## WEEK 2

## HANDS ON EXERCISES

PL/SQL PROGRAMMING:

EXERCISE 1: CONTROL STRUCTURES

SCENARIO 1:

CREATE TABLE bank\_customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(50),

age NUMBER,

interest\_rate NUMBER(5,2)

);

INSERT INTO bank\_customers VALUES (1, 'Srinu', 65, 5.25);

INSERT INTO bank\_customers VALUES (2, 'Nikhil', 45, 4.75);

INSERT INTO bank\_customers VALUES (3, 'Bhupal', 70, 6.00);

INSERT INTO bank\_customers VALUES (4, 'Rahul', 30, 3.50);

INSERT INTO bank\_customers VALUES (5, 'Madhu', 62, 5.00);

INSERT INTO bank\_customers VALUES (6, 'Sudhakar', 58, 4.25);

COMMIT;

BEGIN

UPDATE bank\_customers

SET interest\_rate = interest\_rate - 1

WHERE age > 60;

DBMS\_OUTPUT.PUT\_LINE('Senior customers receiving discount:');

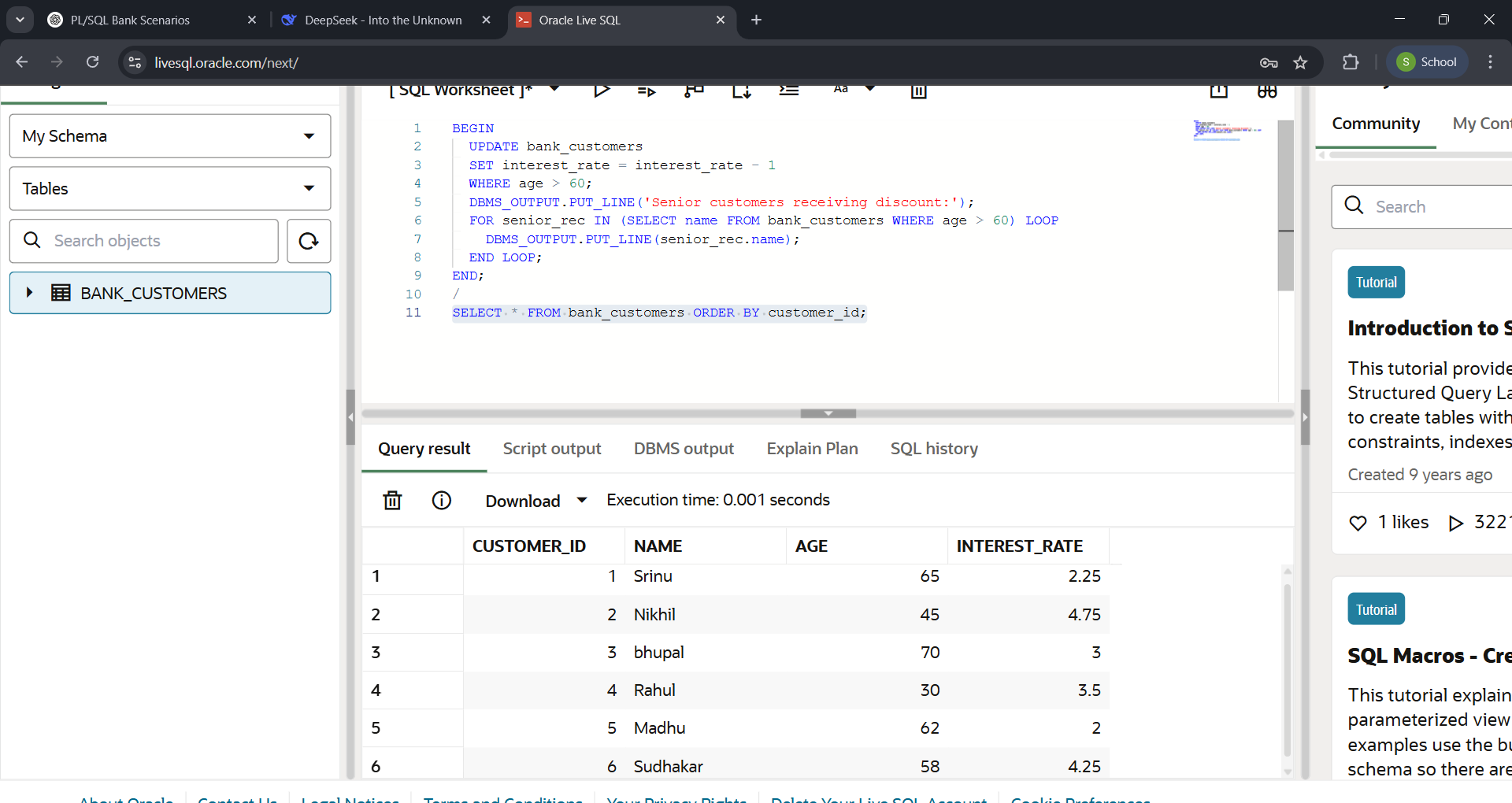
FOR senior\_rec IN (SELECT name FROM bank\_customers WHERE age > 60) LOOP

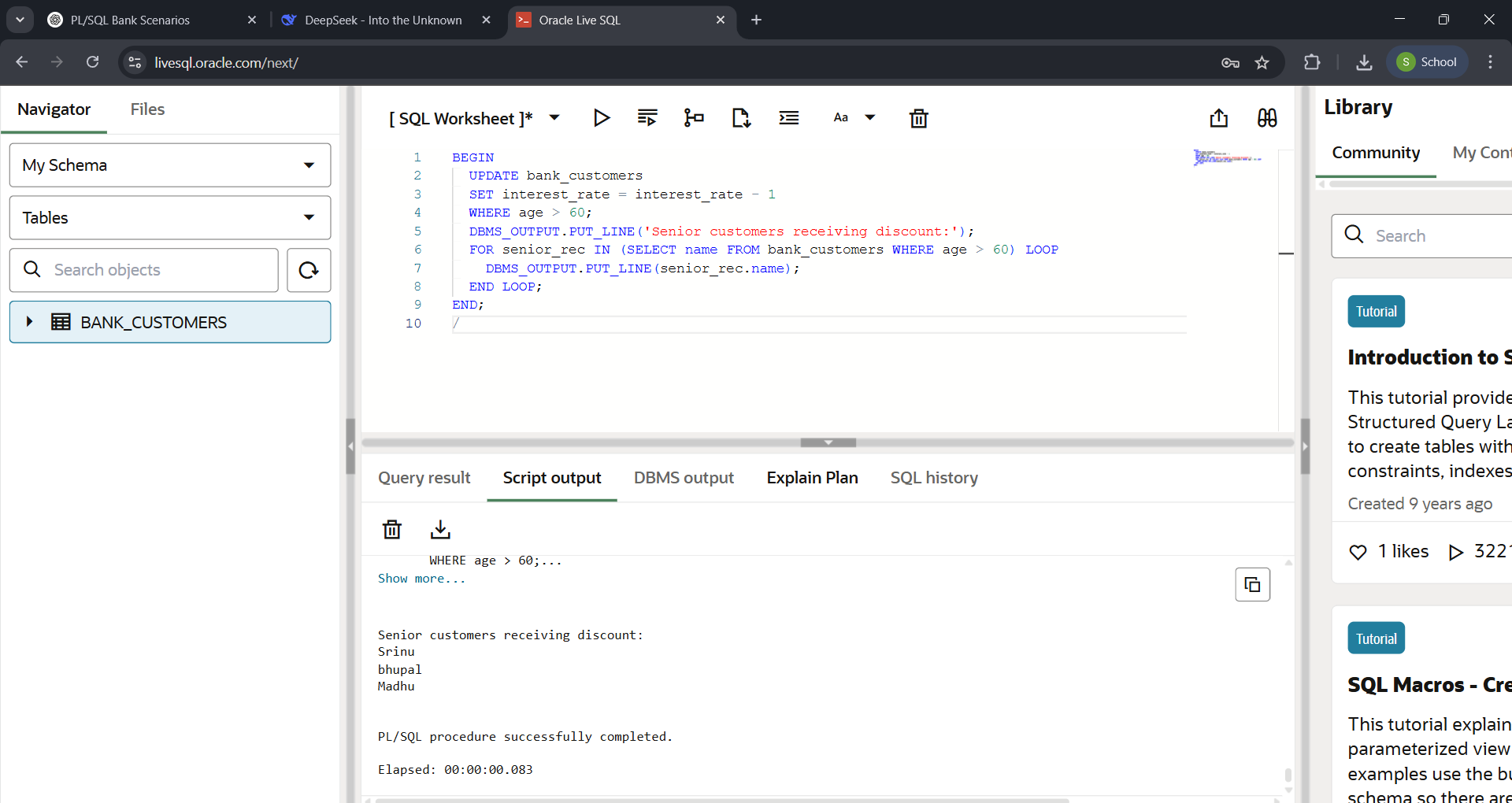
DBMS\_OUTPUT.PUT\_LINE(senior\_rec.name);

END LOOP;

END;

/

OUTPUT:



SCENARIO 2:

CREATE TABLE customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(50),

balance NUMBER(10,2),

is\_vip CHAR(1) DEFAULT 'N'

);

INSERT INTO customers VALUES (1, 'Anil', 5000.00, 'N');

INSERT INTO customers VALUES (2, 'Damodhar', 15000.00, 'N');

INSERT INTO customers VALUES (3, 'Rohan', 3000.00, 'N');

INSERT INTO customers VALUES (4, 'Eshwar', 25000.00, 'N');

INSERT INTO customers VALUES (5, 'Mohan', 8000.00, 'N');

INSERT INTO customers VALUES (6, 'Sunny', 12000.00, 'N');

COMMIT;

SET SERVEROUTPUT ON;

BEGIN

UPDATE customers

SET is\_vip = 'Y'

WHERE balance > 10000;

DBMS\_OUTPUT.PUT\_LINE('New VIP customers:');

FOR vip\_rec IN (SELECT name, balance FROM customers WHERE is\_vip = 'Y') LOOP

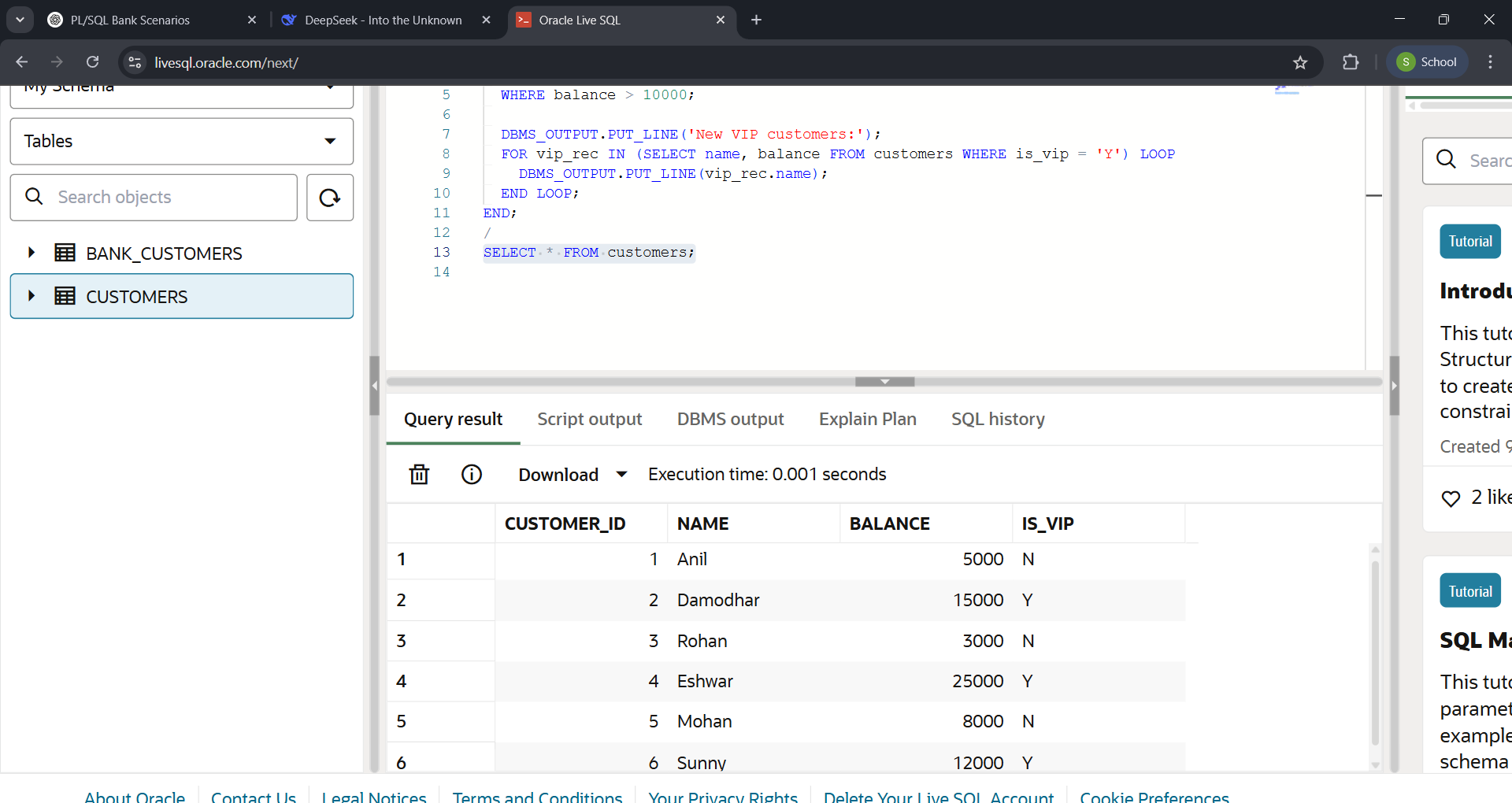
DBMS\_OUTPUT.PUT\_LINE(vip\_rec.name);

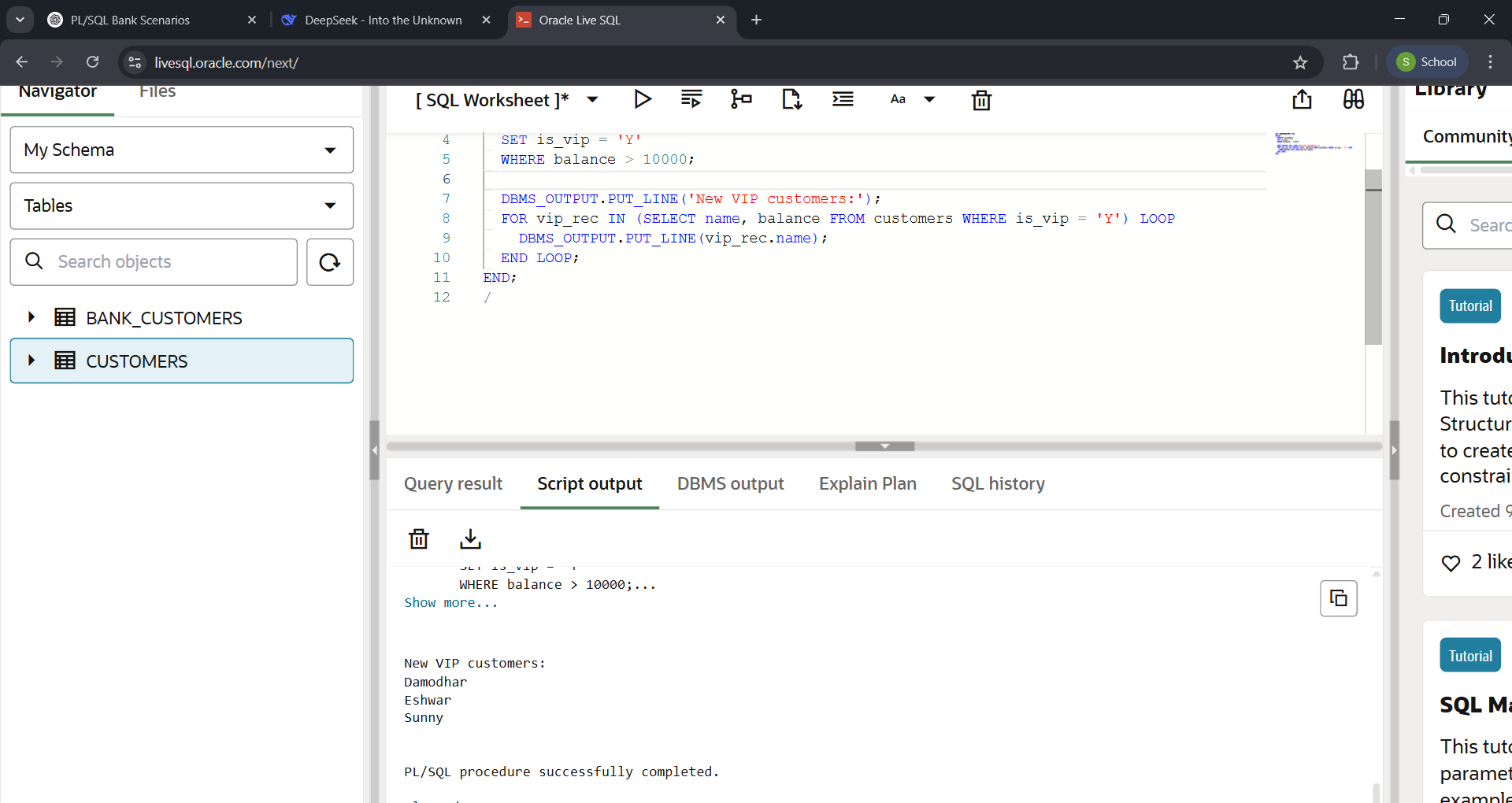
END LOOP;

END;

/

SELECT \* FROM customers;

OUTPUT:



SCENARIO 3:

CREATE TABLE customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(50)

);

CREATE TABLE loans (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER REFERENCES customers(customer\_id),

amount NUMBER,

due\_date DATE

);

INSERT INTO customers VALUES (1, 'Damodhar');

INSERT INTO customers VALUES (2, 'Anil');

INSERT INTO customers VALUES (3, 'Bhupal');

INSERT INTO customers VALUES (4, 'Eshwar');

INSERT INTO customers VALUES (5, 'Madhu');

INSERT INTO loans VALUES (101, 1, 5000, SYSDATE + 5);

INSERT INTO loans VALUES (102, 2, 12000, SYSDATE + 15);

INSERT INTO loans VALUES (103, 3, 8000, SYSDATE + 45);

INSERT INTO loans VALUES (104, 4, 3000, SYSDATE + 25);

INSERT INTO loans VALUES (105, 5, 10000, SYSDATE - 1);

COMMIT;

SET SERVEROUTPUT ON;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('UPCOMING LOAN PAYMENTS (Next 30 Days)');

FOR r IN (

SELECT c.name, l.amount, l.due\_date

FROM customers c JOIN loans l ON c.customer\_id = l.customer\_id

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

ORDER BY l.due\_date

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

r.name || ': $' || r.amount || ' due on ' ||

TO\_CHAR(r.due\_date, 'DD-MON-YYYY')

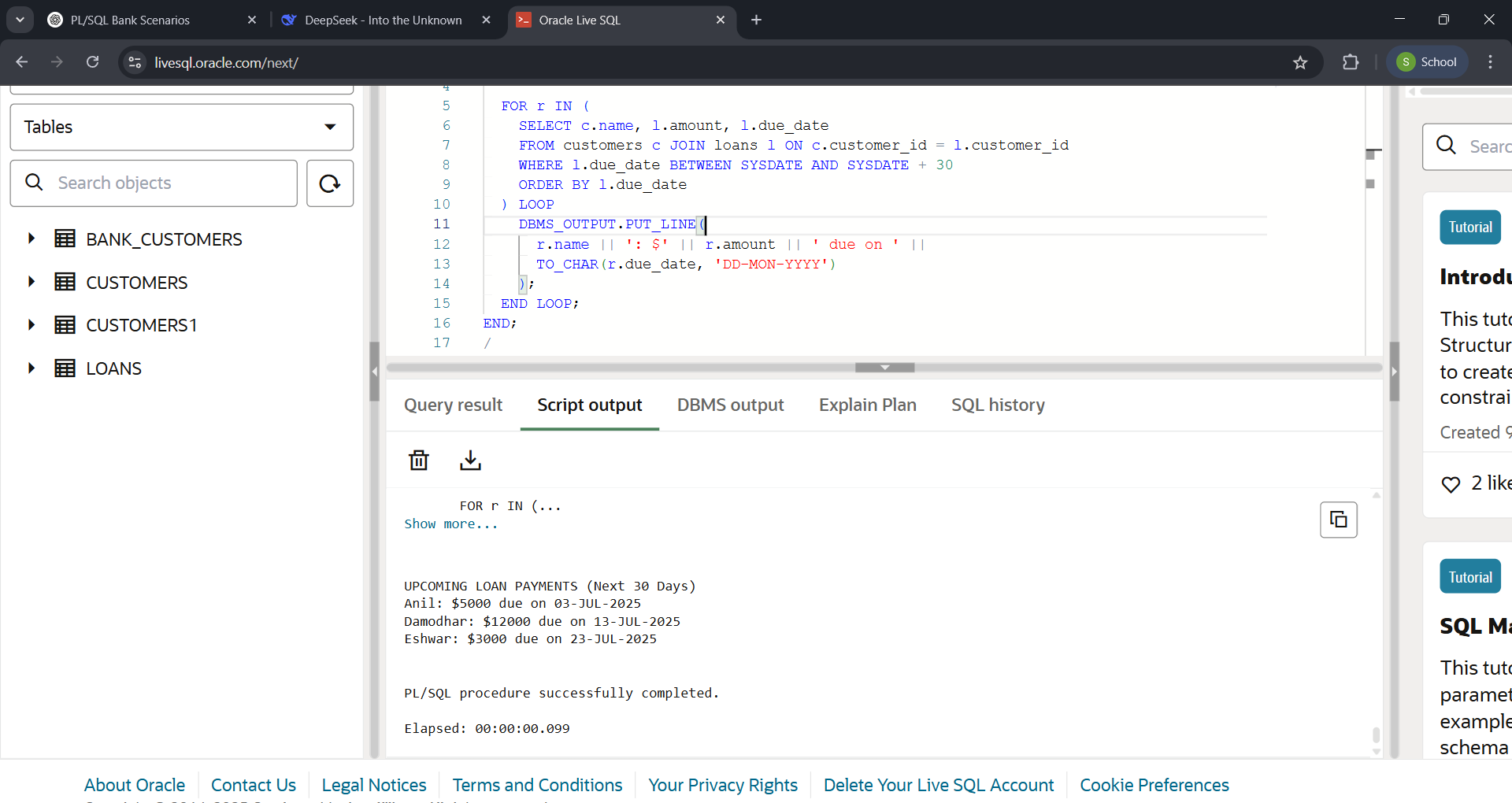
);

END LOOP;

END;

/

OUTPUT:



EXERCISE 3: STORED PROCEDURES

SCENARIO 1:

CREATE TABLE accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

account\_type VARCHAR2(20) CHECK (account\_type IN ('SAVINGS', 'CHECKING')),

balance NUMBER(10,2),

interest\_rate NUMBER(5,2) DEFAULT 1.0

);

CREATE TABLE employees (

employee\_id NUMBER PRIMARY KEY,

name VARCHAR2(50),

department VARCHAR2(50),

salary NUMBER(10,2),

bonus\_percentage NUMBER(5,2)

);

INSERT INTO accounts VALUES (101, 1, 'SAVINGS', 5000.00, 1.0);

INSERT INTO accounts VALUES (102, 1, 'CHECKING', 2500.00, 0.5);

INSERT INTO accounts VALUES (103, 2, 'SAVINGS', 12000.00, 1.2);

INSERT INTO accounts VALUES (104, 3, 'SAVINGS', 8000.00, 1.0);

INSERT INTO accounts VALUES (105, 4, 'CHECKING', 3000.00, 0.3);

INSERT INTO accounts VALUES (106, 5, 'SAVINGS', 15000.00, 1.5);

INSERT INTO employees VALUES (1, 'Joseph, 'LOAN', 5000.00, 5.0);

INSERT INTO employees VALUES (2, 'Anil', 'CUSTOMER SERVICE', 4500.00, 3.0);

INSERT INTO employees VALUES (3, 'Rohan', 'LOAN', 6000.00, 7.0);

INSERT INTO employees VALUES (4, 'Eshwar', 'IT', 7000.00, 10.0);

INSERT INTO employees VALUES (5, 'Madhu', 'CUSTOMER SERVICE', 4800.00, 4.0);

INSERT INTO employees VALUES (6, 'Sudhakar', 'IT', 6500.00, 8.0);

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

FOR rec IN (SELECT account\_id, balance FROM accounts WHERE account\_type = 'SAVINGS') LOOP

UPDATE accounts

SET balance = balance + (balance \* (interest\_rate/100))

WHERE account\_id = rec.account\_id;

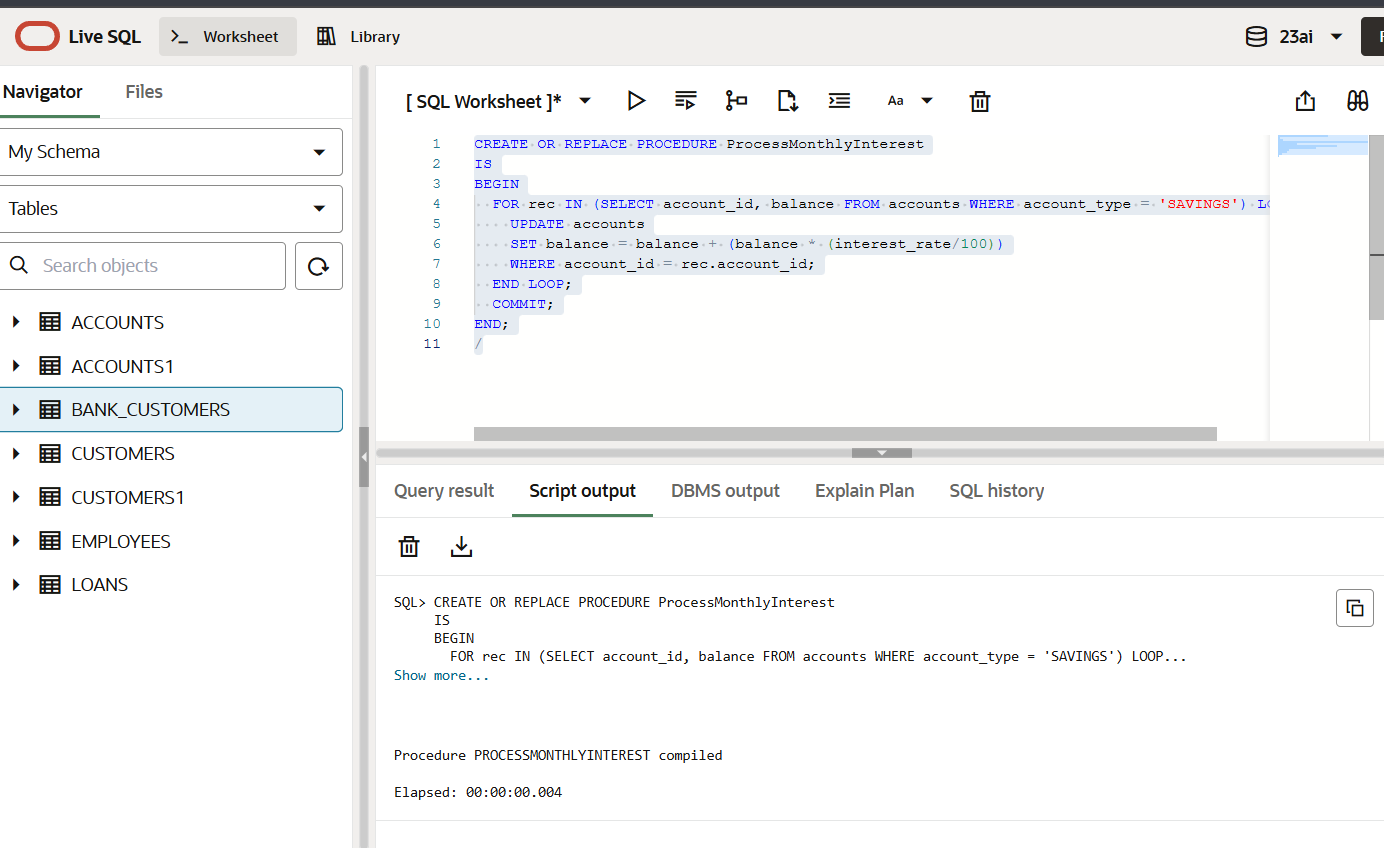
END LOOP;

COMMIT;

END;

/

OUTPUT:



SCENARIO 2:

CREATE TABLE Employees (

employee\_id NUMBER PRIMARY KEY,

name VARCHAR2(50),

department VARCHAR2(50),

salary NUMBER(10,2)

);

INSERT INTO Employees VALUES (1, 'Joseph', 'LOAN', 5000.00);

INSERT INTO Employees VALUES (2, 'Anil', 'CUSTOMER SERVICE', 4500.00);

INSERT INTO Employees VALUES (3, 'Rohan', 'LOAN', 6000.00);

INSERT INTO Employees VALUES (4, 'Madhu', 'IT', 7000.00);

INSERT INTO Employees VALUES (5, 'Manoj', 'CUSTOMER SERVICE', 4800.00);

INSERT INTO Employees VALUES (6, 'Sunny’, 'IT', 6500.00);

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percentage IN NUMBER

)

IS

BEGIN

UPDATE Employees

SET salary = salary + (salary \* p\_bonus\_percentage / 100)

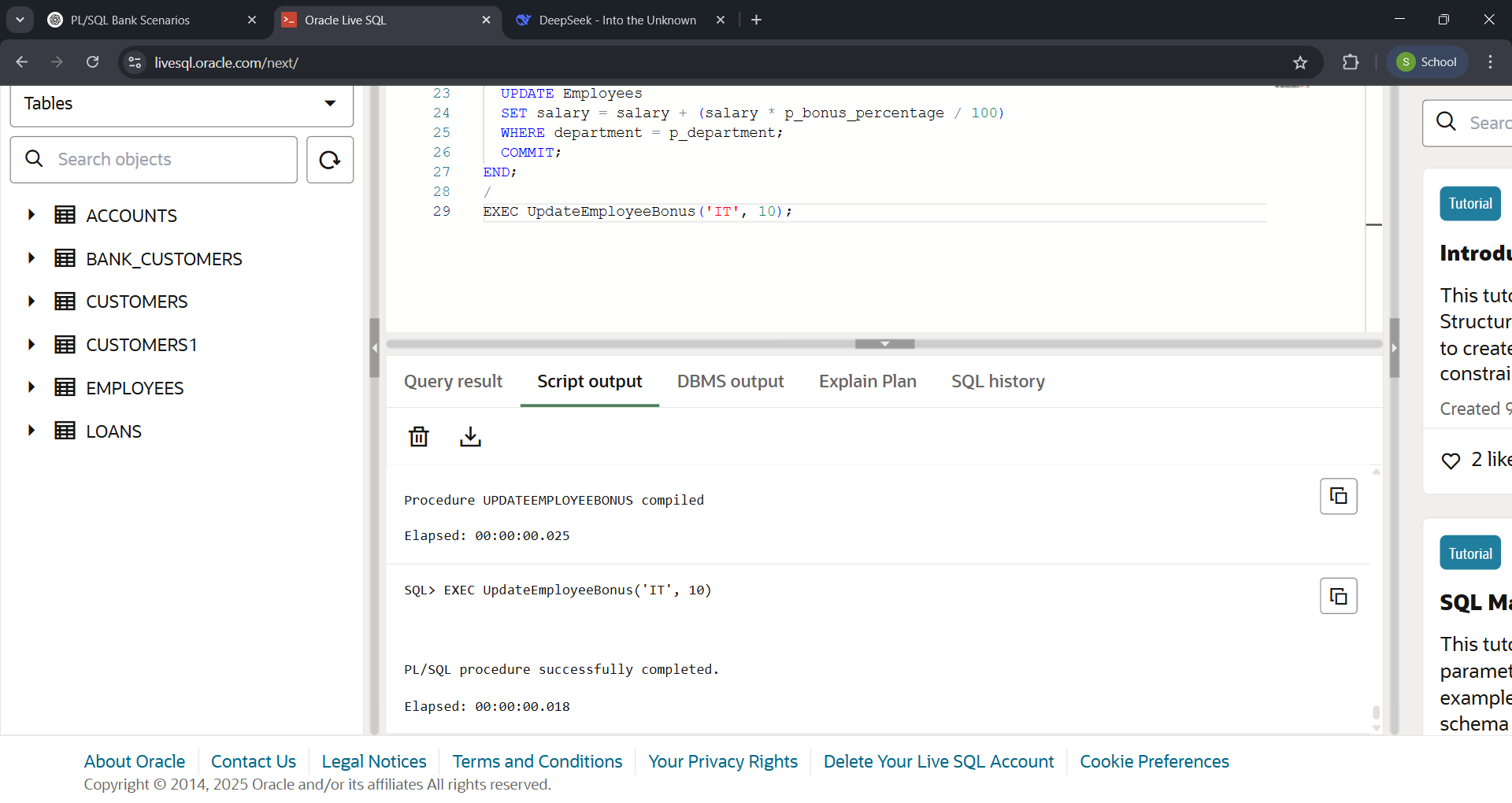
WHERE department = p\_department;

COMMIT;

END;

/

EXEC UpdateEmployeeBonus('IT', 10);

OUTPUT:

SCENARIO 3:

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

Balance NUMBER(10,2)

);

INSERT INTO Accounts VALUES (101, 5000.00);

INSERT INTO Accounts VALUES (102, 2500.00);

INSERT INTO Accounts VALUES (103, 12000.00);

INSERT INTO Accounts VALUES (104, 8000.00);

INSERT INTO Accounts VALUES (105, 3000.00);

INSERT INTO Accounts VALUES (106, 15000.00);

COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_SourceAccountID IN NUMBER,

p\_DestinationAccountID IN NUMBER,

p\_Amount IN NUMBER

)

IS

v\_SourceBalance NUMBER;

BEGIN

SELECT Balance INTO v\_SourceBalance

FROM Accounts

WHERE AccountID = p\_SourceAccountID;

IF v\_SourceBalance >= p\_Amount THEN

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_SourceAccountID;

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_DestinationAccountID;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account.');

END IF;

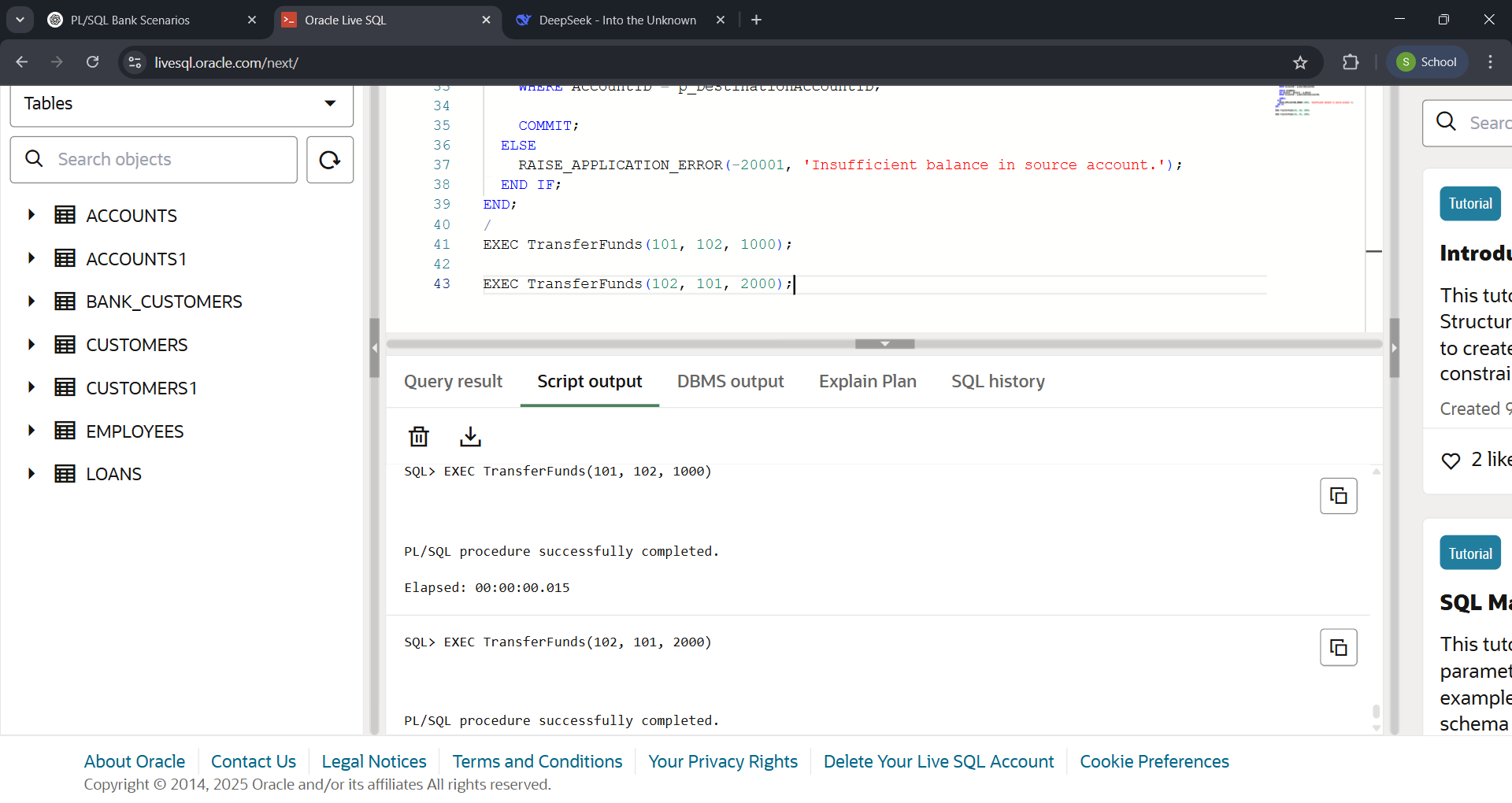
END;

/

EXEC TransferFunds(101, 102, 1000);

EXEC TransferFunds(102, 101, 2000);

OUTPUT:



## JUNIT BASIC TESTING EXERCISES:

EXERCISE 1:

SOURCE CODE:

Calculator.java:

**package** com.example.main;

**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:

**package** com.example.test;

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

**import** org.junit.Test;

**import** com.example.main.Calculator;

**public** **class** CalculatorTest {

@Test

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

Calculator calc = **new** Calculator();

**int** result = calc.add(10, 5);

*assertEquals*(15, result);

}

@Test

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

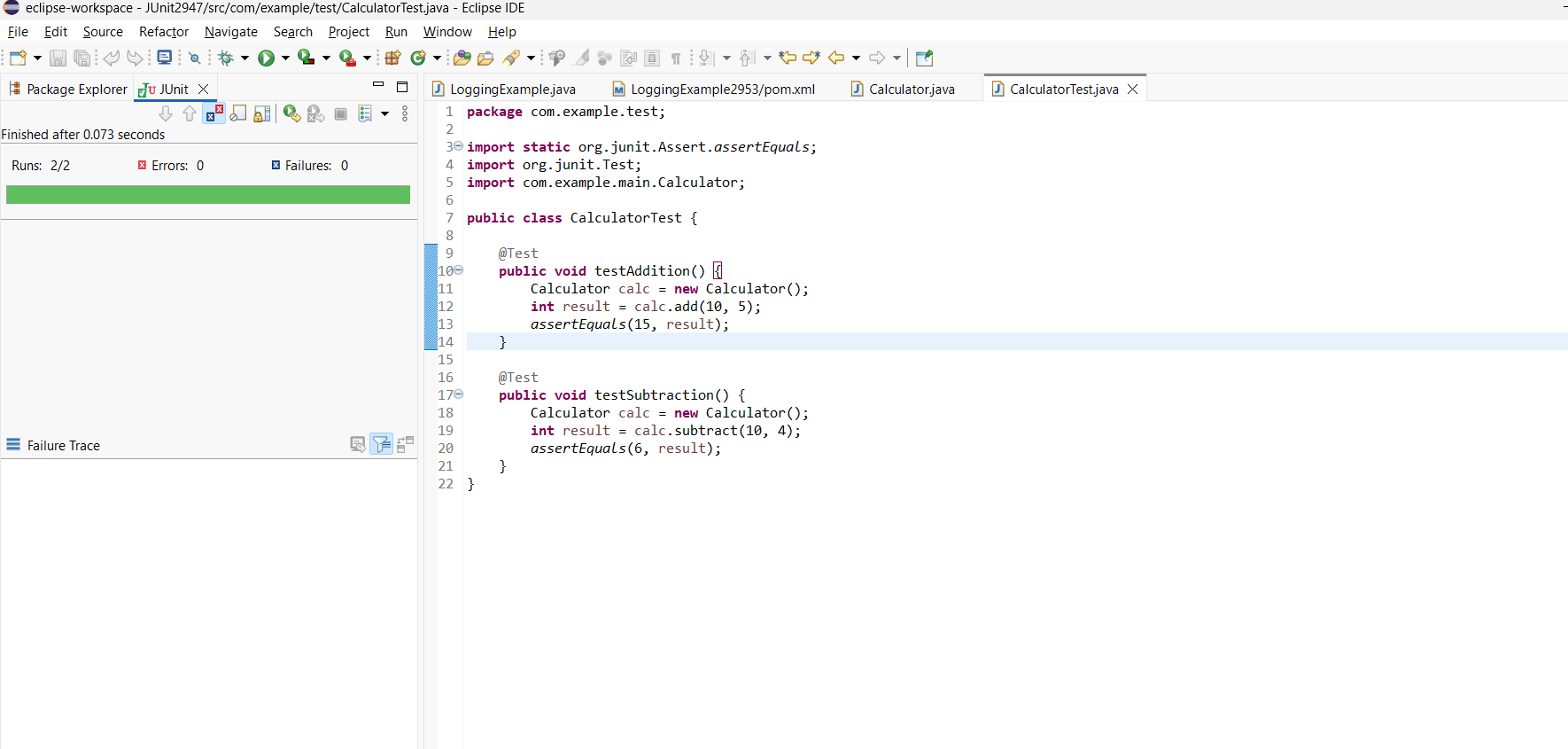
Calculator calc = **new** Calculator();

**int** result = calc.subtract(10, 4);

*assertEquals*(6, result);

}

}

OUTPUT:

EXERCISE 3: ASSERTIONS IN JUNIT

SOURCE CODE:

**package** com.example.test;

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

**import** org.junit.Test;

**public** **class** AssertionsTest {

@Test

**public** **void** testAssertions() {

*assertEquals*(5, 2 + 3);

*assertTrue*(5 > 3);

*assertFalse*(5 < 3);

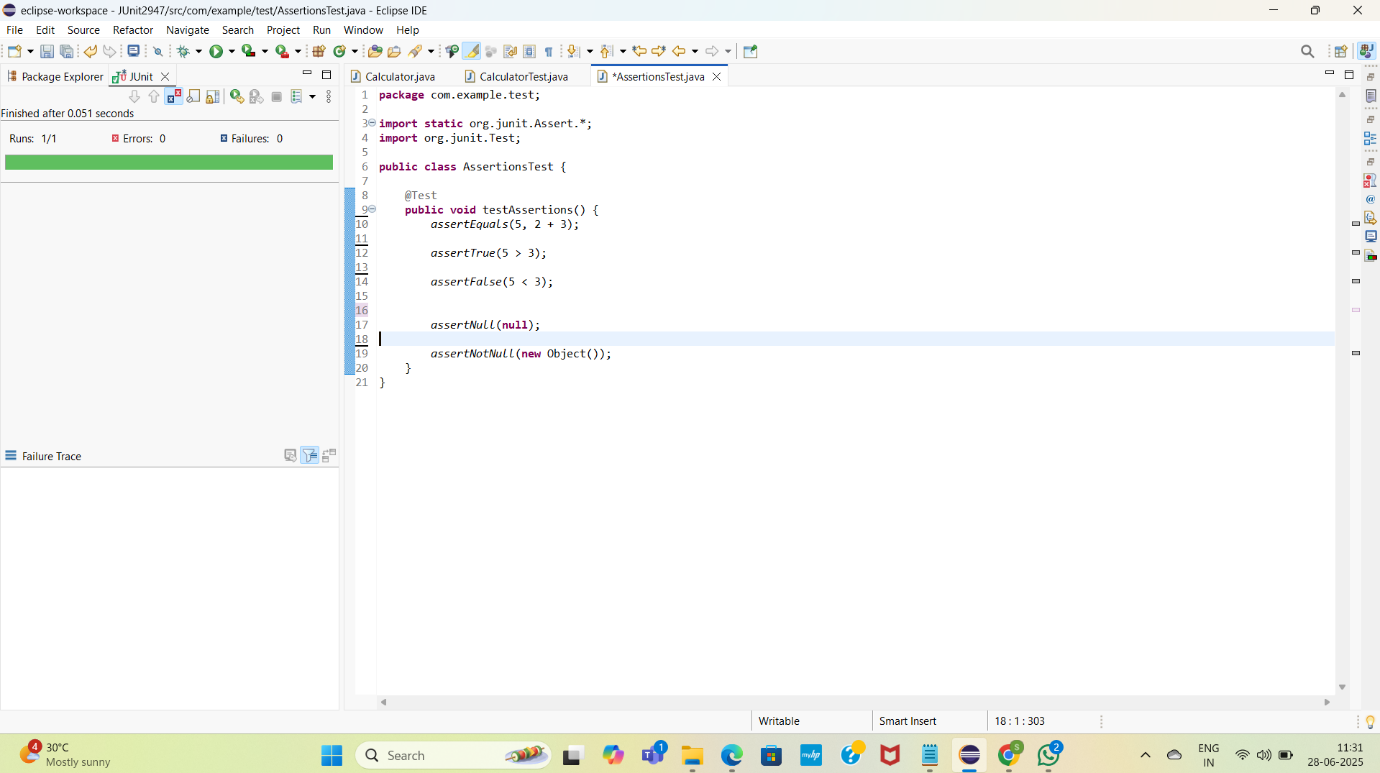
*assertNull*(**null**);

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

}

}

OUTPUT:



EXERCISE 4: AAA PATTERN

SOURCE CODE:

**package** com.example.test;

**import** com.example.main.Calculator;

**import** org.junit.Before;

**import** org.junit.After;

**import** org.junit.Test;

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

**public** **class** CalculatorAAATest {

**private** Calculator calc;

@Before

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

System.***out***.println("Setting up...");

calc = **new** Calculator();

}

@After

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

System.***out***.println("Cleaning up...");

calc = **null**;

}

@Test

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

**int** result = calc.add(4, 6);

*assertEquals*(10, result);

}

@Test

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

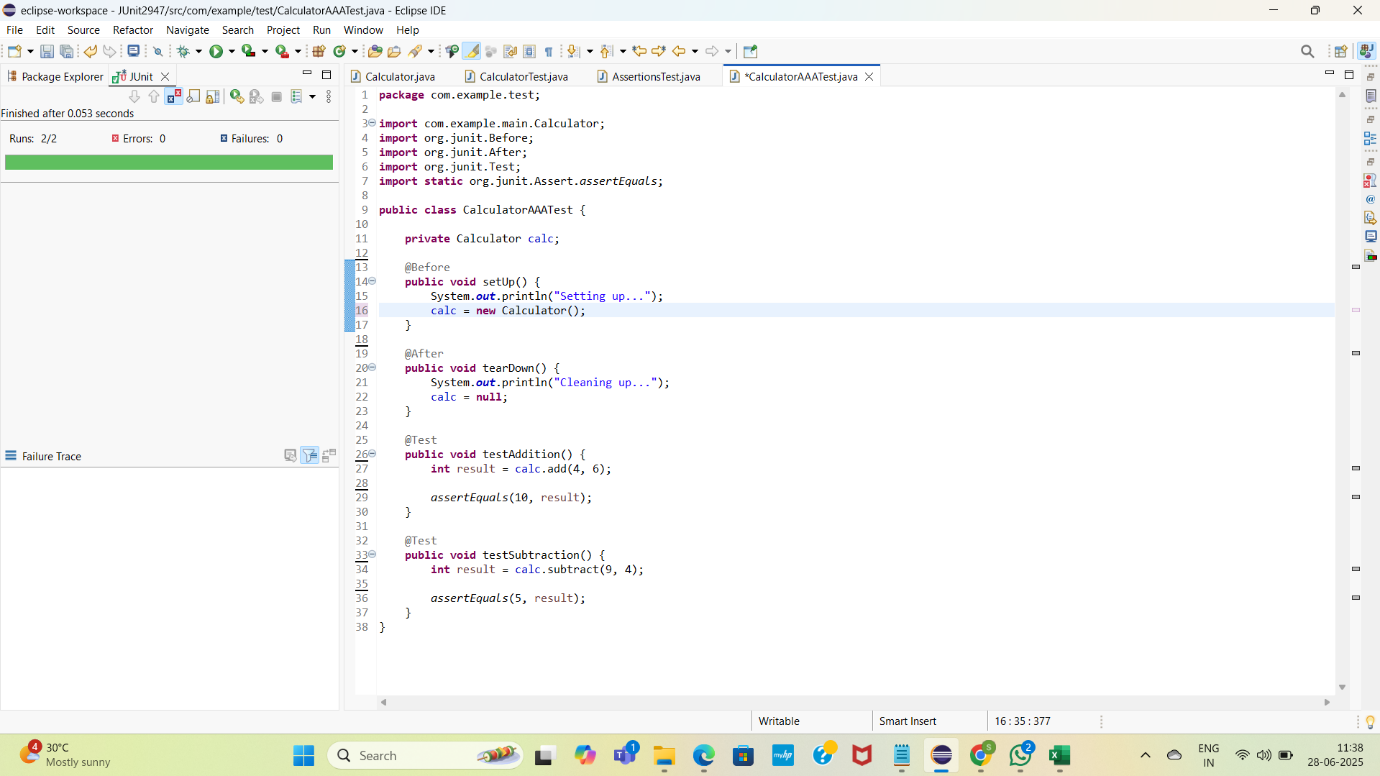
**int** result = calc.subtract(9, 4);

*assertEquals*(5, result);

}

}

OUTPUT:



## **MOKITO EXERCISES:**

EXERCISE 1: MOCKING AND STUBBING

SOURCE CODE:

ExternalAPI.java:

**package** com.example.main;

**public** **interface** ExternalApi {

String getData();

}

Myservice.java:

**package** com.example.main;

**public** **class** MyService {

**private** ExternalApi api;

**public** MyService(ExternalApi api) {

**this**.api = api;

}

**public** String fetchData() {

**return** api.getData();

}

}

MyServiceTest.java:

package com.example.test;

import com.example.main.ExternalApi;

import com.example.main.MyService;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

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);

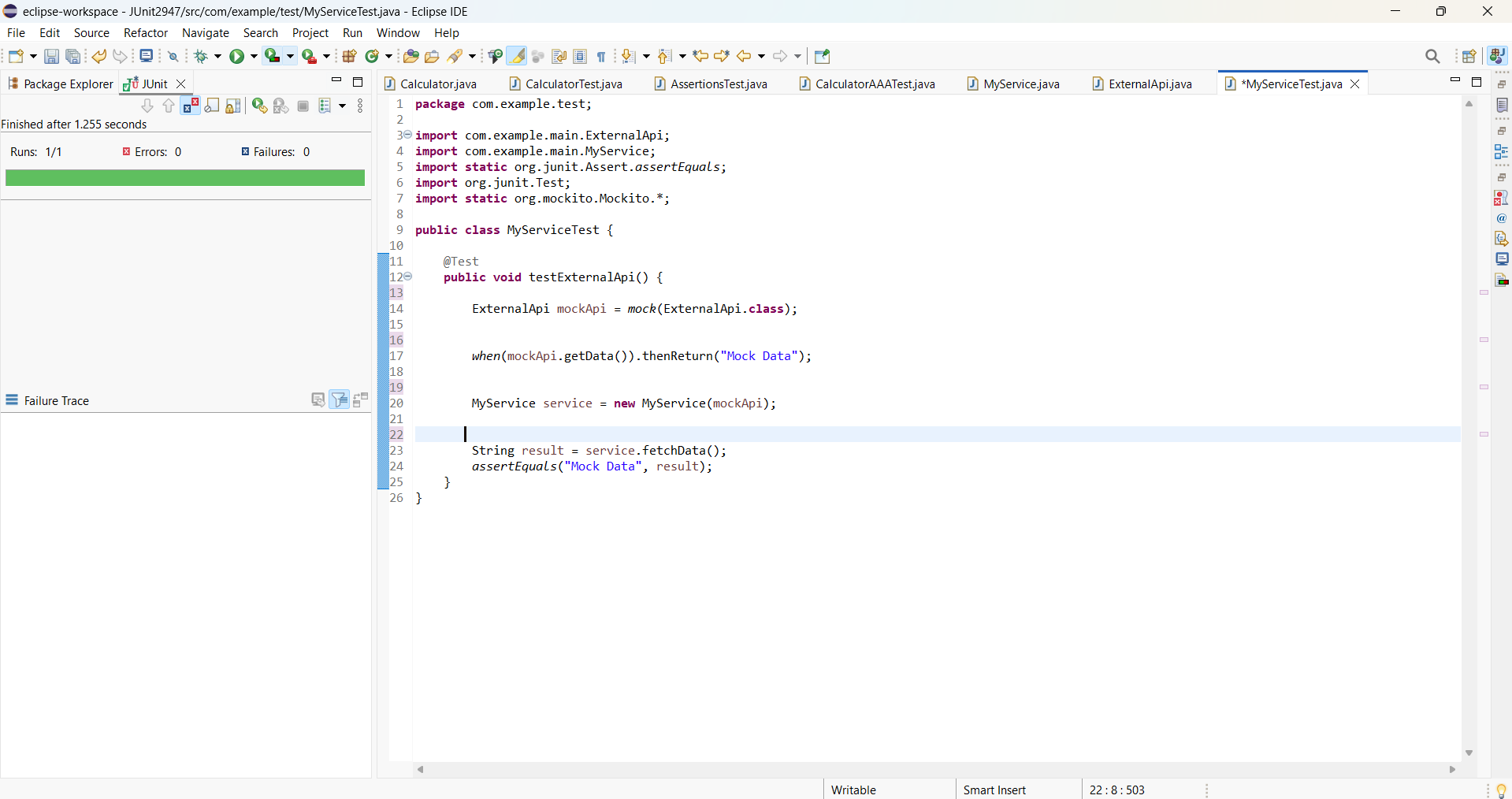
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

OUTPUT:



EXERCISE 2: VERIFYING INTERACTIONS

SOURCE CODE:

MyServiceTest.java:

**package** com.example.test;

**import** **static** org.mockito.Mockito.\*;

**import** com.example.main.ExternalApi;

**import** com.example.main.MyService;

**import** org.junit.Test;

**import** org.mockito.Mockito;

**public** **class** MyServiceTest {

@Test

**public** **void** testVerifyInteraction() {

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

MyService service = **new** MyService(mockApi);

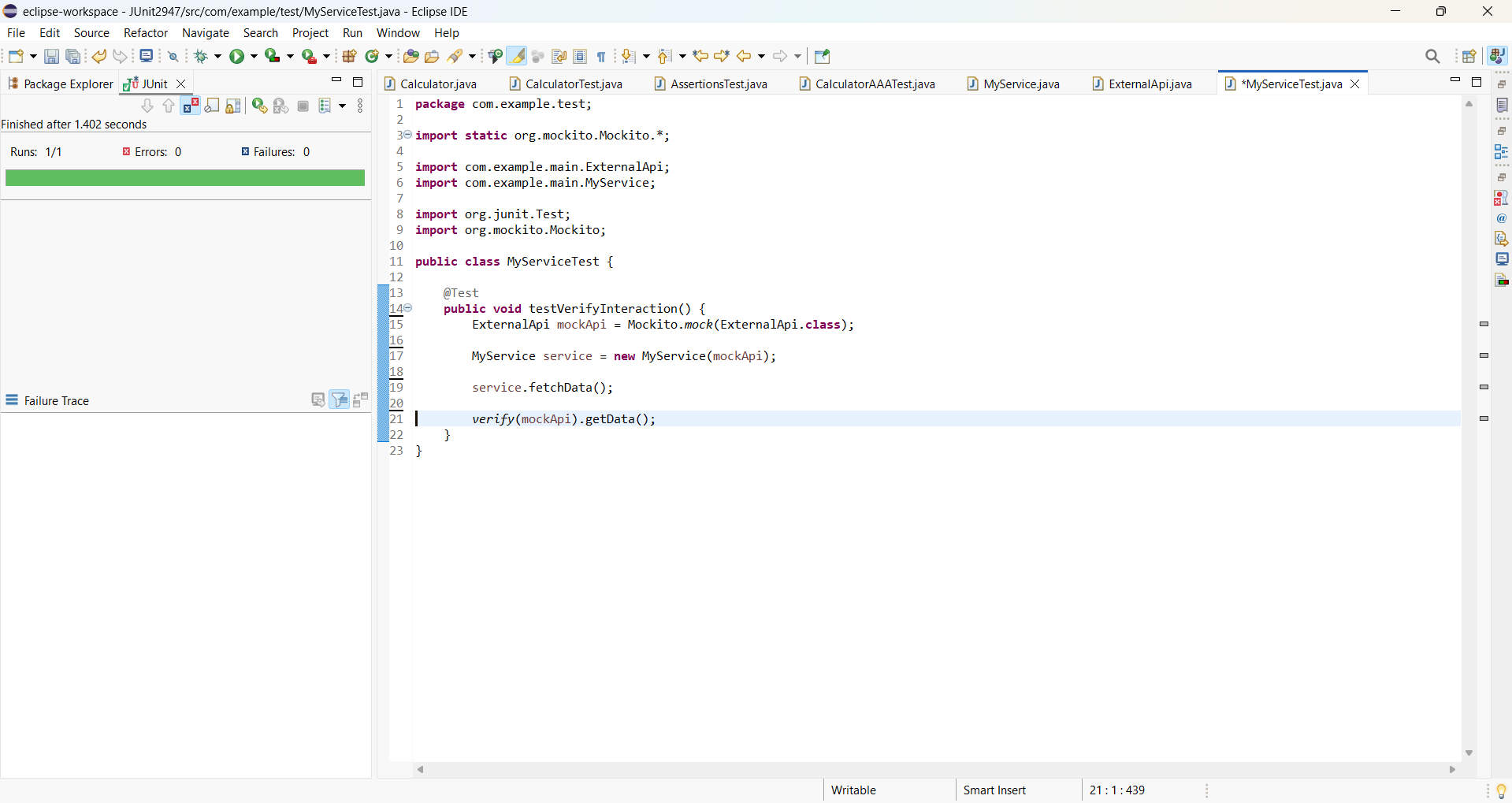
service.fetchData();

*verify*(mockApi).getData();

}

}

OUTPUT:



## **SLF4J EXERCISES:**

Exercise 1: Logging Error Messages and Warning Levels

SOURCE CODE:

LoggingExample.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>LoggingExample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>LoggingExample</name>

<description>A simple SLF4J logging example with Logback</description>

<properties>

<maven.compiler.source>1.8</maven.compiler.source> <!-- Use 17 or higher if you prefer -->

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<!-- SLF4J API -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<!-- Logback: SLF4J Implementation -->

<dependency>

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

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

<!-- Optional: Plugin for running Java from Maven CLI -->

<build>

<plugins>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>3.1.0</version>

<configuration>

<mainClass>com.example.LoggingExample</mainClass>

</configuration>

</plugin>

</plugins>

</build>

</project>

LoggingExample.java:

**package** com.example;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**public** **class** LoggingExample {

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

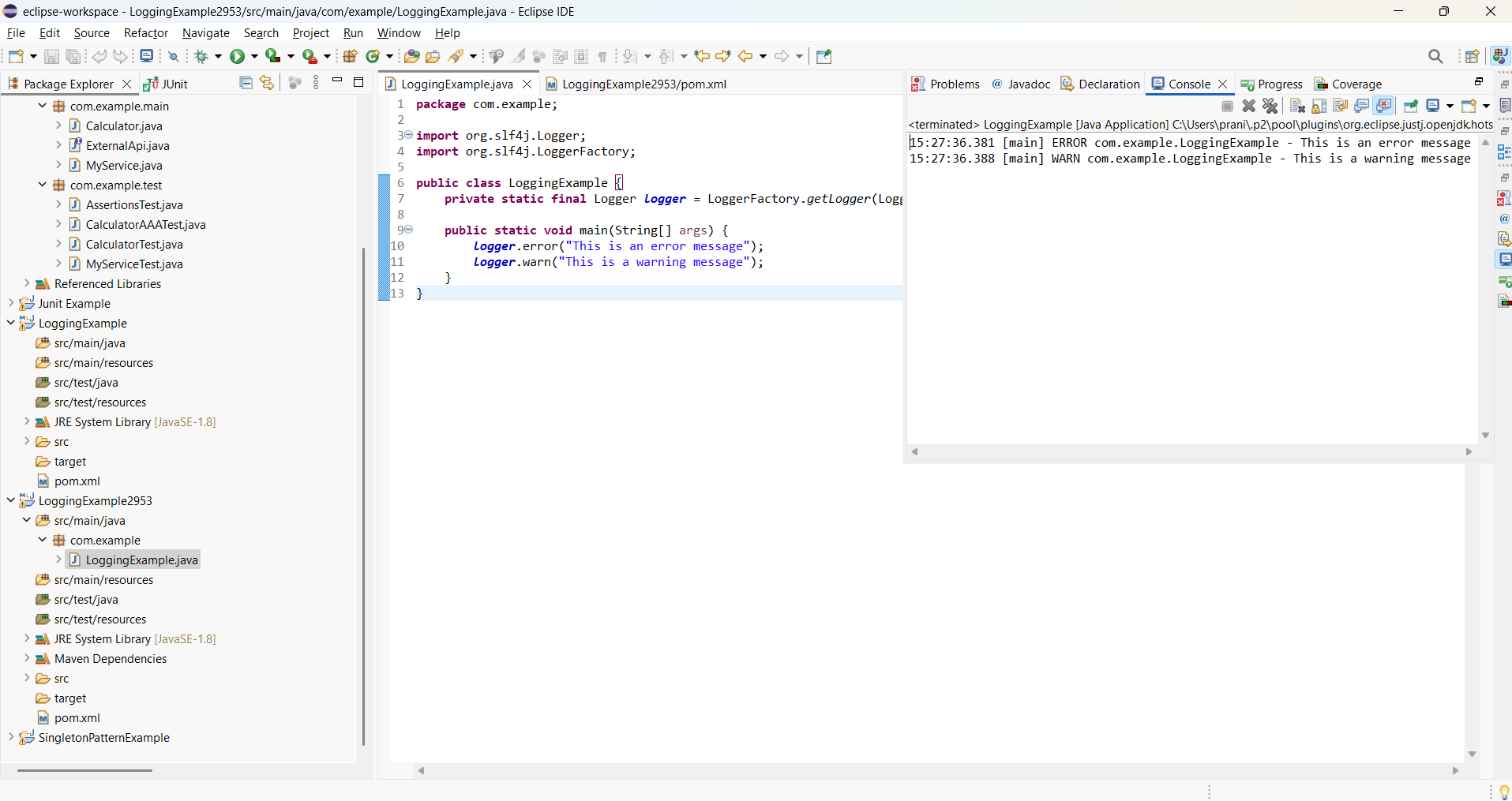
**public** **static** **void** main(String[] args) {

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

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

}

}

OUTPUT: