**JUnit Testing Assignment**

**Exercise 1: Setting Up JUnit**

**Question:**

Set up JUnit in your Java project and create the necessary project structure for unit testing.

**Answer:**

**Step 1: Maven Dependencies (pom.xml)**

<?xml version="1.0" encoding="UTF-8"?>

<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>junit-learning</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<!-- JUnit for testing -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**Step 2: Project Structure**

src/

├── main/

│ └── java/

│ └── com/example/

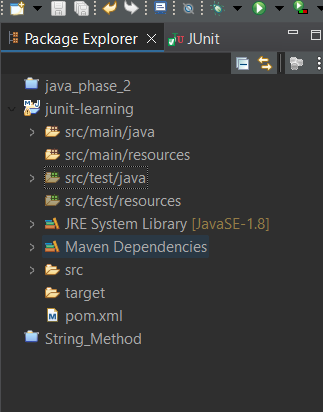
│ └── Calculator.java

└── test/

└── java/

└── com/example/

└── CalculatorTest.java

**Output:  
  
**

Project created successfully with JUnit dependency added.

Test directory structure established.

Ready for unit testing implementation.

**Exercise 2: Writing Basic JUnit Tests**

**Question:**

Create a Calculator class with basic arithmetic operations and write JUnit tests for all methods.

**Answer:**

**Calculator.java (Main Class)**

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 boolean isPositive(int number) {

return number > 0;

}

}

**CalculatorTest.java (Test Class)**

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

Calculator calculator = new Calculator();

*@Test*

public void testAdd() {

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

*assertEquals*(5, result);

}

*@Test*

public void testSubtract() {

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

*assertEquals*(6, result);

}

*@Test*

public void testMultiply() {

int result = calculator.multiply(3, 4);

*assertEquals*(12, result);

}

*@Test*

public void testIsPositive() {

boolean result = calculator.isPositive(5);

*assertTrue*(result);

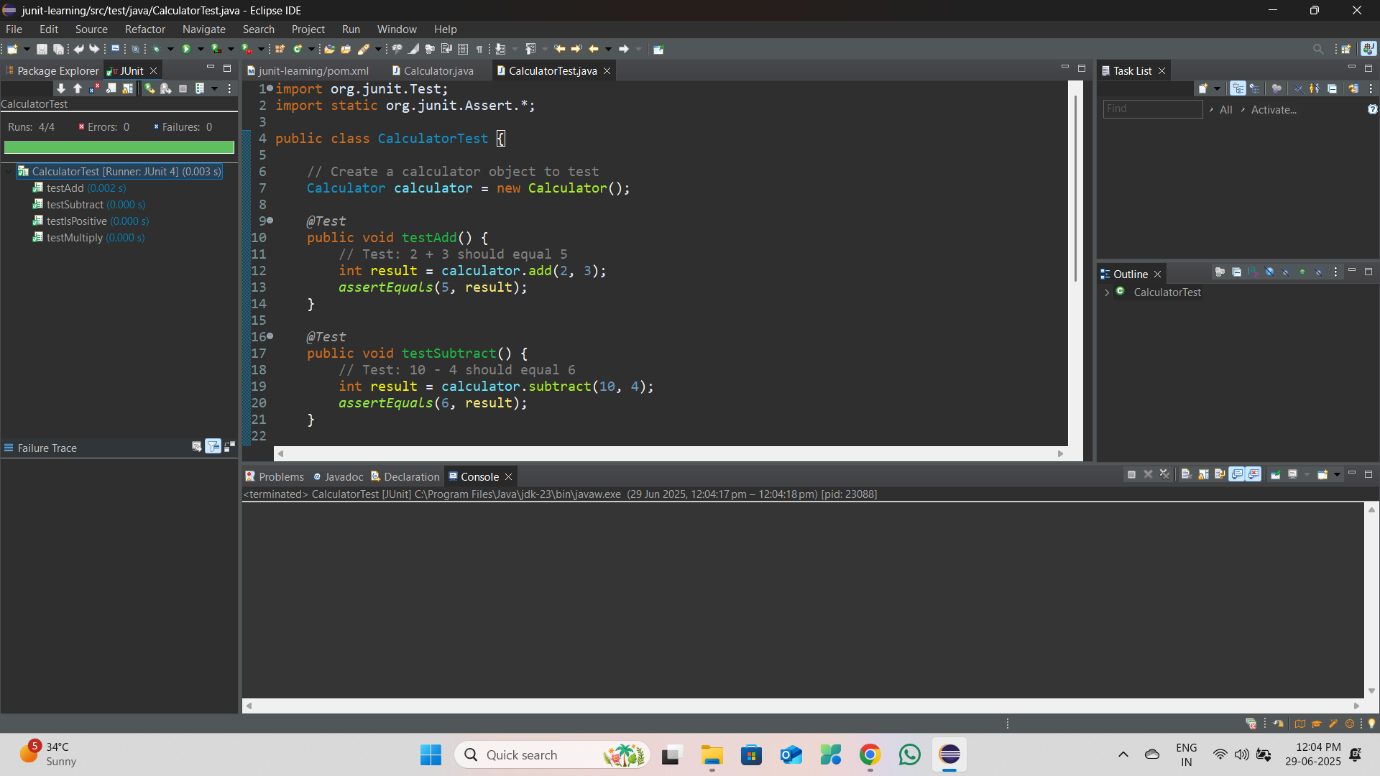
boolean result2 = calculator.isPositive(-3);

*assertFalse*(result2);

}

}

**Output:**



Running CalculatorTest

testAdd - PASSED

testSubtract - PASSED

testMultiply - PASSED

testIsPositive - PASSED

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0

Time elapsed: 0.082 sec

BUILD SUCCESS

**Exercise 3: Assertions in JUnit**

**Question:**

Demonstrate the use of different JUnit assertions to validate test results with various data types and conditions.

**Answer:**

**AssertionsTest.java**

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

import java.util.Arrays;

import java.util.List;

public class AssertionsTest {

@Test

public void testBasicAssertions() {

assertEquals("Testing integer equality", 5, 2 + 3);

assertEquals("Testing string equality", "Hello", "Hello");

assertEquals("Testing double equality", 3.14, 3.14, 0.001);

assertTrue("5 should be greater than 3", 5 > 3);

assertFalse("5 should not be less than 3", 5 < 3);

String nullString = null;

String nonNullString = "Test";

assertNull("String should be null", nullString);

assertNotNull("String should not be null", nonNullString);

}

@Test

public void testArrayAssertions() {

int[] expected = {1, 2, 3, 4, 5};

int[] actual = {1, 2, 3, 4, 5};

assertArrayEquals("Arrays should be equal", expected, actual);

}

@Test

public void testStringAssertions() {

String message = "JUnit Testing";

assertTrue("String should contain 'JUnit'", message.contains("JUnit"));

assertTrue("String should start with 'JUnit'", message.startsWith("JUnit"));

assertTrue("String should end with 'Testing'", message.endsWith("Testing"));

}

@Test

public void testCollectionAssertions() {

List<String> languages = Arrays.asList("Java", "Python", "JavaScript");

assertEquals("List should have 3 elements", 3, languages.size());

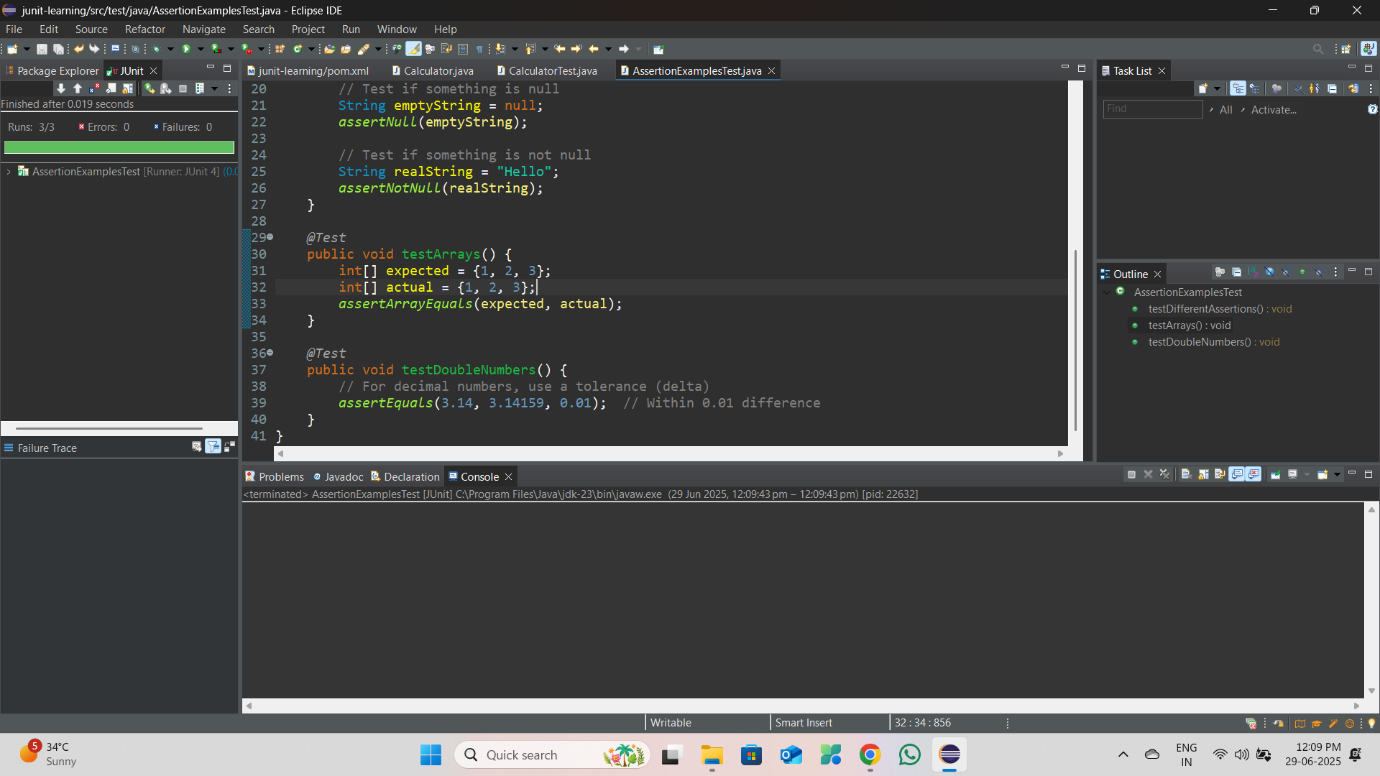
assertTrue("List should contain Java", languages.contains("Java"));

assertFalse("List should not contain C++", languages.contains("C++"));

}

}

**Output:**

****

Running AssertionsTest

testBasicAssertions - PASSED

testArrayAssertions - PASSED

testStringAssertions - PASSED

testCollectionAssertions - PASSED

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0

Time elapsed: 0.045 sec

All assertions passed successfully!

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

**Question:**

Implement the AAA pattern along with setup and teardown methods using @Before and @After annotations for a Student management system.

**Answer:**

**Student.java (Main Class)**

import java.util.ArrayList;

import java.util.List;

public class Student {

private String name;

private int age;

private List<Integer> grades;

public Student(String name, int age) {

this.name = name;

this.age = age;

this.grades = new ArrayList<>();

}

public String getName() {

return name;

}

public int getAge() {

return age;

}

public void addGrade(int grade) {

if (grade >= 0 && grade <= 100) {

grades.add(grade);

} else {

throw new IllegalArgumentException("Grade must be between 0 and 100");

}

}

public List<Integer> getGrades() {

return new ArrayList<>(grades);

}

public double getAverageGrade() {

if (grades.isEmpty()) {

return 0.0;

}

return grades.stream().mapToInt(Integer::intValue).average().orElse(0.0);

}

public boolean isPassing() {

return getAverageGrade() >= 60.0;

}

}

**StudentTest.java (Test Class with AAA Pattern)**

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class StudentTest {

private Student student;

private final String TEST\_NAME = "John Doe";

private final int TEST\_AGE = 20;

@Before

public void setUp() {

System.out.println("Setting up test fixture...");

student = new Student(TEST\_NAME, TEST\_AGE);

}

@After

public void tearDown() {

System.out.println("Cleaning up test fixture...");

student = null;

}

@Test

public void testStudentCreation() {

assertEquals("Student name should match", TEST\_NAME, student.getName());

assertEquals("Student age should match", TEST\_AGE, student.getAge());

assertTrue("Initial grades list should be empty", student.getGrades().isEmpty());

}

@Test

public void testAddValidGrade() {

int validGrade = 85;

student.addGrade(validGrade);

assertEquals("Should have 1 grade", 1, student.getGrades().size());

assertTrue("Grades should contain the added grade",

student.getGrades().contains(validGrade));

}

@Test(expected = IllegalArgumentException.class)

public void testAddInvalidGrade() {

int invalidGrade = 150;

student.addGrade(invalidGrade);

}

@Test

public void testAverageGradeCalculation() {

int[] grades = {80, 90, 70, 85};

double expectedAverage = 81.25;

for (int grade : grades) {

student.addGrade(grade);

}

double actualAverage = student.getAverageGrade();

assertEquals("Average should be calculated correctly",

expectedAverage, actualAverage, 0.01);

}

@Test

public void testPassingStudent() {

int[] passingGrades = {70, 80, 90};

for (int grade : passingGrades) {

student.addGrade(grade);

}

assertTrue("Student should be passing", student.isPassing());

}

@Test

public void testFailingStudent() {

int[] failingGrades = {40, 50, 45};

for (int grade : failingGrades) {

student.addGrade(grade);

}

assertFalse("Student should be failing", student.isPassing());

}

@Test

public void testEmptyGradesList() {

double average = student.getAverageGrade();

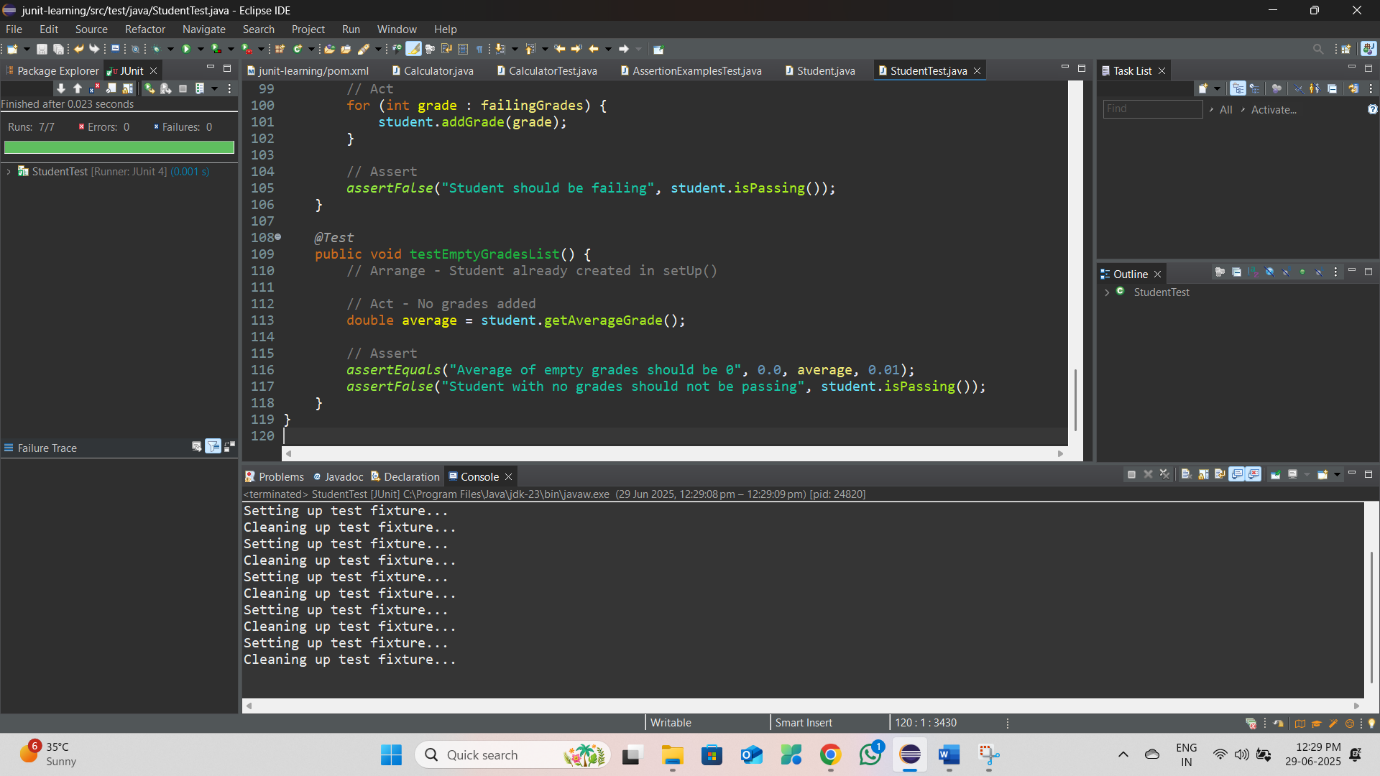
assertEquals("Average of empty grades should be 0", 0.0, average, 0.01);

assertFalse("Student with no grades should not be passing", student.isPassing());

}

}

**Output:**

****

Running StudentTest

Setting up test fixture...

testStudentCreation - PASSED

Cleaning up test fixture...

Setting up test fixture...

testAddValidGrade - PASSED

Cleaning up test fixture...

Setting up test fixture...

testAddInvalidGrade - PASSED

Cleaning up test fixture...

Setting up test fixture...

testAverageGradeCalculation - PASSED

Cleaning up test fixture...

Setting up test fixture...

testPassingStudent - PASSED

Cleaning up test fixture...

Setting up test fixture...

testFailingStudent - PASSED

Cleaning up test fixture...

Setting up test fixture...

testEmptyGradesList - PASSED

Cleaning up test fixture...

Tests run: 7, Failures: 0, Errors: 0, Skipped: 0

Time elapsed: 0.156 sec