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:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

1. Create a new test class in your project.

**Basic Test Class**

java

CopyEdit

import org.junit.Test;

public class SampleTest {

@Test

public void sampleMethodTest() {

System.out.println("JUnit setup is working!");

}

}

# 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.

**Java Class to Test**

java

CopyEdit

public class Calculator {

public int add(int a, int b) { return a + b; }

public int subtract(int a, int b) { return a - b; }

public int multiply(int a, int b) { return a \* b; }

public int divide(int a, int b) {

if (b == 0) throw new IllegalArgumentException("Cannot divide by zero");

return a / b;

}

}

1. Write JUnit tests for these methods.

**JUnit Test Class**

java

CopyEdit

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

assertEquals(5, calc.add(2, 3));

}

@Test

public void testSubtract() {

Calculator calc = new Calculator();

assertEquals(1, calc.subtract(3, 2));

}

@Test

public void testMultiply() {

Calculator calc = new Calculator();

assertEquals(6, calc.multiply(2, 3));

}

@Test(expected = IllegalArgumentException.class)

public void testDivideByZero() {

Calculator calc = new Calculator();

calc.divide(5, 0);

}

}

# 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());

}

}

# 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.

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorAAATest {

private Calculator calc;

@Before

public void setUp() {

// Arrange: Create test fixture

calc = new Calculator();

System.out.println("Setup complete");

}

@After

public void tearDown() {

calc = null;

System.out.println("Teardown complete");

}

@Test

public void testAddition() {

// Act

int result = calc.add(2, 3);

// Assert

assertEquals(5, result);

}

@Test

public void testSubtraction() {

int result = calc.subtract(5, 3);

assertEquals(2, result);

}

}