**WEEK 2**

**TDD USING JUNIT5 AND MOCKITO**

**MANDATORY**

**JUNIT5 TESTING**

1. Setting Up Junit

**import** org.junit.Test;

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

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

@Test

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

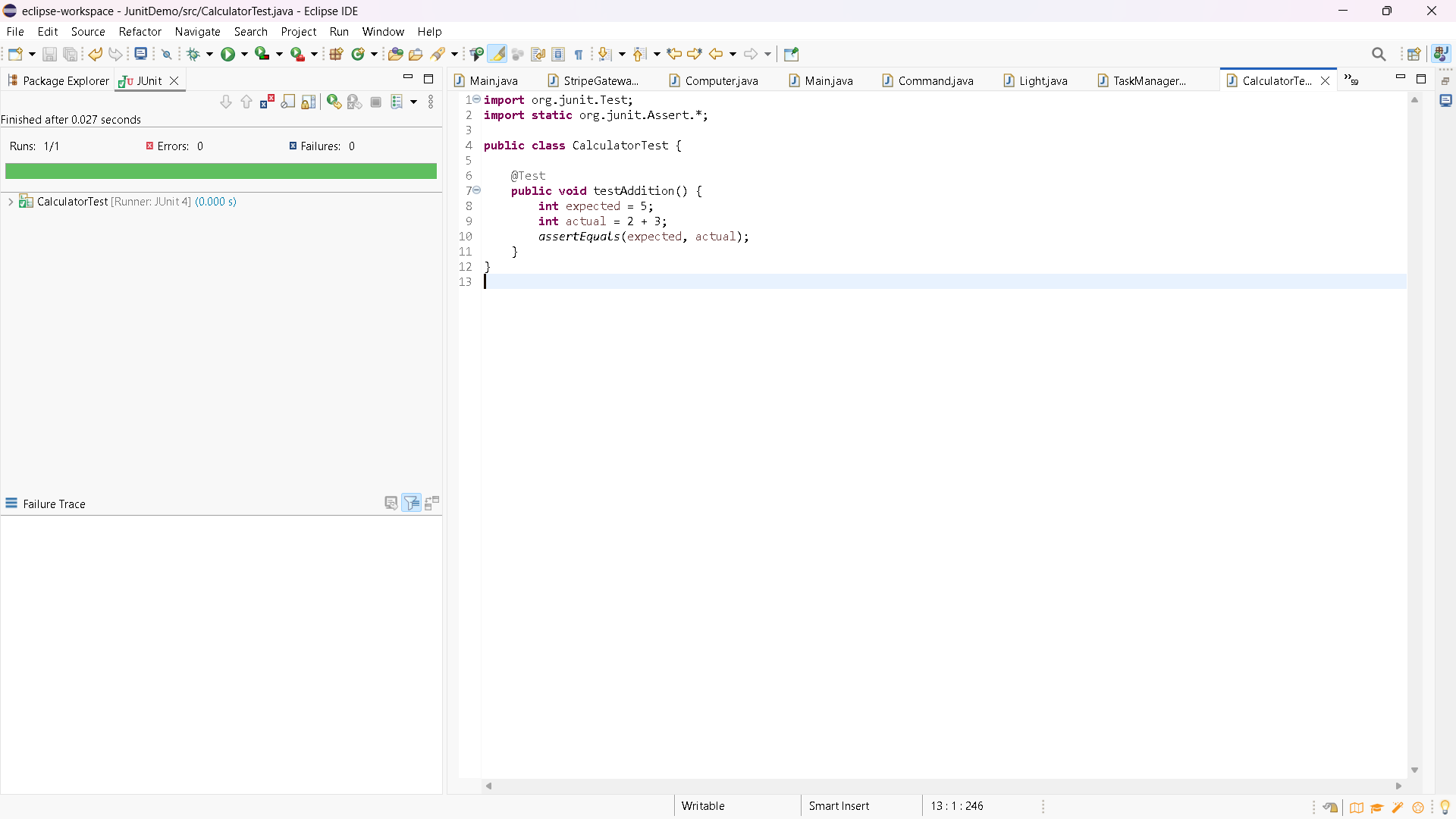
**int** expected = 5;

**int** actual = 2 + 3;

*assertEquals*(expected, actual);

}

}



1. Assertions in Junit

**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);

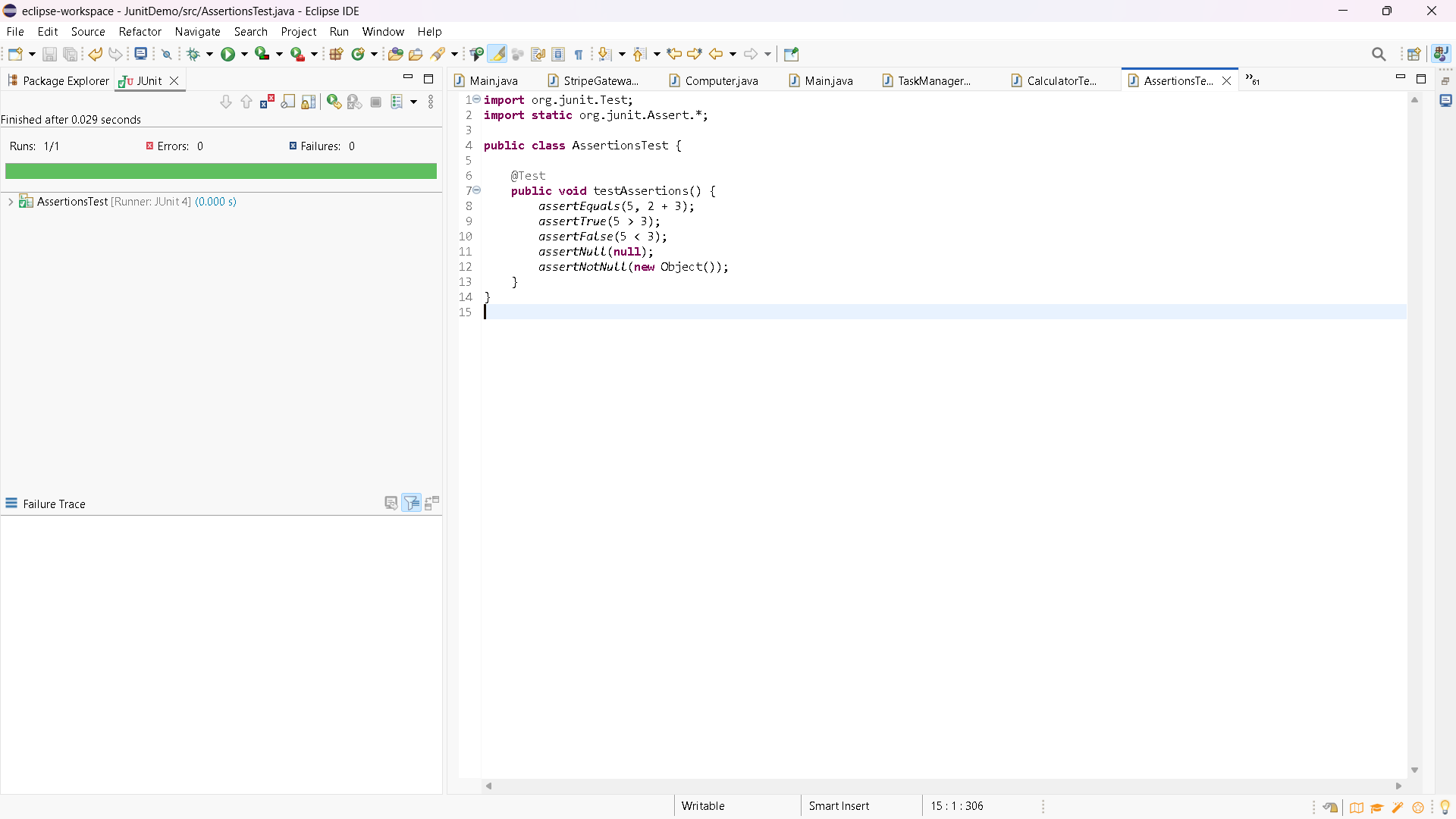
*assertNull*(**null**);

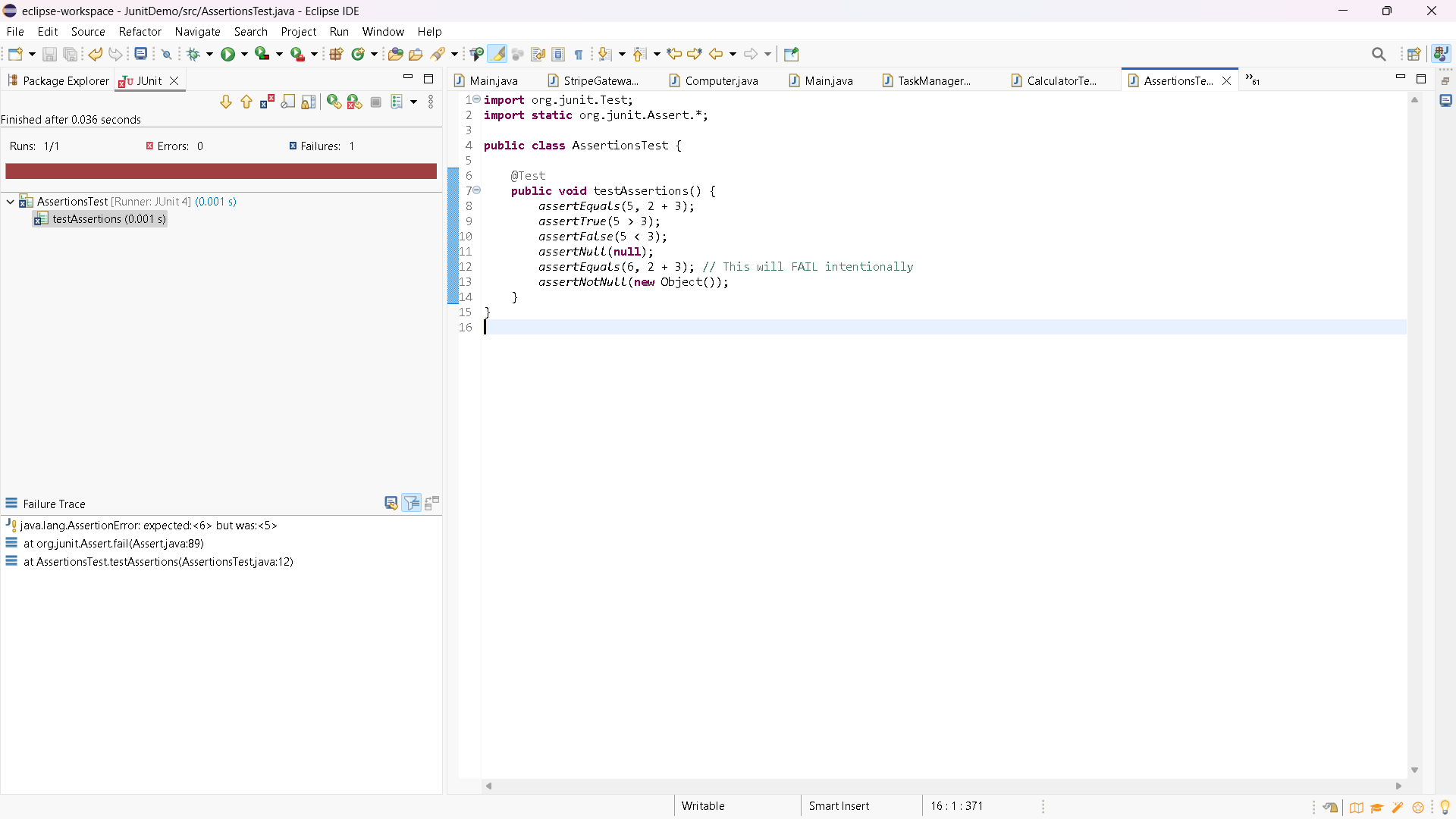
//assertEquals(6, 2 + 3); // This will FAIL intentionally

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

}

}





1. AAA Pattern, Test Fixtures, Setup and Teardown Methods in Junit

Calculator.java

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

// Adds two integers

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

**return** a + b;

}

// Subtracts b from a

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

**return** a - b;

}

// Multiplies two integers

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

**return** a \* b;

}

// Divides a by b

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

**if** (b == 0) {

**throw** **new** ArithmeticException("Cannot divide by zero");

}

**return** a / b;

}

// Optional: Reset method for teardown

**public** **void** reset() {

// Nothing to reset in this simple example, but method is here for structure

}

}

CalculatorTest.java

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

**import** org.junit.\*;

**import** org.junit.Test;

**public** **class** CalculatorTest1 {

**private** Calculator calculator;

@Before

**public** **void** setUp() {

calculator = **new** Calculator();

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

}

@After

**public** **void** tearDown() {

calculator.reset();

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

}

@Test

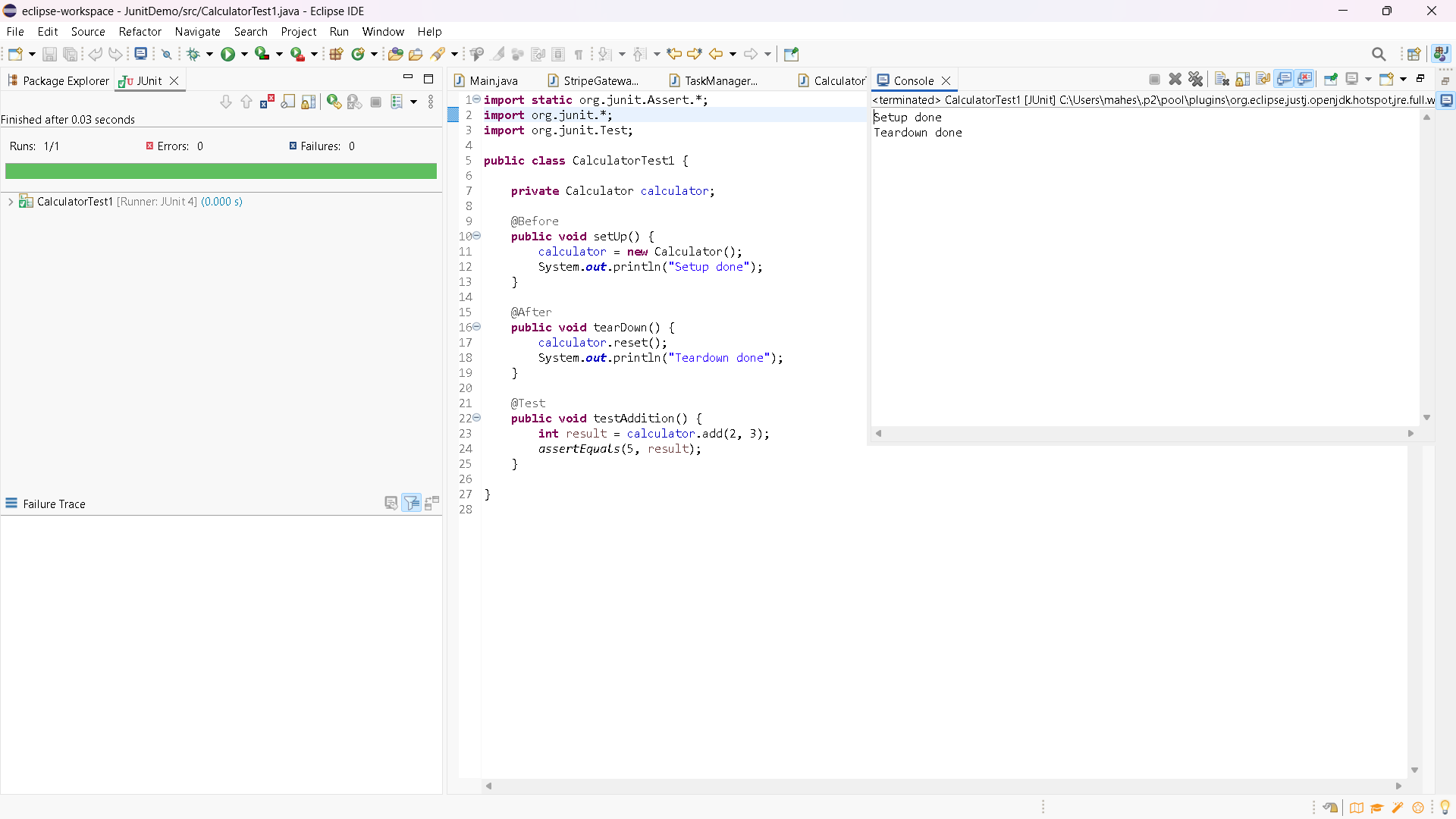
**public** **void** testAddition() {

**int** result = calculator.add(2, 3);

*assertEquals*(5, result);

}

}



**MOCKITO EXERCISES**

1. Mocking and Stubbing

ExternalApi.java(Interface)

**package** myproject;

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

String getData();

}

MyService.java

**package** myproject;

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

**private** ExternalApi api;

**public** MyService(ExternalApi api) {

**this**.api = api;

}

**public** String fetchData() {

**return** api.getData();

}

}

MyServiceTest.java

**package** myproject;

**import** org.junit.Before;

**import** org.junit.Test;

**import** org.mockito.InjectMocks;

**import** org.mockito.Mock;

**import** org.mockito.MockitoAnnotations;

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

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

**public** **class** MyServiceTest {

@Mock

**private** ExternalApi mockApi;

@InjectMocks

**private** MyService service;

@Before

**public** **void** setUp() {

MockitoAnnotations.*openMocks*(**this**); // new way

}

@Test

**public** **void** testFetchData() {

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

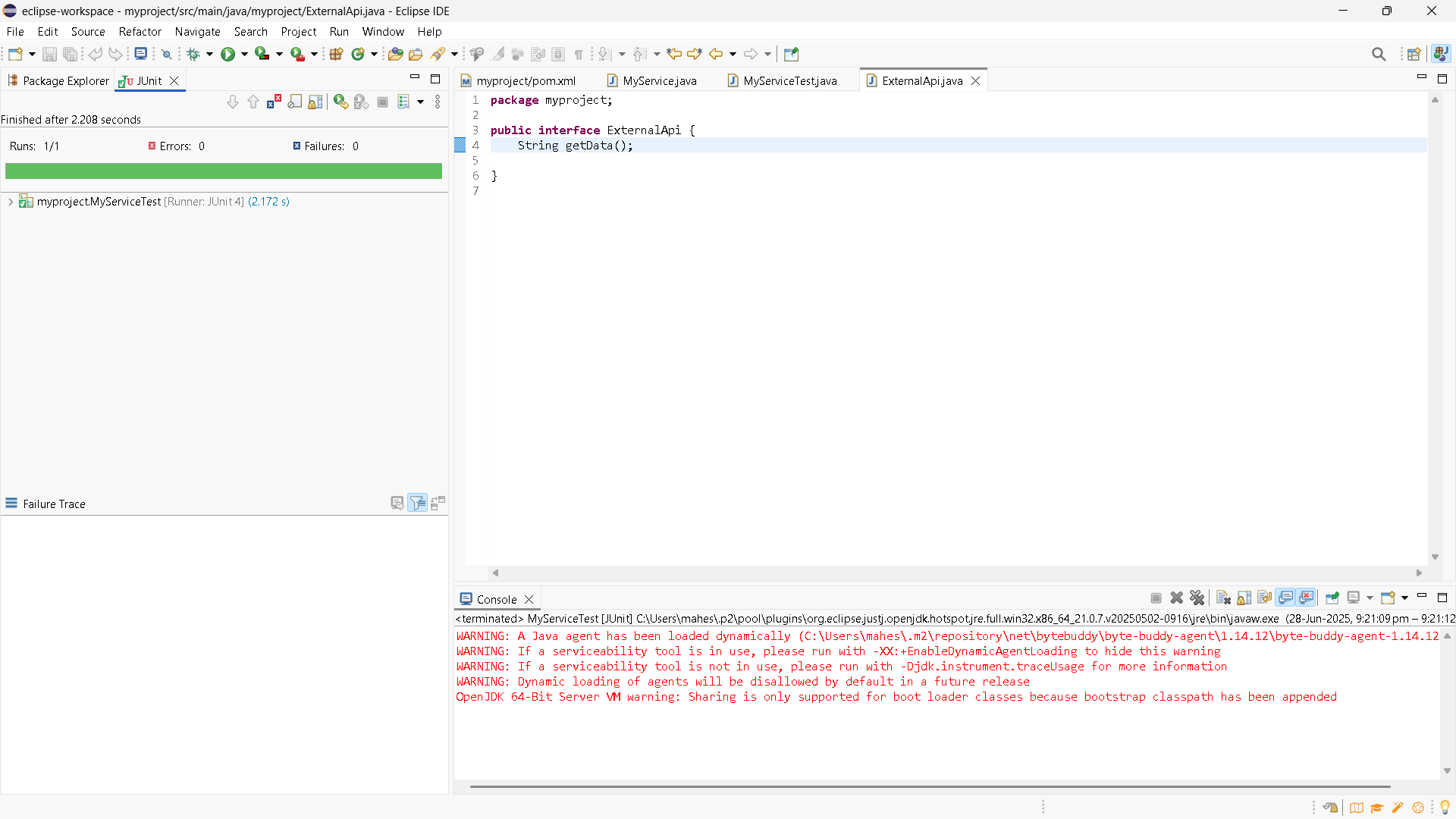
String result = service.fetchData();

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

*verify*(mockApi).getData();

}

}



1. Verifying Interactions

**package** myproject;

**import** org.junit.Test;

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

**public** **class** MyServiceTest {

@Test

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

// Step 1: Create mock object

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

// Step 2: Inject mock into service

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

// Step 3: Call method

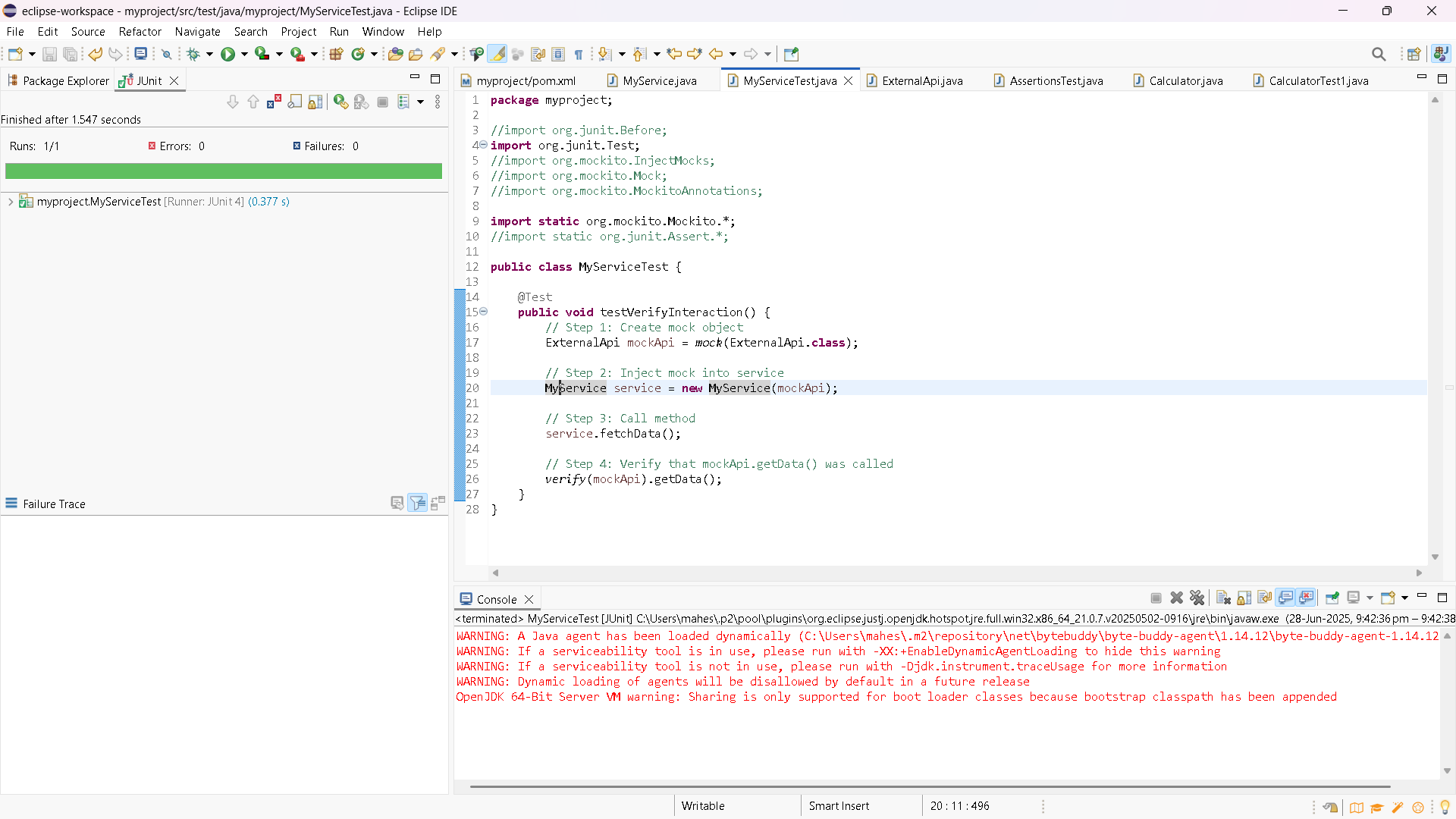
service.fetchData();

// Step 4: Verify that mockApi.getData() was called

*verify*(mockApi).getData();

}

}



**SLFJ LOGGING FRAMEWORK**

1. Logging Error Messages and Warning Levels

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**public** **class** LoggingExample {

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

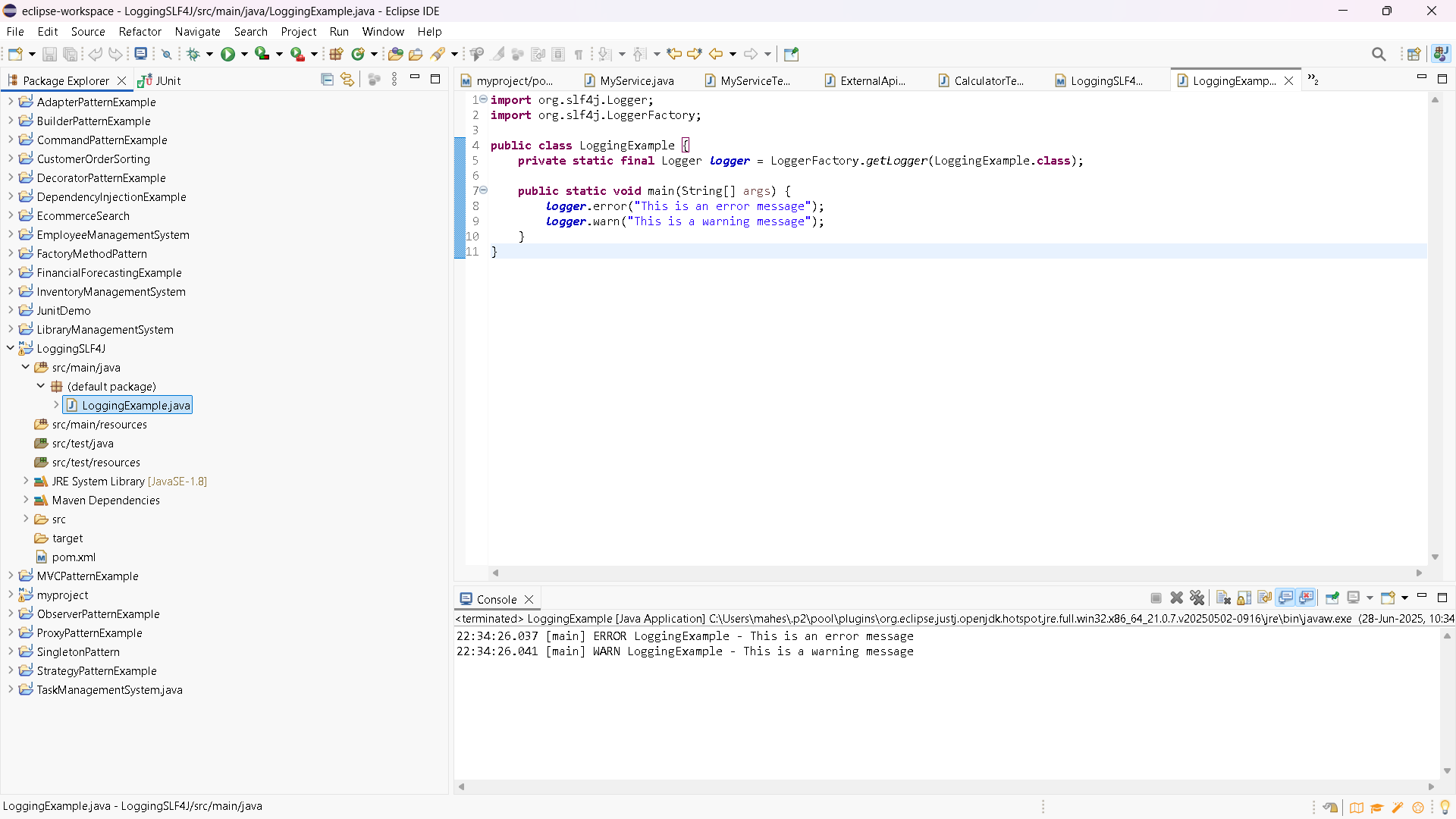
**public** **static** **void** main(String[] args) {

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

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

}

}



**PRACTICE EXERCISES**

**JUNIT TESTING**

1. Parametrized Tests

EvenChecker.java

**public** **class** EvenChecker {

**public** **boolean** isEven(**int** number) {

**return** number % 2 == 0;

}

}

EvenCheckerTest.java

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

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

**import** org.junit.jupiter.params.ParameterizedTest;

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

**public** **class** EvenCheckerTest {

**private** **final** EvenChecker checker = **new** EvenChecker();

@ParameterizedTest

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

**void** testIsEven\_WithEvenNumbers\_ShouldReturnTrue(**int** input) {

*assertTrue*(checker.isEven(input));

}

@ParameterizedTest

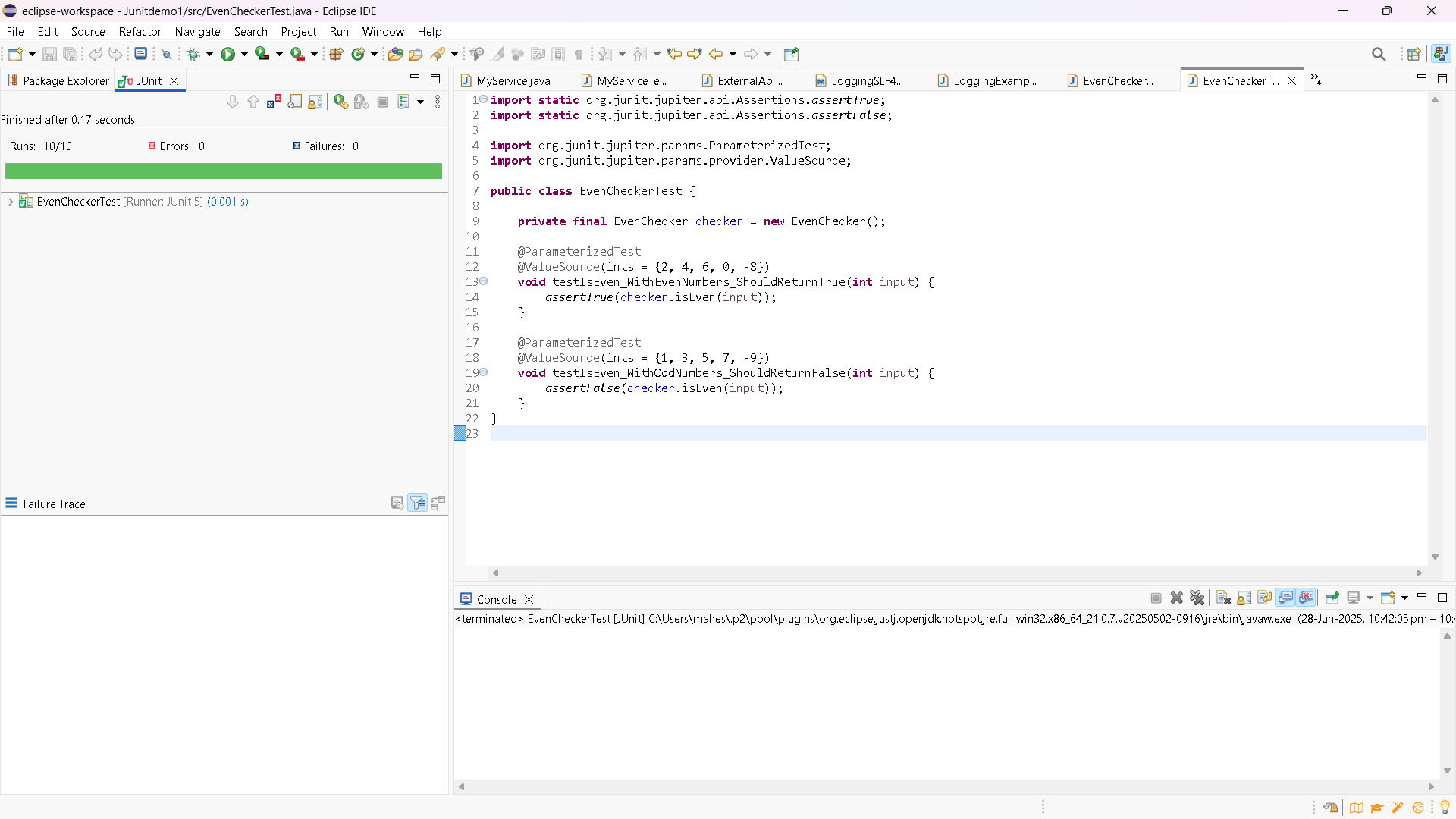
@ValueSource(ints = {1, 3, 5, 7, -9})

**void** testIsEven\_WithOddNumbers\_ShouldReturnFalse(**int** input) {

*assertFalse*(checker.isEven(input));

}

}



1. Test Suites and categories

TestA.java

**package** junitdemo2;

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

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

**public** **class** TestA {

@Test

**void** testA1() {

System.***out***.println("Running TestA.testA1()");

}

}

TestB.java

**package** junitdemo2;

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

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

**public** **class** TestB {

@Test

**void** testB1() {

System.***out***.println("Running TestB.testB1()");

}

}

AllTests.java

**package** junitdemo2;

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

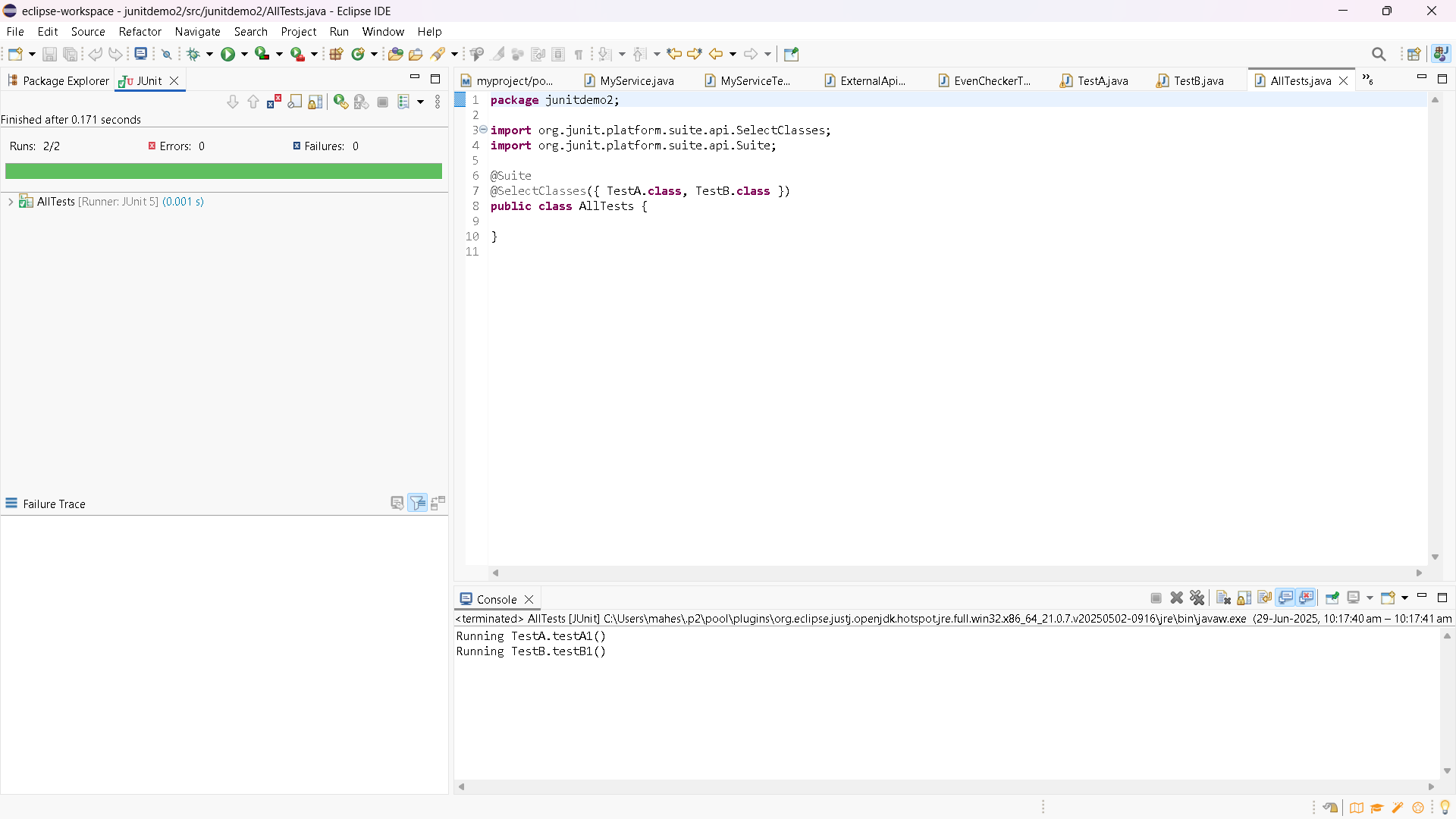
**import** org.junit.platform.suite.api.Suite;

@Suite

@SelectClasses({ TestA.**class**, TestB.**class** })

**public** **class** AllTests {

}



1. Exceptions Testing

ExceptionThrower.java

**package** junitdemo3;

**public** **class** ExceptionThrower {

**public** **void** throwException() {

**throw** **new** IllegalArgumentException("Invalid input");

}

}

ExceptionThrowerTest.java

**package** junitdemo3;

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

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

**class** ExceptionThrowerTest {

@Test

**void** testThrowException() {

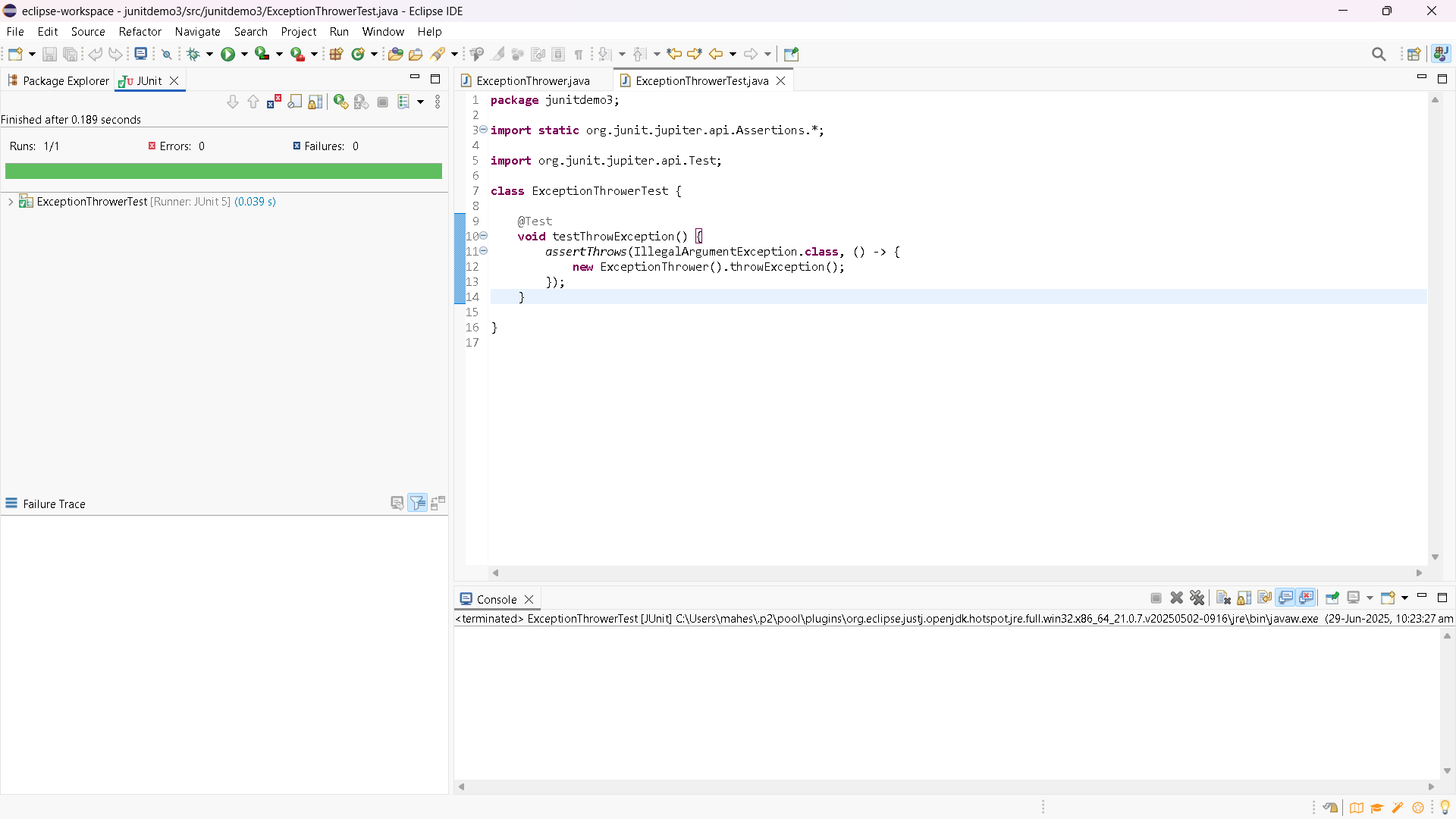
*assertThrows*(IllegalArgumentException.**class**, () -> {

**new** ExceptionThrower().throwException();

});

}

}



1. Test Execution Order

**package** junitdemo4;

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

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

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

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

@TestMethodOrder(MethodOrderer.OrderAnnotation.**class**)

**public** **class** OrderedTests {

@Test

@Order(2)

**void** testSecond() {

System.***out***.println("Running testSecond()");

}

@Test

@Order(1)

**void** testFirst() {

System.***out***.println("Running testFirst()");

}

@Test

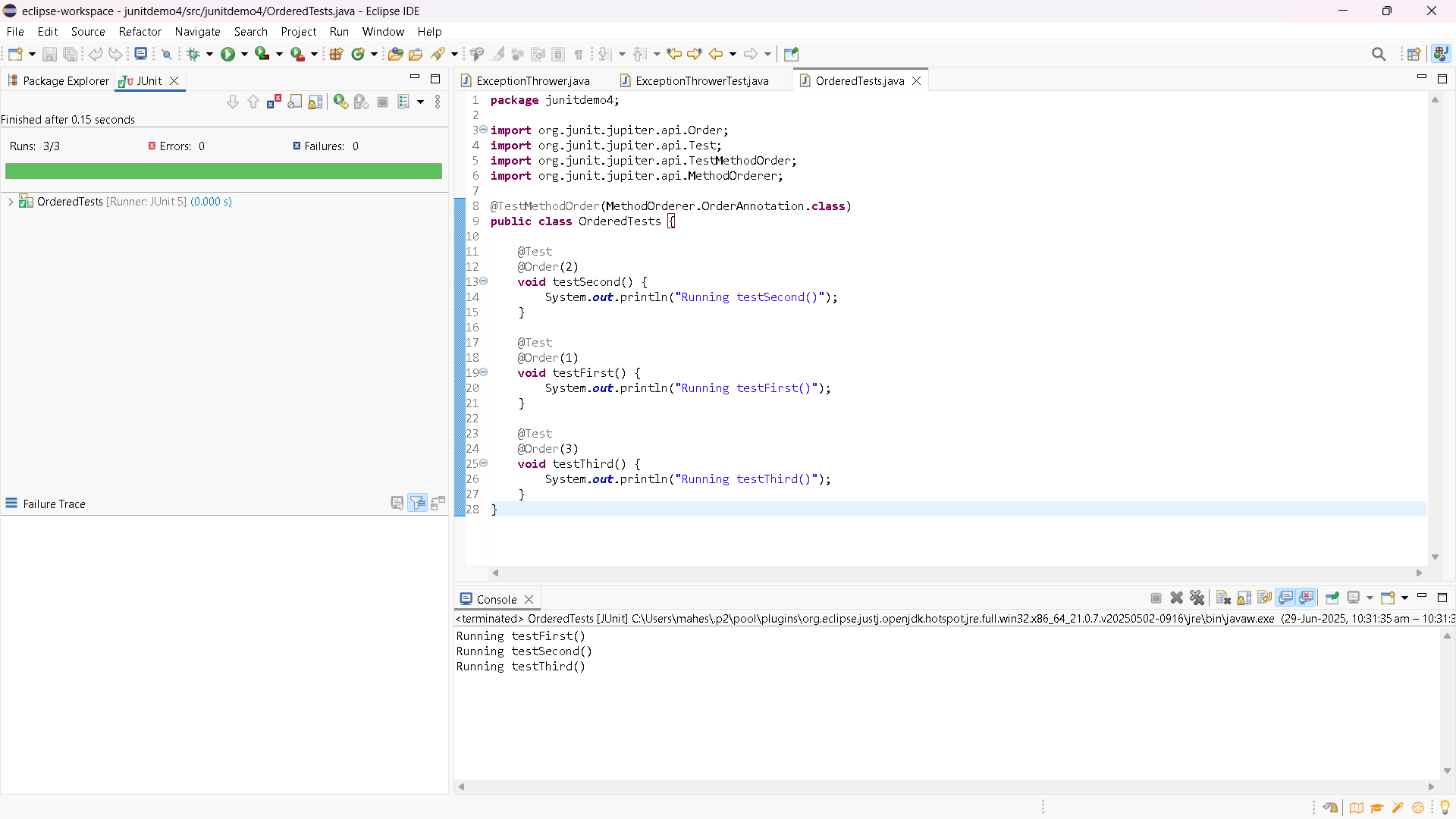
@Order(3)

**void** testThird() {

System.***out***.println("Running testThird()");

}

}



1. Timeout And Performance Testing

PerformanceTester.java

**package** junitdemo5;

**public** **class** PerformanceTester {

**public** **void** performTask() {

**try** {

// Simulate some processing (e.g., database call, computation)

Thread.*sleep*(500); // 500 milliseconds

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

PerformanceTesterTest.java

**package** junitdemo5;

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

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

**import** java.time.Duration;

**public** **class** PerformanceTesterTest {

@Test

**void** testPerformTaskWithinTime() {

PerformanceTester tester = **new** PerformanceTester();

// This test will fail if performTask() takes longer than 1 second

*assertTimeout*(Duration.*ofSeconds*(1), tester::performTask);

}

}

