Week 2 J UNIT

Exercise 1 :

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>

3. Create a new test class in your project.

how to do on eclipseide

**🔹 Step 1: Create Maven Project in VS Code**

1. Open VS Code
2. Open the **Command Palette**: Ctrl+Shift+P
3. Type: Maven: Create Maven Project
4. Choose quickstart template → Press Enter
5. Choose Java version (like 8 or 11)
6. Fill in:
   * **Group Id**: com.example
   * **Artifact Id**: JUnitDemo
7. Choose a folder to save the project
8. VS Code will generate a Maven project

**Step 2: Add JUnit to pom.xml**

1. Open pom.xml from the root of your project
2. Add this inside the <dependencies> tag:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

🔹 Step 3: Add Code to Test

Calculator.java

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

CalculatorTest.java

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

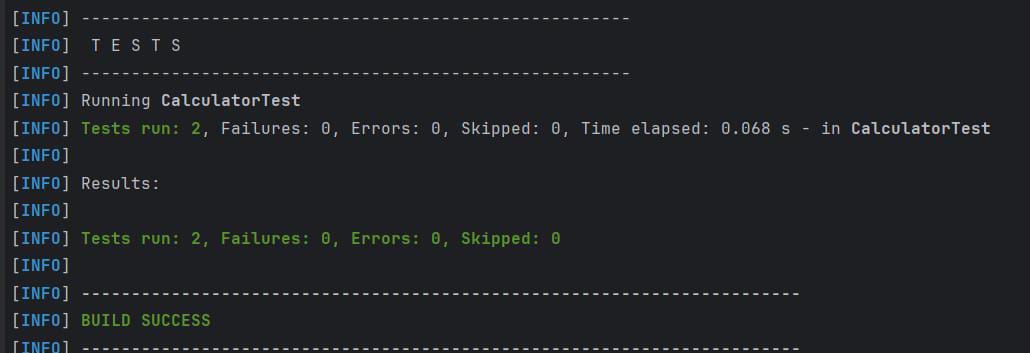
assertEquals(5, calc.add(2, 3));

}

}

🔹 Step 4: Run the JUnit Test

mvn test



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());

}

}

AssertionsTest.java

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*; // <-- Required for assertions

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());

}

}

Run by

mvn test

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.

Project Structure

JUnitDemo/

└── src

├── main/java/com/example/Calculator.java

└── test/java/com/example/CalculatorTest.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; }

public void clear() {

// In real calculators, clear state here

}

}

CalculatorTest.java

package com.example;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

// Setup (called before every test)

@Before

public void setUp() {

calculator = new Calculator(); // Arrange

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

}

// Teardown (called after every test)

@After

public void tearDown() {

calculator.clear(); // Clean-up

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

}

@Test

public void testAddition() {

// Arrange done in setUp()

// Act

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

// Assert

assertEquals(5, result);

}

@Test

public void testSubtraction() {

int result = calculator.subtract(10, 4);

assertEquals(6, result);

}

}

