**Advanced JUnit Testing Exercise**

**Exercise 1: Parameterized Tests:** Y**ou 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.**

**EvenCheckerTest.java**

package com.example.junit\_demo;

import org.junit.jupiter.params.ParameterizedTest;

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

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

public class EvenCheckerTest {

private final EvenChecker checker = new EvenChecker();

*@ParameterizedTest*

*@ValueSource*(ints = {2, 4, 6, 8, 10, 0, -2, -8})

public void testIsEven\_WithEvenNumbers(int number) {

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

}

*@ParameterizedTest*

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

public void testIsEven\_WithOddNumbers(int number) {

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

}

}

**EvenChecker.java**

package com.example.junit\_demo;

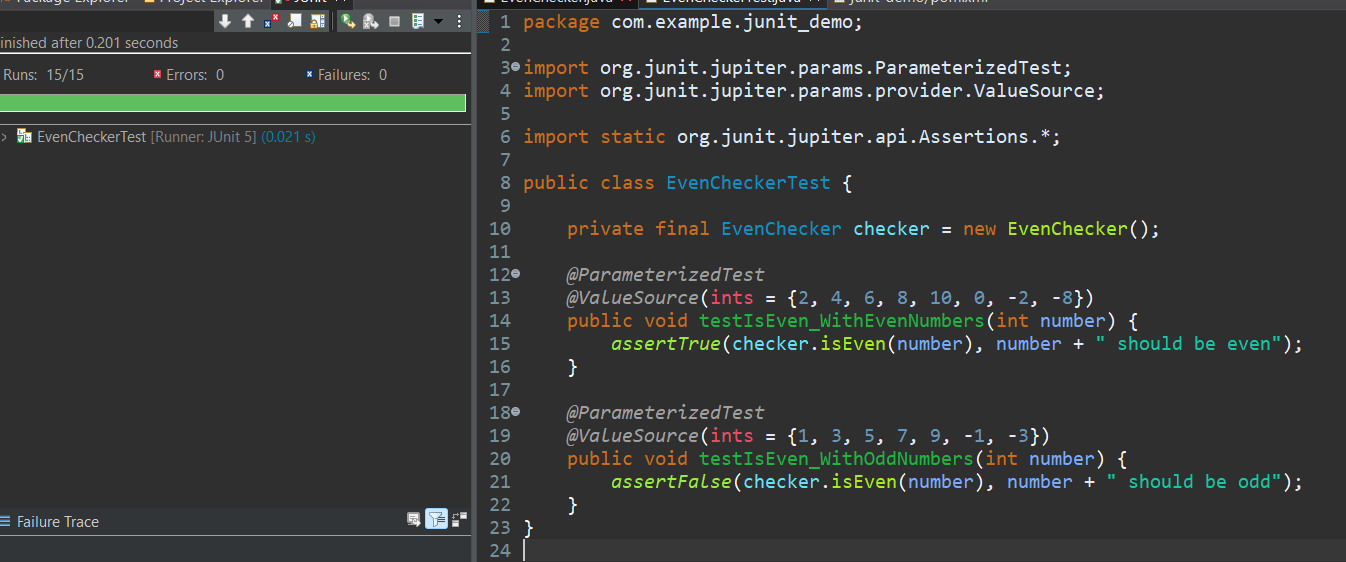
public class EvenChecker {

public boolean isEven(int number) {

return number % 2 == 0;

}

}

****

**Exercise 2: Test Suites and Categories :**

**You want to group related tests into a test suite and categorize them**

**MathUtilsTest.java**

package com.example.junit\_demo;

import org.junit.jupiter.api.Test;

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

public class MathUtilsTest {

MathUtils math = new MathUtils();

*@Test*

public void testAdd() {

*assertEquals*(9, math.add(4, 5));

}

*@Test*

public void testMultiply() {

*assertEquals*(20, math.multiply(4, 5));

}

}

package com.example.junit\_demo;

**MathUtils.java**

public class MathUtils {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

}

package com.example.junit\_demo;

public class EvenChecker {

public boolean isEven(int number) {

return number % 2 == 0;

}

}

package com.example.junit\_demo;

import org.junit.jupiter.params.ParameterizedTest;

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

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

public class EvenCheckerTest {

private final EvenChecker checker = new EvenChecker();

*@ParameterizedTest*

*@ValueSource*(ints = {2, 4, 6, 8, 10, 0, -2, -8})

public void testIsEven\_WithEvenNumbers(int number) {

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

}

*@ParameterizedTest*

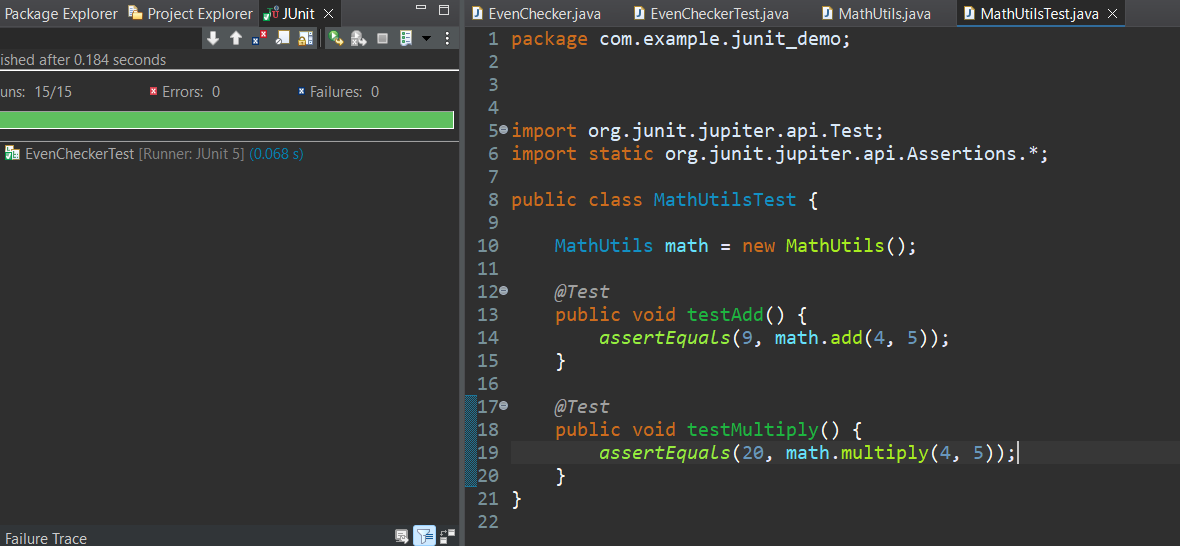
*@ValueSource*(ints = {1, 3, 5, 7, 9, -1, -3})

public void testIsEven\_WithOddNumbers(int number) {

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

}

}

****

**Exercise 3: Test Execution Order**

**Scenario: You want to control the order in which tests are executed.**

**OrderedTests.java**

package com.example.junit\_demo;

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

*@TestMethodOrder*(MethodOrderer.OrderAnnotation.class)

public class OrderedTests {

*@Test*

*@Order*(3)

void testC() {

System.*out*.println("Running Test C");

}

*@Test*

*@Order*(1)

void testA() {

System.*out*.println("Running Test A");

}

*@Test*

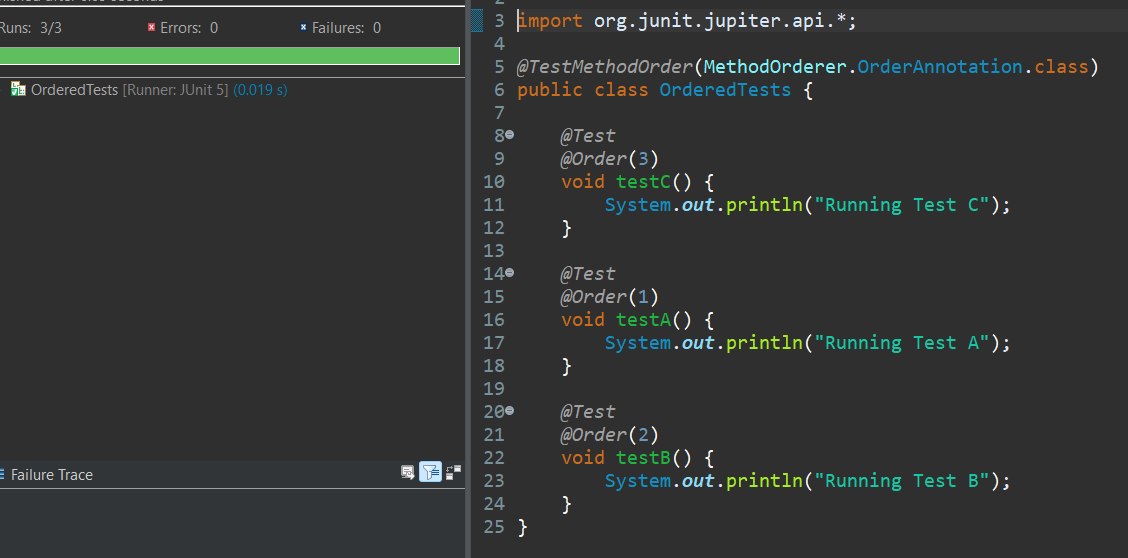
*@Order*(2)

void testB() {

System.*out*.println("Running Test B");

}

}

****

**Exercise 4: Exception Testing**

**Scenario: You want to test that a method throws the expected exception.**

**ExceptionThrowerTest.java**

package com.example.junit\_demo;

import org.junit.jupiter.api.Test;

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

public class ExceptionThrowerTest {

@Test

public void testThrowsException() {

ExceptionThrower et = new ExceptionThrower();

// assertThrows checks for expected exception

assertThrows(IllegalArgumentException.class, () -> {

et.throwException(""); // Invalid input

});

assertThrows(IllegalArgumentException.class, () -> {

et.throwException(null); // Invalid input

});

}

@Test

public void testDoesNotThrowException() {

ExceptionThrower et = new ExceptionThrower();

assertDoesNotThrow(() -> {

et.throwException("Hello JUnit!");

});

}

}

**ExceptionThrower.java**

package com.example.junit\_demo;

public class ExceptionThrower {

public void throwException(String message) {

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

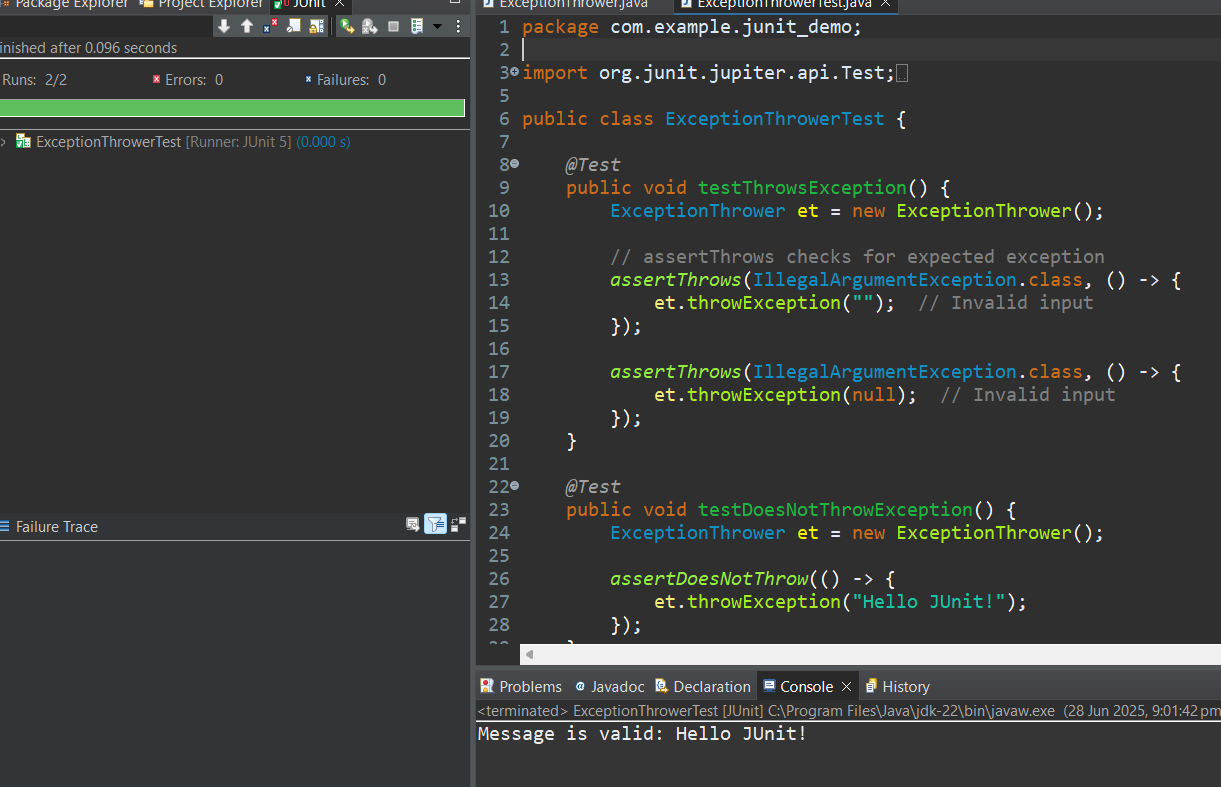
throw new IllegalArgumentException("Message cannot be null or empty");

}

System.*out*.println("Message is valid: " + message);

}

}

****

**Exercise 5: Timeout and Performance Testing**

**Scenario: You want to ensure that a method completes within a specified time limit.**

**PerformanceTesterTest.java**

package com.example.junit\_demo;

import org.junit.jupiter.api.Test;

import com.example.junit\_demo.PerformanceTester;

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

import java.time.Duration;

public class PerformanceTesterTest {

PerformanceTester tester = new PerformanceTester();

*@Test*

public void testPerformTaskWithinTime() {

*assertTimeout*(Duration.*ofSeconds*(2), () -> {

tester.performTask(); // Should pass (1 sec < 2 sec)

});

}

*@Test*

public void testSlowTaskFailsTimeout() {

*assertTimeoutPreemptively*(Duration.*ofSeconds*(6), () -> {

tester.slowTask(); // now 5 sec < 6 sec

});

}

}

**PerformanceTester.java**

package com.example.junit\_demo;

public class PerformanceTester {

public void performTask() {

try {

// Simulating a task that takes 1 second

Thread.*sleep*(1000);

} catch (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

public void slowTask() {

try {

// Simulating a long task (e.g., 5 seconds)

Thread.*sleep*(5000);

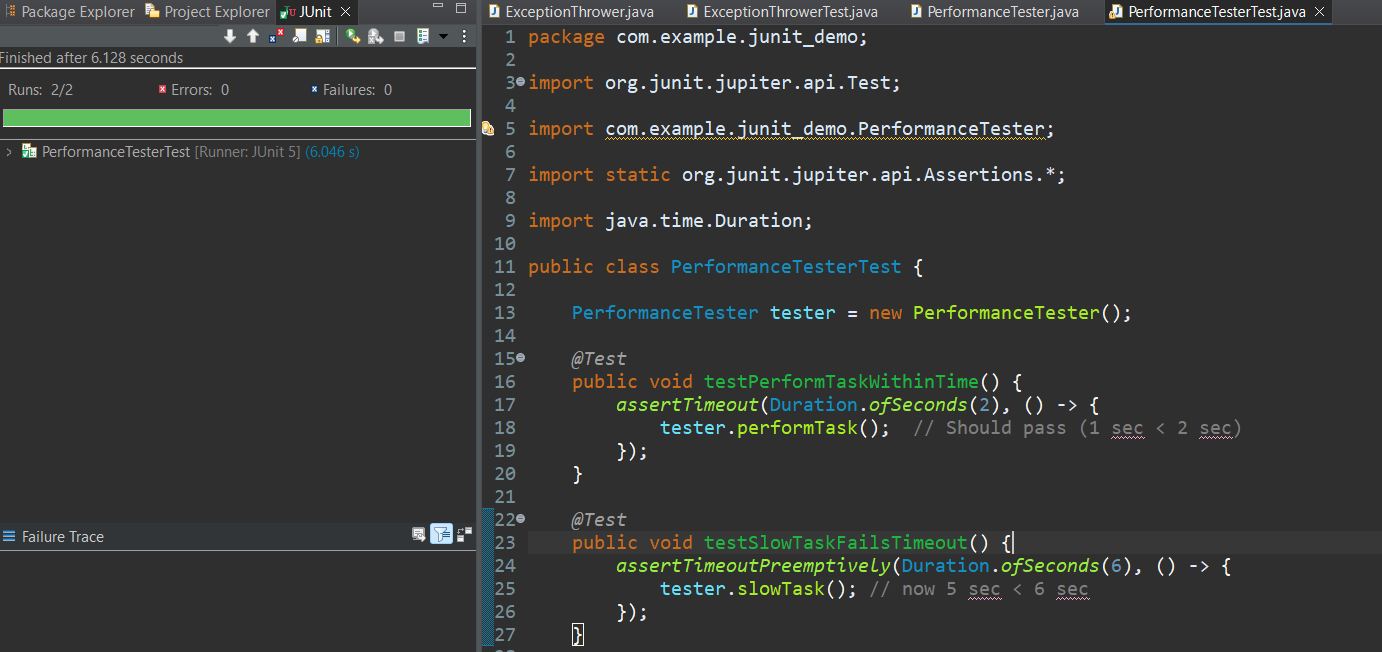
} catch (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

****