TDD using JUnit5 and Mockito

**JUnit Testing Exercises**

**Exercise 1: Setting Up Junit**

**package** com.example.junit;

**import** org.junit.Test;

**import** **static** org.junit.Assert.*assertEquals*;

**public** **class** SampleTest {

@Test

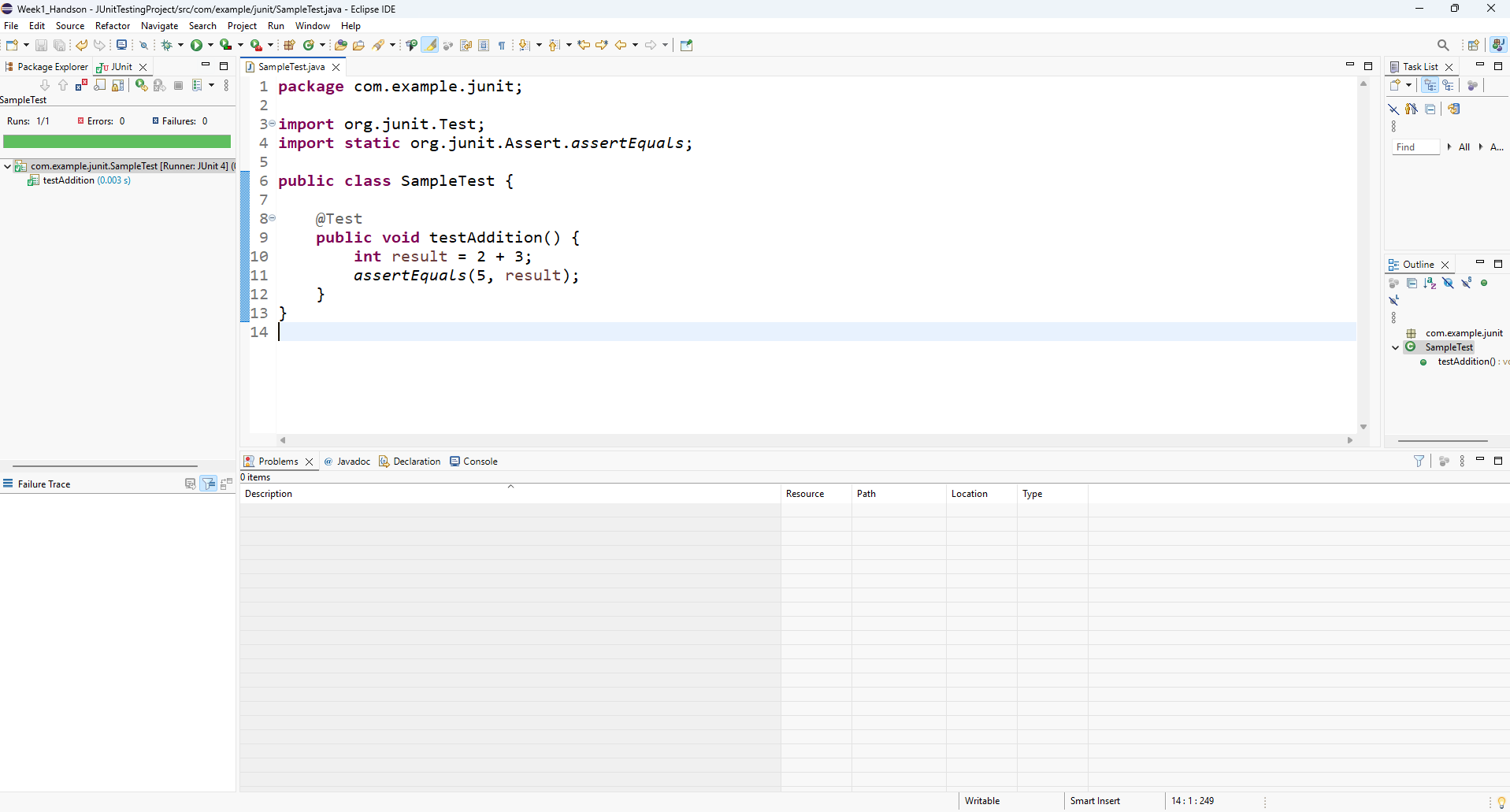
**public** **void** testAddition() {

**int** result = 2 + 3;

*assertEquals*(5, result);

}

}



**Exercise 2: Writing Basic JUnit Tests**

Calculator.java

**package** com.example.junit;

**public** **class** Calculator {

**public** **int** add(**int** a, **int** b) {

**return** a + b;

}

**public** **int** subtract(**int** a, **int** b) {

**return** a - b;

}

**public** **int** multiply(**int** a, **int** b) {

**return** a \* b;

}

**public** **int** divide(**int** a, **int** b) {

**if** (b == 0) **throw** **new** IllegalArgumentException("Division by zero");

**return** a / b;

}

}

CalculatorTest.java

**package** com.example.junit;

**import** org.junit.Test;

**import** **static** org.junit.Assert.\*;

**public** **class** CalculatorTest {

Calculator calc = **new** Calculator();

@Test

**public** **void** testAdd() {

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

}

@Test

**public** **void** testSubtract() {

*assertEquals*(1, calc.subtract(4, 3));

}

@Test

**public** **void** testMultiply() {

*assertEquals*(12, calc.multiply(3, 4));

}

@Test

**public** **void** testDivide() {

*assertEquals*(2, calc.divide(10, 5));

}

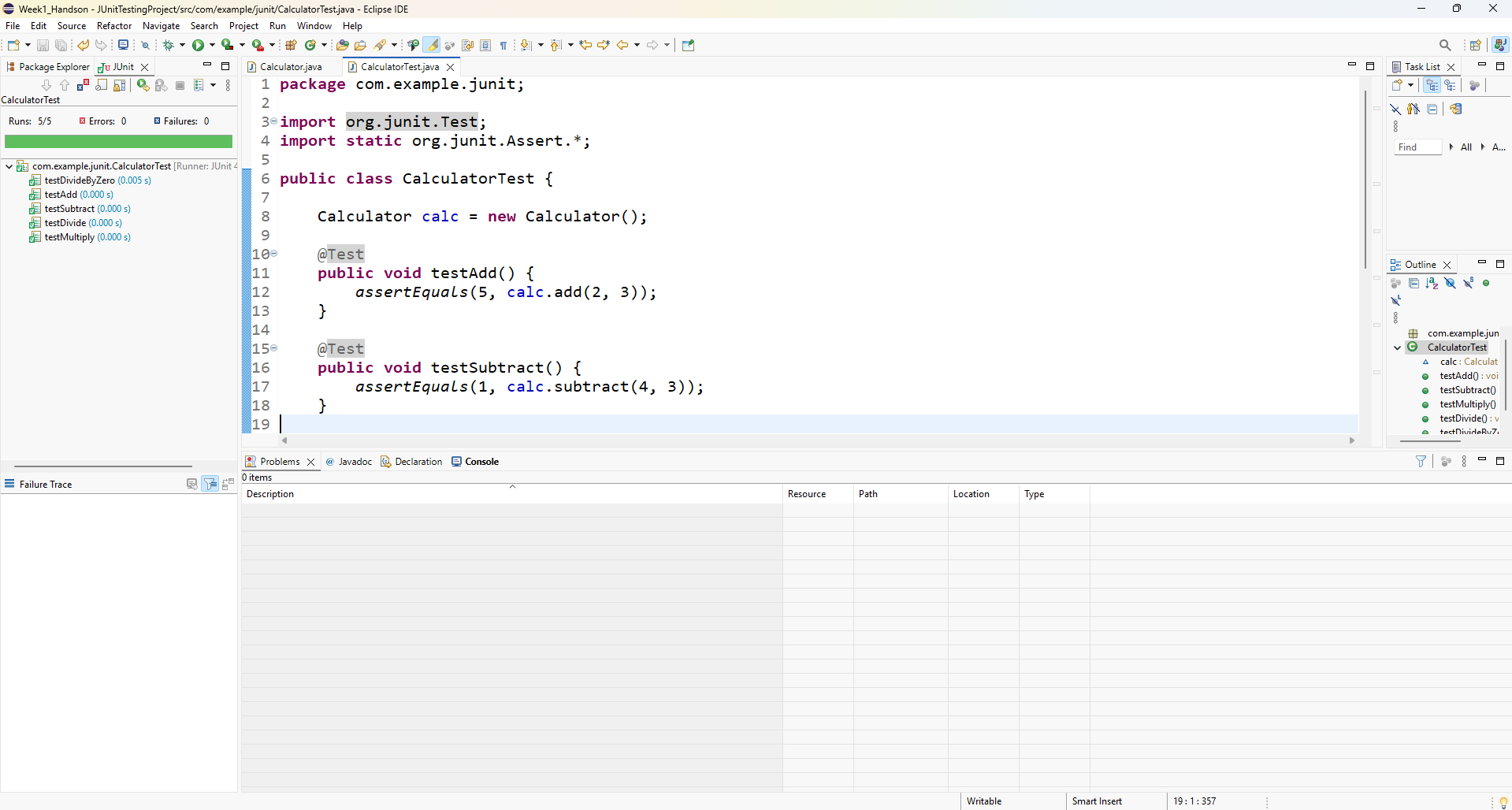
@Test(expected = IllegalArgumentException.**class**)

**public** **void** testDivideByZero() {

calc.divide(5, 0);

}

}



**Exercise 3: Assertions in Junit**

AssertionsTest.java

**package** com.example.junit;

**import** org.junit.Test;

**import** **static** org.junit.Assert.\*;

**public** **class** AssertionsTest {

@Test

**public** **void** testAssertions() {

// Assert equals

*assertEquals*(5, 2 + 3);

// Assert true

*assertTrue*(5 > 3);

// Assert false

*assertFalse*(5 < 3);

// Assert null

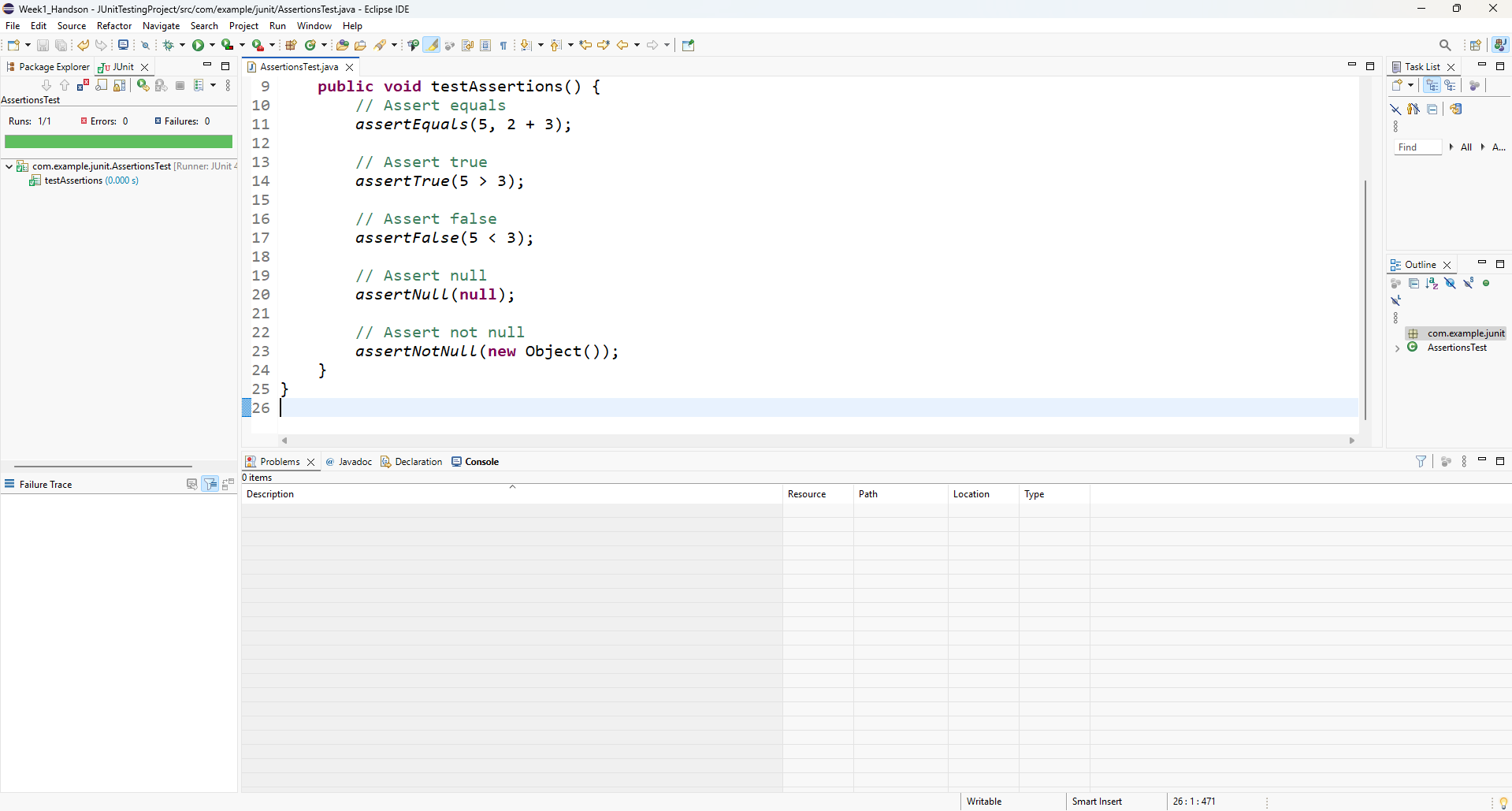
*assertNull*(**null**);

// Assert not null

*assertNotNull*(**new** Object());

}

}



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

AaaPatternTest.java

package com.example.junit;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AaaPatternTest {

private Calculator calc;

// Setup method runs before every @Test

@Before

public void setUp() {

calc = new Calculator();

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

}

// Teardown method runs after every @Test

@After

public void tearDown() {

System.out.println("Test finished");

}

@Test

public void testAddition\_AAA() {

// Arrange

int a = 10;

int b = 5;

// Act

int result = calc.add(a, b);

// Assert

assertEquals(15, result);

}

@Test

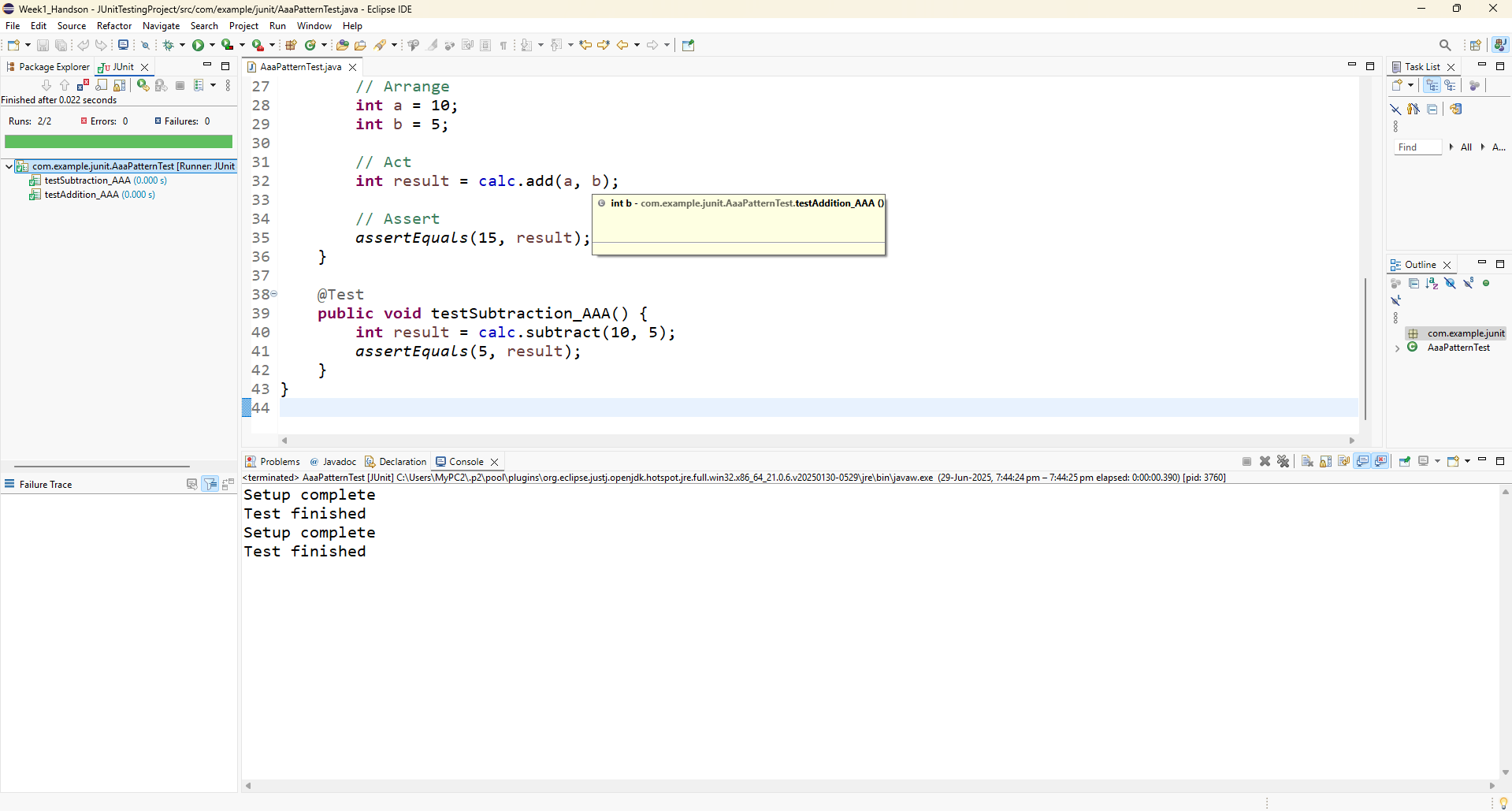
public void testSubtraction\_AAA() {

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

assertEquals(5, result);

}

}



Mockito exercises

Exercise 1: Mocking and Stubbing

pom.xml

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>MockitoDemo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- JUnit 5 -->

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.11.0</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

ExternalApi.java

**package** com.example;

**public** **interface** ExternalApi {

String getData();

}

MyService.java

**package** com.example;

**public** **class** MyService {

**private** ExternalApi api;

**public** MyService(ExternalApi api) {

**this**.api = api;

}

**public** String fetchData() {

**return** api.getData();

}

}

MyServiceTest\_MockingStubbing.java

**package** com.example;

**import** org.junit.jupiter.api.Test;

**import** **static** org.mockito.Mockito.\*;

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

**public** **class** MyServiceTest\_MockingStubbing {

@Test

**public** **void** testExternalApi() {

// Step 1: Create mock object

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

// Step 2: Stub method to return predefined value

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

// Step 3: Use the mock in the service

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

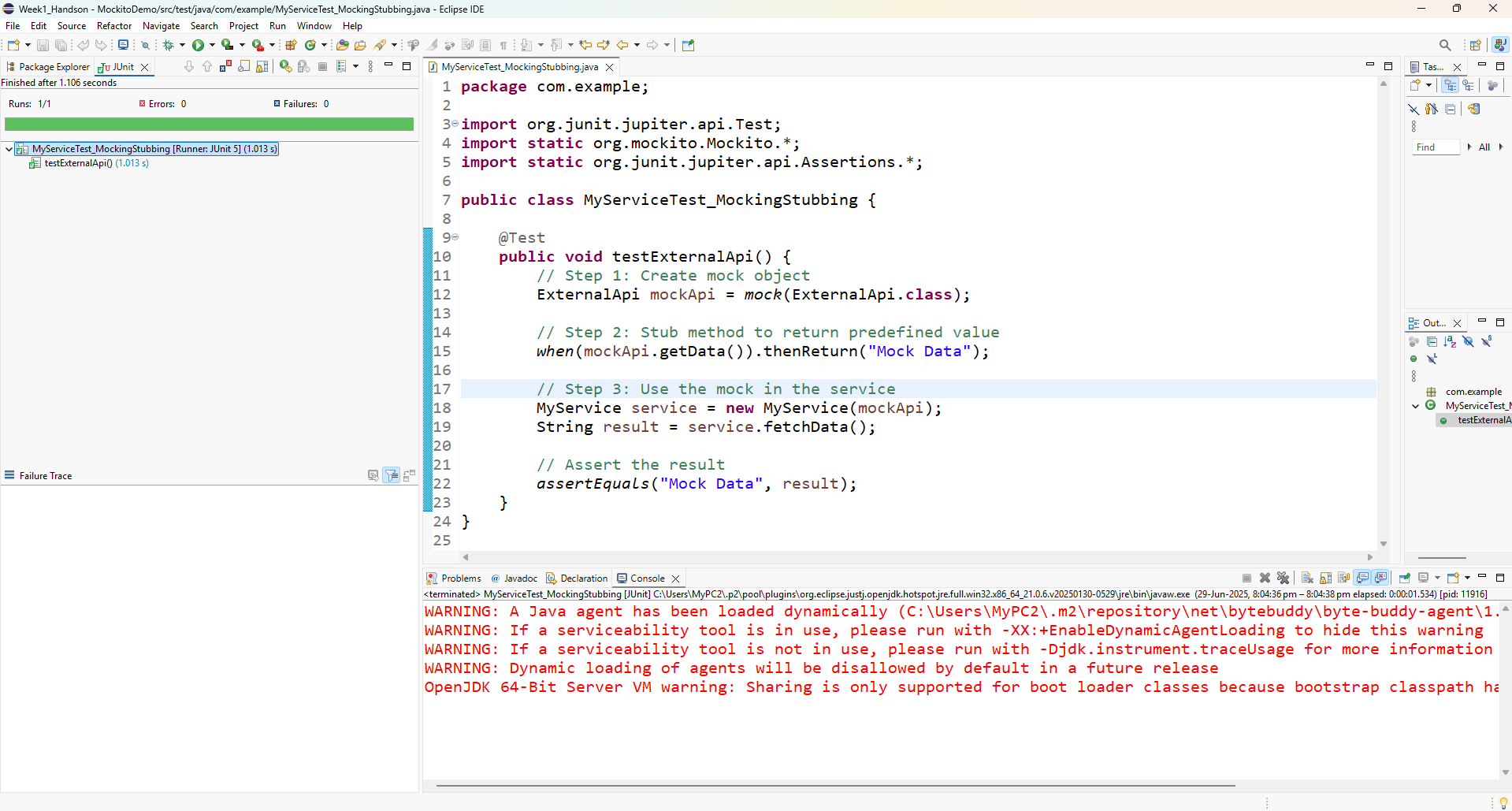
String result = service.fetchData();

// Assert the result

*assertEquals*("Mock Data", result);

}

}



**Exercise 2: Verifying Interactions**

MyServiceTest\_VerifyInteraction.java

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest\_VerifyInteraction {

@Test

public void testVerifyInteraction() {

// Step 1: Create mock object

ExternalApi mockApi = mock(ExternalApi.class);

// Step 2: Use the mock in the service

MyService service = new MyService(mockApi);

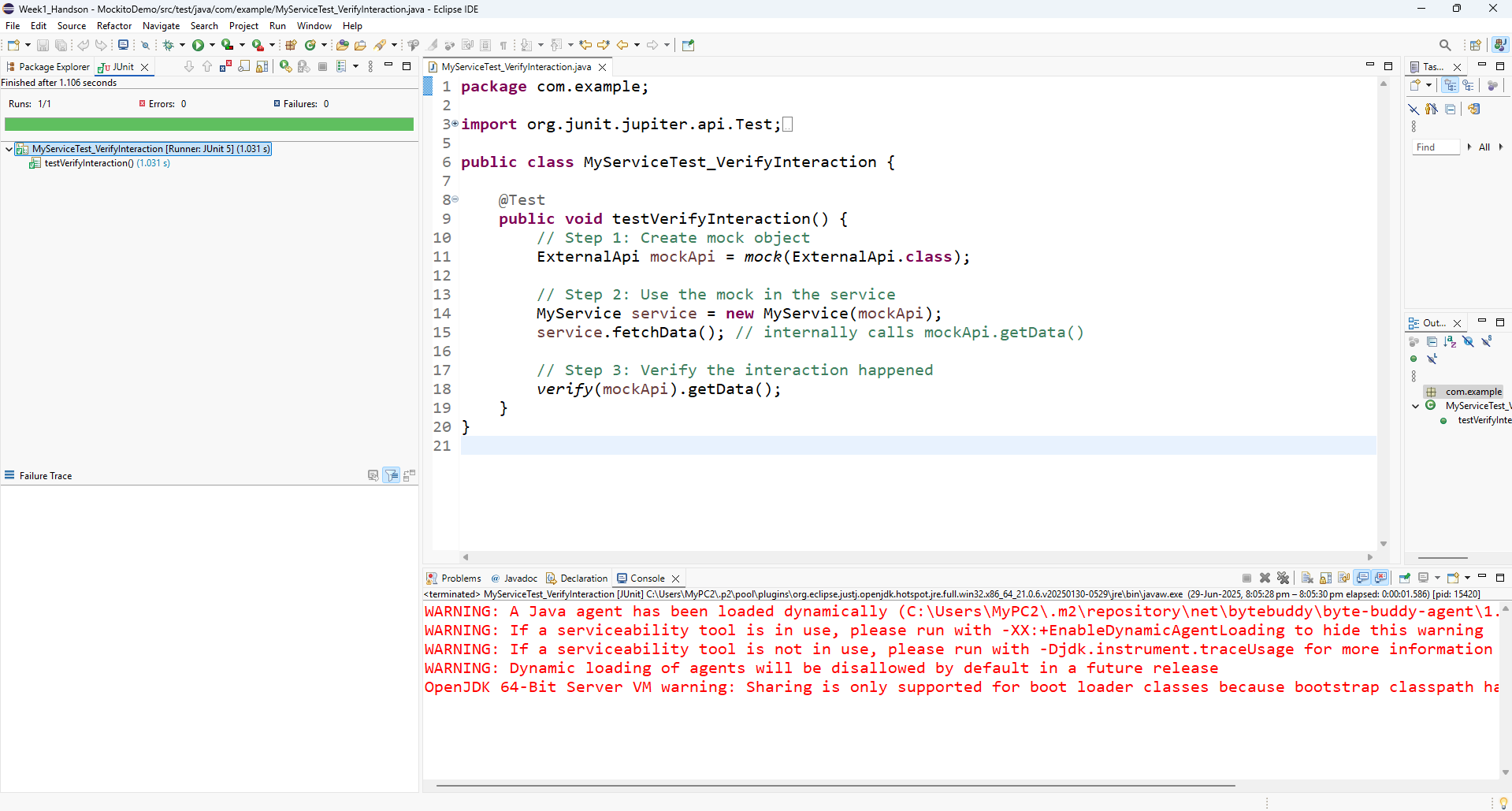
service.fetchData(); // internally calls mockApi.getData()

// Step 3: Verify the interaction happened

verify(mockApi).getData();

}

}



**Exercise 3: Argument Matching**

**MyServiceTest\_ArgumentMatching.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

public class MyServiceTest\_ArgumentMatching {

interface Logger {

void log(String level, String message);

}

@Test

public void testArgumentMatching() {

Logger mockLogger = mock(Logger.class);

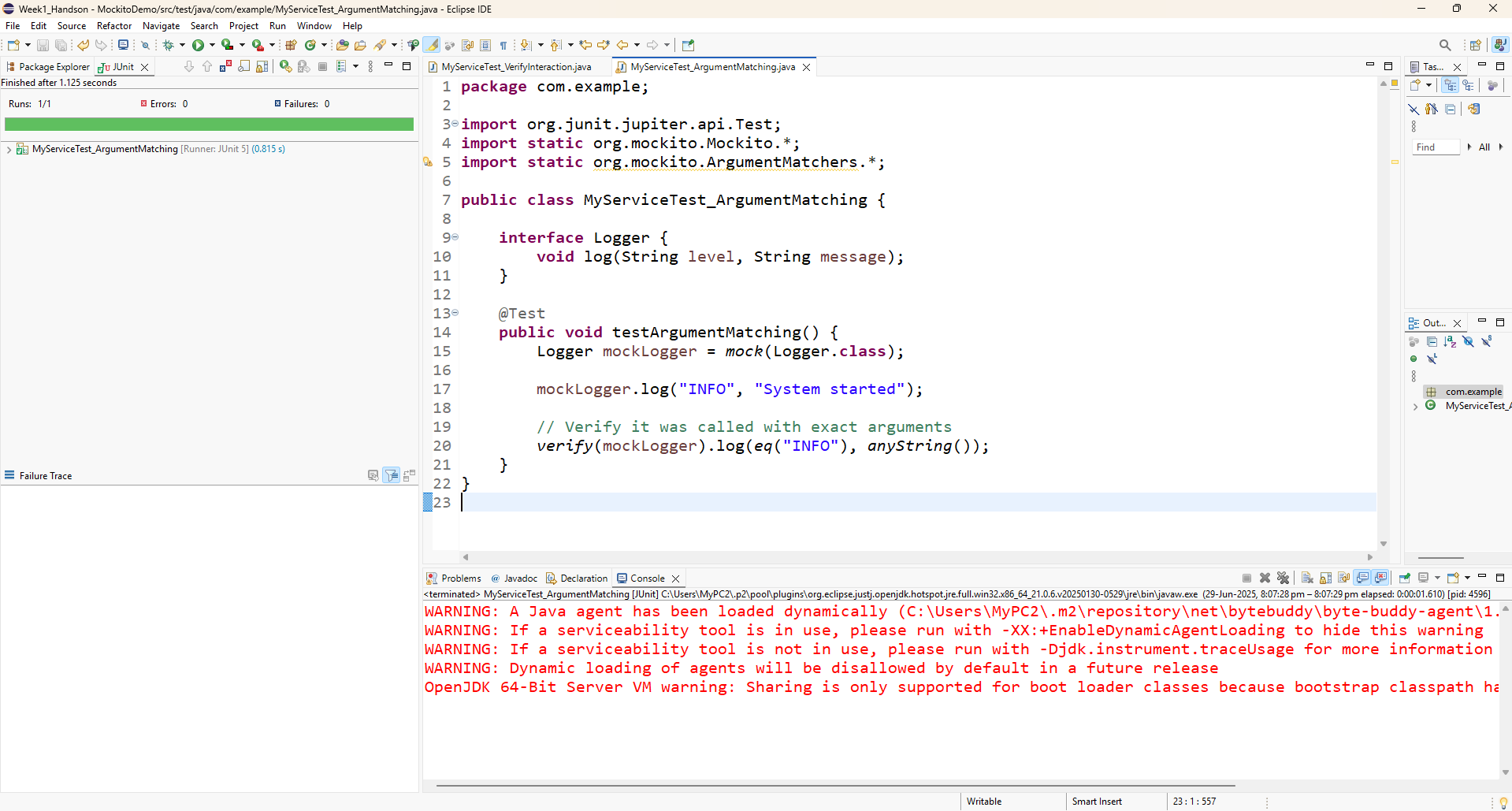
mockLogger.log("INFO", "System started");

// Verify it was called with exact arguments

verify(mockLogger).log(eq("INFO"), anyString());

}

}



**Exercise 4: Handling Void Methods**

MyServiceTest\_VoidMethod.java

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest\_VoidMethod {

interface Notifier {

void send(String message);

}

@Test

public void testVoidMethod() {

Notifier mockNotifier = mock(Notifier.class);

// No need to stub unless you want to suppress side effects

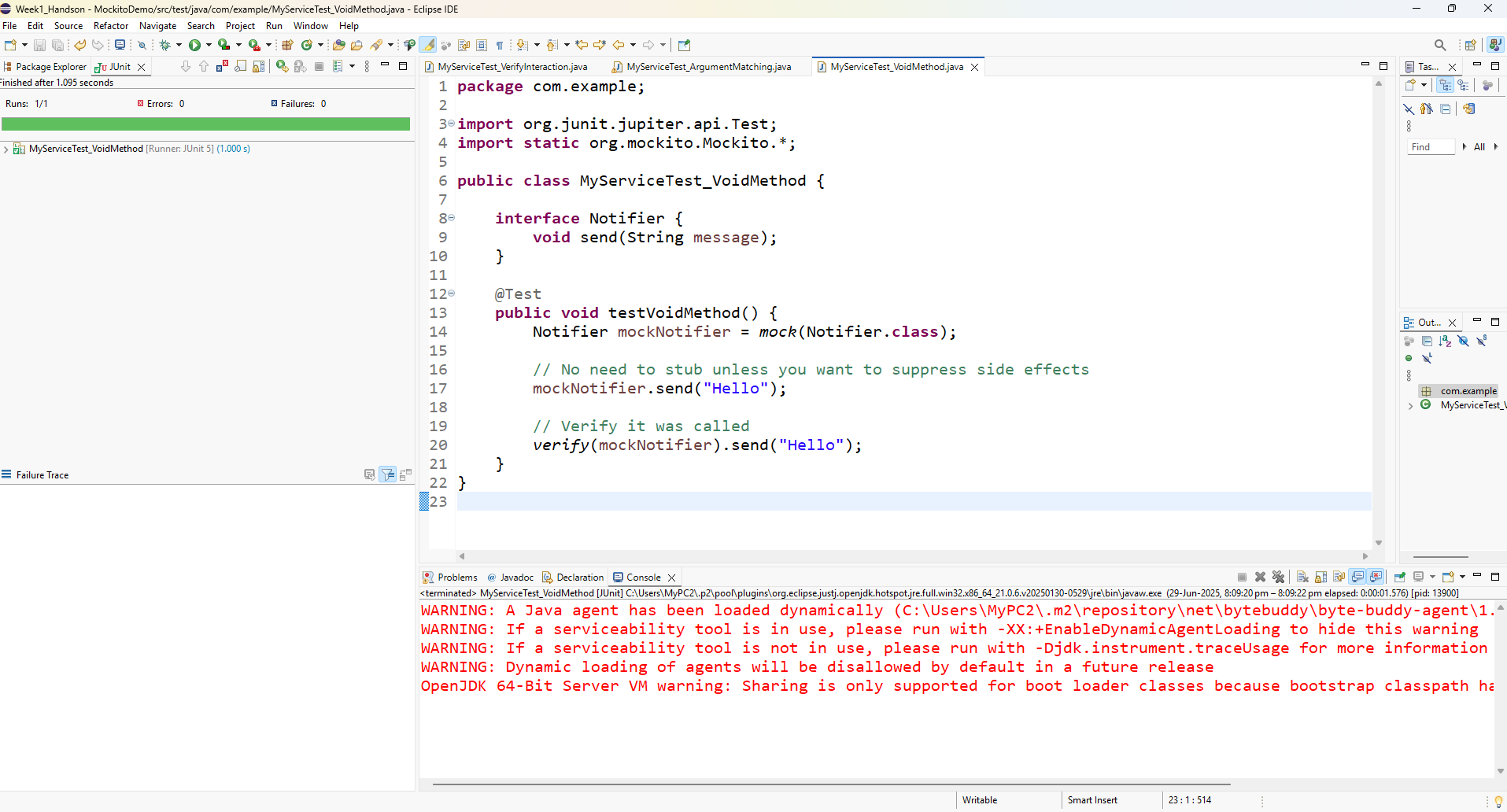
mockNotifier.send("Hello");

// Verify it was called

verify(mockNotifier).send("Hello");

}

}



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

MyServiceTest\_MultipleReturns.java

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class MyServiceTest\_MultipleReturns {

interface ExternalApi {

String getStatus();

}

@Test

public void testMultipleReturns() {

ExternalApi mockApi = mock(ExternalApi.class);

when(mockApi.getStatus())

.thenReturn("Loading")

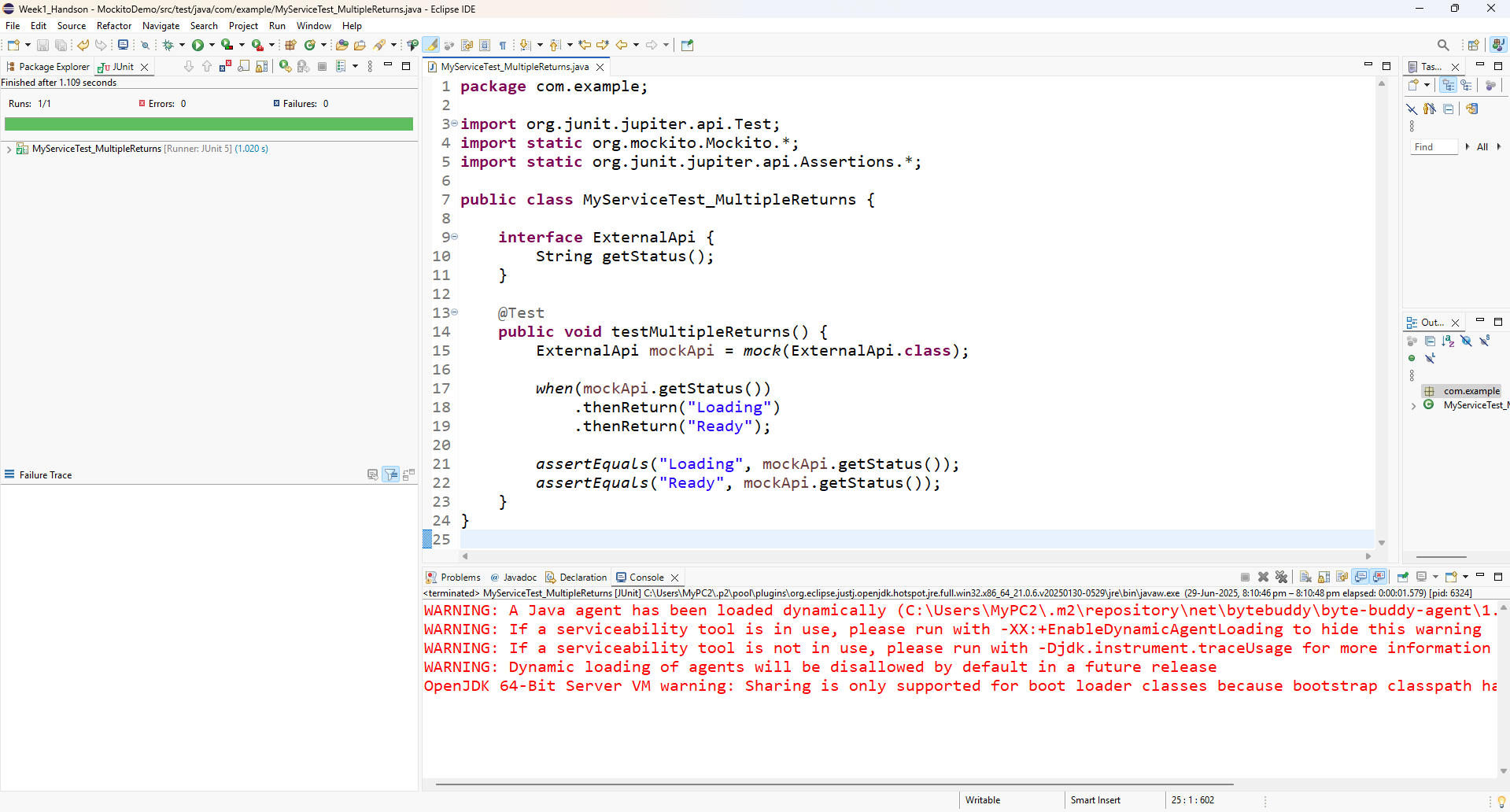
.thenReturn("Ready");

assertEquals("Loading", mockApi.getStatus());

assertEquals("Ready", mockApi.getStatus());

}

}



**Exercise 6: Verifying Interaction Order**

MyServiceTest\_InteractionOrder.java

**package** com.example;

**import** org.junit.jupiter.api.Test;

**import** org.mockito.InOrder;

**import** **static** org.mockito.Mockito.\*;

**public** **class** MyServiceTest\_InteractionOrder {

**interface** Processor {

**void** stepOne();

**void** stepTwo();

}

@Test

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

Processor mockProcessor = *mock*(Processor.**class**);

mockProcessor.stepOne();

mockProcessor.stepTwo();

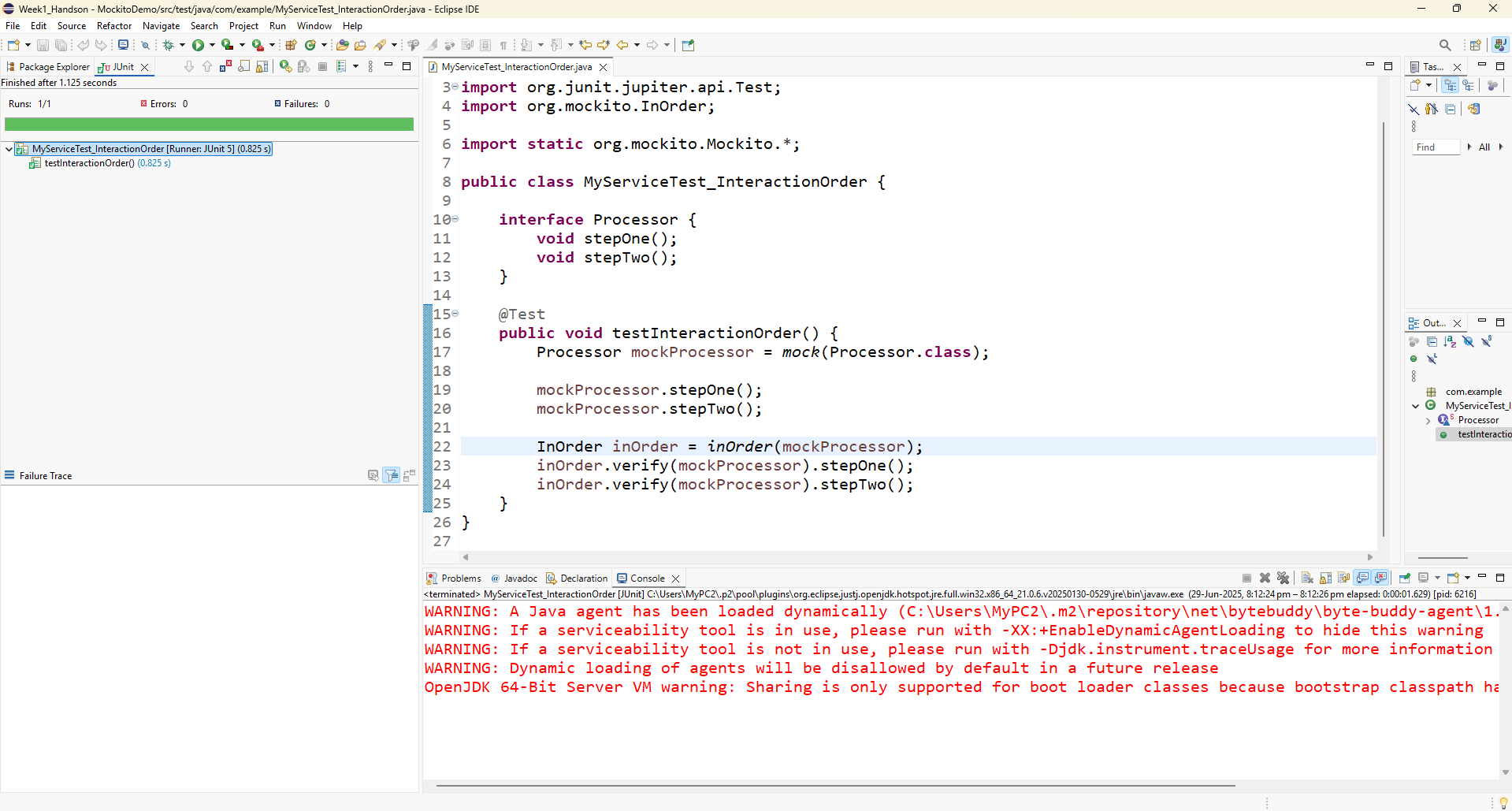
InOrder inOrder = *inOrder*(mockProcessor);

inOrder.verify(mockProcessor).stepOne();

inOrder.verify(mockProcessor).stepTwo();

}

}



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

MyServiceTest\_VoidMethodException.java

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class MyServiceTest\_VoidMethodException {

interface Cleaner {

void clean() throws RuntimeException;

}

@Test

public void testVoidMethodException() {

Cleaner mockCleaner = mock(Cleaner.class);

// Stub void method to throw exception

doThrow(new RuntimeException("Cleaning failed")).when(mockCleaner).clean();

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

mockCleaner.clean();

});

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

verify(mockCleaner).clean();

}

}

