder inOrder = inOrder(mockProcessor);

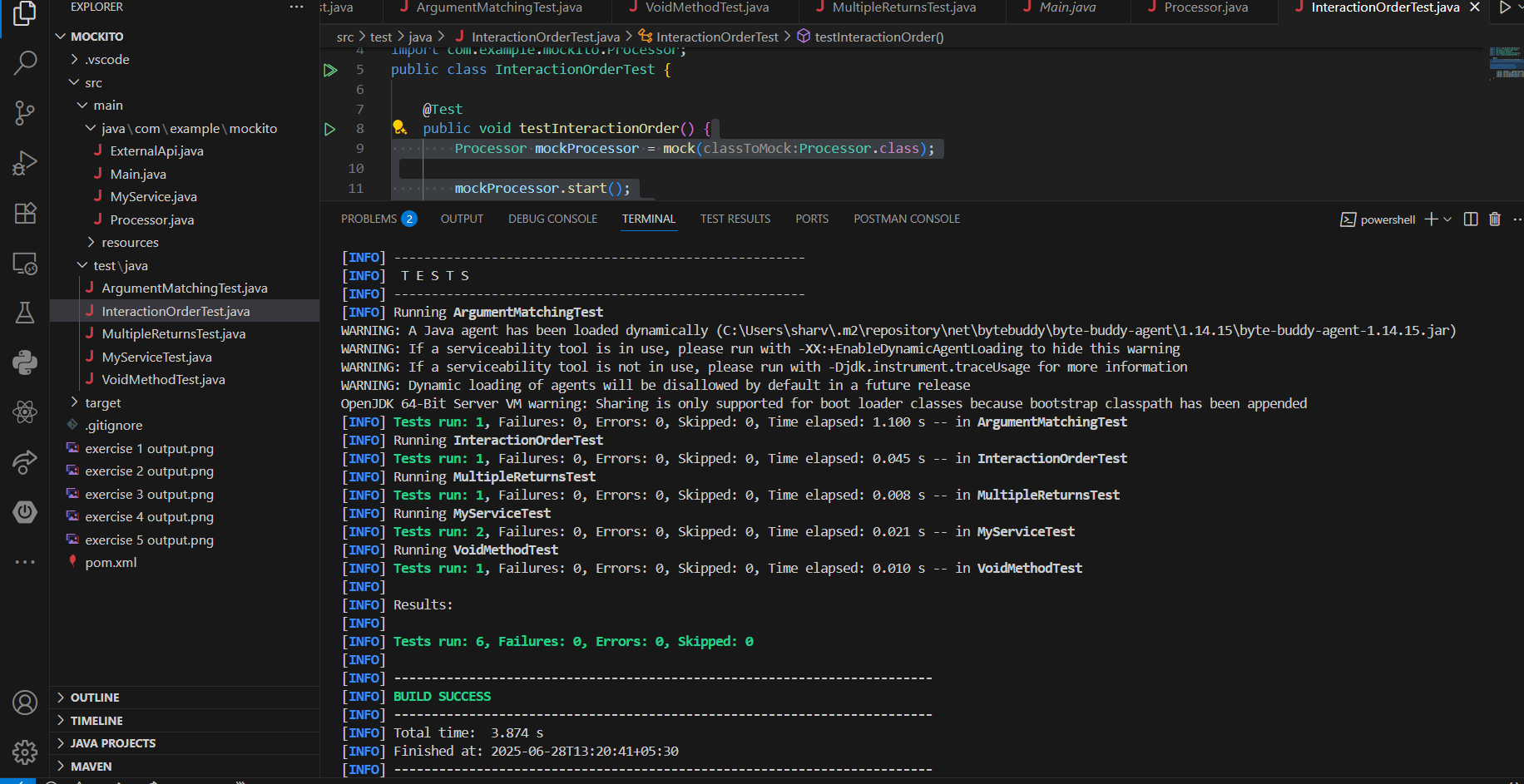
inOrder.verify(mockProcessor).start();

inOrder.verify(mockProcessor).process();

inOrder.verify(mockProcessor).finish();

}

}

**Output:**

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

**Code:**

**//** **VoidMethodExceptionTest.java**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import com.example.mockito.ExternalApi;

public class VoidMethodExceptionTest {

@Test

public void testVoidMethodThrowsException() {

ExternalApi mockLogger = mock(ExternalApi.class);

doThrow(new RuntimeException("Logging failed"))

.when(mockLogger)

.getData("error");

try {

mockLogger.getData("error");

} catch (RuntimeException e) {

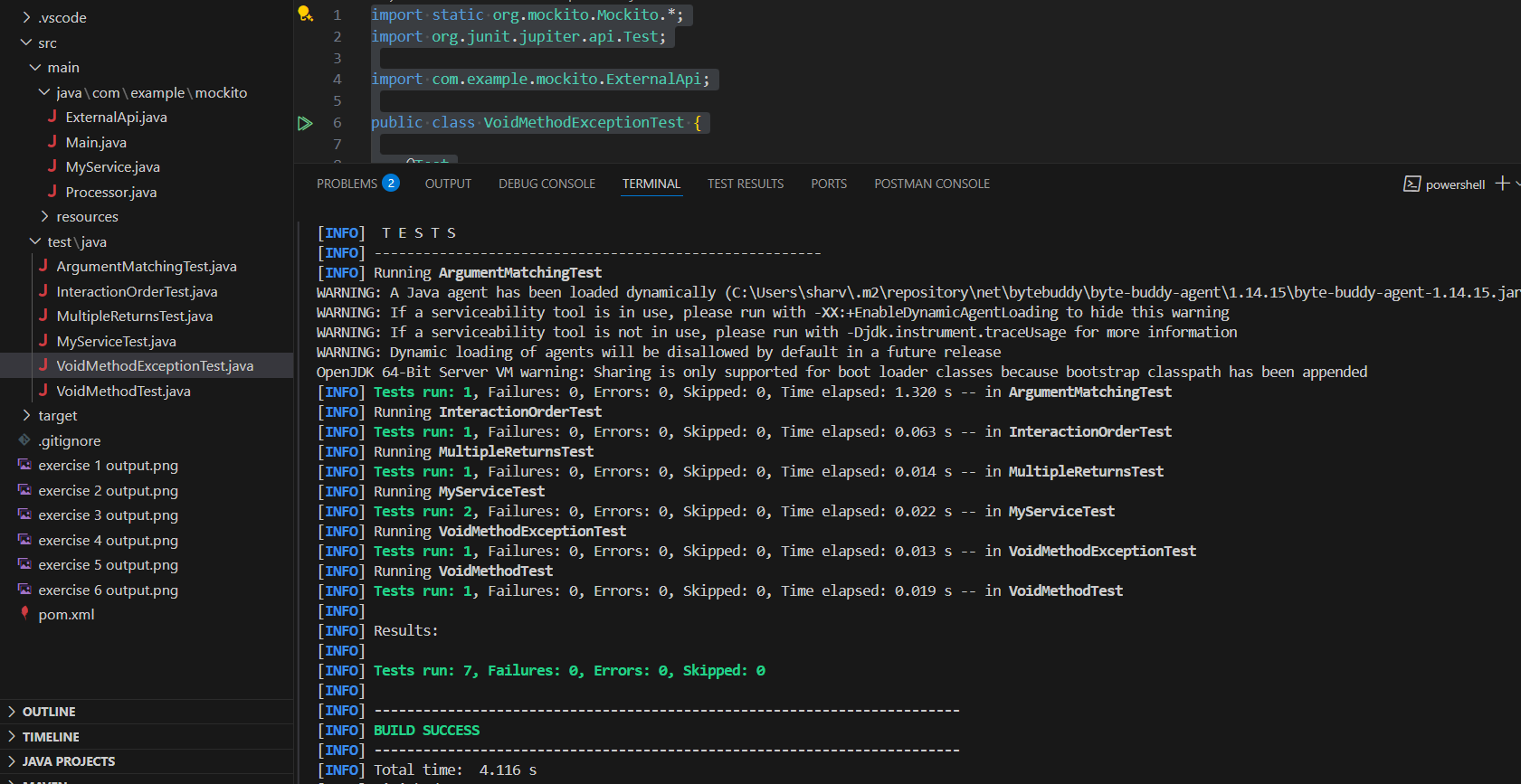
assert e.getMessage().equals("Logging failed");

}

verify(mockLogger).getData("error");

}

}

**Output:**

**JUnit Advanced Exercises**

**Exercise 1: Parameterized Tests**

**Code:**

**//EvenChecker.java**

package com.example.junitadvanced;

public class EvenChecker {

public boolean isEven(int number) {

return number % 2 == 0;

}

}

**//EvenCheckerTest.java**

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

import org.junit.jupiter.params.ParameterizedTest;

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

import com.example.junitadvanced.EvenChecker;

public class EvenCheckerTest {

EvenChecker checker = new EvenChecker();

@ParameterizedTest

@ValueSource(ints = {2, 4, 6, 10, 100})

void testIsEven\_withEvenNumbers(int number) {

assertTrue(checker.isEven(number));

}

@ParameterizedTest

@ValueSource(ints = {1, 3, 5, 7, 99})

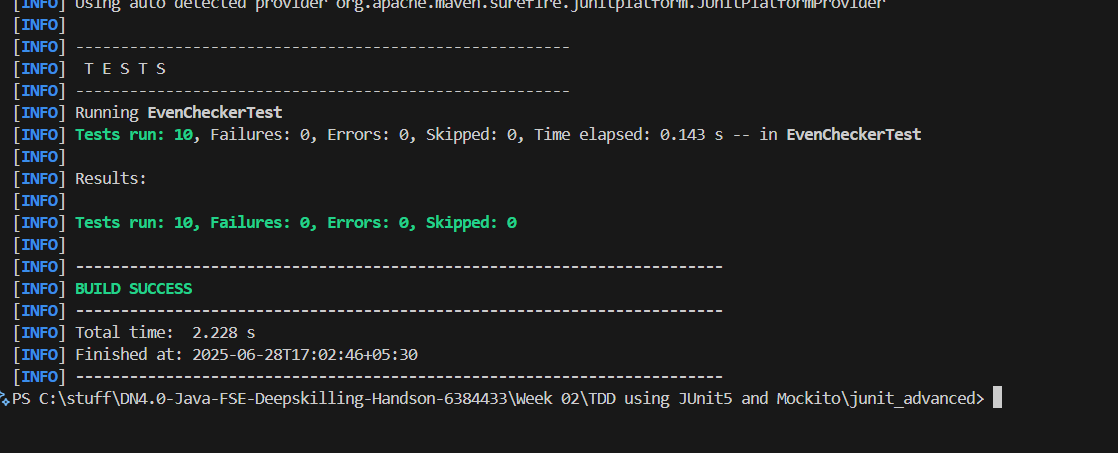
void testIsEven\_withOddNumbers(int number) {

assertFalse(checker.isEven(number));

}

}

**Output:**

****