**Advanced JUnit Testing Exercises (Everything under this section is Additional)**

**Exercise 1: Parameterized Tests (Additional Exercise)**

**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 ParameterizedTest;

public class EvenChecker {

public boolean isEven(int number) {

return number % 2 == 0;

}

}

**EvenCheckerTest.java**

package ParameterizedTest;

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

import org.junit.jupiter.params.ParameterizedTest;

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

public class EvenCheckerTest {

*@ParameterizedTest*

*@ValueSource*(ints = {2, 4, 6, 8, 10, 100})

void testEvenNumbers(int input) {

EvenChecker checker = new EvenChecker();

*assertTrue*(checker.isEven(input), input + " should be even");

}

*@ParameterizedTest*

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

void testOddNumbers(int input) {

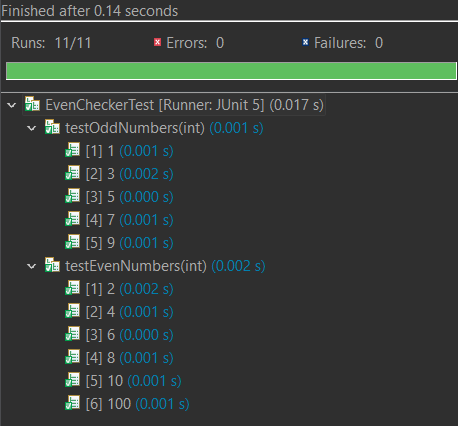
EvenChecker checker = new EvenChecker();

*assertFalse*(checker.isEven(input), input + " should be odd");

}

}

**Output:**

****

**Exercise 2: Test Suites and Categories (Additional Exercise)**

**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.**

**AdditionTest.java**

package SuitesAndCategories;

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

import org.junit.jupiter.api.Test;

public class AdditionTest {

*@Test*

void testAdd() {

*assertEquals*(5, 2 + 3);

}

}

**MultiplicationTest.java**

package SuitesAndCategories;

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

import org.junit.jupiter.api.Test;

public class MultiplicationTest {

*@Test*

void testMultiply() {

*assertEquals*(6, 2 \* 3);

}

}

**AllTests.java**

package SuitesAndCategories;

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

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

*@Suite*

*@SelectClasses*({

AdditionTest.class,

MultiplicationTest.class

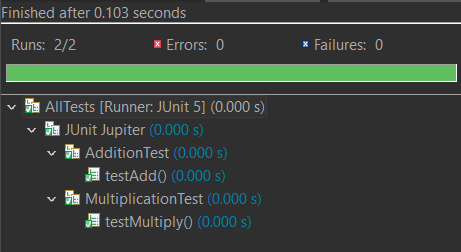
})

public class AllTests {

// No methods needed; this class is just a container for the suite

}

**Output:**

****

**Exercise 3: Test Execution Order (Additional Exercise)**

**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 OrderedTest;

import org.junit.jupiter.api.Order;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestMethodOrder;

import org.junit.jupiter.api.MethodOrderer;

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

*@TestMethodOrder*(MethodOrderer.OrderAnnotation.class)

public class OrderedTest {

*@Test*

*@Order*(3)

void testLogin() {

System.***out***.println("Running: testLogin");

*assertTrue*(true);

}

*@Test*

*@Order*(2)

void testSearch() {

System.***out***.println("Running: testSearch");

*assertTrue*(true);

}

*@Test*

*@Order*(1)

void testLogout() {

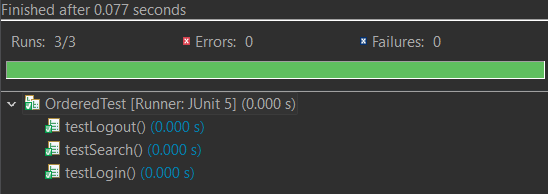
System.***out***.println("Running: testLogout");

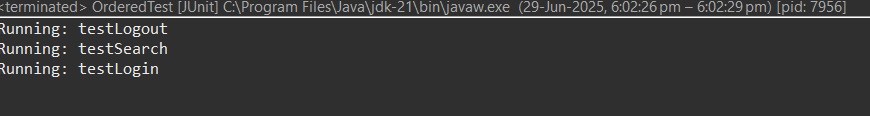
*assertTrue*(true);

}

}

**Output:**





**Exercise 4: Exception Testing (Additional Exercise)**

**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 Exception;

public class ExceptionThrower {

public void throwException(String input) {

if (input == null) {

throw new IllegalArgumentException("Input is null");

}

}

}

**ExceptionThrowerTest.java**

package Exception;

import static org.junit.Assert.\*;

import org.junit.Test;

public class ExceptionThrowerTest {

*@Test*(expected = IllegalArgumentException.class)

public void testThrowsException() {

ExceptionThrower obj = new ExceptionThrower();

obj.throwException(null);

}

*@Test*

public void testDoesNotThrow() {

ExceptionThrower obj = new ExceptionThrower();

try {

obj.throwException("Hello");

} catch (Exception e) {

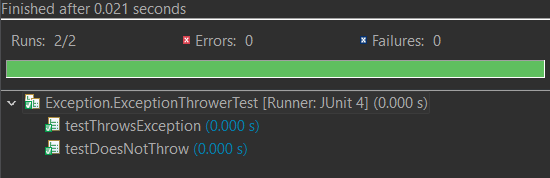
*fail*("Should not throw exception");

}

}

}

**Output:**

****

**Exercise 5: Timeout and Performance Testing (Additional Exercise)**

**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.**

**PerformanceTest.java**

package PerformanceTest;

public class PerformanceTester {

public void performTask() throws InterruptedException {

// Simulates a task that takes 200 ms

Thread.*sleep*(200);

}

}

**PerformanceTesterTest.java**

package PerformanceTest;

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

import java.time.Duration;

import org.junit.jupiter.api.Test;

public class PerformanceTesterTest {

*@Test*

void testPerformTaskWithinTime() {

PerformanceTester tester = new PerformanceTester();

*assertTimeout*(Duration.*ofMillis*(500), () -> {

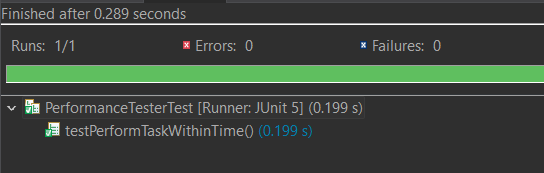
tester.performTask();

});

}

}

**Output:**

****