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 com.example.junit\_demo;**

**public class EvenChecker {**

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

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

**}**

**}**

**EvenCheckerTest.java**

package com.example.junit\_demo;

import org.junit.jupiter.params.ParameterizedTest;

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

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

public class EvenCheckerTest {

EvenChecker checker = new EvenChecker();

*@ParameterizedTest*

*@ValueSource*(ints = {2, 4, 6, 8, 10})

void testEvenNumbers(int number) {

System.***out***.println("Testing number: " + number);

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

}

*@ParameterizedTest*

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

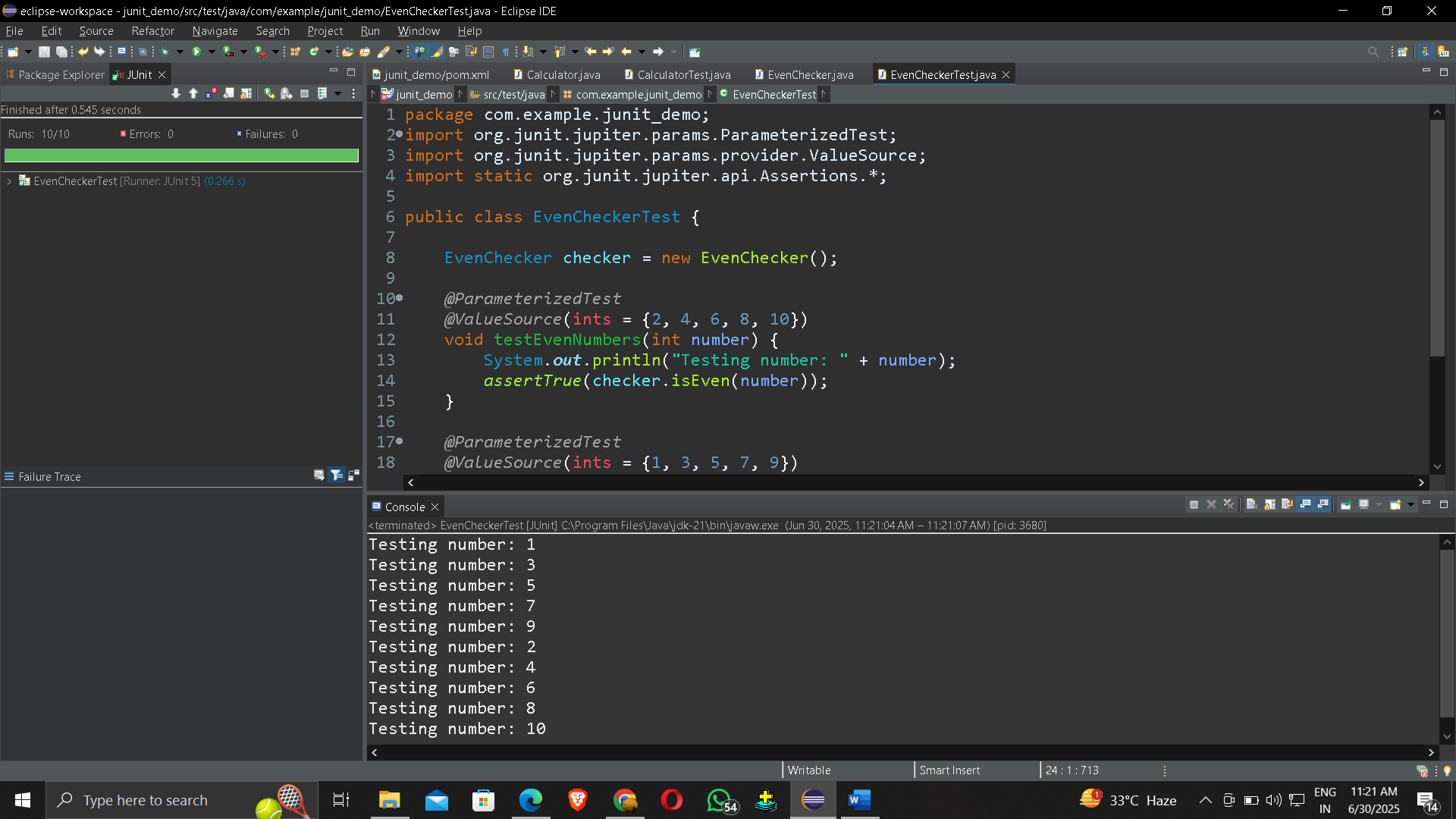
void testOddNumbers(int number) {

System.***out***.println("Testing number: " + number);

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

}

}



# 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.

package com.example.junit\_demo;

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

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

*@Suite*

*@SelectClasses*({

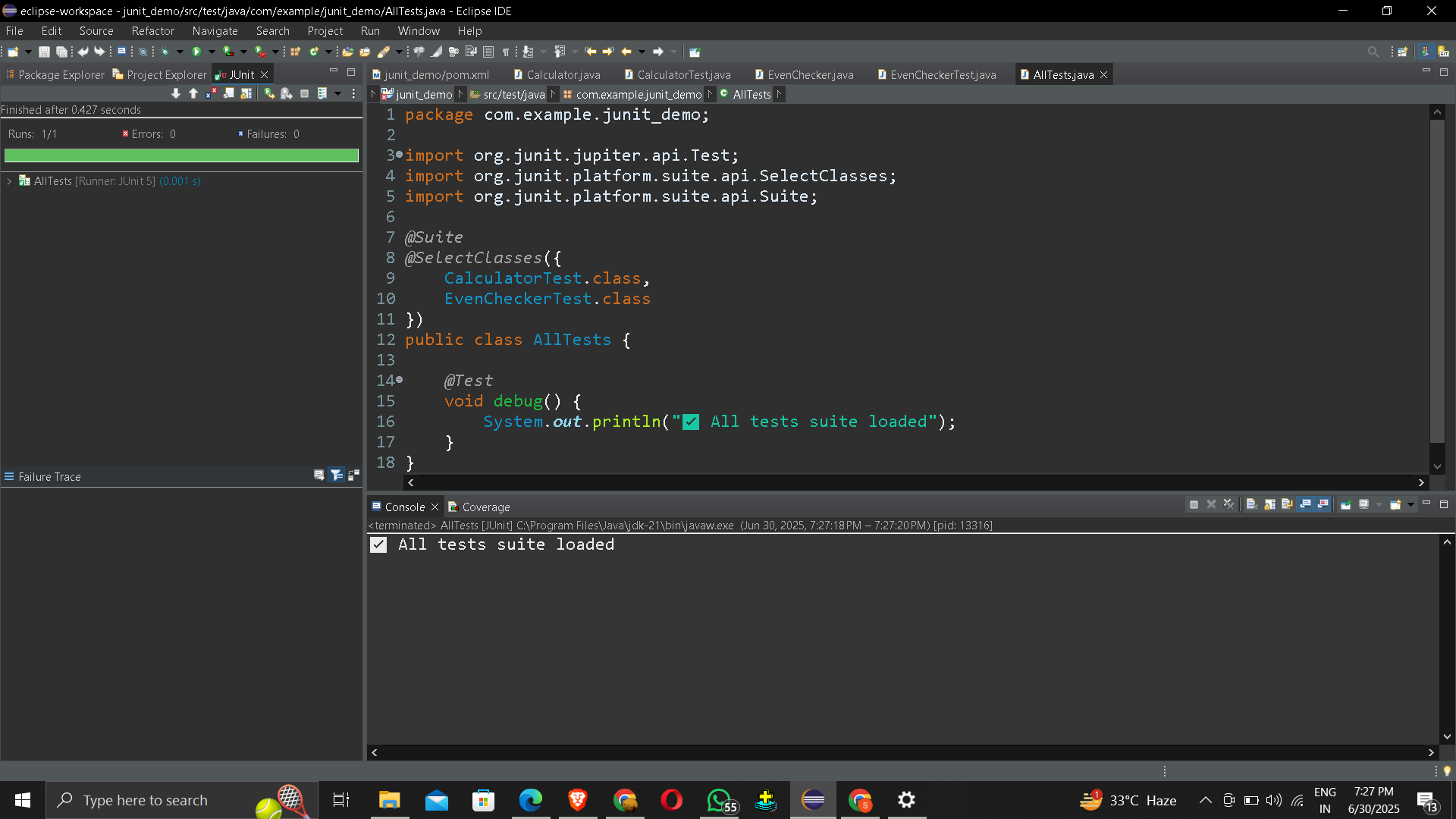
CalculatorTest.class, EvenCheckerTest.class

})

public class AllTests {

// No @Test method here

}



# 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.

package com.example.junit\_demo;

import org.junit.jupiter.api.MethodOrderer;

import org.junit.jupiter.api.Order;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestMethodOrder;

*@TestMethodOrder*(MethodOrderer.OrderAnnotation.class)

public class OrderedTests {

*@Test*

*@Order*(3)

void testC() {

System.***out***.println("Running testC (Order 3)");

}

*@Test*

*@Order*(1)

void testA() {

System.***out***.println("Running testA (Order 1)");

}

*@Test*

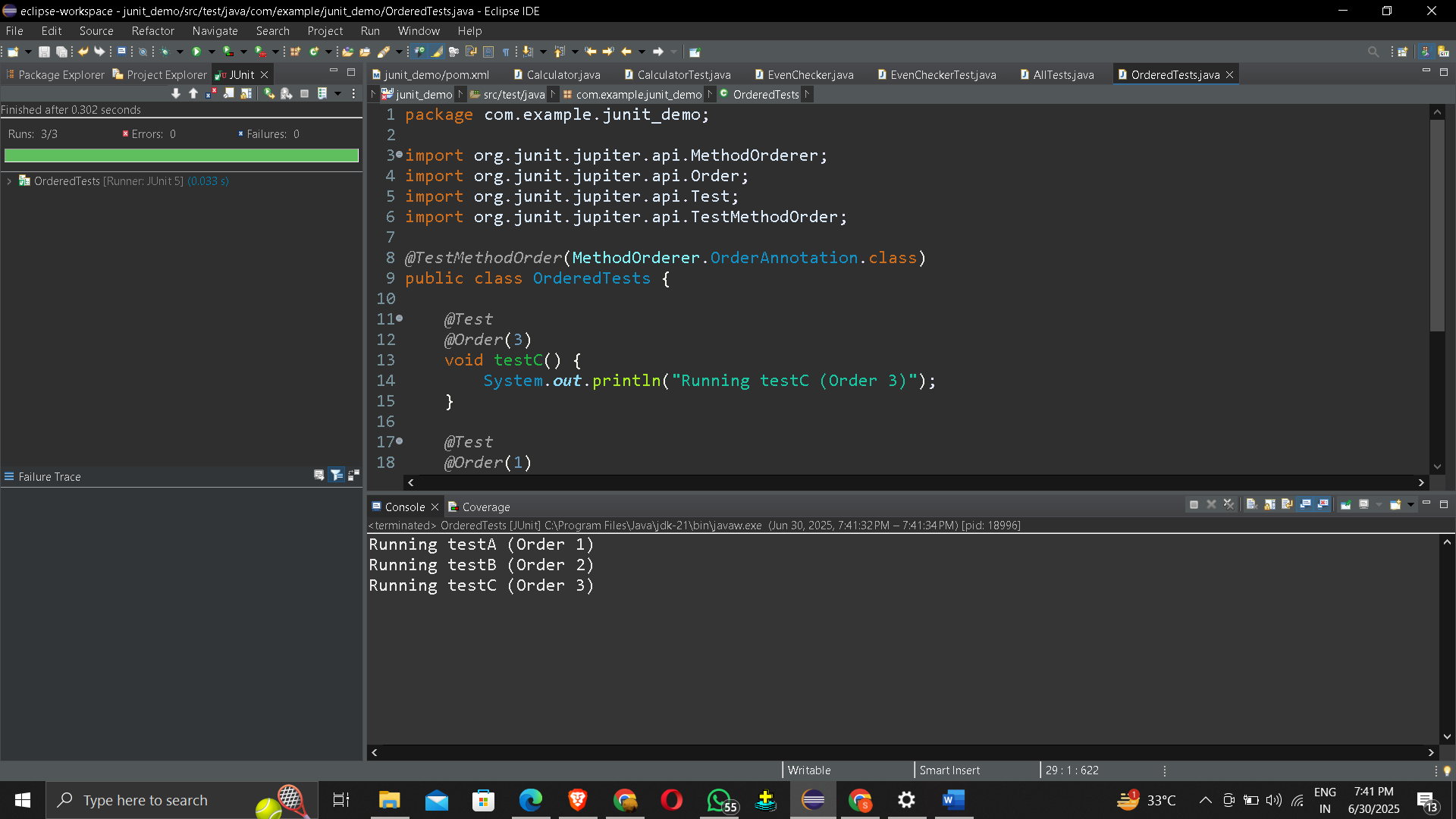
*@Order*(2)

void testB() {

System.***out***.println("Running testB (Order 2)");

}

}



# 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 com.example.junit\_demo;

public class ExceptionThrower {

public void throwException(String input) {

if (input == null || input.isEmpty()) {

throw new IllegalArgumentException("Input cannot be null or empty");

}

System.***out***.println("Valid input: " + input);

}

}

ExceptionThrowerTest.java

package com.example.junit\_demo;

import org.junit.jupiter.api.Test;

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

public class ExceptionThrowerTest {

ExceptionThrower thrower = new ExceptionThrower();

*@Test*

void testThrowExceptionWithNull() {

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

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

thrower.throwException(null);

});

}

*@Test*

void testThrowExceptionWithEmpty() {

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

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

thrower.throwException("");

});

}

*@Test*

void testThrowExceptionWithValidInput() {

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

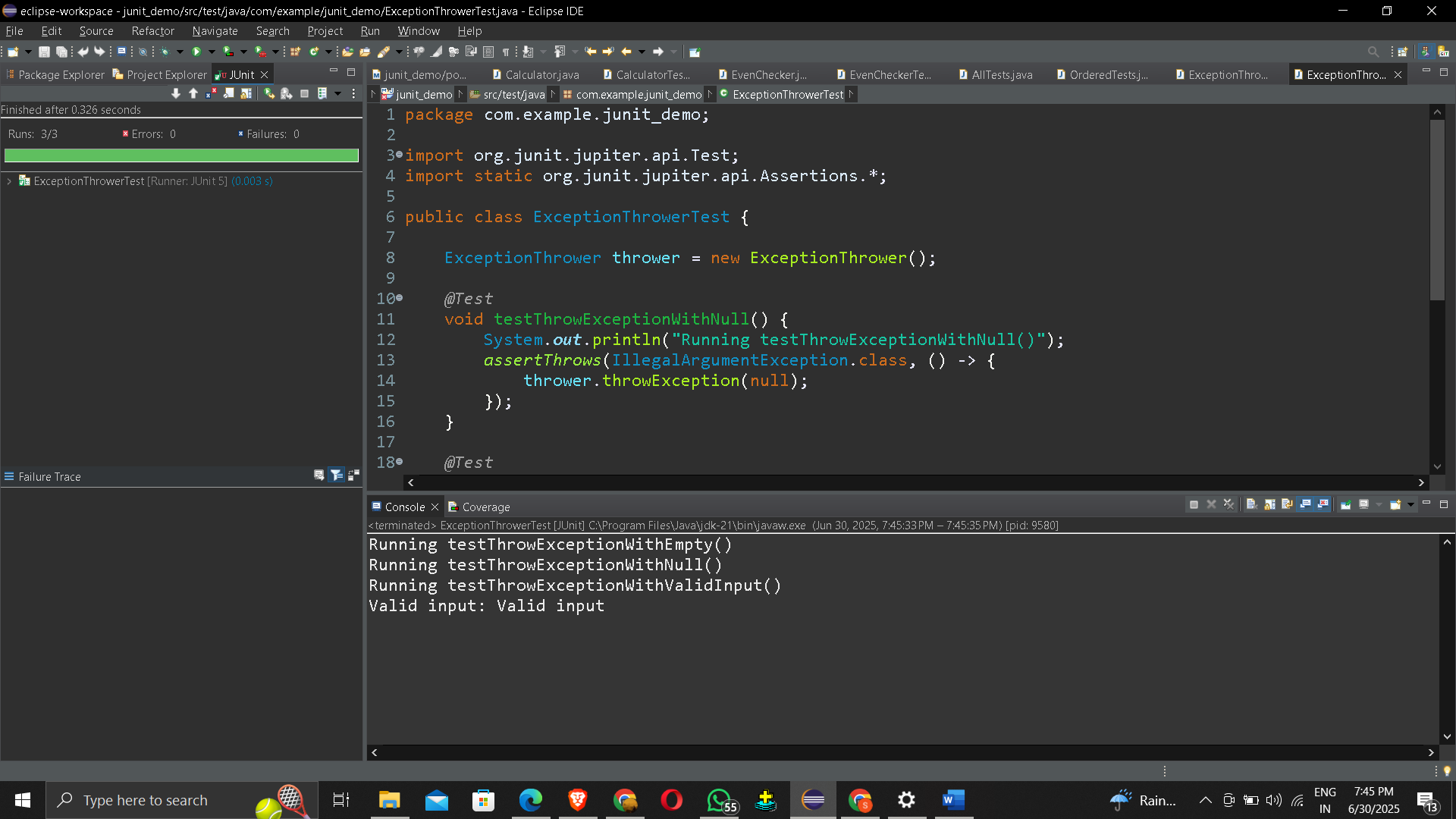
*assertDoesNotThrow*(() -> {

thrower.throwException("Valid input");

});

}

}



# 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 com.example.junit\_demo;

public class PerformanceTester {

public void performTask() {

// Simulate some work

try {

System.***out***.println("Starting task...");

Thread.*sleep*(800); // 800 milliseconds delay

System.***out***.println("Task completed");

} catch (InterruptedException e) {

throw new RuntimeException("Task interrupted", e);

}

}

}

PerformanceTesterTest.java

package com.example.junit\_demo;

import org.junit.jupiter.api.Test;

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

import java.time.Duration;

public class PerformanceTesterTest {

PerformanceTester tester = new PerformanceTester();

*@Test*

void testPerformTaskWithinTimeLimit() {

System.***out***.println("🚀 Running testPerformTaskWithinTimeLimit()");

*assertTimeout*(Duration.*ofSeconds*(1), () -> {

tester.performTask();

});

}

}

