**WEEK 2**

**PL/SQL - EXERCISE 1:**

**CODE:**

-- DROP TABLES IF THEY ALREADY EXIST

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions';

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- SCHEMA CREATION

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE,

IsVIP VARCHAR2(5)

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

-- SAMPLE DATA INSERTION

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified, IsVIP)

VALUES (1, 'John Doe', TO\_DATE('1950-05-15', 'YYYY-MM-DD'), 12000, SYSDATE, 'FALSE');

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified, IsVIP)

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 9000, SYSDATE, 'FALSE');

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 12000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 9000, SYSDATE);

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 8000, 6, SYSDATE, SYSDATE + 20); -- Due in 20 days

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

-- ENABLE OUTPUT

SET SERVEROUTPUT ON;

**-- SCENARIO 1: Interest discount for customers > 60**

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Scenario 1: Interest Discount for Seniors ---');

FOR c IN (

SELECT CustomerID, Name, DOB, TRUNC(MONTHS\_BETWEEN(SYSDATE, DOB) / 12) AS Age

FROM Customers

) LOOP

IF c.Age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = c.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Interest rate discounted for: ' || c.Name || ' (Age: ' || c.Age || ')');

END IF;

END LOOP;

END;

/

**-- SCENARIO 2: VIP status for high balance customers**

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Scenario 2: VIP Status Update ---');

FOR r IN (

SELECT CustomerID, Name, Balance FROM Customers

) LOOP

IF r.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = r.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Promoted to VIP: ' || r.Name || ' (Balance: $' || r.Balance || ')');

END IF;

END LOOP;

END;

/

**-- SCENARIO 3: Loan reminders for upcoming due dates**

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Scenario 3: Loan Due Reminders ---');

FOR l IN (

SELECT l.LoanID, l.CustomerID, l.EndDate, c.Name

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate <= SYSDATE + 30

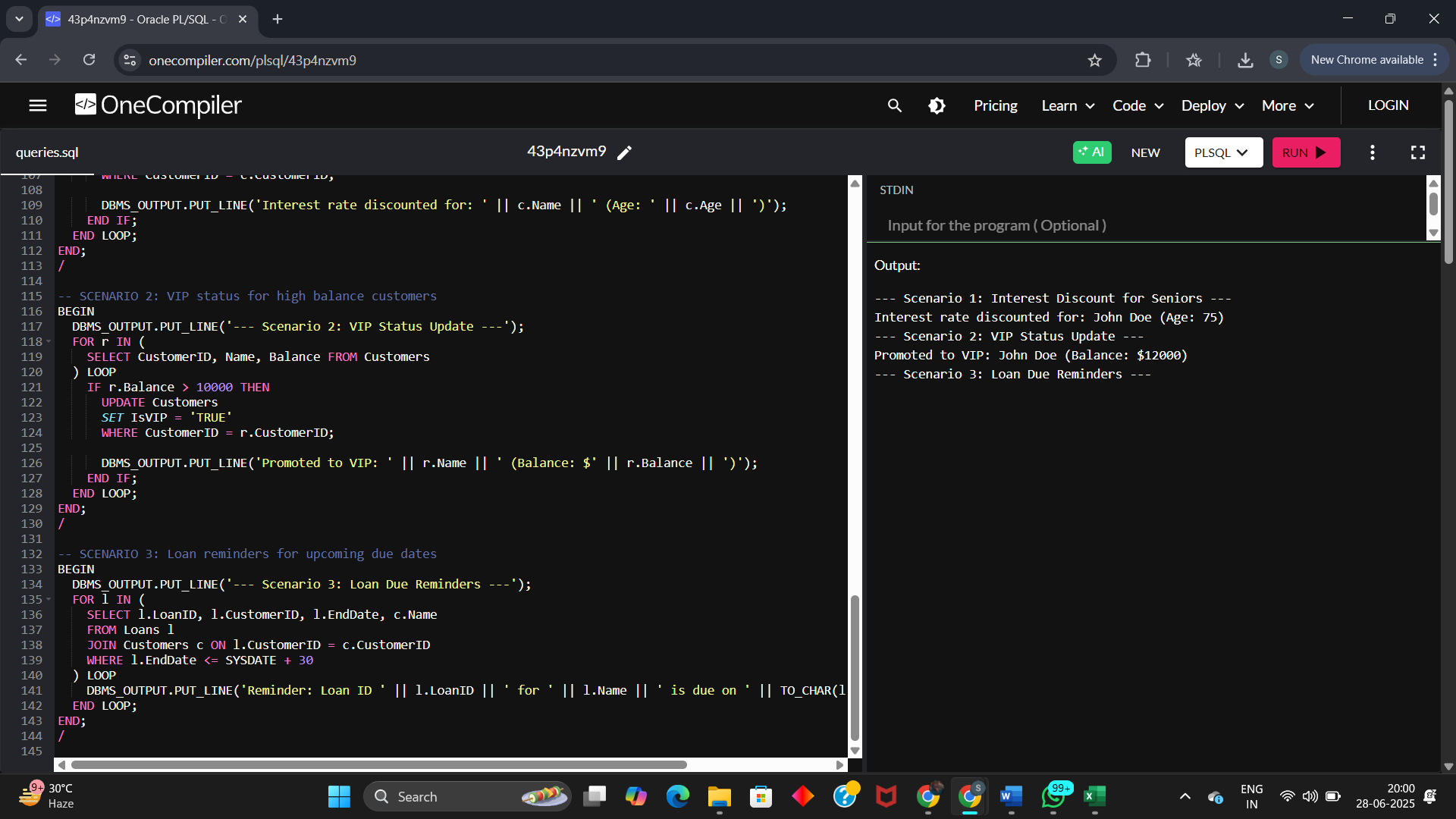
) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || l.LoanID || ' for ' || l.Name || ' is due on ' || TO\_CHAR(l.EndDate, 'YYYY-MM-DD'));

END LOOP;

END;

/

**RESULT:**

**PL/SQL - EXERCISE 3:**

**CODE:**

-- Enable output

SET SERVEROUTPUT ON;

-- Clean up old tables if they exist

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions';

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- SCHEMA SETUP

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE,

IsVIP VARCHAR2(5)

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

-- SAMPLE DATA

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1950-05-15','YYYY-MM-DD'), 12000, SYSDATE, 'FALSE');

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20','YYYY-MM-DD'), 9000, SYSDATE, 'FALSE');

INSERT INTO Accounts VALUES (1, 1, 'Savings', 12000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 9000, SYSDATE);

INSERT INTO Transactions VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Loans VALUES (2, 2, 8000, 6, SYSDATE, SYSDATE + 20);

INSERT INTO Employees VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15','YYYY-MM-DD'));

INSERT INTO Employees VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20','YYYY-MM-DD'));

**-- SCENARIO 1: Apply Monthly Interest on Savings Accounts**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Applying Monthly Interest to Savings Accounts ---');

FOR acc IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

) LOOP

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01),

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest added to Account ' || acc.AccountID ||

'. New Balance: ' || TO\_CHAR(acc.Balance \* 1.01));

END LOOP;

END;

/

**-- SCENARIO 2: Update Employee Bonus by Department**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Adding Bonus to Employees in Department: ' || p\_department || ' ---');

FOR emp IN (

SELECT EmployeeID, Name, Salary

FROM Employees

WHERE Department = p\_department

) LOOP

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_percent / 100)

WHERE EmployeeID = emp.EmployeeID;

DBMS\_OUTPUT.PUT\_LINE('Bonus added to ' || emp.Name ||

'. New Salary: ' || TO\_CHAR(emp.Salary \* (1 + p\_bonus\_percent / 100)));

END LOOP;

END;

/

**-- SCENARIO 3: Transfer Funds Between Accounts**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account

FOR UPDATE;

IF v\_balance < p\_amount THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient funds in Account ' || p\_from\_account);

RETURN;

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account;

DBMS\_OUTPUT.PUT\_LINE('Transferred $' || p\_amount || ' from Account ' || p\_from\_account || ' to Account ' || p\_to\_account);

END;

/

-- EXECUTE PROCEDURES

BEGIN

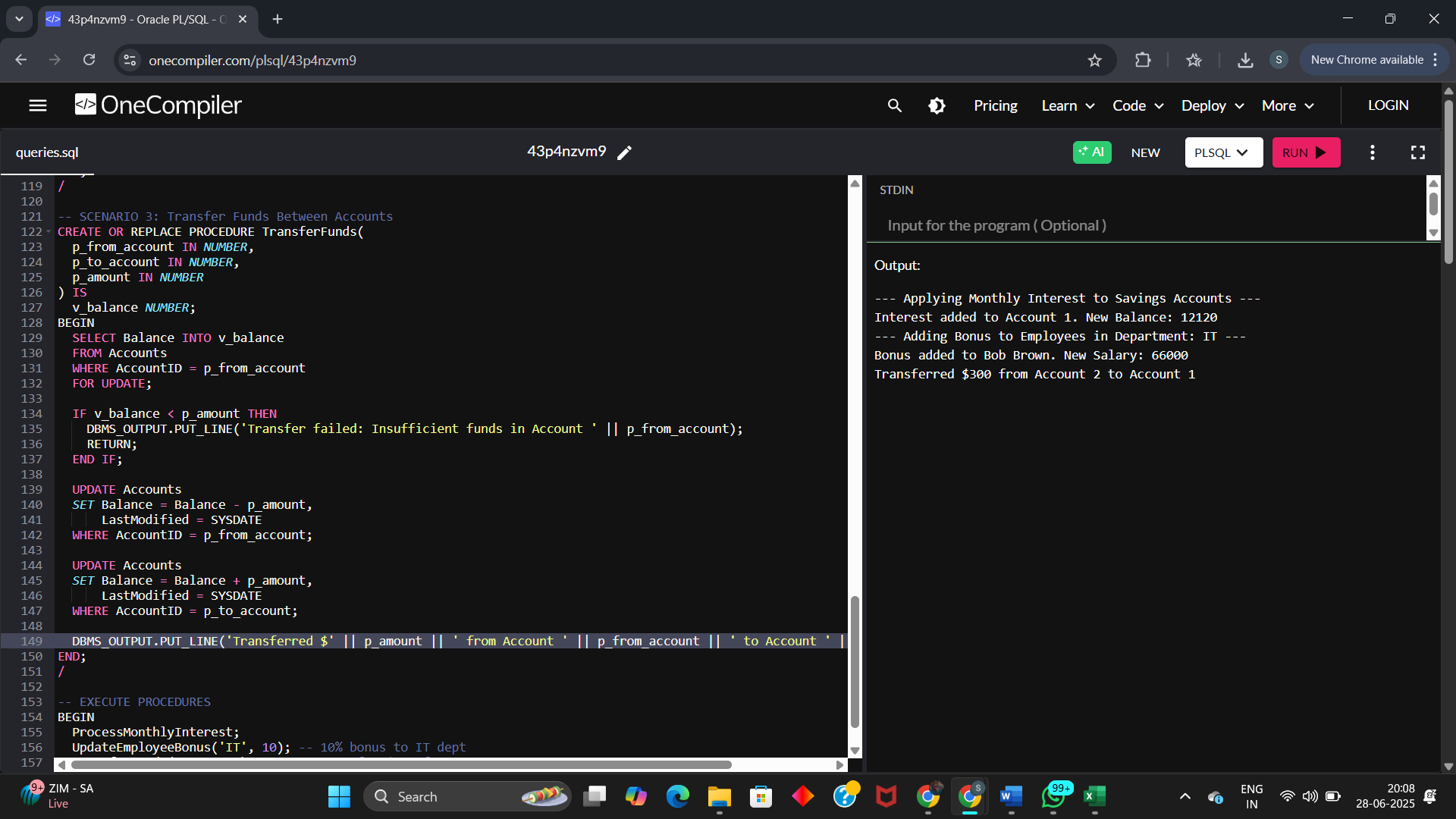
ProcessMonthlyInterest;

UpdateEmployeeBonus('IT', 10); -- 10% bonus to IT dept

TransferFunds(2, 1, 300); -- Transfer $300 from Account 2 → 1

END;

/

**RESULT:**

**Junit – EXERCISE 1:**

**CODE:**

**In JUnitSetupDemo/pom.xml / Dependency:**

<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>JUnitSetupDemo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.1.2</version>

</plugin>

</plugins>

</build>

</project>

**In CalTest.java:**

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalTest {

*@*Test

public void testAddition() {

int result = 2 + 3;

assertEquals(5, result);

}

*@*Test

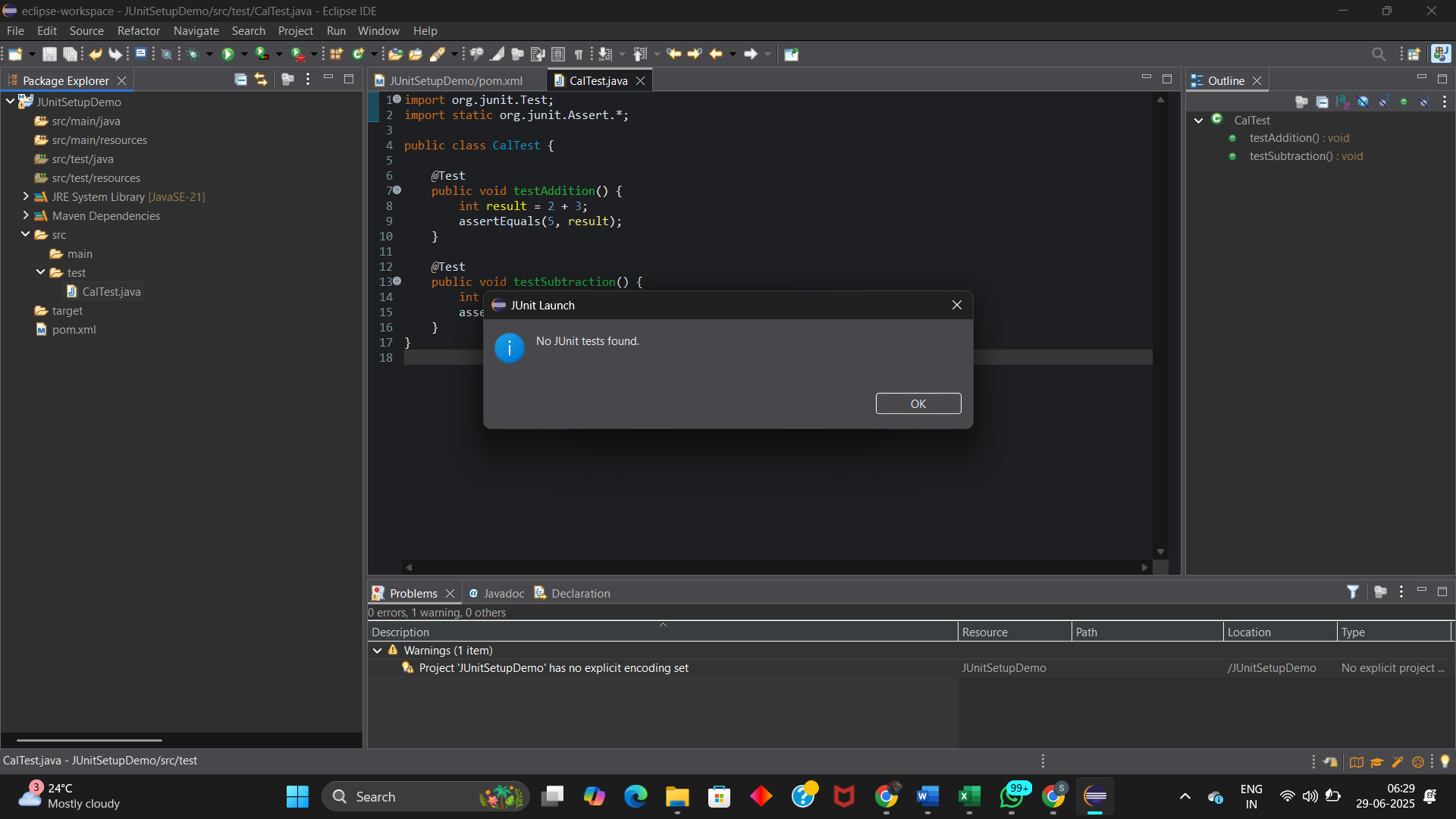
public void testSubtraction() {

int result = 3 - 2;

assertEquals(6, result);

}

}

**RESULT:**

**Junit EXERCISE 3:**

**CODE:**

**Dependency:**

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

**AssertionsTest.java:**

import org.junit.jupiter.api.Test;

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

public class AssertionsTest {

*@Test*

public void testAssertions() {

*assertEquals*(5, 2 + 3);

*assertTrue*(5 > 3);

*assertFalse*(5 < 3);

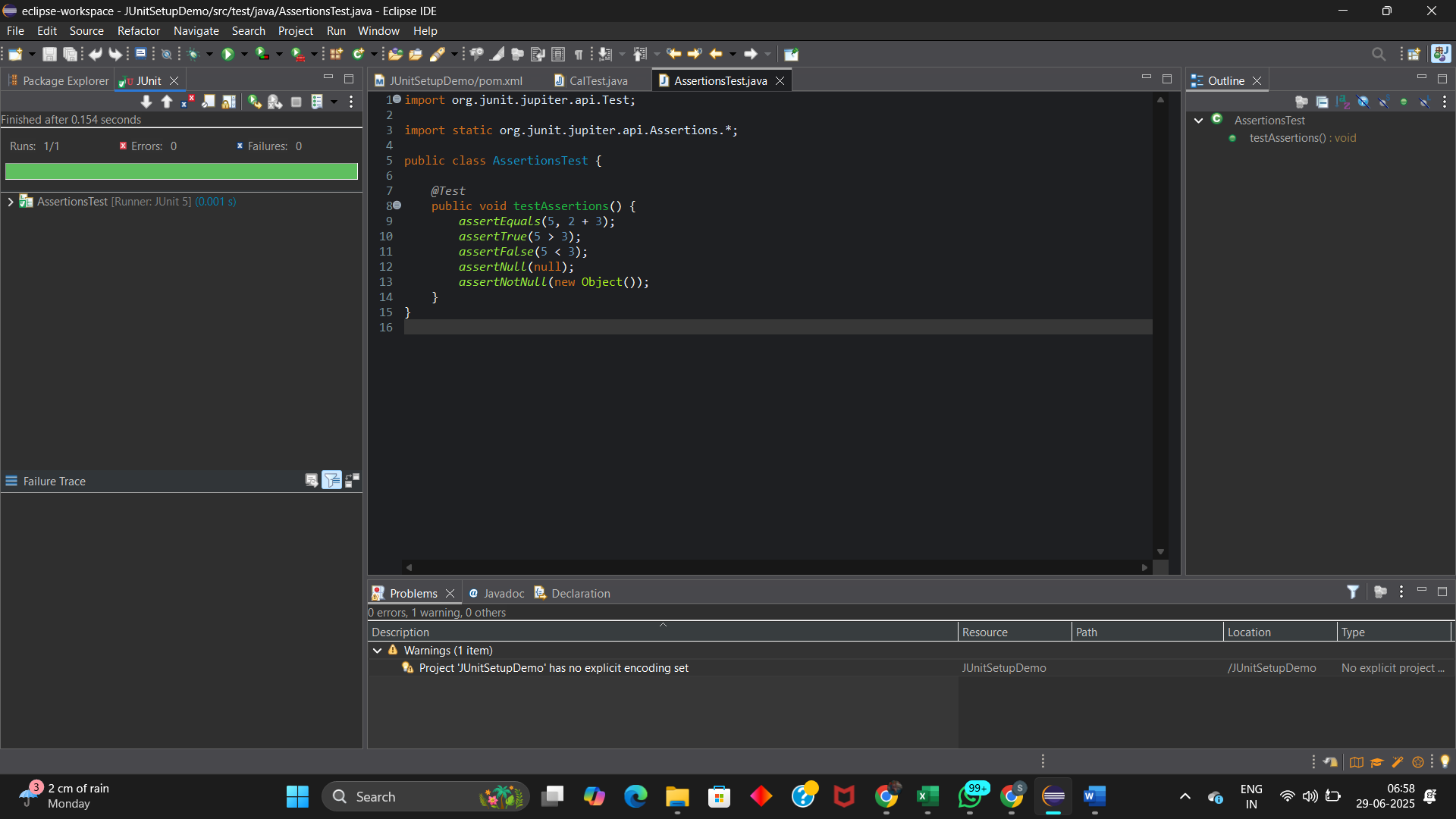
*assertNull*(null);

*assertNotNull*(new Object());

}

}

**RESULT:**



**Junit – EXERCISE 4:**

**CODE:**

**Calculator.java:**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

**CalculatorTest.java:**

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

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

public class CalculatorTest {

private Calculator calculator;

*@BeforeEach*

void setUp() {

// Setup (runs before each test)

calculator = new Calculator();

System.***out***.println("BeforeEach - setUp");

}

*@AfterEach*

void tearDown() {

// Teardown (runs after each test)

System.***out***.println("AfterEach - tearDown");

}

*@Test*

void testAddition() {

// Arrange

int a = 5;

int b = 3;

// Act

int result = calculator.add(a, b);

// Assert

*assertEquals*(8, result);

}

*@Test*

void testSubtraction() {

// Arrange

int a = 10;

int b = 4;

// Act

int result = calculator.subtract(a, b);

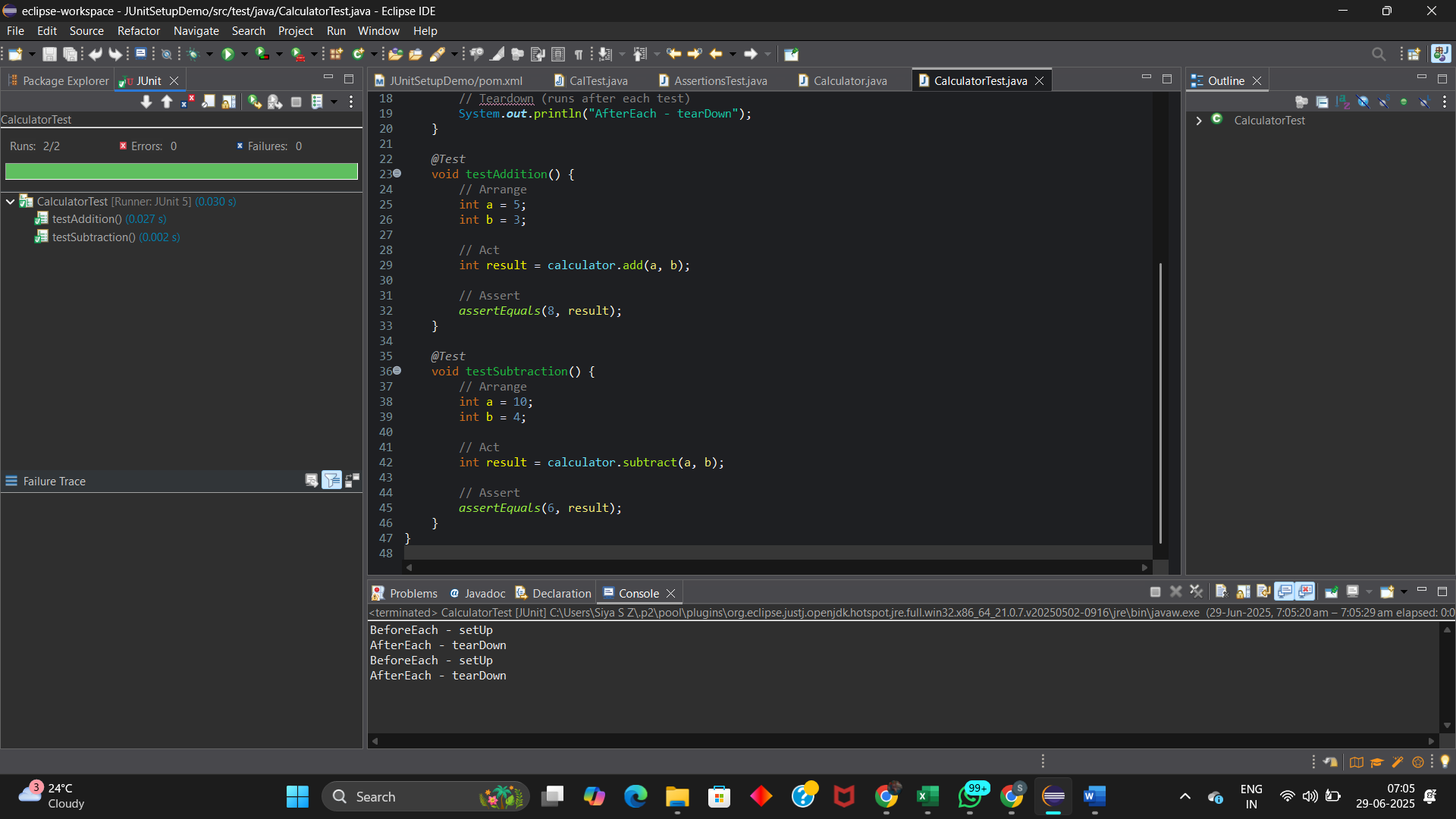
// Assert

*assertEquals*(6, result);

}

}

**RESULT:**



**MOCKITO – EXERCISE 1:**

**CODE:**

**Dependency:**

<dependencies>

**<!-- JUnit 5 -->**

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

**<!-- Mockito -->**

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.12.0</version>

<scope>test</scope>

</dependency>

</dependencies>

**ExternalApi:**

public interface ExternalApi {

String getData();

}

**MyService.java:**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java:**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

*@Test*

public void testExternalApi() {

//Creating a mock of the external API

ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);

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

//Mocking

MyService service = new MyService(mockApi);

String result = service.fetchData();

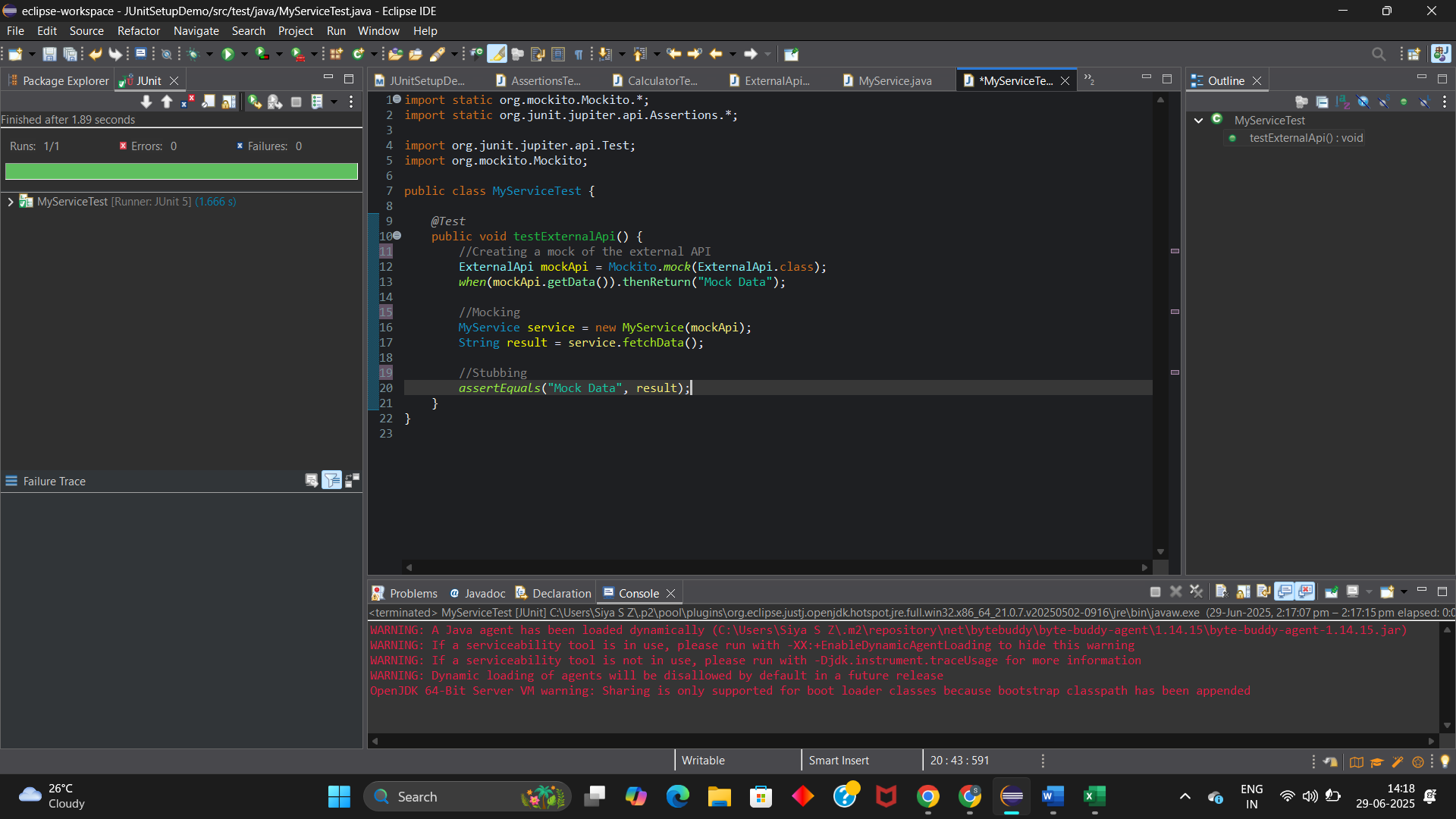
//Stubbing

*assertEquals*("Mock Data", result);

}

}

**RESULT:**



**MOCKITO – EXERCISE 2:**

**CODE:**

**Dependency:**

<dependencies>

**<!-- JUnit 5 -->**

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

**Mockito**

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.12.0</version>

<scope>test</scope>

</dependency>

</dependencies>

**ExternalApi:**

public interface ExternalApi {

String getData();

}

**MyService.java:**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

*@Test*

public void testVerifyInteraction() {

ExternalApi mockApi = Mockito.*mock*(ExternalApi.class); // 1. Mock object

MyService service = new MyService(mockApi); // 2. Inject mock

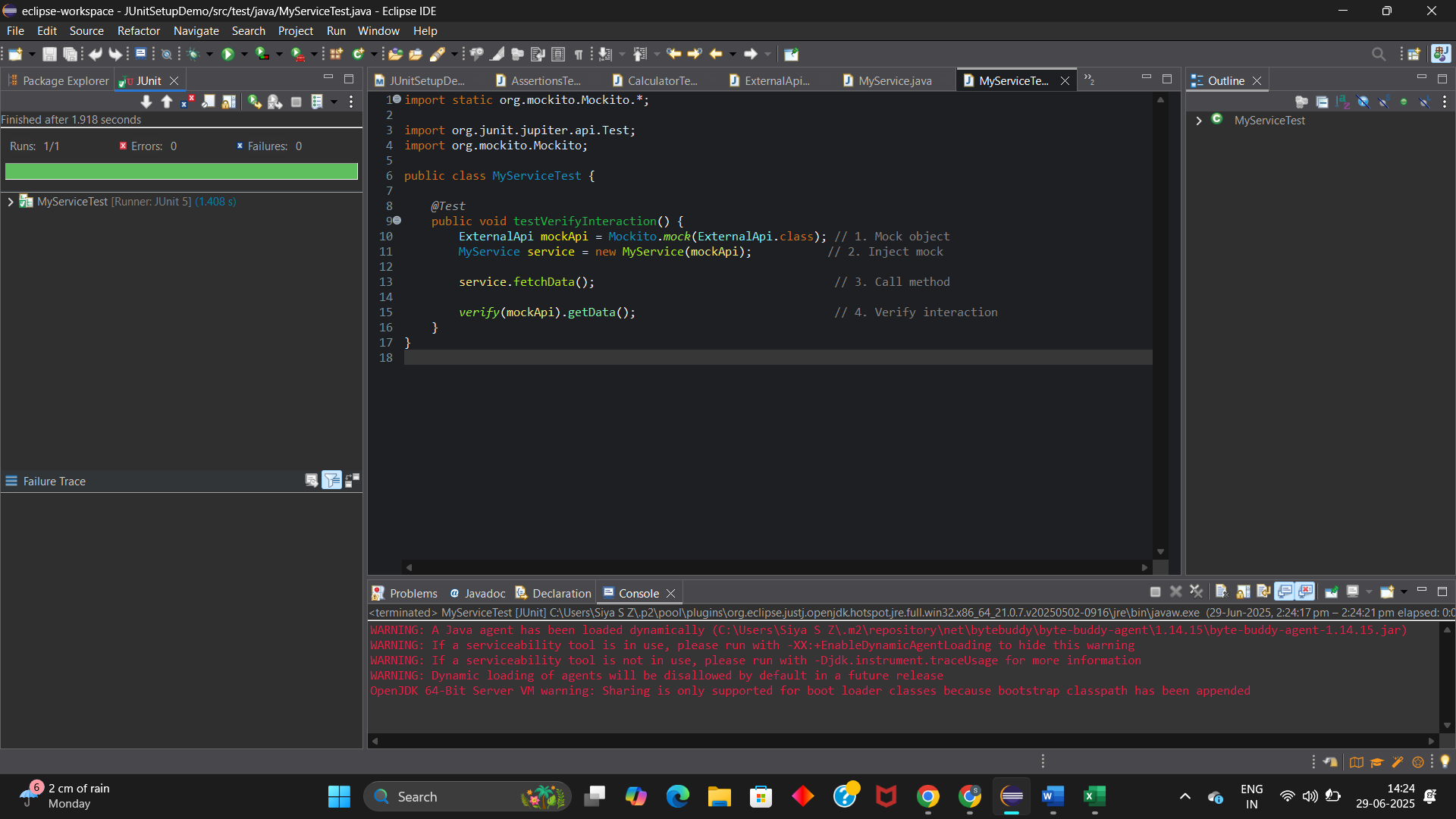
service.fetchData(); // 3. Call method

*verify*(mockApi).getData(); // 4. Verify interaction

}

}

**RESULT:**



**Logging using SLF4J – EXERCISE 1:**

**CODE:**

**Dependency:**

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

**LoggingExample.java:**

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

private static final Logger ***logger*** = LoggerFactory.*getLogger*(LoggingExample.class);

public static void main(String[] args) {

***logger***.error("This is an error message");

***logger***.warn("This is a warning message");

}

}

**RESULT:**

