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.

Answer:

**EvenChecker.java:**

package Junit;

public class EvenChecker {

public boolean isEven(int number) {

return number % 2 == 0;

}

}

**EvenCheckerTest.java:**

package Junit;

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

import org.junit.jupiter.params.ParameterizedTest;

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

public class EvenCheckerTest {

EvenChecker checker = new EvenChecker();

@ParameterizedTest

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

void testEvenNumbers(int num) {

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

}

@ParameterizedTest

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

void testOddNumbers(int num) {

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

}

}

# 

# 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.

**AllTests.java:**

package Junit;

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

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

@Suite

@SelectClasses({

EvenCheckerTest.class,

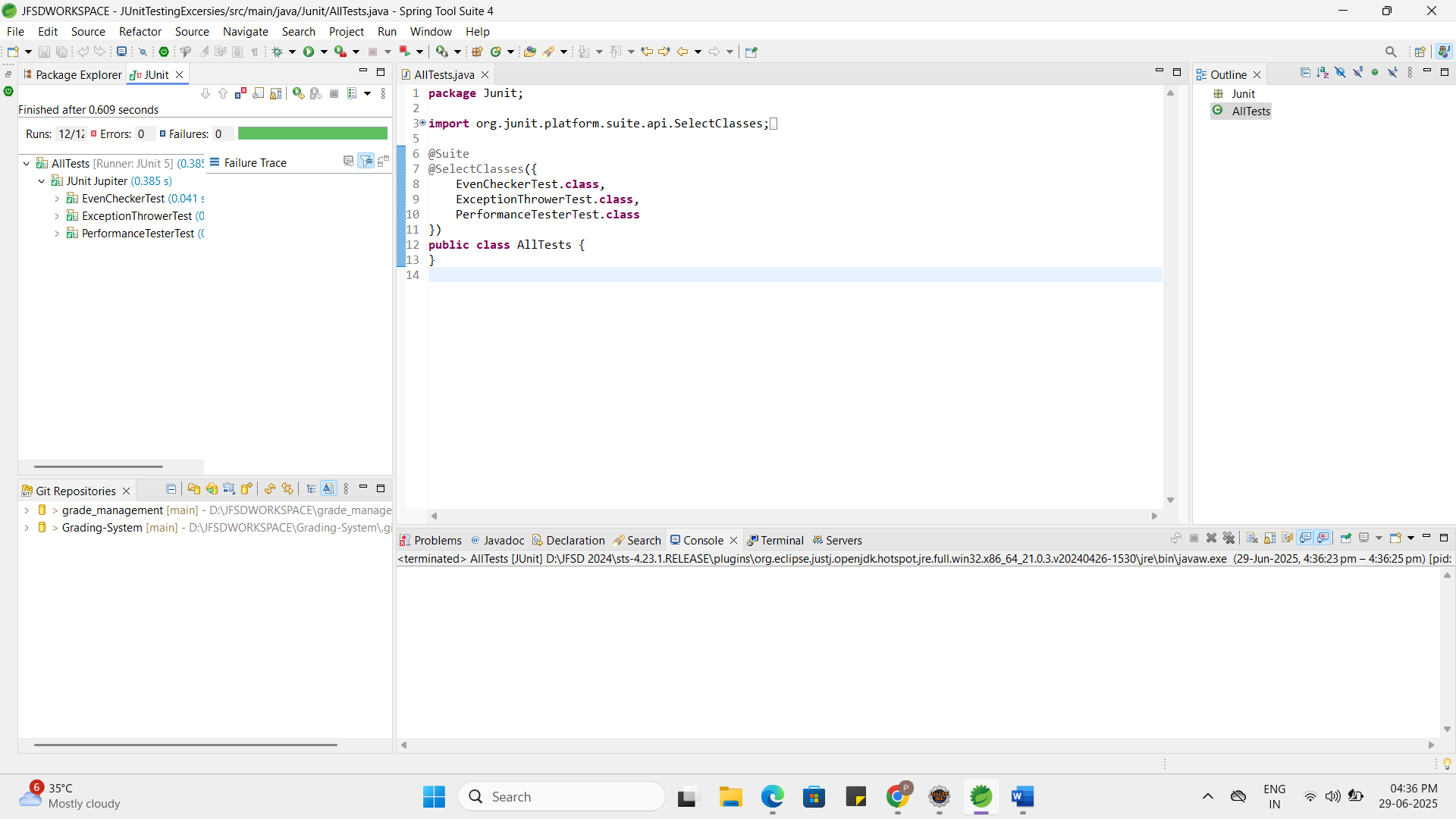
ExceptionThrowerTest.class,

PerformanceTesterTest.class

})

public class AllTests {

}



# 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.

**OrderedTests.java:**

package Junit;

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

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)

public class OrderedTests {

@Test

@Order(2)

void testB() {

System.*out*.println("Test B");

}

@Test

@Order(1)

void testA() {

System.*out*.println("Test A");

}

@Test

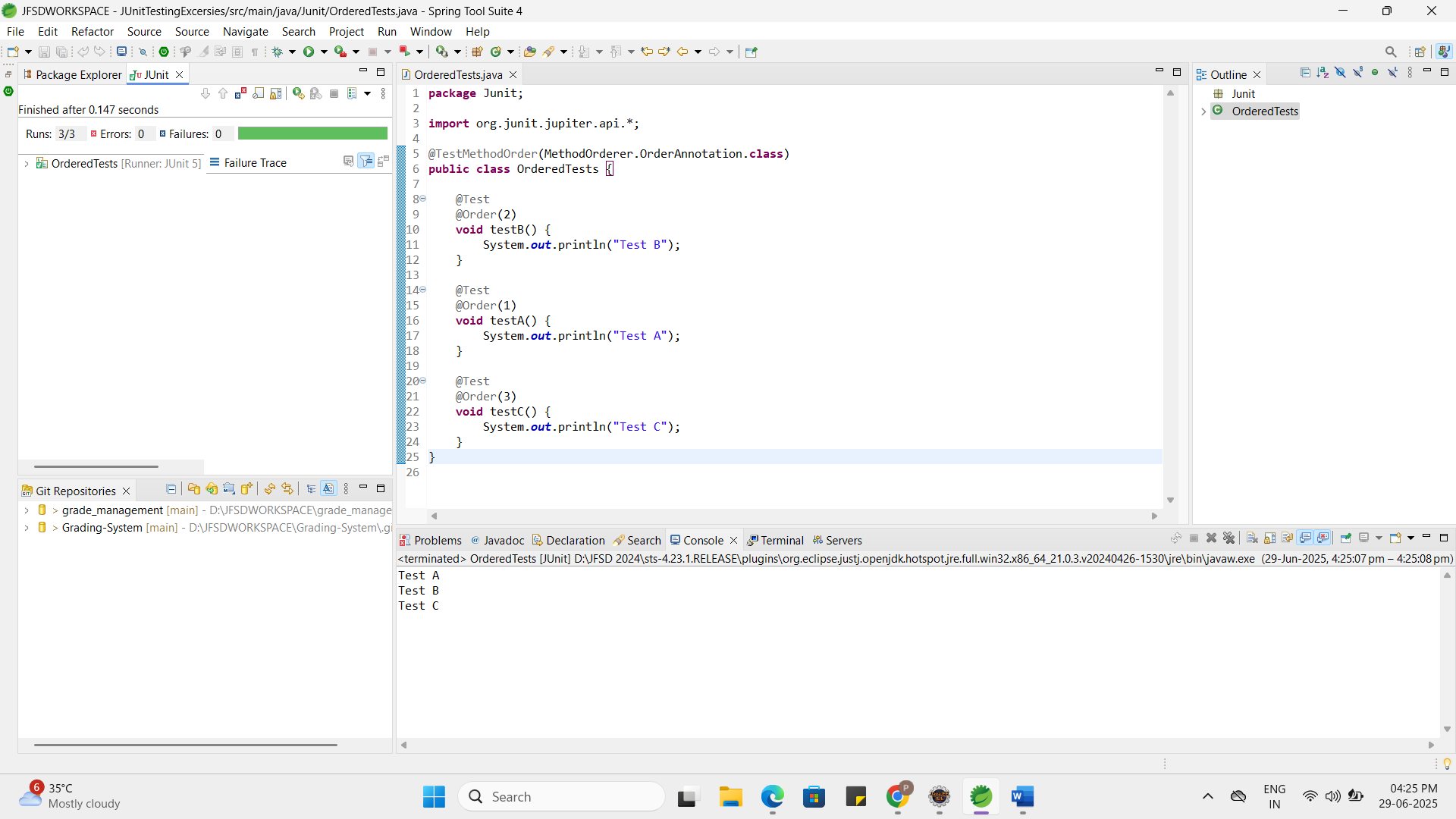
@Order(3)

void testC() {

System.*out*.println("Test C");

}

}



# 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 Junit;

public class ExceptionThrower {

public void throwException() throws IllegalArgumentException {

throw new IllegalArgumentException("Invalid input!");

}

}

**ExceptionThrowerTest.java:**

package Junit;

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

import org.junit.jupiter.api.Test;

public class ExceptionThrowerTest {

@Test

void testThrowException() {

ExceptionThrower et = new ExceptionThrower();

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

}

}

A screenshot of a computer

AI-generated content may be incorrect.

# 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 Junit;

public class PerformanceTester {

public void performTask() {

try {

Thread.*sleep*(300);

} catch (InterruptedException e) {

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

}

}

}

**PerformanceTesterTest.java:**

package Junit;

import org.junit.jupiter.api.Test;

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

import java.time.Duration;

public class PerformanceTesterTest {

@Test

void testPerformance() {

PerformanceTester tester = new PerformanceTester();

*assertTimeout*(Duration.*ofMillis*(500), tester::performTask);

}

}

