JUNIT BASIC TESTING EXERCISES

**Exercise 1: Setting Up JUnit**

Scenario:

You need to set up JUnit in your Java project to start writing unit tests.

Steps:

1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).

2. Add JUnit dependency to your project. If you are using Maven, edit pom.xml accordingly

3. Create a new test class in your project.

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>JUnitDemo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- JUnit 5 API -->

<dependency>

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

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

<version>5.10.2</version>

<scope>test</scope>

</dependency>

<!-- JUnit 5 Engine -->

<dependency>

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

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

<version>5.10.2</version>

<scope>test</scope>

</dependency>

<!-- Mockito Core -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.12.0</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**Exercise 2: Writing Basic JUnit Tests**

Scenario:

You need to write basic JUnit tests for a simple Java class.

Steps:

1. Create a new Java class with some methods to test.

2. Write JUnit tests for these methods.

**src/main/java**

**Calculator.java**  
  
**package** com.example;

**public** **class** Calculator {

**public** **int** add(**int** a, **int** b) {

**return** a + b;

}

**public** **int** subtract(**int** a, **int** b) {

**return** a - b;

}

}

**src/test/java**

**CalculatorTest.java**

package com.example;

import org.junit.jupiter.api.Test;

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

public class CalculatorTest {

@Test

void testAdd() {

Calculator calc = new Calculator();

assertEquals(5, calc.add(2, 3), "2 + 3 should be 5");

}

@Test

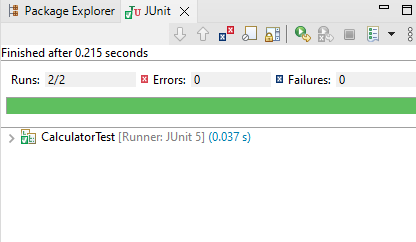
void testSubtract() {

Calculator calc = new Calculator();

assertEquals(1, calc.subtract(5, 4), "5 - 4 should be 1");

}

}

`

**Exercise 3: Assertions in JUnit**

Scenario:

You need to use different assertions in JUnit to validate your test results.

Steps: 1. Write tests using various JUnit assertions.

**src/test/java**

**AssertionsTest.java**

package com.example;

import org.junit.jupiter.api.Test;

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

public class AssertionsTest {

@Test

void testAssertions() {

// Assert equals

assertEquals(5, 2 + 3);

// Assert true

assertTrue(5 > 3);

// Assert false

assertFalse(5 < 3);

// Assert null

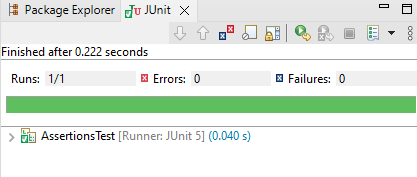
assertNull(null);

// Assert not null

assertNotNull(new Object());

}

}



**Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and**

**Teardown Methods in JUnit**

Scenario:

You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup

and teardown methods.

Steps:

1. Write tests using the AAA pattern.

2. Use @Before and @After annotations for setup and teardown methods

**src/test/java**

**CalculatorAAA.java**

package com.example;

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

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

public class CalculatorAAA {

private Calculator calculator;

@BeforeEach

void setUp() {

System.out.println("BeforeEach: Setting up calculator instance");

calculator = new Calculator();

}

@AfterEach

void tearDown() {

System.out.println("AfterEach: Tearing down calculator instance");

calculator = null;

}

@Test

void additionTest() {

System.out.println("Test: Running additionTest");

int result = calculator.add(10, 5);

assertEquals(15, result, "10 + 5 should be 15");

}

@Test

void subtractionTest() {

System.out.println("Test: Running subtractionTest");

int result = calculator.subtract(10, 5);

assertEquals(5, result, "10 - 5 should be 5");

}

}

