**“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.**

***SOLUTION:***

**1.EvenChecker.java**

package Utilities;

public class EvenChecker {

public static boolean isEven(int number) {

return number % 2 == 0;

}

}

**2.EvenCheckerTest.java**

package Utilities;

import org.junit.jupiter.params.ParameterizedTest;

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

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

public class EvenCheckerTest {

// ✅ Test with even numbers

@ParameterizedTest

@ValueSource(ints = {2, 4, 6, 8, 10, 100})

void testIsEvenWithEvenNumbers(int number) {

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

}

// ✅ Test with odd numbers

@ParameterizedTest

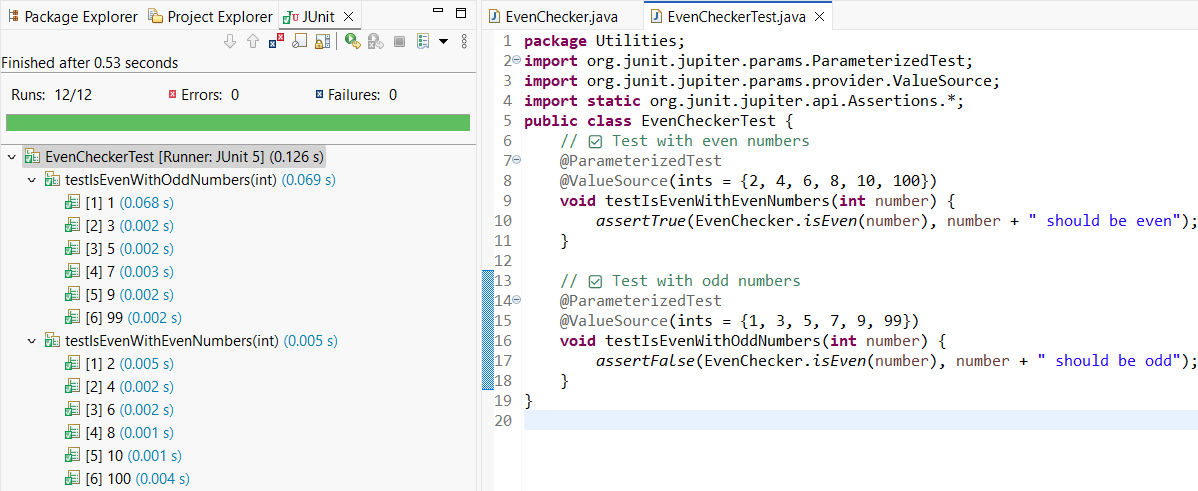
@ValueSource(ints = {1, 3, 5, 7, 9, 99})

void testIsEvenWithOddNumbers(int number) {

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

}

}



**---------------------------------------------------------------------------------------------------------------------------**

**Exercise 2: Test Suites and Categories , Scenario: You want to group related tests into a test suite and categorize them.**

***SOLUTION:***

**1.MathTest.java**

package Utilities;

import org.junit.jupiter.api.Test;

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

public class MathTest {

@Test

void testAdd() {

*assertEquals*(5, 2 + 3);

}

@Test

void testSubtract() {

*assertEquals*(1, 4 - 3);

}

}

**2.StringTest.java**

package Utilities;

import org.junit.jupiter.api.Test;

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

public class StringTest {

@Test

void testUpperCase() {

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

}

@Test

void testLength() {

*assertEquals*(4, "test".length());

}

}

**3.AllTests.java**

package Utilities;

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

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

@Suite

@SelectClasses({

MathTest.class,

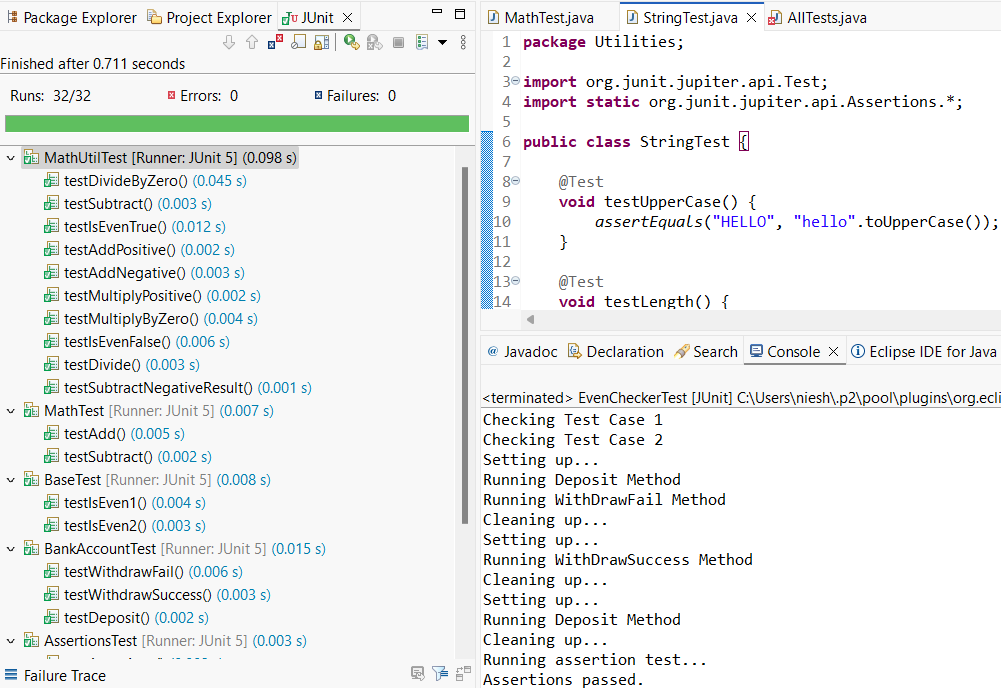
StringTest.class

})

public class AllTests {

// No code needed; this class acts as a suite holder

}

****

**---------------------------------------------------------------------------------------------------------------------------**

**Exercise 3: Test Execution Order Scenario: You want to control the order in which tests are executed.**

**SOLUTION:**

**1.OrderedTests.java**

package Utilities;

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

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

@TestMethodOrder(MethodOrderer.OrderAnnotation.class) // Enables @Order

public class OrderedTests {

@Test

@Order(1)

void testInitialize() {

System.*out*.println("Test 1 - Initialization");

*assertTrue*(true);

}

@Test

@Order(2)

void testProcessData() {

System.*out*.println("Test 2 - Processing Data");

*assertEquals*(10, 5 + 5);

}

@Test

@Order(3)

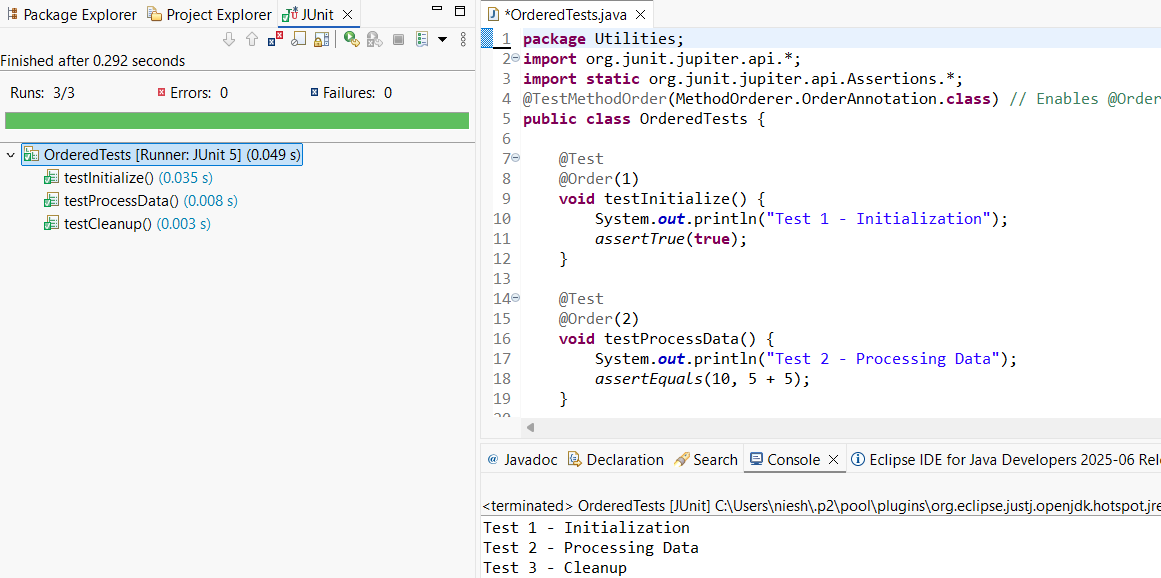
void testCleanup() {

System.*out*.println("Test 3 - Cleanup");

*assertNotNull*("done");

}

}

****

**----------------------------------------------------------------------------------------------------------------**

**Exercise 4: Exception Testing Scenario: You want to test that a method throws the expected exception.**

***SOLUTION:***

**1.ExceptionThrower.java**

package Utilities;

public class ExceptionThrower {

public void throwException(String input) {

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

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

}

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

}

}

**2.ExceptionThrowerTest.java**

package Utilities;

import org.junit.jupiter.api.Test;

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

public class ExceptionThrowerTest {

@Test

void testThrowsExceptionForNull() {

ExceptionThrower et = new ExceptionThrower();

// Test for null input

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

et.throwException(null);

});

}

@Test

void testThrowsExceptionForEmptyString() {

ExceptionThrower et = new ExceptionThrower();

// Test for empty string

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

et.throwException("");

});

}

@Test

void testValidInput() {

ExceptionThrower et = new ExceptionThrower();

// No exception should be thrown

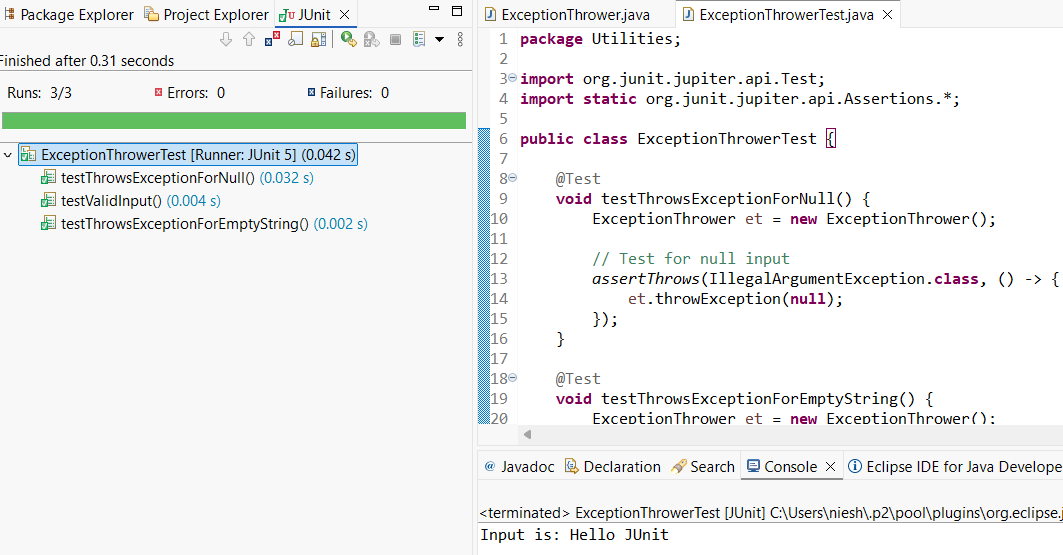
*assertDoesNotThrow*(() -> {

et.throwException("Hello JUnit");

});

}

}

****

**---------------------------------------------------------------------------------------------------------------------------**

**Exercise 5: Timeout and Performance Testing**

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

***SOLUTION:***

**1.PerformanceTester.java**

package Utilities;

public class PerformanceTester {

public void performTask() {

// Simulate a task that takes time

try {

Thread.*sleep*(300); // 300 milliseconds

} catch (InterruptedException e) {

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

}

}

}

**2.PerformanceTesterTest.java**

package Utilities;

import org.junit.jupiter.api.Test;

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

public class PerformanceTesterTest {

@Test

void testPerformTaskWithinTimeout() {

PerformanceTester tester = new PerformanceTester();

// Assert that the task completes within 500 milliseconds

*assertTimeout*(

java.time.Duration.*ofMillis*(500),

tester::performTask,

"performTask() should complete within 500 milliseconds"

);

}

@Test

void testPerformTaskExceedsTimeout() {

PerformanceTester tester = new PerformanceTester();

// Example: force failure by expecting it within a shorter time (like 100ms)

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

*assertTimeout*(

java.time.Duration.*ofMillis*(100),

tester::performTask,

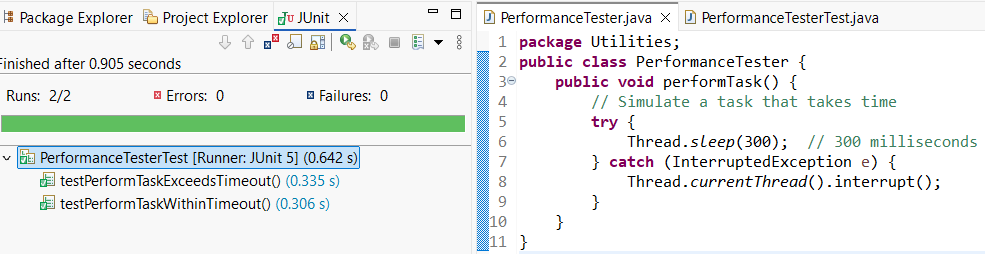
"Expected to fail - performTask() should not complete within 100ms"

);

});

}

}

****

**---------------------------------------------------------------------------------------------------------------------------**