**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.

**Code:**

**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>JUnitParameterizedExample</artifactId>

<version>1.0</version>

<name>JUnitParameterizedExample</name>

<dependencies>

<!-- JUnit 5 API -->

<dependency>

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

<artifactId>junit-jupiter-api</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- JUnit 5 Engine -->

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Maven Surefire Plugin for JUnit 5 -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0</version>

</plugin>

</plugins>

</build>

</project>

**EvenChecker.java**

package com.example.junit;

public class EvenChecker {

public boolean isEven(int number) {

return number % 2 == 0;

}

}

**EvenCheckerTest.java**

package com.example.junit;

import org.junit.jupiter.params.ParameterizedTest;

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

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

public class EvenCheckerTest {

private final EvenChecker checker = new EvenChecker();

*@ParameterizedTest*

*@ValueSource*(ints = {2, 4, 6, 8, 10, 12})

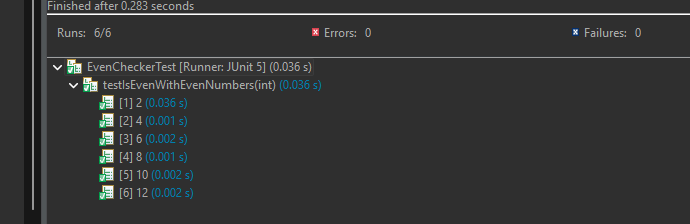
void testIsEvenWithEvenNumbers(int number) {

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

}

}

**Output:**

****

# 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.

**Code:**

**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>JUnitTestSuiteExample</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- JUnit 5 Jupiter API -->

<dependency>

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

<artifactId>junit-jupiter-api</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- JUnit 5 Engine -->

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- JUnit Platform Suite API for @Suite and @SelectClasses -->

<dependency>

<groupId>org.junit.platform</groupId>

<artifactId>junit-platform-suite-api</artifactId>

<version>1.10.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Maven Surefire Plugin to enable JUnit 5 support -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0</version>

</plugin>

</plugins>

</build>

</project>

**MathUtils.java**

package com.example.junit;

public class MathUtils {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

}

**AddTest.java**

package com.example.junit;

import org.junit.jupiter.api.Test;

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

public class AddTest {

*@Test*

void testAdd() {

MathUtils math = new MathUtils();

*assertEquals*(7, math.add(3, 4));

}

}

**MultiplyTest.java**

package com.example.junit;

import org.junit.jupiter.api.Test;

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

public class MultiplyTest {

*@Test*

void testMultiply() {

MathUtils math = new MathUtils();

*assertEquals*(12, math.multiply(3, 4));

}

}

**AllTests.java**

package com.example.junit;

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

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

*@Suite*

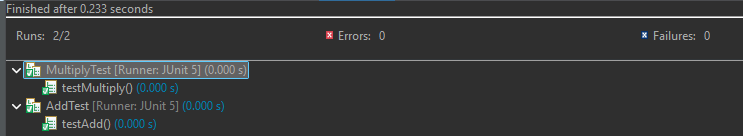
*@SelectClasses*({ AddTest.class, MultiplyTest.class })

public class AllTests {

// This class remains empty; used only as a holder for annotations

}

**Output:**

****

# 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.

**Code:**

**pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>JUnitExecutionOrderExample</artifactId>

<version>1.0</version>

<name>JUnitExecutionOrderExample</name>

<dependencies>

<!-- JUnit Jupiter API -->

<dependency>

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

<artifactId>junit-jupiter-api</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- JUnit Jupiter Engine -->

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Required for JUnit 5 tests -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0</version>

</plugin>

</plugins>

</build>

</project>

# OrderedTests.java

package com.example.junit;

import org.junit.jupiter.api.Order;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestMethodOrder;

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

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

*@TestMethodOrder*(OrderAnnotation.class)

public class OrderedTests {

*@Test*

*@Order*(3)

void testC() {

System.***out***.println("Running testC (Order 3)");

*assertTrue*(true);

}

*@Test*

*@Order*(1)

void testA() {

System.***out***.println("Running testA (Order 1)");

*assertTrue*(true);

}

*@Test*

*@Order*(2)

void testB() {

System.***out***.println("Running testB (Order 2)");

*assertTrue*(true);

}

}

# Output:

# 

# 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.

**Code:**

**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>JUnitExceptionTestExample</artifactId>

<version>1.0</version>

<name>JUnitExceptionTestExample</name>

<dependencies>

<dependency>

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

<artifactId>junit-jupiter-api</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0</version>

</plugin>

</plugins>

</build>

</project>

**ExceptionThrower.java**

package com.example.junit;

public class ExceptionThrower {

public void throwException(String input) {

if (input == null || input.isEmpty()) {

throw new IllegalArgumentException("Input must not be null or empty");

}

// normal processing logic

System.***out***.println("Processing input: " + input);

}

}

**ExceptionThrowerTest.java**

package com.example.junit;

import org.junit.jupiter.api.Test;

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

public class ExceptionThrowerTest {

*@Test*

void testThrowExceptionWithNull() {

ExceptionThrower thrower = new ExceptionThrower();

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

thrower.throwException(null);

});

}

*@Test*

void testThrowExceptionWithEmpty() {

ExceptionThrower thrower = new ExceptionThrower();

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

thrower.throwException("");

});

}

*@Test*

void testThrowExceptionWithValidInput() {

ExceptionThrower thrower = new ExceptionThrower();

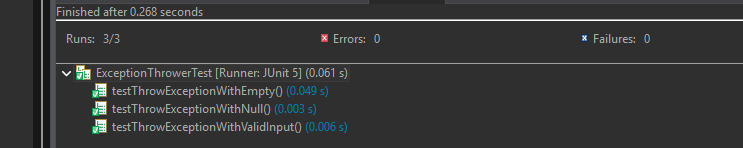
// Should NOT throw any exception

thrower.throwException("Hello");

}

}

**Output:**

****

# 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.

**Code:**

**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>JUnitTimeoutExample</artifactId>

<version>1.0</version>

<name>JUnitTimeoutExample</name>

<dependencies>

<!-- JUnit 5 API -->

<dependency>

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

<artifactId>junit-jupiter-api</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- JUnit 5 Engine -->

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0</version>

</plugin>

</plugins>

</build>

</project>

**PerformanceTester.java**

package com.example.junit;

public class PerformanceTester {

public void performTask() {

try {

// Simulate a task that takes 300 milliseconds

Thread.*sleep*(300);

} catch (InterruptedException e) {

throw new RuntimeException("Task interrupted", e);

}

}

}

**PerformanceTesterTest.java**

package com.example.junit;

import org.junit.jupiter.api.Test;

import java.time.Duration;

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

public class PerformanceTesterTest {

*@Test*

void testPerformTaskCompletesInTime() {

PerformanceTester tester = new PerformanceTester();

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

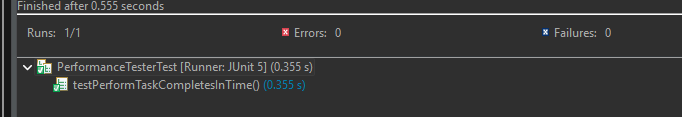
tester.performTask();

});

}

}

**Output:**

****