**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.

Program:

EvenChecker.java

package com.example;

public class EvenChecker {

public boolean isEven(int number) {

return number % 2 == 0;

}

}

**EvenCheckerTest.java**

package com.example;

import static org.junit.Assert.\*;

import java.util.Arrays;

import java.util.Collection;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.junit.runners.Parameterized;

*@RunWith*(Parameterized.class)

public class EvenCheckerTest {

private int input;

private boolean expected;

public EvenCheckerTest(int input, boolean expected) {

this.input = input;

this.expected = expected;

}

*@*Parameterized.*Parameters*

public static Collection<Object[]> data() {

return Arrays.*asList*(new Object[][] {

{2, true},

{3, false},

{4, true},

{7, false},

{0, true},

{-1, false}

});

}

*@Test*

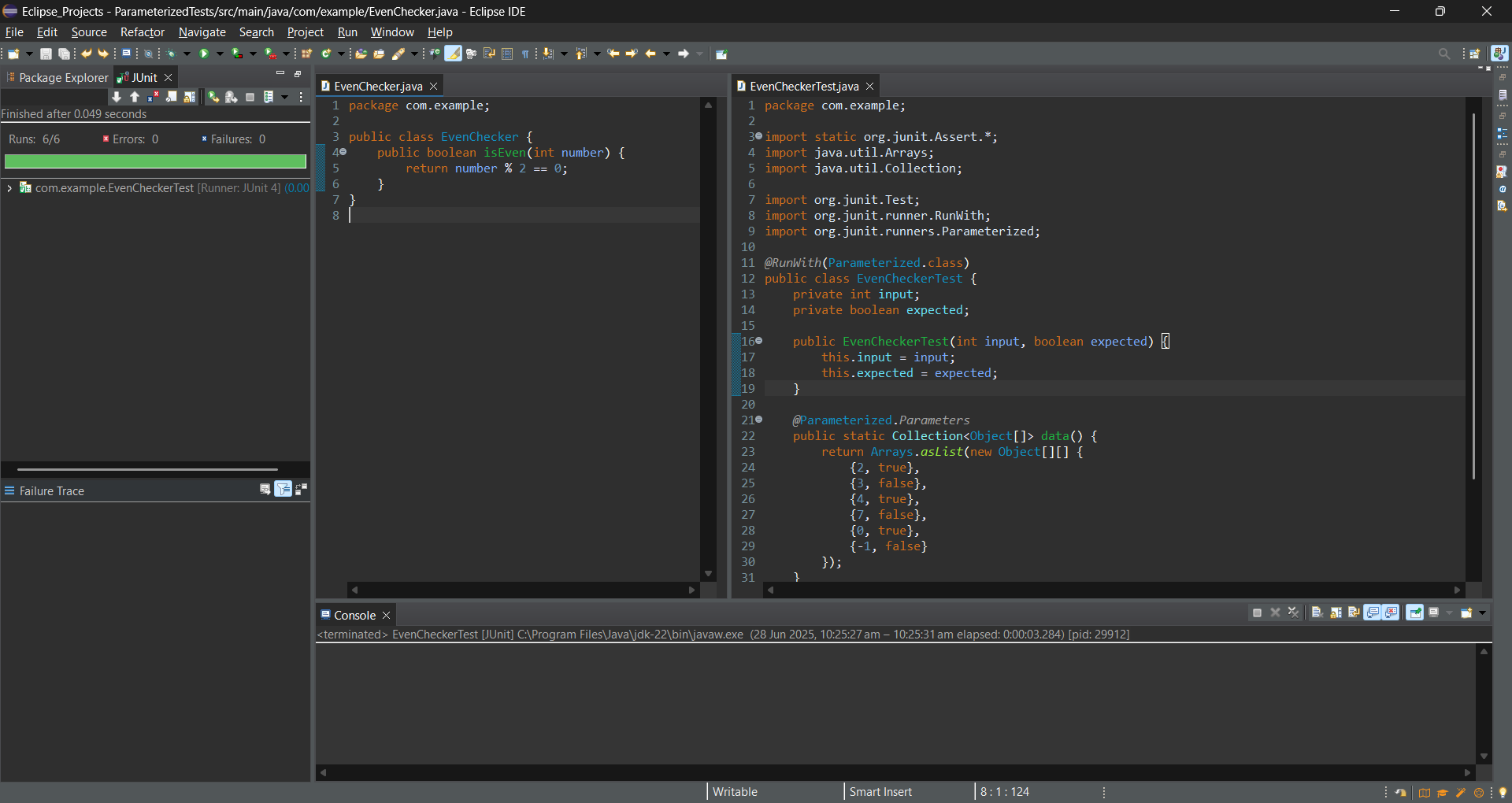
public void testIsEven() {

*assertEquals*(expected, new EvenChecker().isEven(input));

}

}

**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.

**Program:**

CalculatorTest.java

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

*@Test*

public void testAdd() {

*assertEquals*(10, 5 + 5);

}

}

EvenCheckerTest.java

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class EvenCheckerTest {

*@Test*

public void testIsEven() {

*assertTrue*(4 % 2 == 0);

}

}

**AllTests.java**

package com.example;

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

*@RunWith*(Suite.class)

*@*Suite.*SuiteClasses*({

CalculatorTest.class,

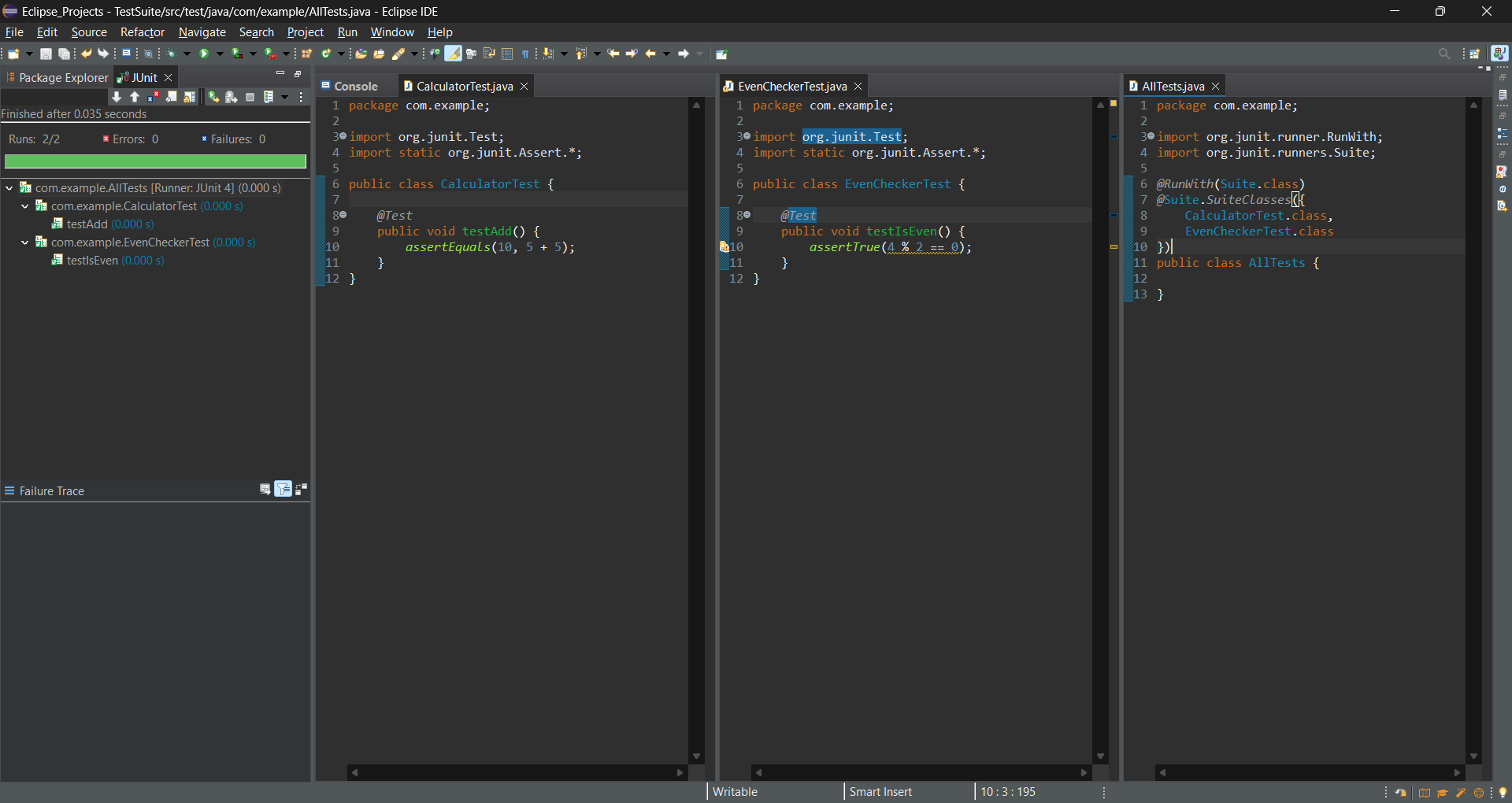
EvenCheckerTest.class

})

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.

**Program:**

OrderedTests.java

package com.example;

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

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

*@TestMethodOrder*(MethodOrderer.OrderAnnotation.class)

public class OrderedTests {

*@Test*

*@Order*(1)

void testA\_First() {

System.***out***.println("Running testA\_First");

*assertTrue*(true);

}

*@Test*

*@Order*(2)

void testB\_Second() {

System.***out***.println("Running testB\_Second");

*assertTrue*(true);

}

*@Test*

*@Order*(3)

void testC\_Third() {

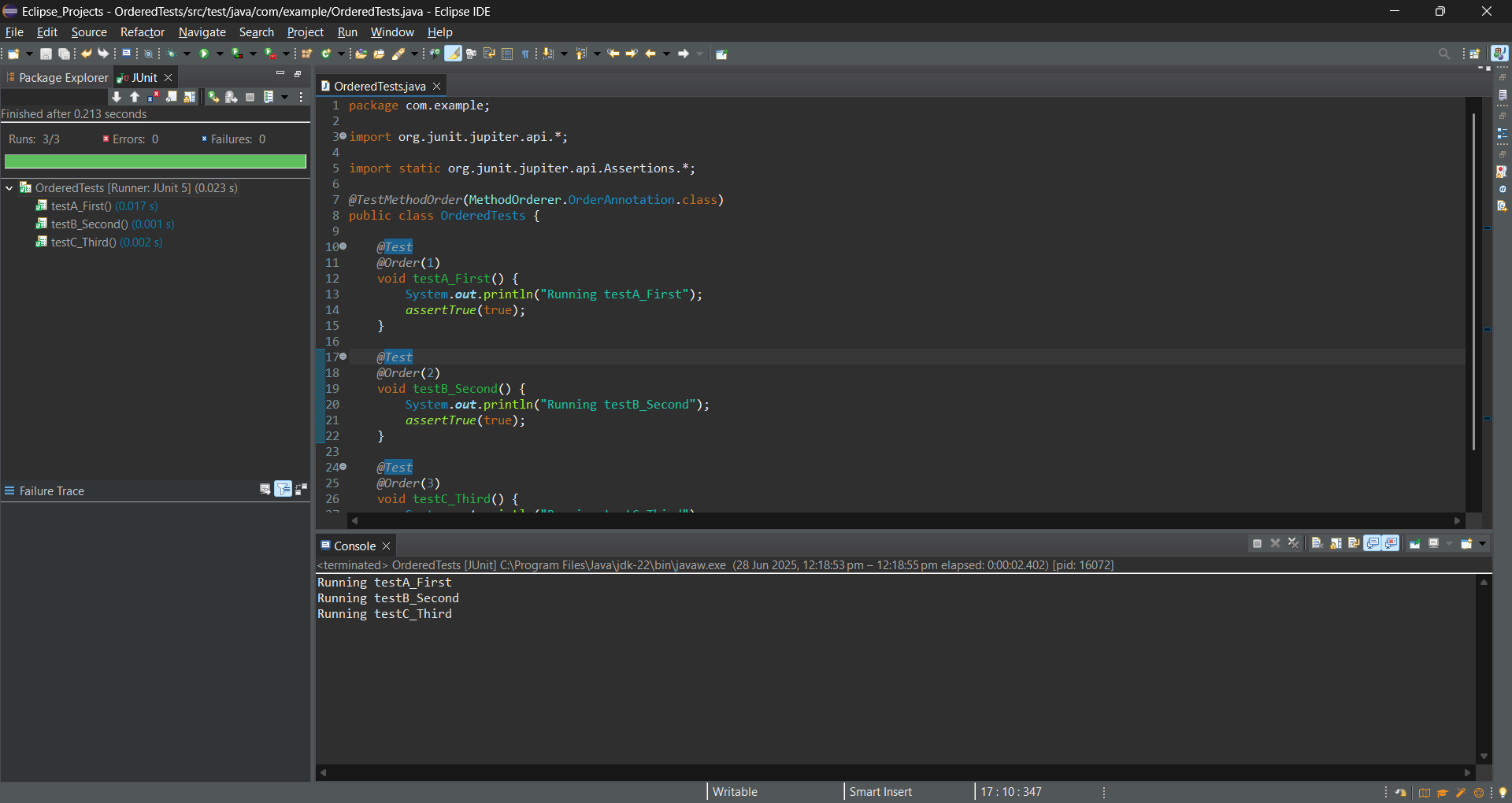
System.***out***.println("Running testC\_Third");

*assertTrue*(true);

}

}

**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.

**Program:**

ExceptionThrower.java

package com.example;

public class ExceptionThrower {

public int divide(int a, int b) {

return a / b;

}

}

ExceptionThrowerTest.java

package com.example;

import org.junit.Test;

public class ExceptionThrowerTest {

*@Test*(expected = ArithmeticException.class)

public void testDivideByZero() {

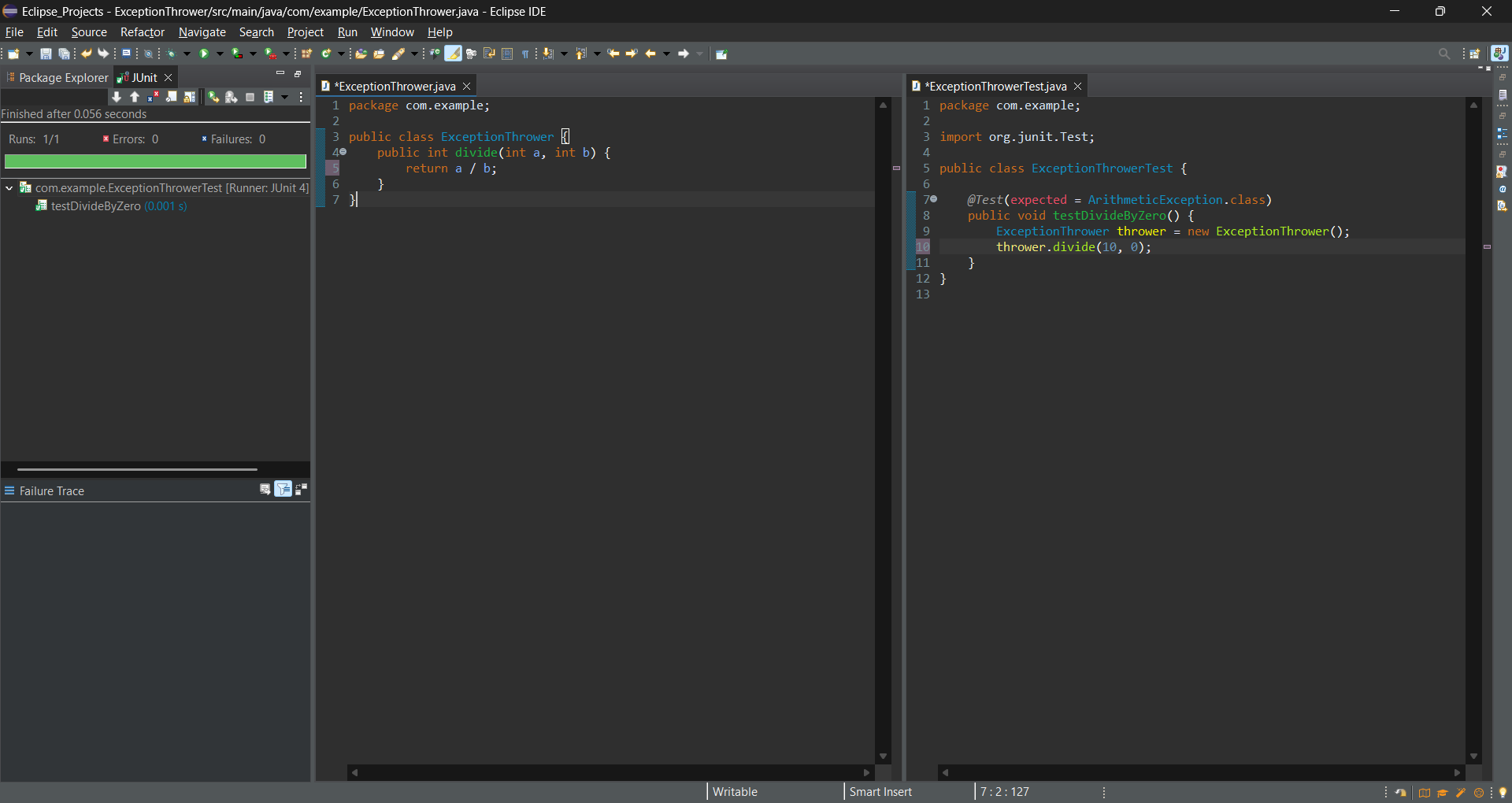
ExceptionThrower thrower = new ExceptionThrower();

thrower.divide(10, 0);

}

}

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.

Program:

PerformanceTester.java

package com.example;

public class PerformanceTester {

public void performTask() throws InterruptedException {

Thread.*sleep*(400);

}

}

PerformanceTesterTest.java

package com.example;

import org.junit.Test;

public class PerformanceTesterTest {

*@Test*(timeout = 500)

public void testPerformTaskCompletesInTime() throws InterruptedException {

PerformanceTester tester = new PerformanceTester();

tester.performTask();

}

}

Output:

