**Advanced JUnit Testing Exercises - Answers**

**Exercise 1: Parameterized Tests**

**Solution:**

**1. Create**EvenChecker**class:**

java

public class EvenChecker {

public boolean isEven(int number) {

return number % 2 == 0;

}

}

**2. Write**EvenCheckerTest**with**@ParameterizedTest**:**

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 {

private final EvenChecker checker = new EvenChecker();

@ParameterizedTest

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

public void testIsEven\_ReturnsTrue(int number) {

assertTrue(checker.isEven(number));

}

@ParameterizedTest

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

public void testIsEven\_ReturnsFalse(int number) {

assertFalse(checker.isEven(number));

}

}

**Key Points:**

* Uses @ParameterizedTest to run the same test with multiple inputs.
* @ValueSource provides test data.

**Exercise 2: Test Suites and Categories**

**Solution:**

**1. Create a test suite**AllTests**:**

java

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

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

@Suite

@SelectClasses({

CalculatorTest.class,

EvenCheckerTest.class,

PerformanceTesterTest.class

})

public class AllTests {

*// Suite runs all selected test classes*

}

**Key Points:**

* @Suite marks the class as a test suite.
* @SelectClasses lists the test classes to include.

**Exercise 3: Test Execution Order**

**Solution:**

**1. Create**OrderedTests**with**@Order**:**

java

import org.junit.jupiter.api.MethodOrderer;

import org.junit.jupiter.api.Order;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestMethodOrder;

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

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)

public class OrderedTests {

private static int counter = 0;

@Test

@Order(1)

public void testFirst() {

counter += 1;

assertEquals(1, counter);

}

@Test

@Order(2)

public void testSecond() {

counter += 2;

assertEquals(3, counter);

}

}

**Key Points:**

* @TestMethodOrder enforces ordered execution.
* @Order specifies the sequence.

**Exercise 4: Exception Testing**

**Solution:**

**1. Create**ExceptionThrower**class:**

java

public class ExceptionThrower {

public void throwException() throws IllegalArgumentException {

throw new IllegalArgumentException("Test exception");

}

}

**2. Write**ExceptionThrowerTest**:**

java

import org.junit.jupiter.api.Test;

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

public class ExceptionThrowerTest {

@Test

public void testThrowException\_ThrowsIllegalArgumentException() {

ExceptionThrower thrower = new ExceptionThrower();

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

}

}

**Key Points:**

* assertThrows verifies the expected exception.

**Exercise 5: Timeout and Performance Testing**

**Solution:**

**1. Create**PerformanceTester**class:**

java

public class PerformanceTester {

public void performTask() throws InterruptedException {

Thread.sleep(1000); *// Simulate a slow task*

}

}

**2. Write**PerformanceTesterTest**:**

java

import org.junit.jupiter.api.Test;

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

import static java.time.Duration.ofMillis;

public class PerformanceTesterTest {

@Test

public void testPerformTask\_CompletesWithinTimeout() {

PerformanceTester tester = new PerformanceTester();

assertTimeoutPreemptively(ofMillis(1500), tester::performTask);

}

}

**Key Points:**

* assertTimeoutPreemptively fails if the method exceeds the timeout.