**WEEK – 2**

**PLSQL\_Exercises**

**Exercise 1: Control Structures**

-- DROP existing tables if rerunning

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE loans';

EXECUTE IMMEDIATE 'DROP TABLE customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- Create customers table

CREATE TABLE customers (

cust\_id NUMBER PRIMARY KEY,

name VARCHAR2(50),

age NUMBER,

balance NUMBER,

isvip CHAR(1) DEFAULT 'N'

);

-- Create loans table

CREATE TABLE loans (

loan\_id NUMBER PRIMARY KEY,

cust\_id NUMBER,

interest\_rate NUMBER(5,2),

due\_date DATE

);

-- Insert sample data

INSERT INTO customers VALUES (1, 'Alice', 65, 15000, 'N');

INSERT INTO customers VALUES (2, 'Bob', 45, 8000, 'N');

INSERT INTO customers VALUES (3, 'Charlie', 70, 5000, 'N');

INSERT INTO loans VALUES (101, 1, 5.5, SYSDATE + 10);

INSERT INTO loans VALUES (102, 2, 6.0, SYSDATE + 40);

INSERT INTO loans VALUES (103, 3, 7.2, SYSDATE + 20);

COMMIT;

BEGIN

FOR rec IN (SELECT \* FROM customers WHERE age > 60) LOOP

UPDATE loans

SET interest\_rate = interest\_rate - (interest\_rate \* 0.01)

WHERE cust\_id = rec.cust\_id;

DBMS\_OUTPUT.PUT\_LINE('Discount applied to Loan(s) of Customer ID: ' || rec.cust\_id);

END LOOP;

END;

/

BEGIN

FOR rec IN (SELECT \* FROM customers WHERE balance > 10000) LOOP

UPDATE customers

SET isvip = 'Y'

WHERE cust\_id = rec.cust\_id;

DBMS\_OUTPUT.PUT\_LINE('Customer ' || rec.name || ' promoted to VIP');

END LOOP;

END;

/

BEGIN

FOR rec IN (

SELECT l.loan\_id, c.name, l.due\_date

FROM loans l

JOIN customers c ON l.cust\_id = c.cust\_id

WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

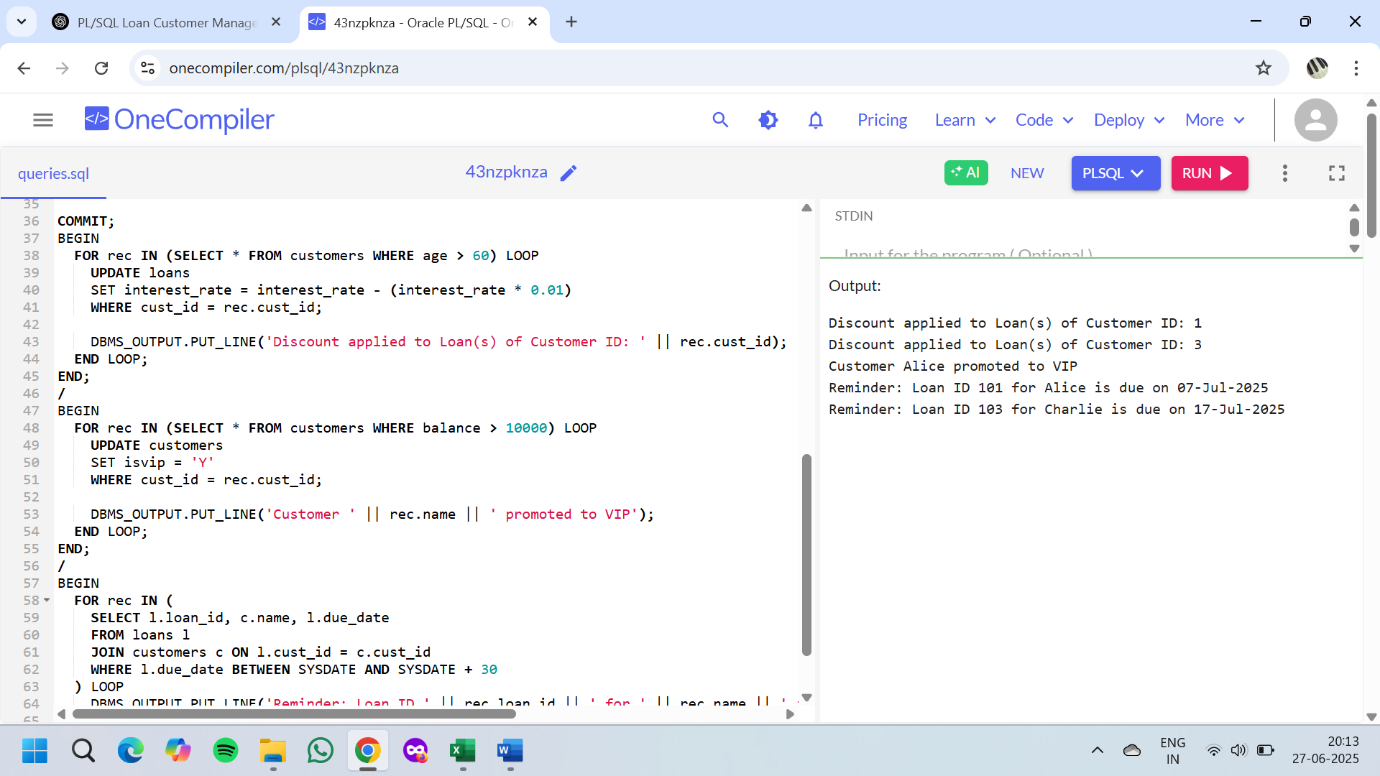
DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.loan\_id || ' for ' || rec.name || ' is due on ' || TO\_CHAR(rec.due\_date, 'DD-Mon-YYYY'));

END LOOP;

END;

/

**OUTPUT :**

****

**Exercise 3: Stored Procedures**

-- Enable DBMS\_OUTPUT

SET SERVEROUTPUT ON;

-- Drop old tables

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE savings\_accounts';

EXECUTE IMMEDIATE 'DROP TABLE employees';

EXECUTE IMMEDIATE 'DROP TABLE accounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- Create tables

CREATE TABLE savings\_accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

balance NUMBER(10,2)

);

CREATE TABLE employees (

emp\_id NUMBER PRIMARY KEY,

name VARCHAR2(50),

department VARCHAR2(30),

salary NUMBER(10,2)

);

CREATE TABLE accounts (

acc\_no NUMBER PRIMARY KEY,

customer\_id NUMBER,

balance NUMBER(10,2)

);

-- Insert sample data

INSERT INTO savings\_accounts VALUES (1, 101, 1000);

INSERT INTO savings\_accounts VALUES (2, 102, 2000);

INSERT INTO employees VALUES (1, 'Alice', 'Sales', 50000);

INSERT INTO employees VALUES (2, 'Bob', 'Sales', 60000);

INSERT INTO employees VALUES (3, 'Charlie', 'HR', 55000);

INSERT INTO accounts VALUES (111, 201, 5000);

INSERT INTO accounts VALUES (222, 202, 3000);

COMMIT;

-- Scenario 1: ProcessMonthlyInterest Procedure

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR rec IN (SELECT \* FROM savings\_accounts) LOOP

UPDATE savings\_accounts

SET balance = balance + (balance \* 0.01)

WHERE account\_id = rec.account\_id;

DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID: ' || rec.account\_id);

END LOOP;

END;

/

-- Call the procedure

BEGIN

ProcessMonthlyInterest;

END;

/

-- Scenario 2: UpdateEmployeeBonus Procedure

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_name IN VARCHAR2,

bonus\_pct IN NUMBER

) IS

BEGIN

FOR rec IN (SELECT \* FROM employees WHERE department = dept\_name) LOOP

UPDATE employees

SET salary = salary + (salary \* bonus\_pct / 100)

WHERE emp\_id = rec.emp\_id;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to Employee ID: ' || rec.emp\_id);

END LOOP;

END;

/

-- Call the procedure

BEGIN

UpdateEmployeeBonus('Sales', 10);

END;

/

-- Scenario 3: TransferFunds Procedure

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_acc IN NUMBER,

to\_acc IN NUMBER,

amount IN NUMBER

) IS

from\_balance NUMBER;

BEGIN

SELECT balance INTO from\_balance FROM accounts WHERE acc\_no = from\_acc;

IF from\_balance < amount THEN

DBMS\_OUTPUT.PUT\_LINE('Insufficient funds in account: ' || from\_acc);

ELSE

UPDATE accounts SET balance = balance - amount WHERE acc\_no = from\_acc;

UPDATE accounts SET balance = balance + amount WHERE acc\_no = to\_acc;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || amount || ' from ' || from\_acc || ' to ' || to\_acc);

END IF;

END;

/

-- Call the procedure

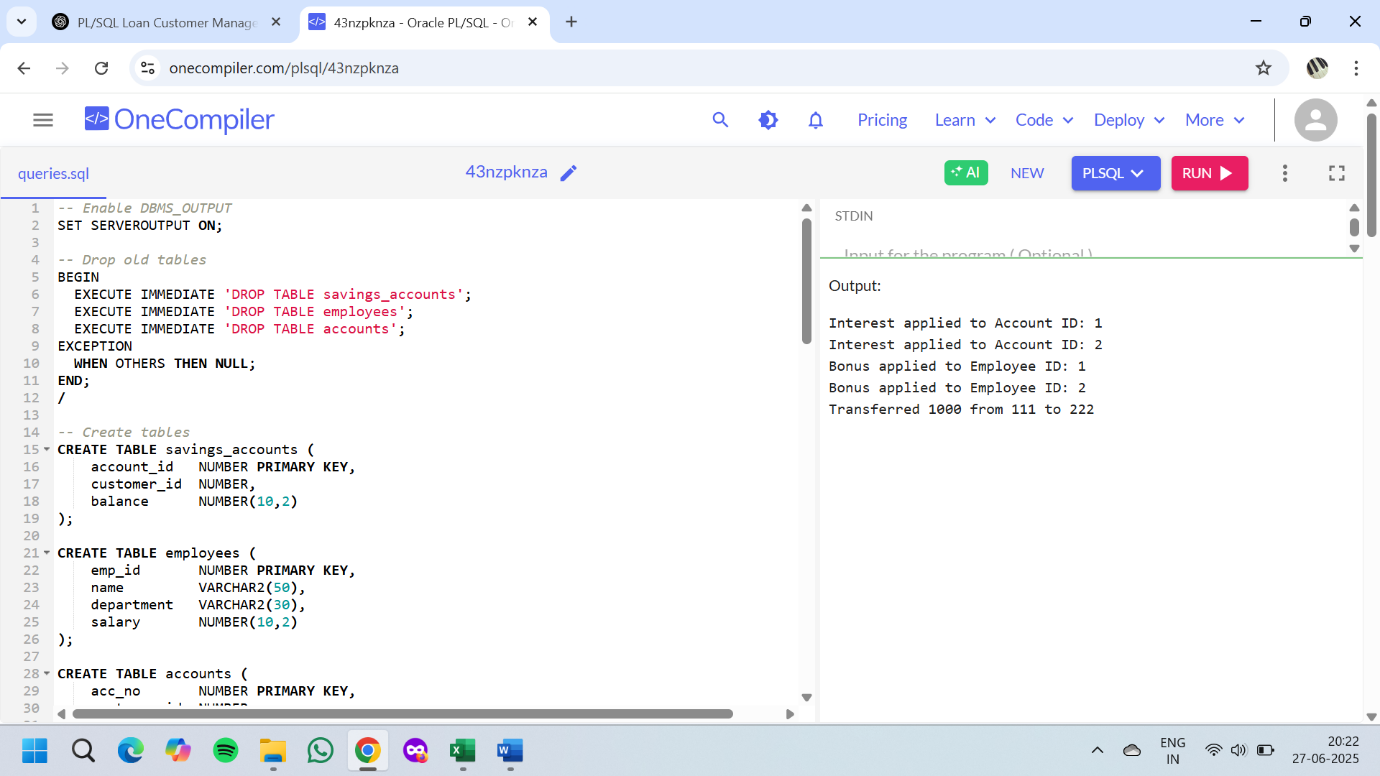
BEGIN

TransferFunds(111, 222, 1000);

END;

/

**OUTPUT:**



**JUnit\_Basic Testing Exercises**

**Exercise 1: Setting Up JUnit**

**File Name: Calculator.java**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

}

**File Name: CalculatorTest.java**

package calculator;

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 testMultiply() {

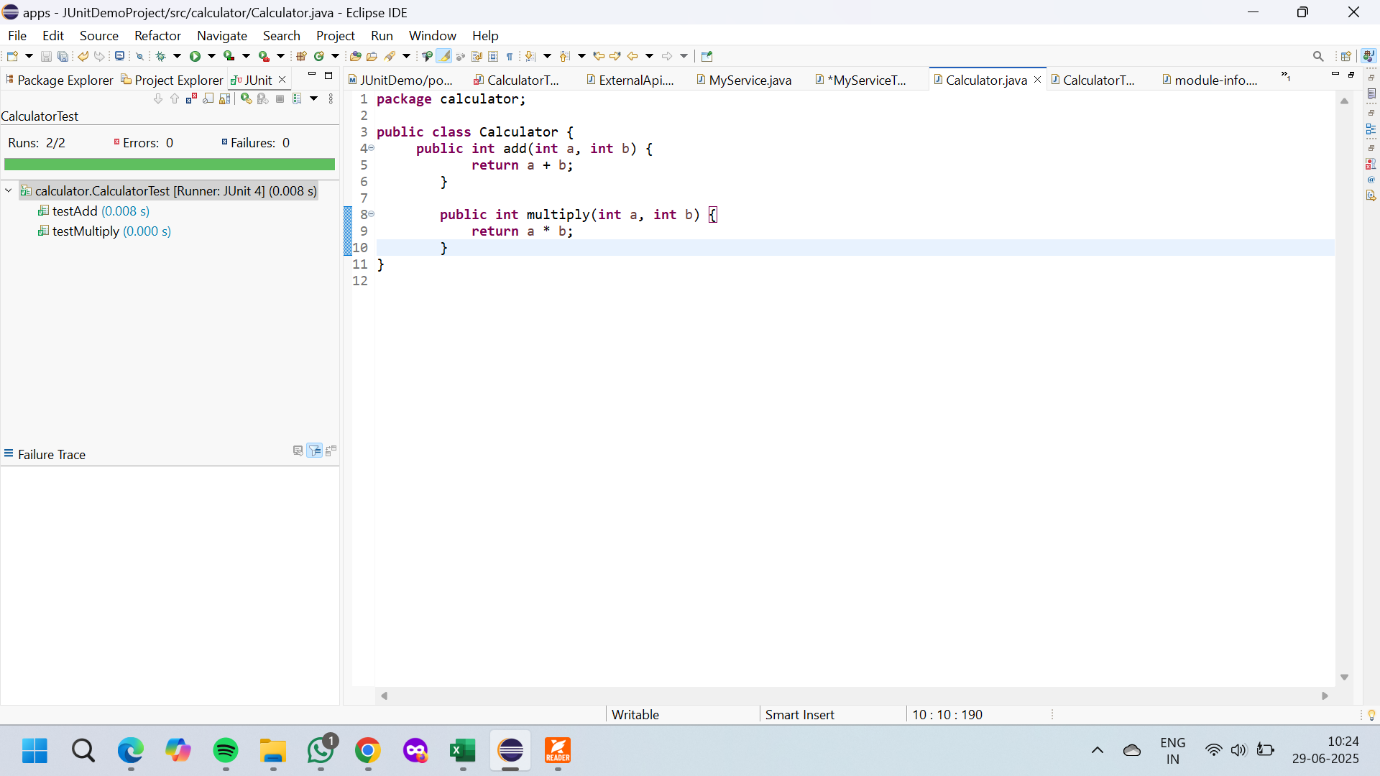
Calculator calc = new Calculator();

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

}

}

**OUTPUT:**

****

**Exercise 3: Assertions in Junit**

**File Name:AssertionsType.java**

package calculator;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsType {

@Test

public void testAssertions() {

// assertEquals

assertEquals(5, 2 + 3);

// assertTrue

assertTrue(5 > 3);

// assertFalse

assertFalse(5 < 3);

// assertNull

assertNull(null);

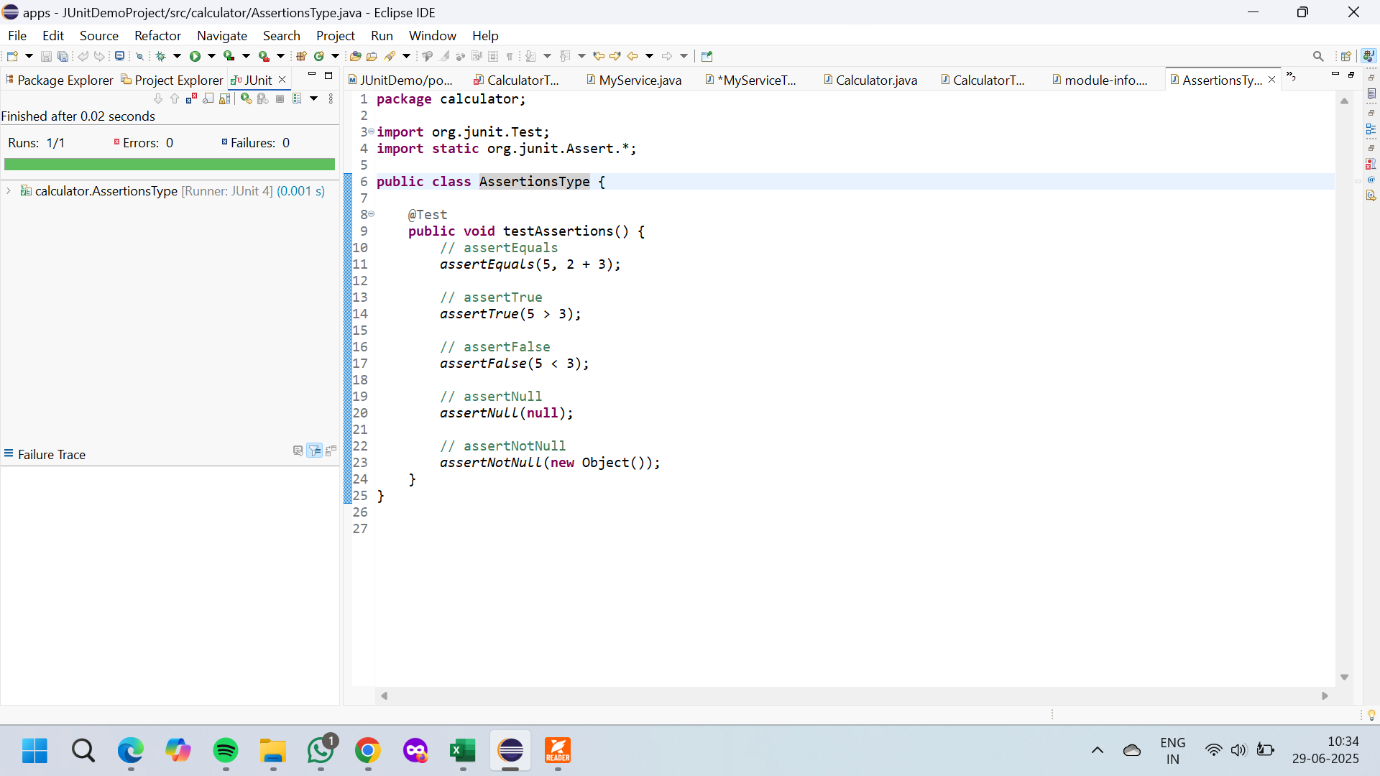
// assertNotNull

assertNotNull(new Object());

}

}

**OUTPUT:**

****

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

**File Name: Calculator.java**

package calculator;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

}

**File Name: CalculatorTest.java**

package calculator;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

// Setup method (runs before each test)

@Before

public void setUp() {

calculator = new Calculator(); // Arrange

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

}

// Teardown method (runs after each test)

@After

public void tearDown() {

calculator = null;

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

}

@Test

public void testAdd() {

// Arrange - done in setUp()

// Act

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

// Assert

assertEquals(5, result);

}

@Test

public void testMultiply() {

// Arrange - done in setUp()

// Act

int result = calculator.multiply(2, 4);

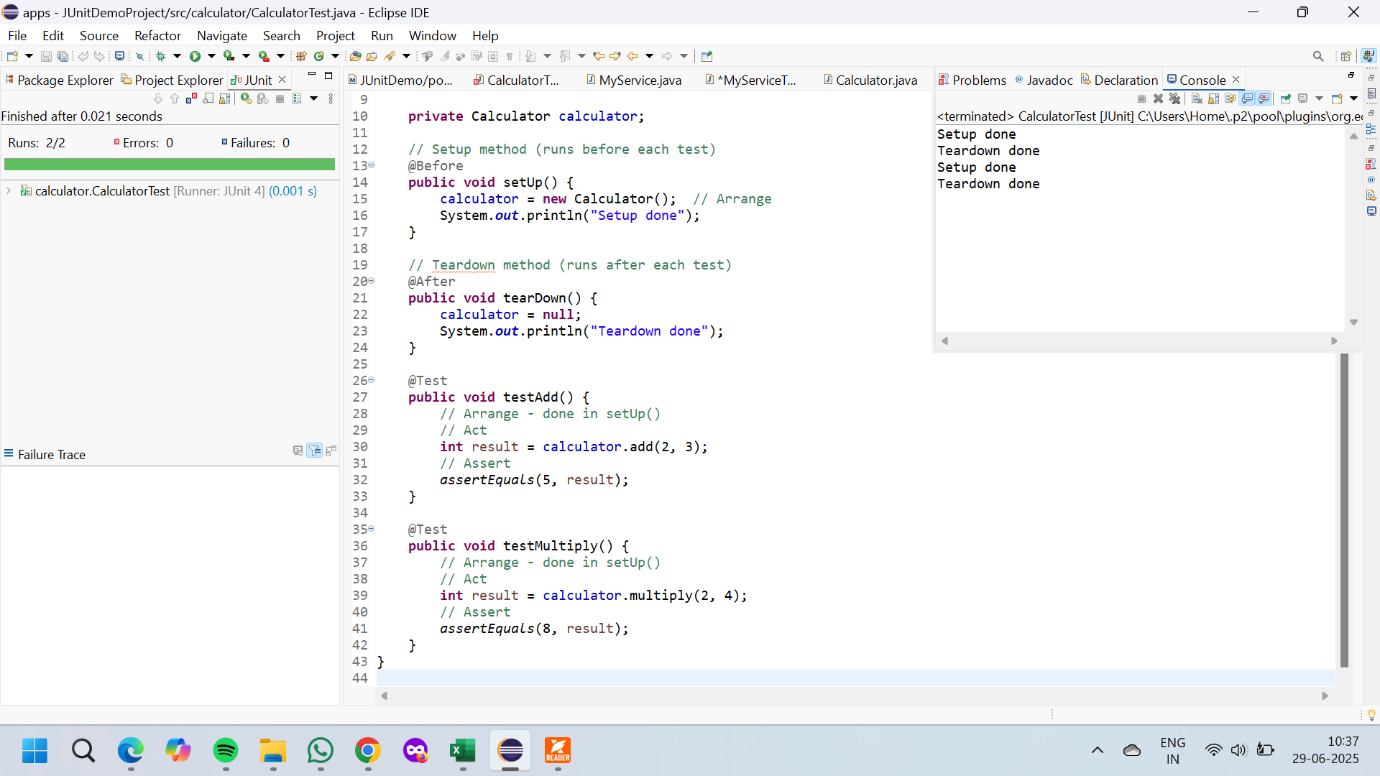
// Assert

assertEquals(8, result);

}

}

**OUTPUT:**

****

**Mockito exercises**

**Exercise 1: Mocking and Stubbing**

**File Name:ExternalApi.java**

package com.example.mocktest;

public interface ExternalApi {

String getData();

}

**File Name:MyService.java**

package com.example.mocktest;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**File Name:MyServiceTest.java**

package com.example.mocktest;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = mock(ExternalApi.class);

when(mockApi.getData()).thenReturn("Mock Data");

MyService service = new MyService(mockApi);

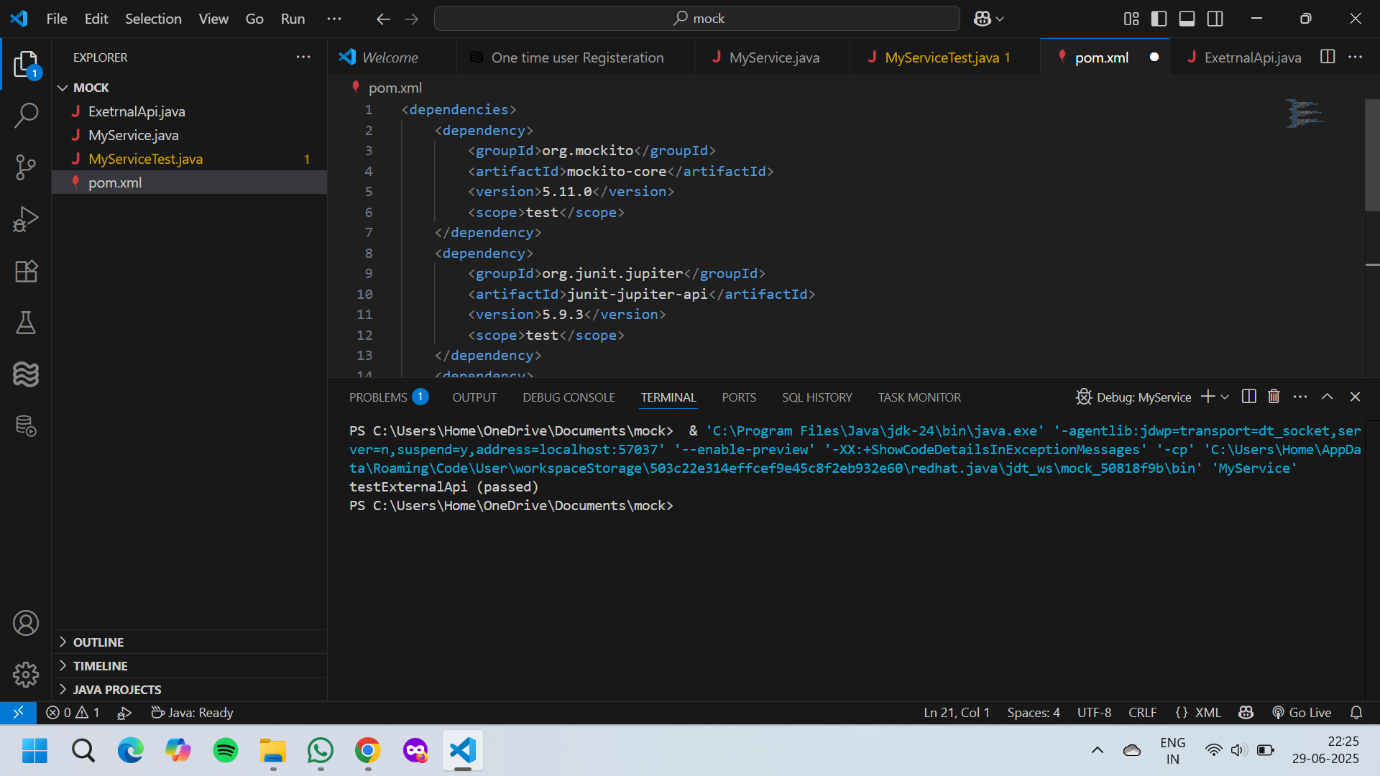
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**OUTPUT:**

****

**Exercise 2: Verifying Interactions**

**File Name:ExternalApi.java**

package com.example.mockverify;

public interface ExternalApi {

String getData();

}

**File Name:MyService.java**

package com.example.mockverify;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**File Name:MyServiceTest.java**

package com.example.mockverify;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

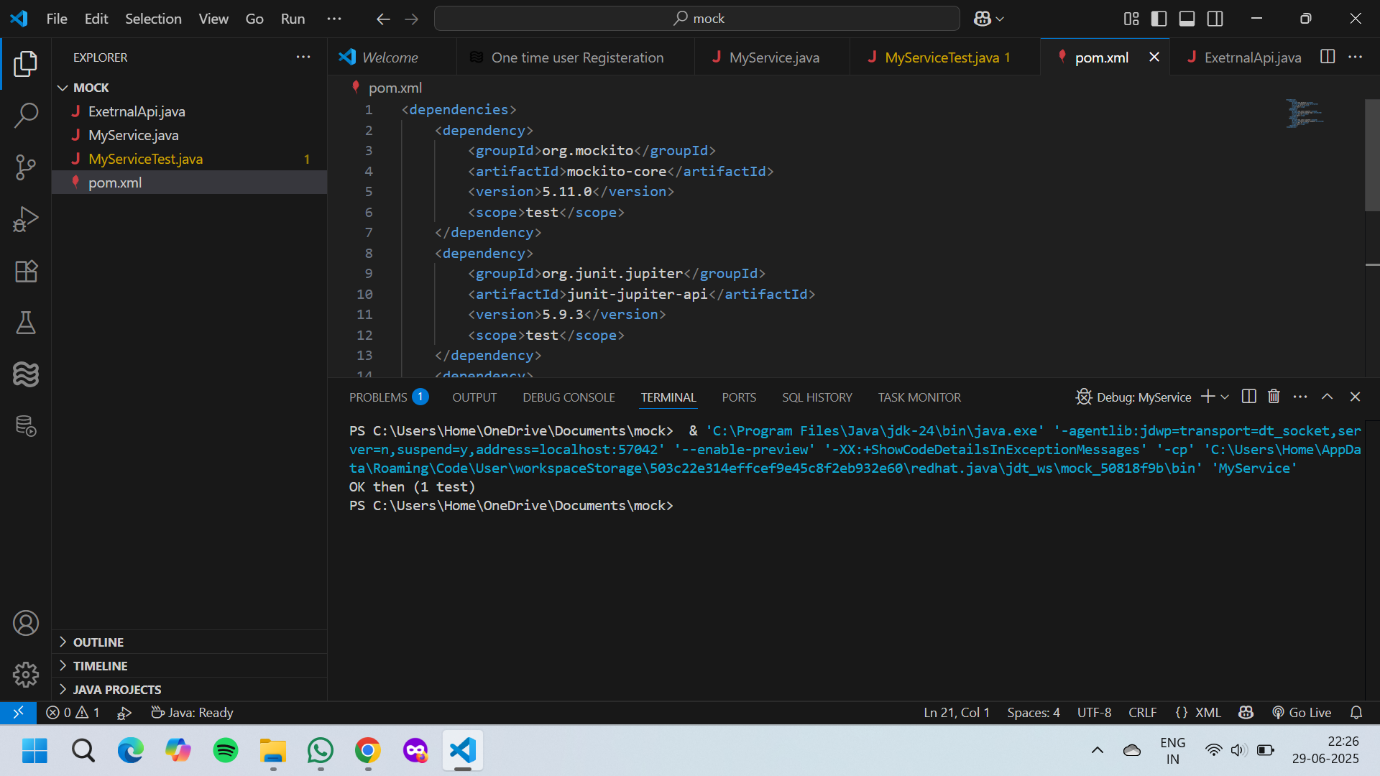
// Verify if getData() was called exactly once

verify(mockApi).getData();

}

}

**OUTPUT:**

****

**Spring Core\_Maven**

**Exercise 1: Logging Error Messages and Warning Levels**

**File Name:LoggingExample.java**

public class LoggingExample {

public static void main(String[] args) {

logError("This is an error message");

logWarning("This is a warning message");

}

public static void logError(String message) {

System.out.println("[ERROR] " + message);

}

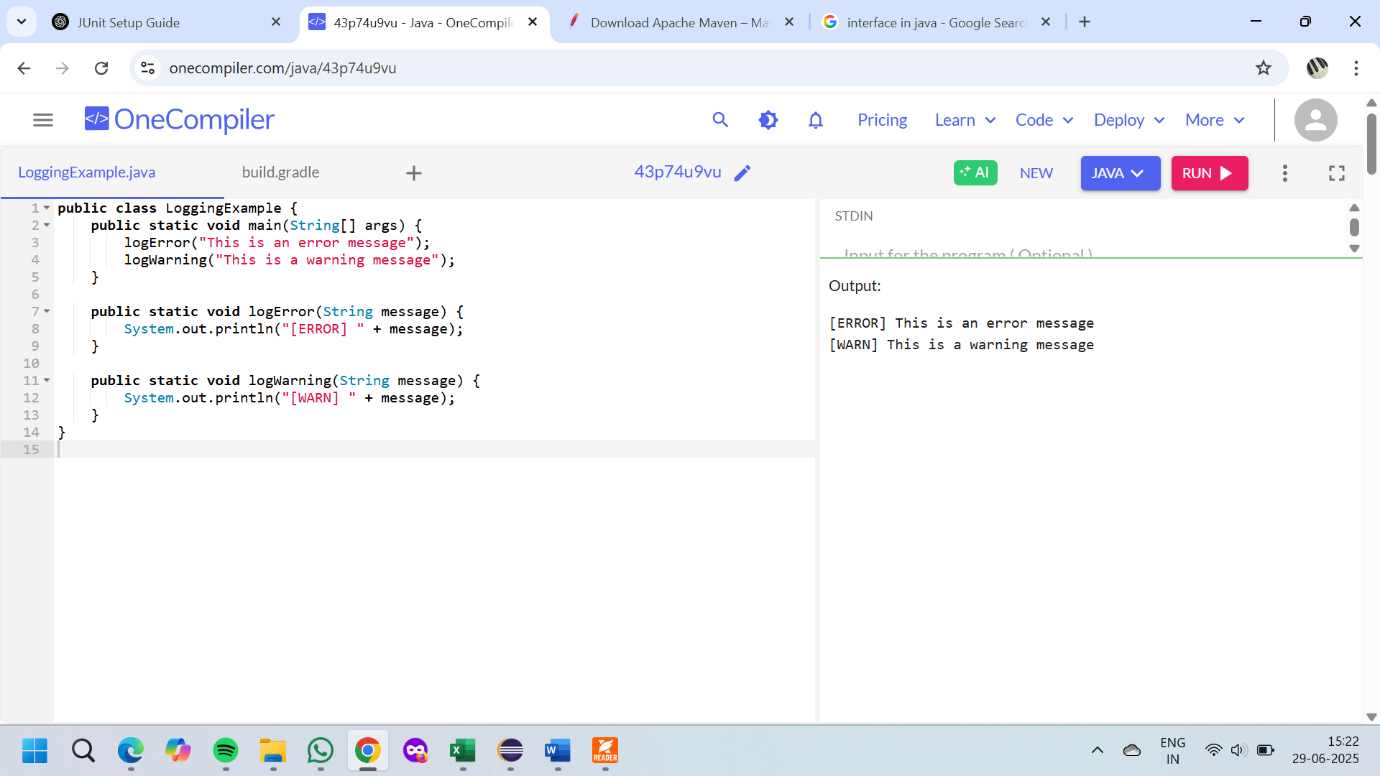
public static void logWarning(String message) {

System.out.println("[WARN] " + message);

}

}

**OUTPUT:**

****