**JUnit Testing**

**Exercise 1: Setting Up Junit**

**pom.xml :**

<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-setup-demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>junit-setup-demo</name>

<url>http://maven.apache.org</url>

<properties>

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

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**Calculator.java :**

package com.example.junit\_setup\_demo;

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;

}

}

**CalculatorTest.java:**

package com.example.junit\_setup\_demo;

import static org.junit.Assert.\*;

import org.junit.Test;

public class CalculatorTest {

private Calculator calculator = new Calculator();

@Test

public void testAdd() {

*assertEquals*("Addition should work", 5, calculator.add(2, 3));

}

@Test

public void testSubtract() {

*assertEquals*("Subtraction should work", 1, calculator.subtract(3, 2));

}

@Test

public void testMultiply() {

*assertEquals*("Multiplication should work", 6, calculator.multiply(2, 3));

}

@Test

public void testDivide() {

*assertEquals*("Division should work", 2, calculator.divide(6, 3));

}

@Test(expected = IllegalArgumentException.class)

public void testDivideByZero() {

calculator.divide(5, 0);

}

@Test

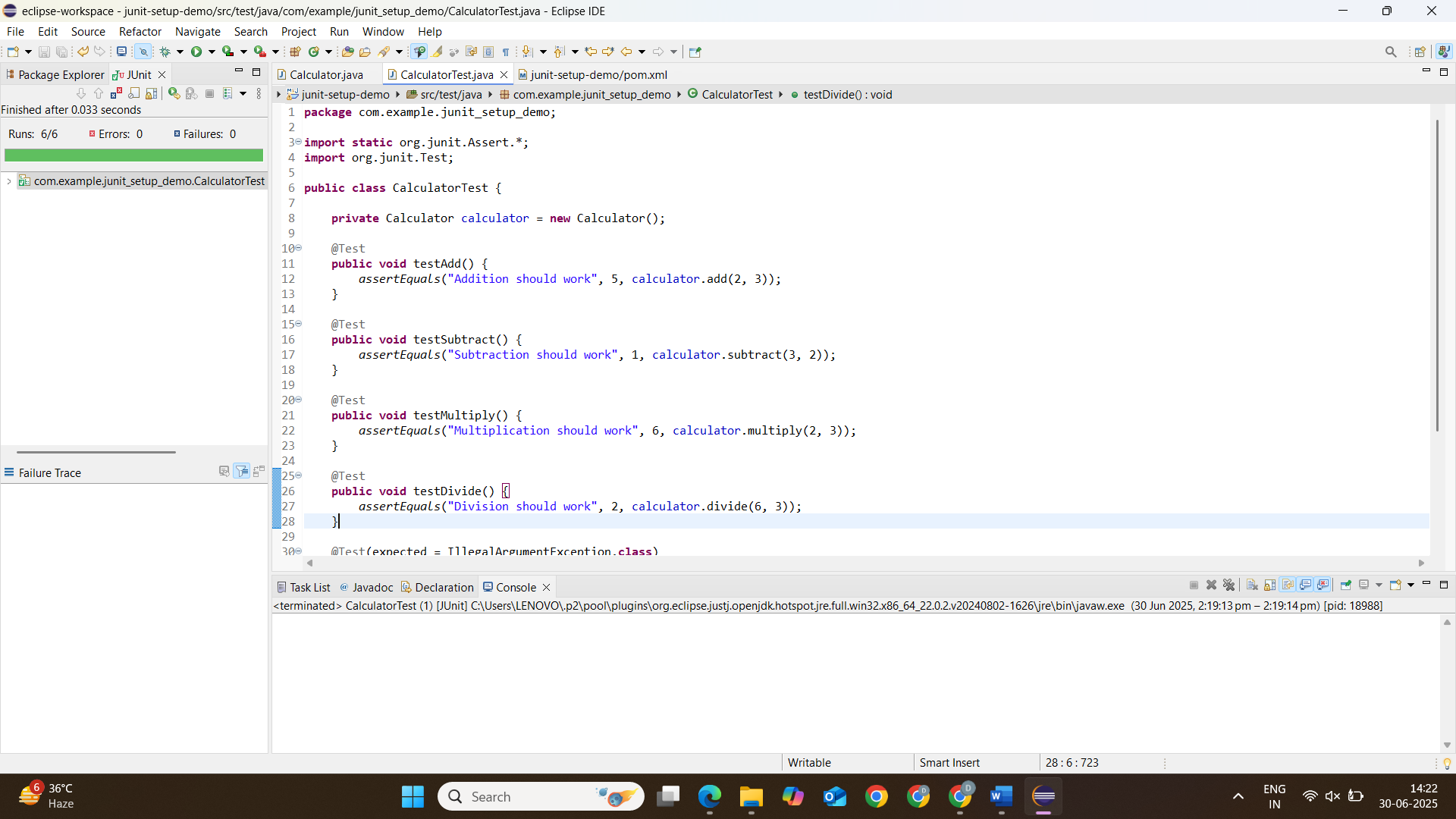
public void testMultipleOperations() {

*assertTrue*("Combined operations test",

calculator.add(calculator.multiply(2, 3), calculator.subtract(5, 1)) == 10);

}

}



**Exercise 3: Assertions in Junit**

**pom.xml :**

<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>JunitAsserts</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>JunitAsserts</name>

<url>http://maven.apache.org</url>

<properties>

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

</properties>

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.8.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**AssertionsTest.java :**

package com.example.JunitAsserts;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.Test;

class AssertionsTest {

@Test

void testBasicAssertions() {

// Assert equals

*assertEquals*(5, 2 + 3, "2 + 3 should equal 5");

// Assert true

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

// Assert false

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

// Assert null

String nullString = null;

*assertNull*(nullString, "The object should be null");

// Assert not null

*assertNotNull*(new Object(), "The object should not be null");

}

@Test

void testArrayAssertions() {

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

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

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

}

@Test

void testExceptionAssertions() {

// Verify exception is thrown

*assertThrows*(ArithmeticException.class, () -> {

int result = 10 / 0;

}, "Should throw ArithmeticException");

}

@Test

void testAllAssertions() {

// Group multiple assertions

*assertAll*(

() -> *assertEquals*(4, 2 \* 2),

() -> *assertTrue*(10 > 5),

() -> *assertNotNull*("Hello")

);

}

@Test

void testAdvancedAssertions() {

String message = "Hello JUnit";

// Multiple assertions on same object

*assertAll*("message",

() -> *assertNotNull*(message),

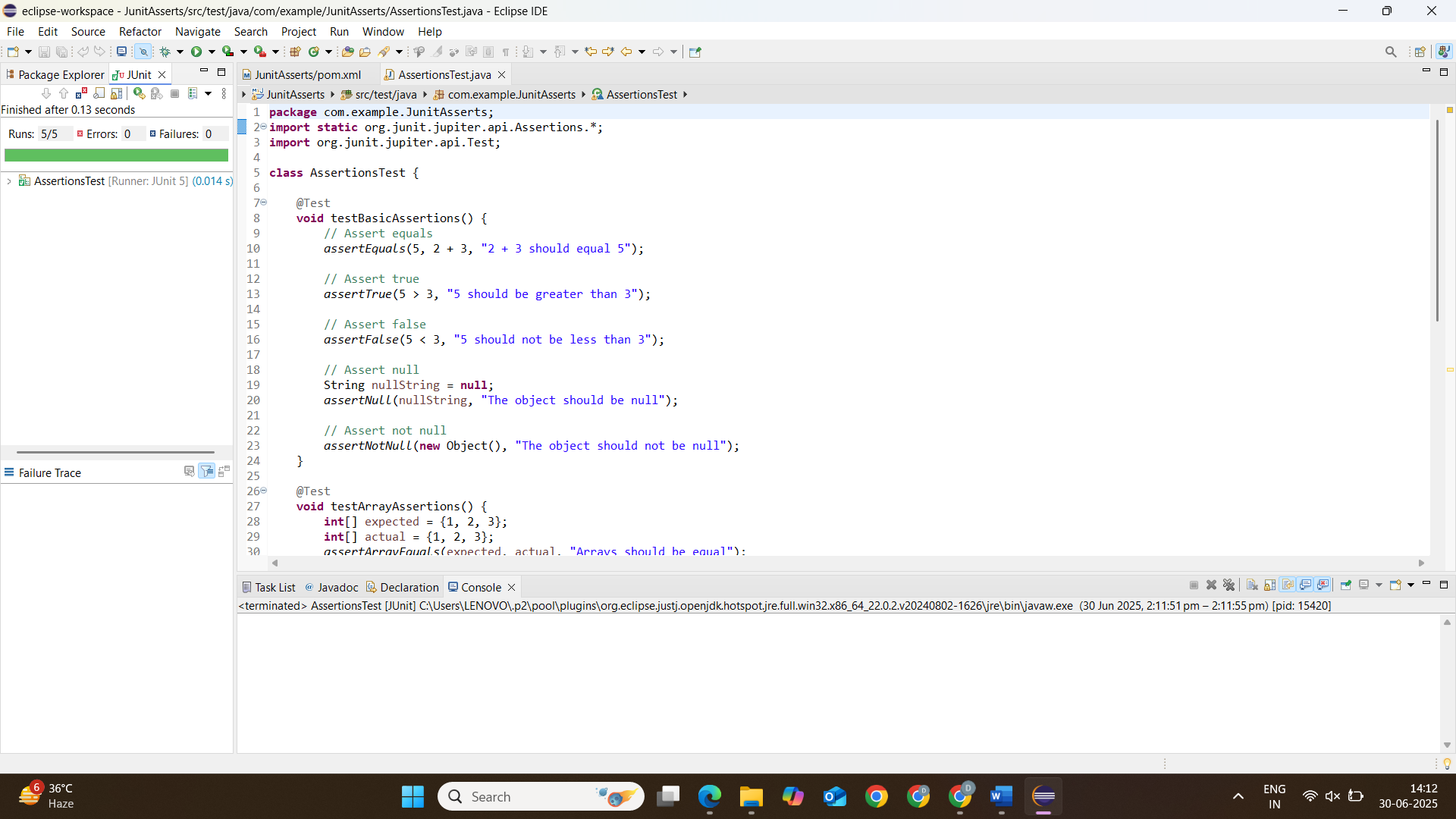
() -> *assertEquals*("Hello JUnit", message),

() -> *assertTrue*(message.startsWith("Hello"))

);

}

}



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

**pom.xml :**

<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>JunitAssertionDemo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>JunitAssertionDemo</name>

<url>http://maven.apache.org</url>

<properties>

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

</properties>

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.8.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**BankAccount.java :**

package com.example.JunitAssertionDemo;

public class BankAccount {

private String accountNumber;

private String accountHolder;

private double balance;

private boolean isActive;

public BankAccount(String accountNumber, String accountHolder) {

this.accountNumber = accountNumber;

this.accountHolder = accountHolder;

this.balance = 0.0;

this.isActive = true;

}

// Add these getter methods

public String getAccountNumber() {

return accountNumber;

}

public String getAccountHolder() {

return accountHolder;

}

public void deposit(double amount) {

if (amount <= 0) {

throw new IllegalArgumentException("Deposit amount must be positive");

}

balance += amount;

}

public void withdraw(double amount) {

if (amount <= 0) {

throw new IllegalArgumentException("Withdrawal amount must be positive");

}

if (amount > balance) {

throw new IllegalStateException("Insufficient funds");

}

balance -= amount;

}

public void transfer(BankAccount destination, double amount) {

this.withdraw(amount);

destination.deposit(amount);

}

public void deactivate() {

isActive = false;

}

public double getBalance() {

return balance;

}

public boolean isActive() {

return isActive;

}

}

**BankAccountTest.java :**

package com.example.JunitAssertionDemo;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

class BankAccountTest {

private BankAccount account;

private BankAccount recipientAccount;

private static final String *TEST\_ACCOUNT\_NUMBER* = "123456789";

private static final String *TEST\_ACCOUNT\_HOLDER* = "John Doe";

private static final String *RECIPIENT\_ACCOUNT\_NUMBER* = "987654321";

private static final String *RECIPIENT\_ACCOUNT\_HOLDER* = "Jane Smith";

@BeforeEach

void setUp() {

// Arrange - Common setup for all tests

account = new BankAccount(*TEST\_ACCOUNT\_NUMBER*, *TEST\_ACCOUNT\_HOLDER*);

recipientAccount = new BankAccount(*RECIPIENT\_ACCOUNT\_NUMBER*, *RECIPIENT\_ACCOUNT\_HOLDER*);

account.deposit(1000.0); // Initial balance for most tests

}

@AfterEach

void tearDown() {

account = null;

recipientAccount = null;

}

@Test

void testInitialState() {

// Arrange - Done in setUp()

// Act - Checking initial state

// Assert

*assertEquals*(*TEST\_ACCOUNT\_NUMBER*, account.getAccountNumber());

*assertEquals*(*TEST\_ACCOUNT\_HOLDER*, account.getAccountHolder());

*assertEquals*(1000.0, account.getBalance(), 0.001);

*assertTrue*(account.isActive());

}

@Test

void testDepositPositiveAmount() {

// Arrange - Initial state set in setUp()

double depositAmount = 500.0;

double expectedBalance = 1500.0;

// Act

account.deposit(depositAmount);

// Assert

*assertEquals*(expectedBalance, account.getBalance(), 0.001);

}

@Test

void testDepositNegativeAmountThrowsException() {

// Arrange

double invalidAmount = -100.0;

// Act & Assert

*assertThrows*(IllegalArgumentException.class, () -> {

account.deposit(invalidAmount);

});

}

@Test

void testWithdrawSufficientFunds() {

// Arrange

double withdrawalAmount = 300.0;

double expectedBalance = 700.0;

// Act

account.withdraw(withdrawalAmount);

// Assert

*assertEquals*(expectedBalance, account.getBalance(), 0.001);

}

@Test

void testWithdrawInsufficientFundsThrowsException() {

// Arrange

double excessiveAmount = 1500.0;

// Act & Assert

*assertThrows*(IllegalStateException.class, () -> {

account.withdraw(excessiveAmount);

});

}

@Test

void testTransferBetweenAccounts() {

// Arrange

double transferAmount = 200.0;

double expectedSourceBalance = 800.0;

double expectedRecipientBalance = 200.0;

// Act

account.transfer(recipientAccount, transferAmount);

// Assert

*assertEquals*(expectedSourceBalance, account.getBalance(), 0.001);

*assertEquals*(expectedRecipientBalance, recipientAccount.getBalance(), 0.001);

}

@Test

void testDeactivateAccount() {

// Arrange - Initial state set in setUp();

// Act

account.deactivate();

// Assert

*assertFalse*(account.isActive());

}

}

