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.

EvenChecker Class:

Code:

public class EvenChecker {

public static boolean isEven(int number) {

return number % 2 == 0;

}

}

Add JUnit 5 Library:

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, 100, 0})

void testEvenNumbers(int number) {

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

}

@ParameterizedTest

@ValueSource(ints = {1, 3, 5, 99, -1})

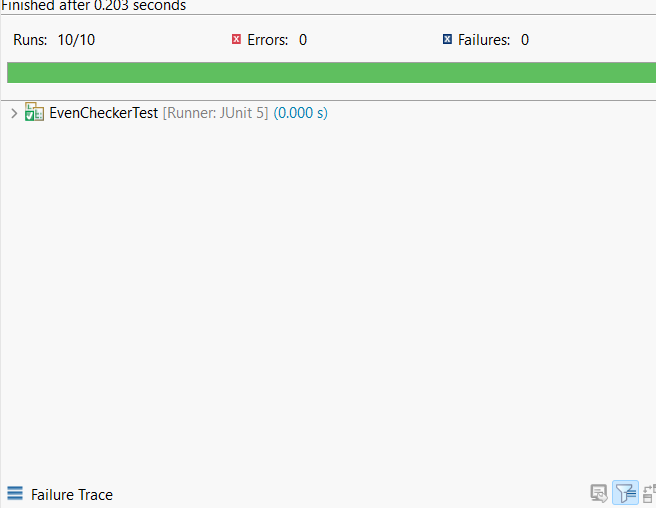
void testOddNumbers(int number) {

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

}

}

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.

MathUtilsTest:

Code:

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

import org.junit.jupiter.api.Test;

public class MathUtilsTest {

@Test

void testAdd() {

assertEquals(4, 2 + 2);

}

@Test

void testSubtract() {

assertEquals(3, 6 - 3);

}

}

StringUtilsTest:

Code:

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

import org.junit.jupiter.api.Test;

public class StringUtilsTest {

@Test

void testToUpperCase() {

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

}

@Test

void testLength() {

assertEquals(5, "Hello".length());

}

}

Code:

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

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

@Suite

@SelectClasses({

MathUtilsTest.class,

StringUtilsTest.class

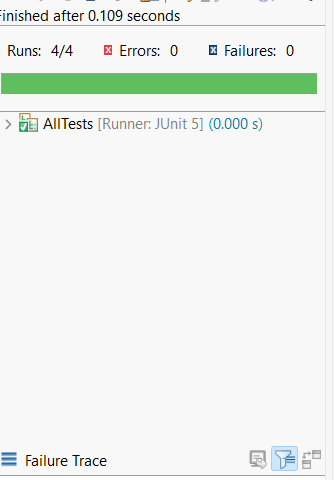
})

public class AllTests {

// No methods needed. Just annotations to group the tests.

}

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.

OrderedTests:

Code:

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(1)

void testStart() {

System.out.println("Test 1: Start");

}

@Test

@Order(3)

void testEnd() {

System.out.println("Test 3: End");

}

@Test

@Order(2)

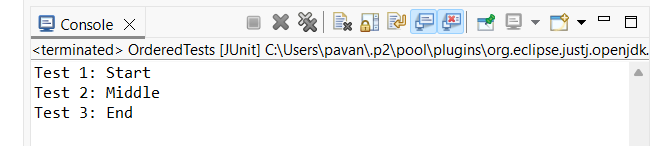
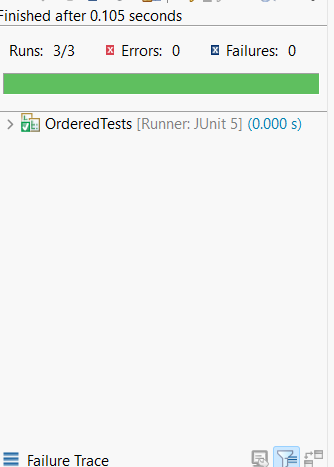
void testMiddle() {

System.out.println("Test 2: Middle");

}

}

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.

ExceptionThrower:

Code:

public class ExceptionThrower {

public static void throwException() {

throw new IllegalArgumentException("Invalid argument provided");

}

}

ExceptionThrowerTest:

Code:

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

import org.junit.jupiter.api.Test;

public class ExceptionThrowerTest {

@Test

void testExceptionIsThrown() {

assertThrows(IllegalArgumentException.class, () -> {

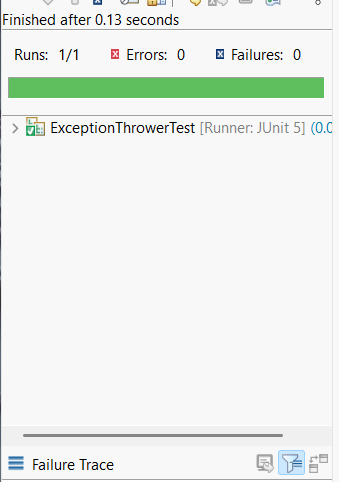
ExceptionThrower.throwException();

});

}

}

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.

PerformanceTester:

Code:

public class PerformanceTester {

public static void performTask() {

try {

// Simulate a task that takes some time (e.g., 500ms)

Thread.sleep(500);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

PerformanceTesterTest:

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

import java.time.Duration;

import org.junit.jupiter.api.Test;

public class PerformanceTesterTest {

@Test

void testPerformTaskCompletesInTime() {

assertTimeout(Duration.ofMillis(1000), () -> {

PerformanceTester.performTask();

});

}

}

Output:

