Skill – TDD using JUnit5 and Mockito

(Unit Testing In Java)

# Exercise 1: Setting Up JUnit

## Purpose:

The purpose of this exercise is to set up JUnit in a Java project and write basic unit tests using JUnit framework. It enables developers to test individual units of code to ensure correctness and simplify debugging.

## junitdemo/pom.xml (CODE) :

<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 https://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 4 dependency -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

## Calculator.java (Code):

**package** com.example;

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

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

**return** a + b;

}

}

## CalculatorTest.java (Code) :

package junitdemo;

import static org.junit.Assert.\*;

import org.junit.Test;

import com.example.Calculator;

public class CalculatorTest {

@Test

public void testAddition() {

Calculator calculator = new Calculator();

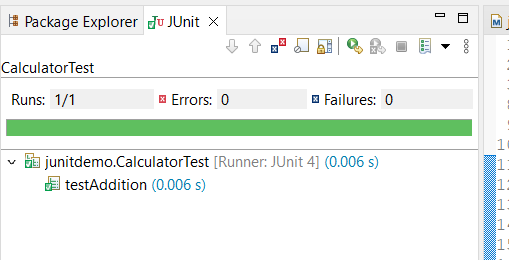
int result = calculator.add(2, 3);

assertEquals(5, result);

}

}

## Output:



# Exercise 3: Assertions in JUnit

## Purpose:

The purpose of this exercise is to demonstrate the usage of various assertions in JUnit, which are crucial for verifying that the program behaves as expected.

## Implementation:

### AssertionsTest.java ( Code ) :

**package com.example;**

**import org.junit.Test;**

**import static org.junit.Assert.\*;**

**public class AssertionsTest {**

**@Test**

**public void testAssertions() {**

**// Assert equality**

**assertEquals(5, 2 + 3);**

**// Assert that condition is true**

**assertTrue(5 > 3);**

**// Assert that condition is false**

**assertFalse(5 < 3);**

**// Assert that value is null**

**assertNull(null);**

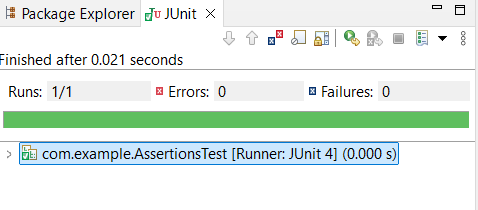
**// Assert that value is not null**

**assertNotNull(new Object());**

**}**

**}**

## Output:



## Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit

## Purpose:

The purpose of this exercise is to learn how to structure unit tests using the Arrange-Act-Assert (AAA) pattern and manage common setup and cleanup operations using JUnit's @Before and @After annotations. This ensures consistency, readability, and reusability in test code.

## Implementation:

## Calculate.java ( Code ) :

**package** com.example;

**public** **class** Calculate {

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

**return** a + b;

}

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

**return** a - b;

}

}

## CalculateTest.java ( Code ) :

**package** com.example;

**import** org.junit.Before;

**import** org.junit.After;

**import** org.junit.Test;

**import** **static** org.junit.Assert.\*;

**public** **class** CalculateTest {

**private** Calculate calculator;

// Setup before each test

@Before

**public** **void** setUp() {

calculator = **new** Calculate(); // Arrange

System.***out***.println("Setup: Calculator created");

}

// Teardown after each test

@After

**public** **void** tearDown() {

calculator = **null**;

System.***out***.println("Teardown: Calculator cleaned up");

}

@Test

**public** **void** testAddition() {

// Act

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

// Assert

*assertEquals*(5, result);

}

@Test

**public** **void** testSubtraction() {

// Act

**int** result = calculator.subtract(10, 7);

// Assert

*assertEquals*(3, result);

}

}

## Output:

