COGNIZANT

Digital Nurture 4.0

Deep Skilling - Java FSE

WEEK-2 HANDS ON

By Kaviya P

**Advanced JUnit Testing Exercises**

Exercise 1: Parameterized Tests

Scenario:

You want to test a method that checks if a number is even. Instead of writing multiple test

cases, you will use parameterized tests to run the same test with different inputs.

Steps:

1. Create a new Java class `EvenChecker` with a method `isEven(int number)`.

2. Write a parameterized test class `EvenCheckerTest` that tests the `isEven` method with

different inputs.

3. Use JUnit's `@ParameterizedTest` and `@ValueSource` annotations.

**EvenChecker.java**

**package** org.sample.my\_first\_maven;

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

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

{

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

}

}

**EvenCheckerTest.java**

**package** org.sample.my\_first\_maven;

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

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

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

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

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

EvenChecker ev=**new** EvenChecker();

@ParameterizedTest

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

**void** testEventNumbers(**int** num)

{

*assertTrue*(ev.isEven(num));

}

@ParameterizedTest

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

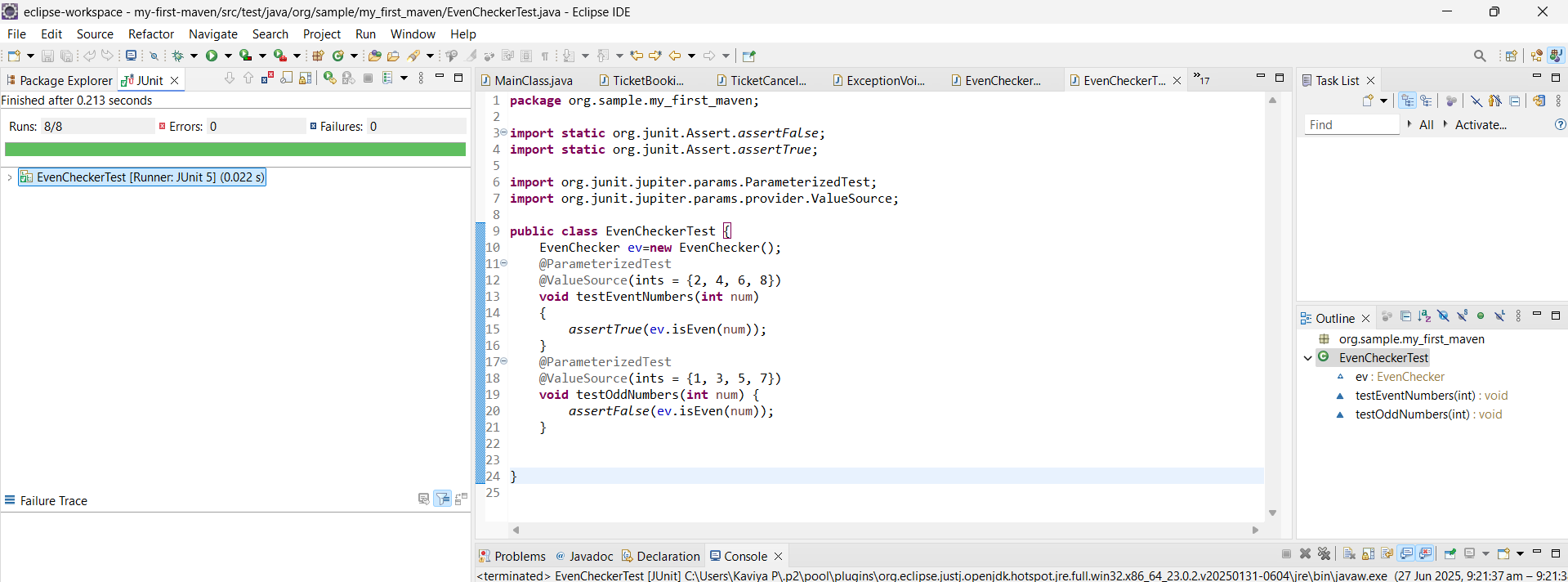
**void** testOddNumbers(**int** num) {

*assertFalse*(ev.isEven(num));

}

}

**OUTPUT:**



**Exercise 2: Test Suites and Categories**

**Scenario:**

**You want to group related tests into a test suite and categorize them.**

**Steps:**

**1. Create a new test suite class `AllTests`.**

**2. Add multiple test classes to the suite.**

**3. Use JUnit's `@Suite` and `@SelectClasses` annotations.**

**TestOne.java**

**package** org.sample.my\_first\_maven;

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

**public** **class** TestOne {

@Test

**void** test1() {

System.***out***.println("Running TestOne.test1");

}

}

**TestTwo.java**

**package** org.sample.my\_first\_maven;

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

**public** **class** TestTwo {

@Test

**void** test2() {

System.***out***.println("Running TestTwo.test2");

}

}

**AllTests.java**

**package** org.sample.my\_first\_maven;

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

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

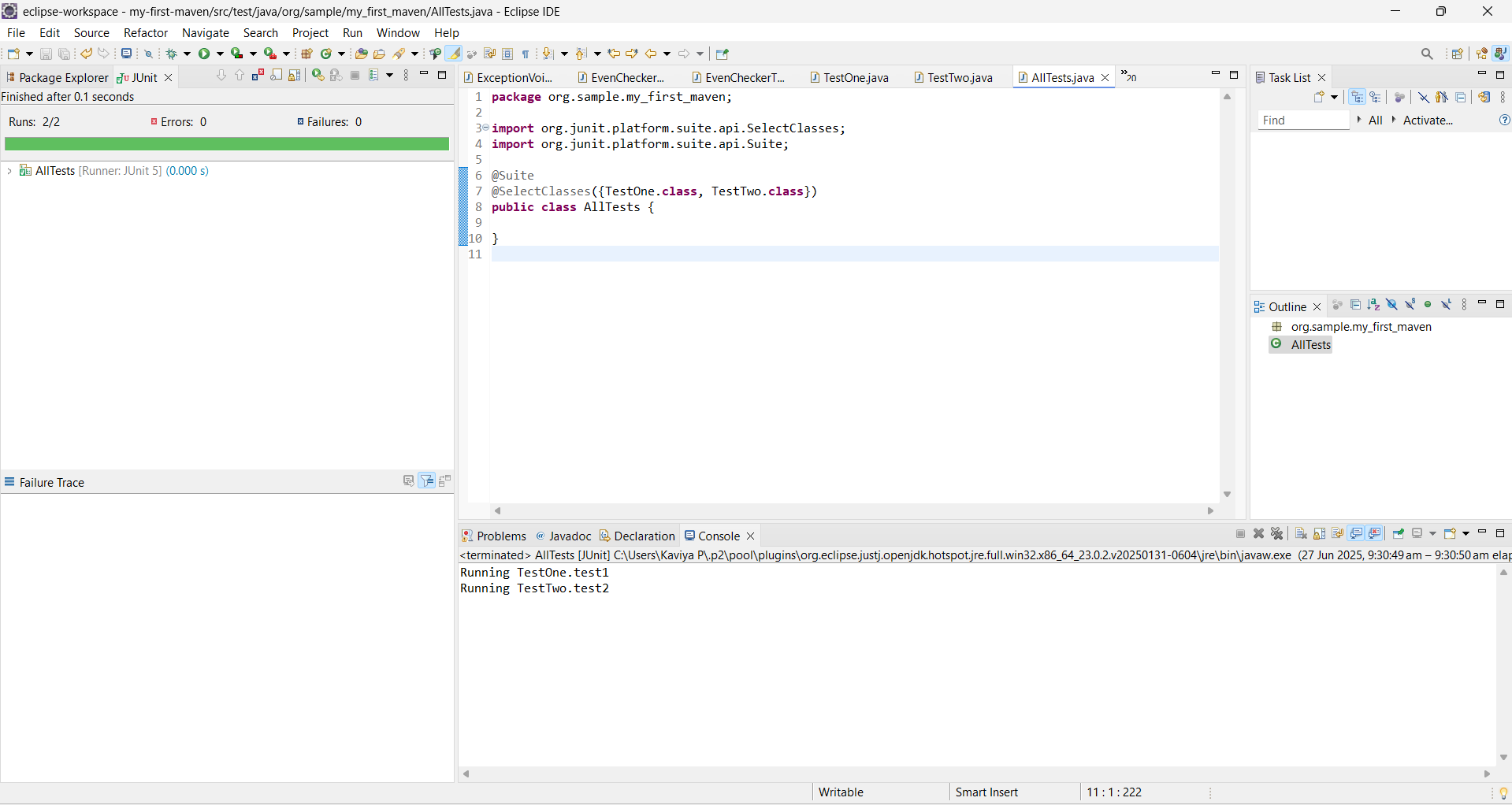
@Suite

@SelectClasses({TestOne.**class**, TestTwo.**class**})

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

}

**OUTPUT:**

****

**Exercise 3: Test Execution Order**

**Scenario:**

**You want to control the order in which tests are executed.**

**Steps:**

**1. Create a test class `OrderedTests`.**

**2. Use JUnit's `@TestMethodOrder` and `@Order` annotations.**

**OrderedTest.java**

**package** org.sample.my\_first\_maven;

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

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

**public** **class** OrderedTest {

@Test

@Order(1)

**void** testFirst() {

System.***out***.println("First test");

}

@Test

@Order(2)

**void** testSecond() {

System.***out***.println("Second test");

}

@Test

@Order(3)

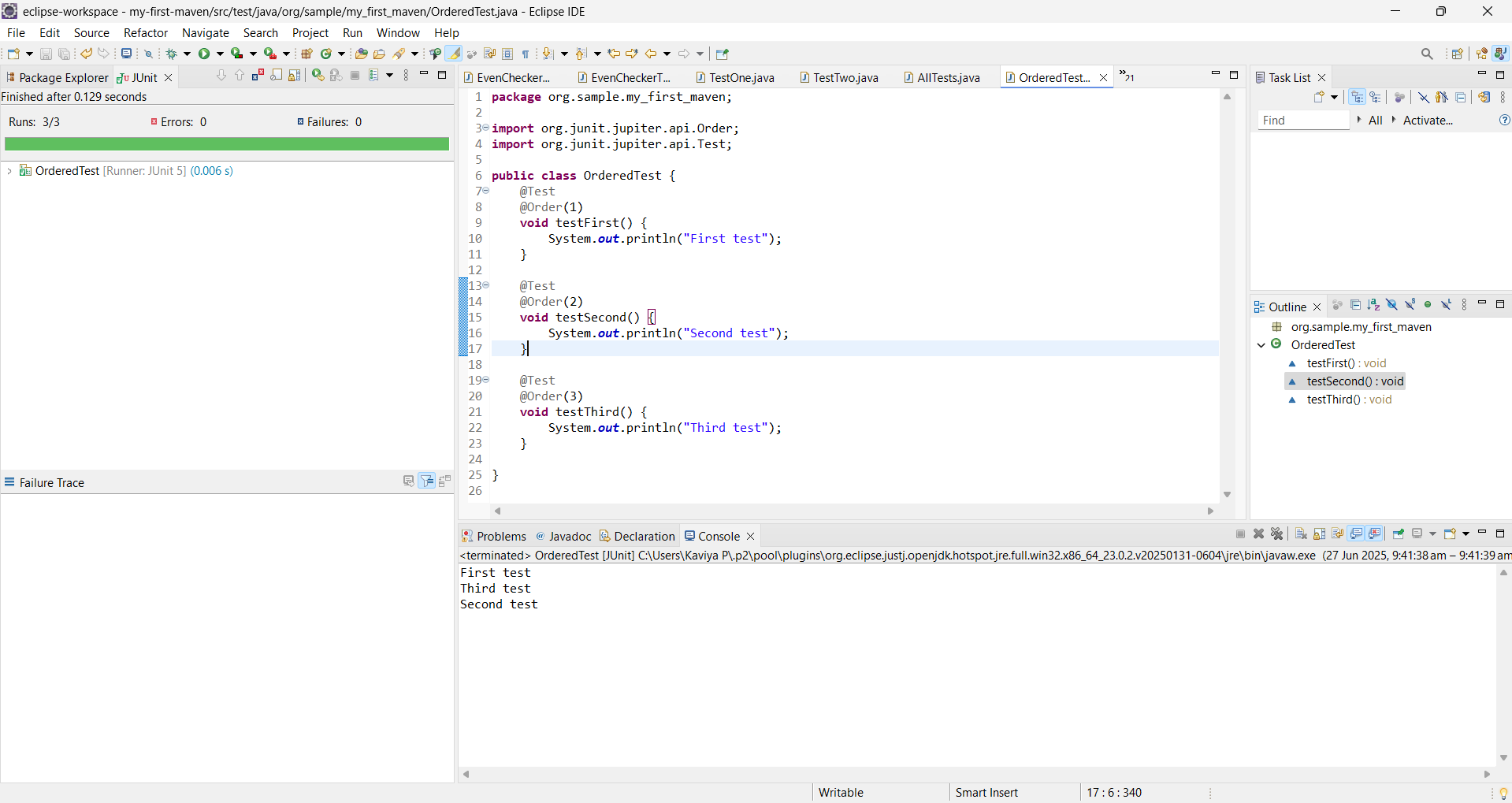
**void** testThird() {

System.***out***.println("Third test");

}

}

**OUTPUT:**

****

**Exercise 4: Exception Testing**

**Scenario:**

**You want to test that a method throws the expected exception.**

**Steps:**

**1. Create a class `ExceptionThrower` with a method `throwException`.**

**2. Write a test class `ExceptionThrowerTest` that tests the method for the expected**

**exception.**

**ExceptionThrower.java**

**package** org.sample.my\_first\_maven;

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

**public** **void** throwException() **throws** IllegalArgumentException {

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

}

}

**ExceptionThrowerTest.java**

**package** org.sample.my\_first\_maven;

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

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

**public** **class** ExceptionThrowerTest {

@Test

**void** testThrowException() {

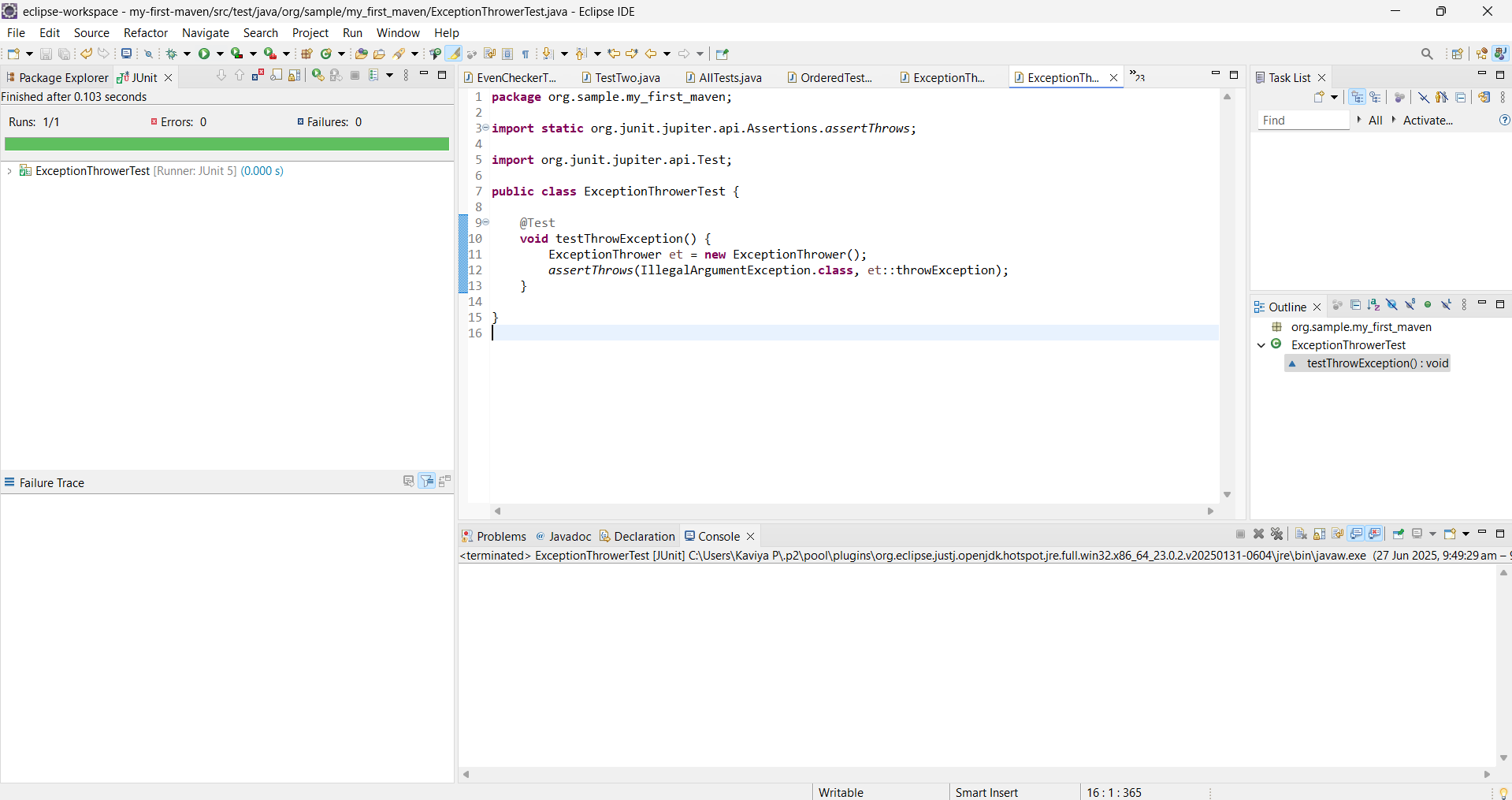
ExceptionThrower et = **new** ExceptionThrower();

*assertThrows*(IllegalArgumentException.**class**, et::throwException);

}

}

**OUTPUT:**

****

**Exercise 5: Timeout and Performance Testing**

**Scenario:**

**You want to ensure that a method completes within a specified time limit.**

**Steps:**

**1. Create a class `PerformanceTester` with a method `performTask`.**

**2. Write a test class `PerformanceTesterTest` that tests the method for timeout.**

**PerformanceTester.java**

**package** org.sample.my\_first\_maven;

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

**public** **void** performTask() **throws** InterruptedException {

Thread.*sleep*(100); // Simulates task

}

}

**PerformanceTesterTest.java**

**package** org.sample.my\_first\_maven;

**import** java.util.concurrent.TimeUnit;

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

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

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

@Test

@Timeout(value = 500, unit = TimeUnit.***MILLISECONDS***)

**void** testPerformTaskWithinTime() **throws** InterruptedException {

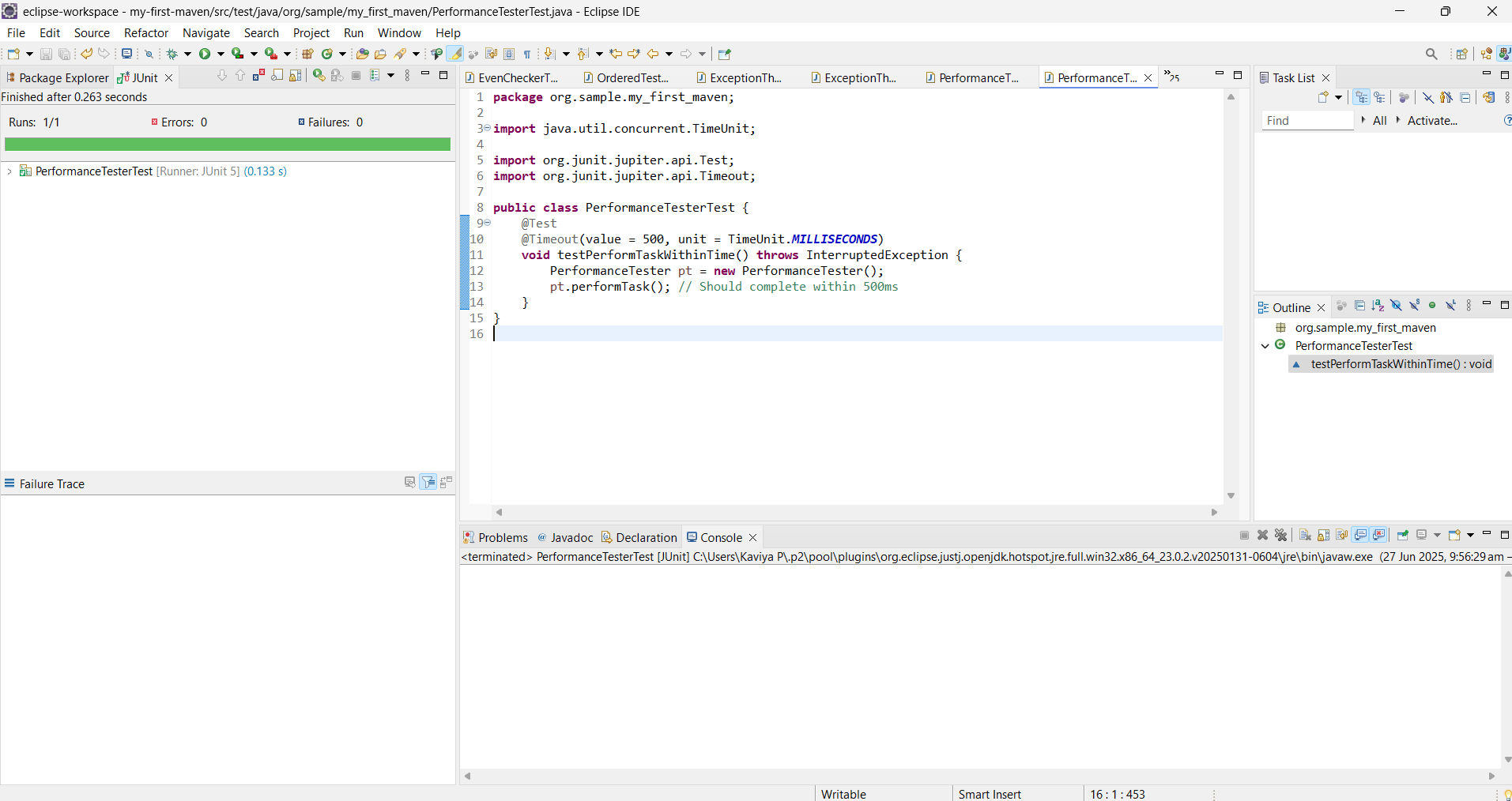
PerformanceTester pt = **new** PerformanceTester();

pt.performTask(); // Should complete within 500ms

}

}

**OUTPUT:**

****