JUnit 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, add the following to your pom.xml: junit junit 4.13.2 test

3. Create a new test class in your project.

SOLUTION:

**Scenario:**

You are required to configure JUnit in your Java application to begin writing unit tests.

**Steps:**

1. Launch your preferred Java IDE and start a new Java project.
2. Open the pom.xml file and include the JUnit dependency as shown:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

3. Inside the src/test/java directory, create a new Java class file for your test cases. Assign it a name that reflects the class you intend to test, followed by Test. For example: CalculatorTest.java.

4. Define test methods in this class using the @Test annotation from JUnit.

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.

Solution Code:

public class AssertionsTest {

@Test

public 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());

}

}

SOLUTION:

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals(7, 4 + 3);

assertTrue("Java".contains("a"));

assertFalse(10 < 5);

assertNull(getNullValue());

assertNotNull("JUnit");

}

private Object getNullValue() {

return null;

}

}

OUTPUT:

Test passed: testAssertions

| **Assertion** | **Result** | **Reason** |
| --- | --- | --- |
| assertEquals(7, 4 + 3) | ✅ | 4 + 3 = 7, which matches expected value |
| "Java".contains("a") | ✅ | True — "a" is in "Java" |
| assertFalse(10 < 5) | ✅ | 10 is not less than 5 |
| assertNull(getNullValue()) | ✅ | Method returns null |
| assertNotNull("JUnit") | ✅ | "JUnit" is a non-null string |

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.

SOLUTION:

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

@Before

public void setUp() {

calculator = new Calculator();

}

@After

public void tearDown() {

calculator = null;

}

@Test

public void testAddition() {

int a = 10;

int b = 5;

int result = calculator.add(a, b);

assertEquals("Sum should be 15", 15, result);

}

@Test

public void testSubtraction() {

int a = 10;

int b = 4;

int result = calculator.subtract(a, b);

assertEquals("Difference should be 6", 6, result);

}

}

public class Calculator {

public int add(int x, int y) {

return x + y;

}

public int subtract(int x, int y) {

return x - y;

}

}

OUTPUT:

Tests passed: 2 of 2 tests

✔ testAddition

✔ testSubtraction