**WEEK 2**

**Advanced JUnit Testing Exercises**

**Exercise 1: Parameterized Tests**

**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.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, 0, -2})

public void testIsEvenWithEvenNumbers(int number) {

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

}

*@ParameterizedTest*

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

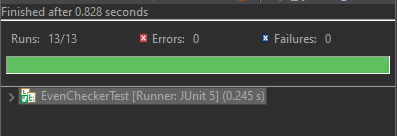
public void testIsEvenWithOddNumbers(int number) {

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

}

}

**OUTPUT**



**Exercise 2: Test Suites and Categories**

**CalculatorTest.java**

package com.example.tests;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

public class CalculatorTest {

*@*Test

void testAddition() {

assertEquals(4, 2 + 2);

}

*@*Test

void testSubtraction() {

assertEquals(0, 2 - 2);

}

}

**StringUtilTest.java**

package com.example.tests;

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

import org.junit.jupiter.api.Test;

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

public class StringUtilTest {

*@Test*

void testStringContains() {

*assertTrue*("JUnit 5".contains("JUnit"));

}

*@Test*

void testStringLength() {

*assertEquals*(6, "OpenAI".length());

}

}

**AllTests.java**

package com.example.tests;

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

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

*@Suite*

*@SelectClasses*({

CalculatorTest.class,

StringUtilTest.class

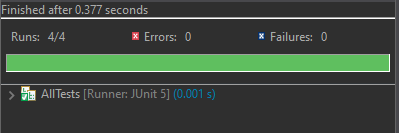
})

public class AllTests {

// No code needed inside

}

**OUTPUT**

****

**Exercise 3: Test Execution Order**

**OrderedTests.java**

package com.example.tests;

import org.junit.jupiter.api.MethodOrderer.OrderAnnotation;

import org.junit.jupiter.api.Order;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestMethodOrder;

*@TestMethodOrder*(OrderAnnotation.class)

public class OrderedTests {

*@Test*

*@Order*(3)

void testC() {

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

}

*@Test*

*@Order*(1)

void testA() {

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

}

*@Test*

*@Order*(2)

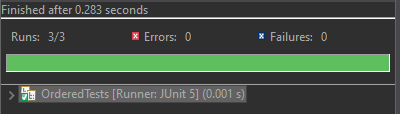
void testB() {

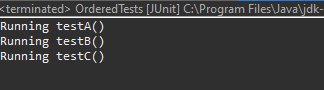
System.***out***.println("Running testB()");

}

}

**OUTPUT**

****

****

**Exercise 4: Exception Testing**

**ExceptionThrower.java**

package com.example;

public class ExceptionThrower {

public void throwException(String input) {

if (input == null) {

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

}

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

}

}

**ExceptionThrowerTest.java**

package com.example.tests;

import com.example.ExceptionThrower;

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

import org.junit.jupiter.api.Test;

public class ExceptionThrowerTest {

*@Test*

void testThrowExceptionWithNull() {

ExceptionThrower thrower = new ExceptionThrower();

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

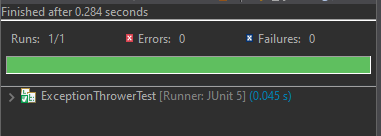
thrower.throwException(null);

});

}

}

**OUTPUT**

****

**Exercise 5: Timeout and Performance Testing**

**PerformanceTester.java**

package com.example;

public class PerformanceTester {

public void performTask() {

// Simulating a task that takes some time (e.g., 500 milliseconds)

try {

Thread.*sleep*(500); // 500 milliseconds delay

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

**PerformanceTesterTest.java**

package com.example.tests;

import com.example.PerformanceTester;

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

import java.time.Duration;

import org.junit.jupiter.api.Test;

public class PerformanceTesterTest {

*@Test*

void testPerformTaskCompletesInTime() {

PerformanceTester tester = new PerformanceTester();

// Assert that the task completes within 1 second

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

tester.performTask();

});

}

}

**OUTPUT**

