**ADVANCED JUNIT TESTING**

**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.

**SOLUTION:**

**EVENTCHECKER.java**

public class EvenChecker {

public static boolean isEven(int number) {

return number % 2 == 0;

}

}

**EVENTCHECKERTEST.java**

import org.junit.jupiter.params.ParameterizedTest;

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

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

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

public class EvenCheckerTest {

@ParameterizedTest

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

void testEvenNumbersShouldReturnTrue(int number) {

assertTrue(EvenChecker.isEven(number));

}

@ParameterizedTest

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

void testOddNumbersShouldReturnFalse(int number) {

assertFalse(EvenChecker.isEven(number));

}

}

**DEPENDENCIES(pom.xml)**

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

</dependencies>

**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.

**SOLUTION:**

**CalculatorTest.java**

import org.junit.jupiter.api.Test;

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

public class CalculatorTest {

@Test

void testAddition() {

assertEquals(4, 2 + 2);

}

@Test

void testSubtraction() {

assertEquals(0, 2 - 2);

}

}

**StringUtilsTest.java**

import org.junit.jupiter.api.Test;

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

public class StringUtilsTest {

@Test

void testToUpperCase() {

assertEquals("HELLO", "hello".toUpperCase());

}

@Test

void testTrim() {

assertEquals("hello", " hello ".trim());

}

}

**AllTests.java**

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

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

@Suite

@SelectClasses({ CalculatorTest.class, StringUtilsTest.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.

**SOLUTION:**

**OrderedTests.java**

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

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)

public class OrderedTests {

@Test

@Order(3)

void testC() {

System.out.println("Test C - Order 3");

}

@Test

@Order(1)

void testA() {

System.out.println("Test A - Order 1");

}

@Test

@Order(2)

void testB() {

System.out.println("Test B - Order 2");

}

}

**OUTPUT:**

Test A - Order 1

Test B - Order 2

Test C - Order 3

**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.

**SOLUTION:**

**ExceptionThrower.java**

public class ExceptionThrower {

public void throwException() {

throw new IllegalArgumentException("Invalid argument provided");

}

}

ExceptionThrowerTest.java

import org.junit.jupiter.api.Test;

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

public class ExceptionThrowerTest {

@Test

void testThrowsException() {

ExceptionThrower thrower = new ExceptionThrower();

Exception exception = assertThrows(IllegalArgumentException.class, thrower::throwException);

assertEquals("Invalid argument provided", exception.getMessage());

}

}

**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.

**SOLUTION:**

**PerformanceTester.java**

public class PerformanceTester {

public void performTask() {

try {

// Simulate a task that takes time (e.g., 300ms)

Thread.sleep(300);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

**PerformanceTesterTest.java**

import org.junit.jupiter.api.Test;

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

import java.time.Duration;

public class PerformanceTesterTest {

@Test

void testPerformTaskCompletesInTime() {

PerformanceTester tester = new PerformanceTester();

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

}

}