**Week-2**

**PL/SQL programming & JUnit, Mockito and SL4J**

**1: Control Structures**

**Scenario 1**: The bank wants to apply a discount to loan interest rates for customers above 60 years old.

* + Question: Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

**Scenario 2**: A customer can be promoted to VIP status based on their balance.

* + Question: Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

* + Question: Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

**Bank\_System.sql :-**

**Input\_Bank\_Data.sql**

CREATE DATABASE IF NOT EXISTS bank\_system;

USE bank\_system;

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100),

Age INT,

Balance DECIMAL(10,2),

IsVIP BOOLEAN DEFAULT FALSE

);

CREATE TABLE Loans (

LoanID INT PRIMARY KEY AUTO\_INCREMENT,

CustomerID INT,

InterestRate DECIMAL(5,2),

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

Sample Data for Verification:-

INSERT INTO Customers (Name, Age, Balance) VALUES

('John Smith', 65, 12000),

('Alice Brown', 45, 8000),

('Mark Davis', 70, 15000),

('Sara Lee', 30, 10500);

INSERT INTO Loans (CustomerID, InterestRate, DueDate) VALUES

(1, 7.5, CURDATE() + INTERVAL 10 DAY),

(2, 8.0, CURDATE() + INTERVAL 40 DAY),

(3, 6.9, CURDATE() + INTERVAL 25 DAY),

(4, 8.2, CURDATE() + INTERVAL 5 DAY);

**SeniorCitizen\_Discount\_Proc.sql**

DELIMITER //

CREATE PROCEDURE ApplySeniorCitizenDiscount()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE cust\_id INT;

DECLARE cust\_cursor CURSOR FOR

SELECT CustomerID FROM Customers WHERE Age > 60;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cust\_cursor;

read\_loop: LOOP

FETCH cust\_cursor INTO cust\_id;

IF done THEN

LEAVE read\_loop;

END IF;

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust\_id;

END LOOP;

CLOSE cust\_cursor;

END;

//

DELIMITER ;

-- Run the procedure

CALL ApplySeniorCitizenDiscount();

-- Check the updated loan interest rates

SELECT \* FROM Loans;

**Promote\_VIP\_Customers\_Proc.sql**

-- Procedure to promote customers with Balance > 10000 to VIP status

DELIMITER //

CREATE PROCEDURE PromoteVIPCustomers()

BEGIN

UPDATE Customers

SET IsVIP = TRUE

WHERE Balance > 10000;

END;

//

DELIMITER ;

-- Run the procedure

CALL PromoteVIPCustomers();

-- Check which customers are now VIP

SELECT \* FROM Customers;

**Loan\_Reminder\_Notification\_Proc.sql**

-- Procedure to send reminders for loans due within the next 30 days

DELIMITER //

CREATE PROCEDURE SendLoanReminders()

BEGIN

SELECT

c.Name AS CustomerName,

l.LoanID,

l.DueDate,

CONCAT('Reminder: Dear ', c.Name, ', your loan (ID: ', l.LoanID, ') is due on ',

DATE\_FORMAT(l.DueDate, '%d-%b-%Y')) AS Message

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate BETWEEN CURDATE() AND CURDATE() + INTERVAL 30 DAY;

END;

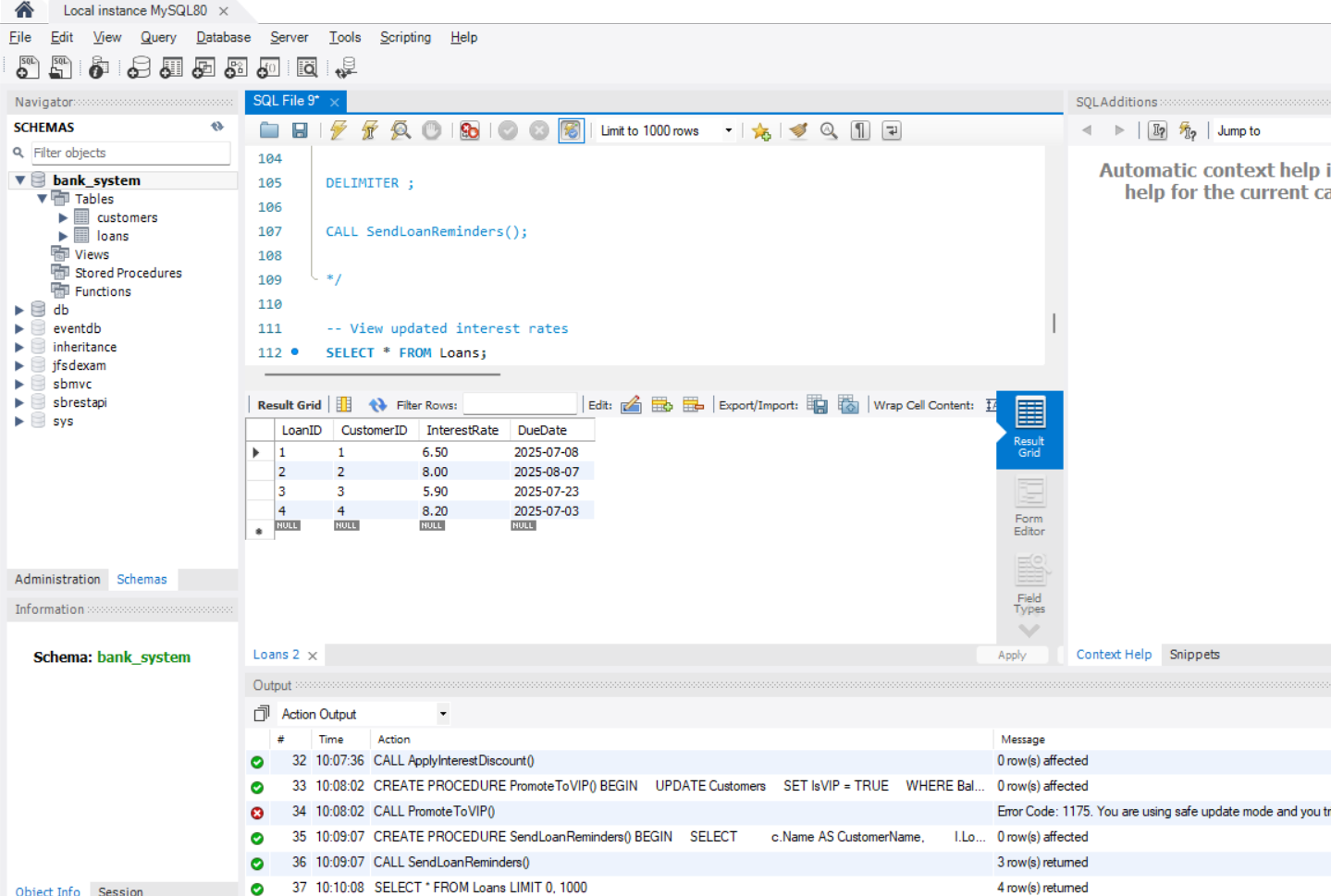
//

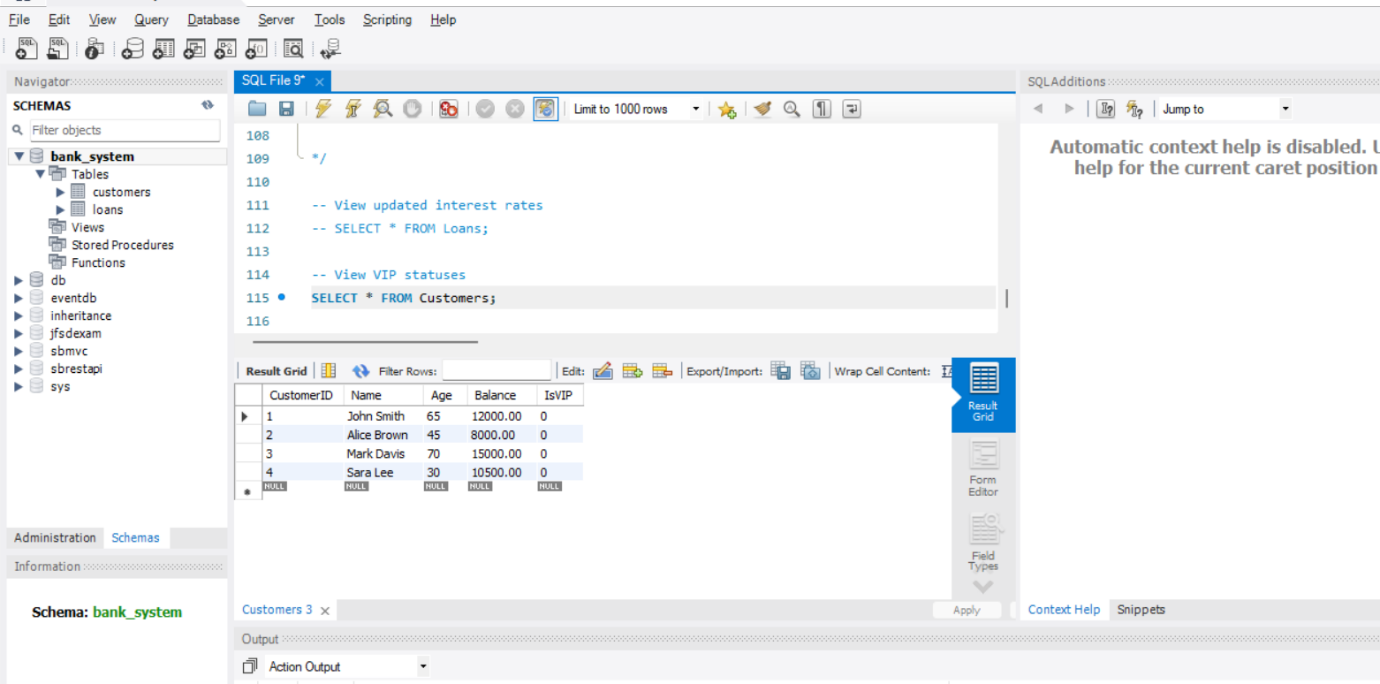
DELIMITER ;

-- Run the procedure

CALL SendLoanReminders();

**Outputs:-**





A screenshot of a computer

AI-generated content may be incorrect.

**2**. **Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

* + **Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

* + **Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

**Scenario 3:** Customers should be able to transfer funds between their accounts.

* + **Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

**Bank\_operations\_ex3:-**

**Input\_data.sql:-**

-- Create and select schema

CREATE DATABASE IF NOT EXISTS bank\_operations\_ex3;

USE bank\_operations\_ex3;

CREATE TABLE SavingsAccounts (

AccountID INT PRIMARY KEY AUTO\_INCREMENT,

CustomerName VARCHAR(100),

Balance DECIMAL(10, 2)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100),

Department VARCHAR(50),

Salary DECIMAL(10, 2)

);

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY AUTO\_INCREMENT,

CustomerName VARCHAR(100),

Balance DECIMAL(10, 2)

);

Sample data for Verification:-

INSERT INTO SavingsAccounts (CustomerName, Balance) VALUES

('Amit Sharma', 10000),

('Neha Rao', 15000),

('Ravi Kumar', 20000);

INSERT INTO Employees (Name, Department, Salary) VALUES

('Arun', 'Sales', 50000),

('Divya', 'Sales', 52000),

('Karan', 'IT', 60000);

INSERT INTO Accounts (CustomerName, Balance) VALUES

('Suresh', 8000),

('Ramesh', 5000);

**Monthly\_interest\_proc.sql:-**

-- Use the correct schema

USE bank\_operations\_ex3;

-- Procedure to apply 1% interest to all savings accounts

DELIMITER //

CREATE PROCEDURE ProcessMonthlyInterest()

BEGIN

UPDATE SavingsAccounts

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

END;

//

DELIMITER ;

-- Call the procedure

CALL ProcessMonthlyInterest();

-- View updated balances

SELECT \* FROM SavingsAccounts;E

**Employee\_bonus\_procd.sql:-**

-- Use the correct schema

USE bank\_operations\_ex3;

-- Procedure to update employee salary with bonus percent

DELIMITER //

CREATE PROCEDURE UpdateEmployeeBonus(

IN deptName VARCHAR(50),

IN bonusPercent DECIMAL(5,2)

)

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* bonusPercent / 100)

WHERE Department = deptName;

END;

//

DELIMITER ;

-- Call the procedure (example: 10% bonus for Sales)

CALL UpdateEmployeeBonus('Sales', 10);

-- View updated salaries

SELECT \* FROM Employees;

**Transfer\_Funds.sql:-**

-- Use the correct schema

USE bank\_operations\_ex3;

-- Procedure to transfer funds between accounts

DELIMITER //

CREATE PROCEDURE TransferFunds(

IN fromAccount INT,

IN toAccount INT,

IN amount DECIMAL(10,2)

)

BEGIN

DECLARE fromBalance DECIMAL(10,2);

SELECT Balance INTO fromBalance

FROM Accounts

WHERE AccountID = fromAccount;

IF fromBalance >= amount THEN

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountID = fromAccount;

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = toAccount;

ELSE

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Insufficient funds in source account.';

END IF;

END;

//

DELIMITER ;

-- Call the procedure (example: transfer 2000 from Account 1 to 2)

CALL TransferFunds(1, 2, 2000);

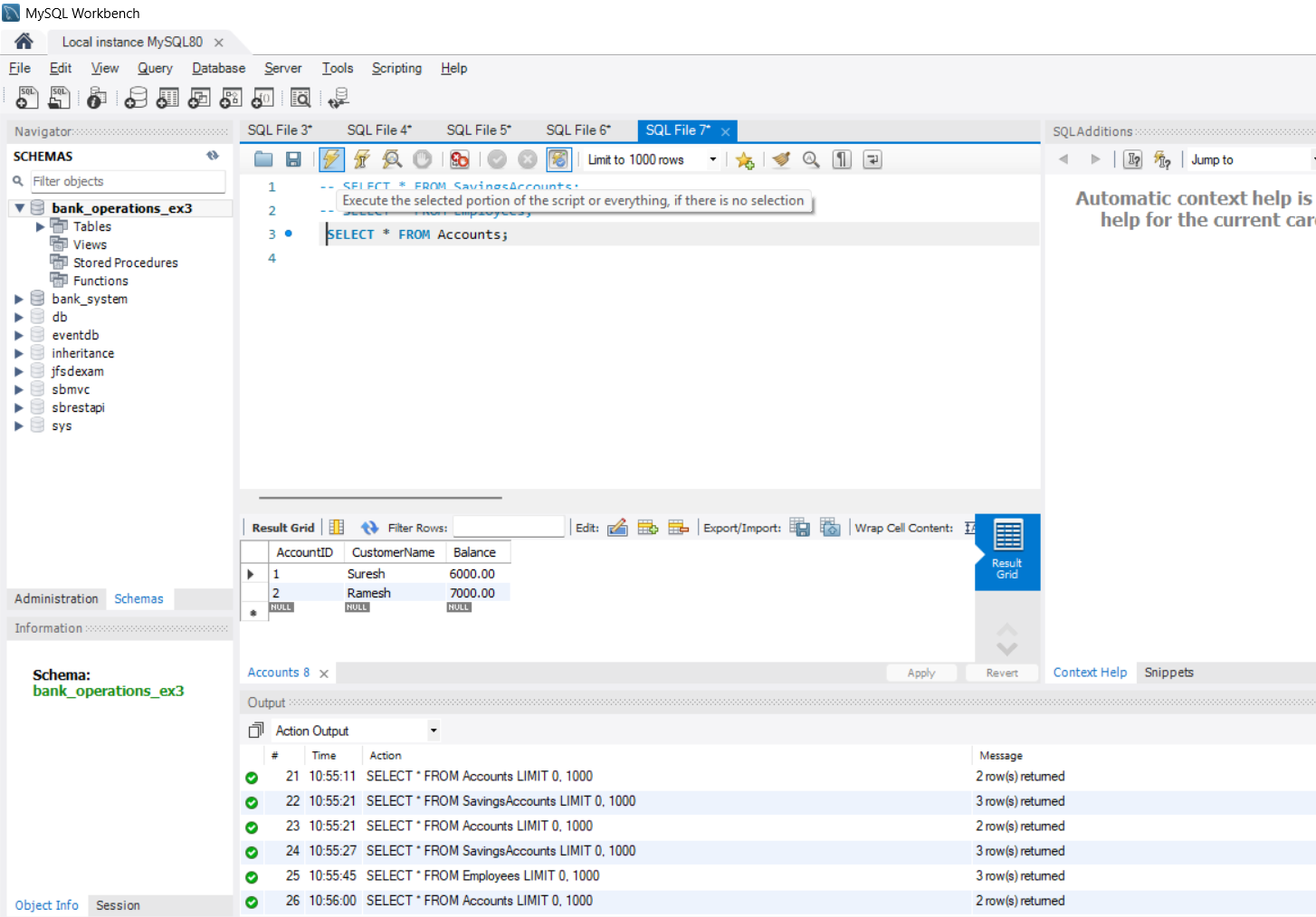
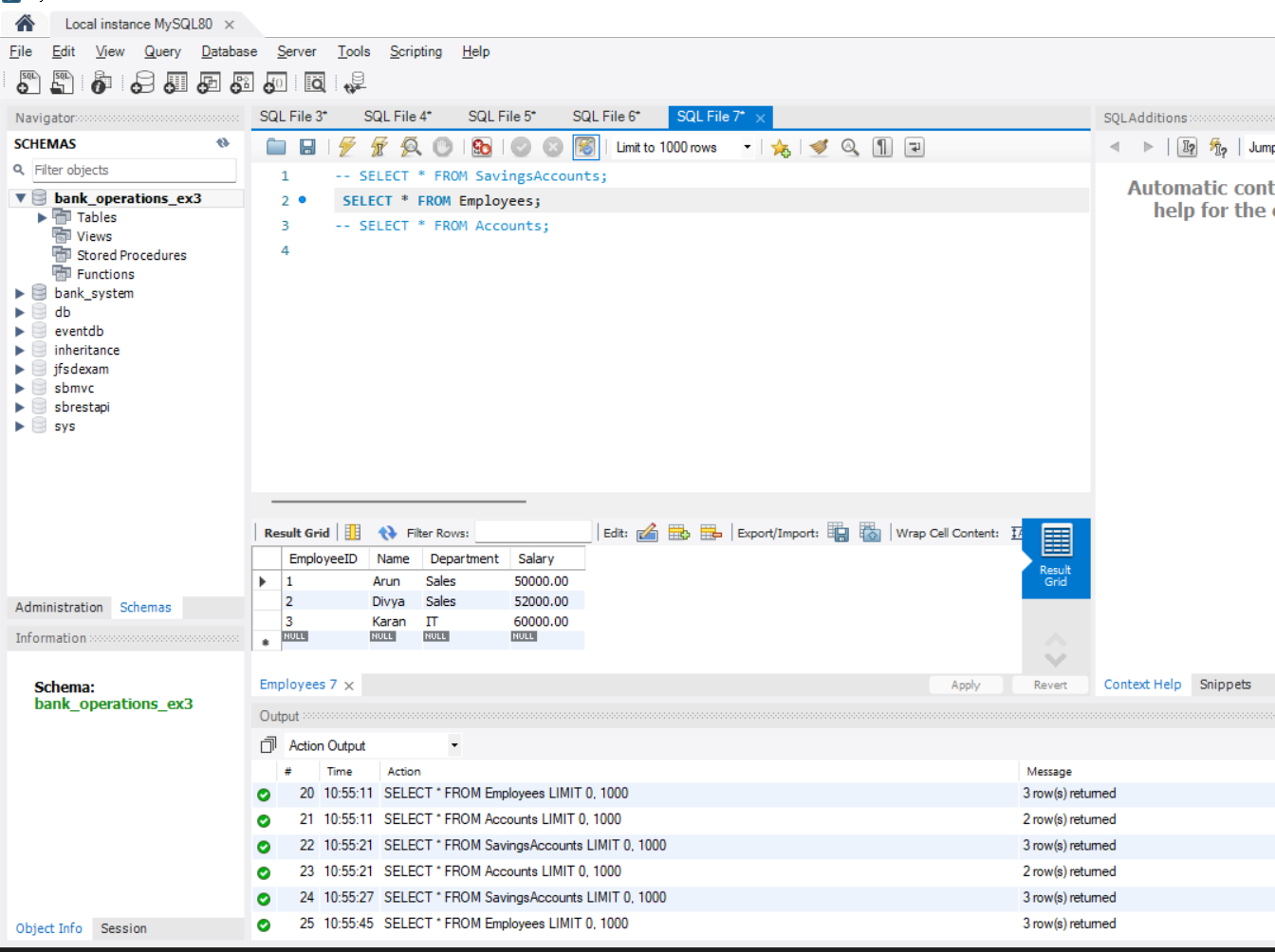
-- View updated balances

SELECT \* FROM Accounts;

**Output:-**

**A screenshot of a computer

AI-generated content may be incorrect.**

****

**JUnit, Mockito and SL4J**

**3. Setting Up Junit**

**Scenario:**

You need to set up JUnit in your Java project to start writing unit tests.

Steps:

I. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).

2. Add JUnit dependency to your project. If you are using Maven, add the following to your

pom.xml:

<dependency>

<groupld>junit</groupld>

<artifactld>junit</artifactld>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

3. Create a new test class in your project.

**Step 1:** Created a new Maven project in Eclipse with Group ID com.example and Artifact ID junit-demo.

**Step 2:** Add JUnit Dependency to pom.xml

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

**Step 3: Calculator.java** under src/main/java/com/example/**Addition**.java

package com.example;

public class Addition {

public int add(int a, int b) {

return a + b;

}}

**Step4:AdditionTest.java** under src/test/java/com/example/**AdditionTest.jav**

package com.example;

import static org.junit.Assert.\*;

import org.junit.Test;

public class AdditionTest {

@Test

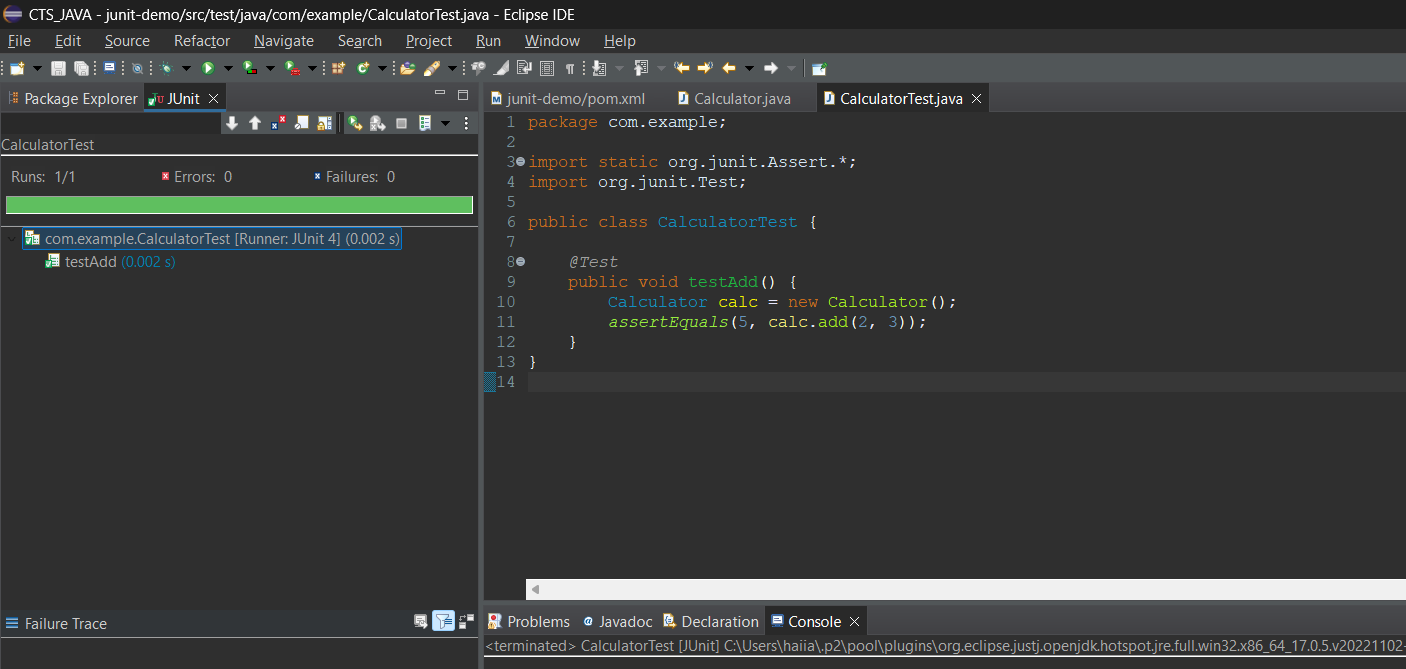
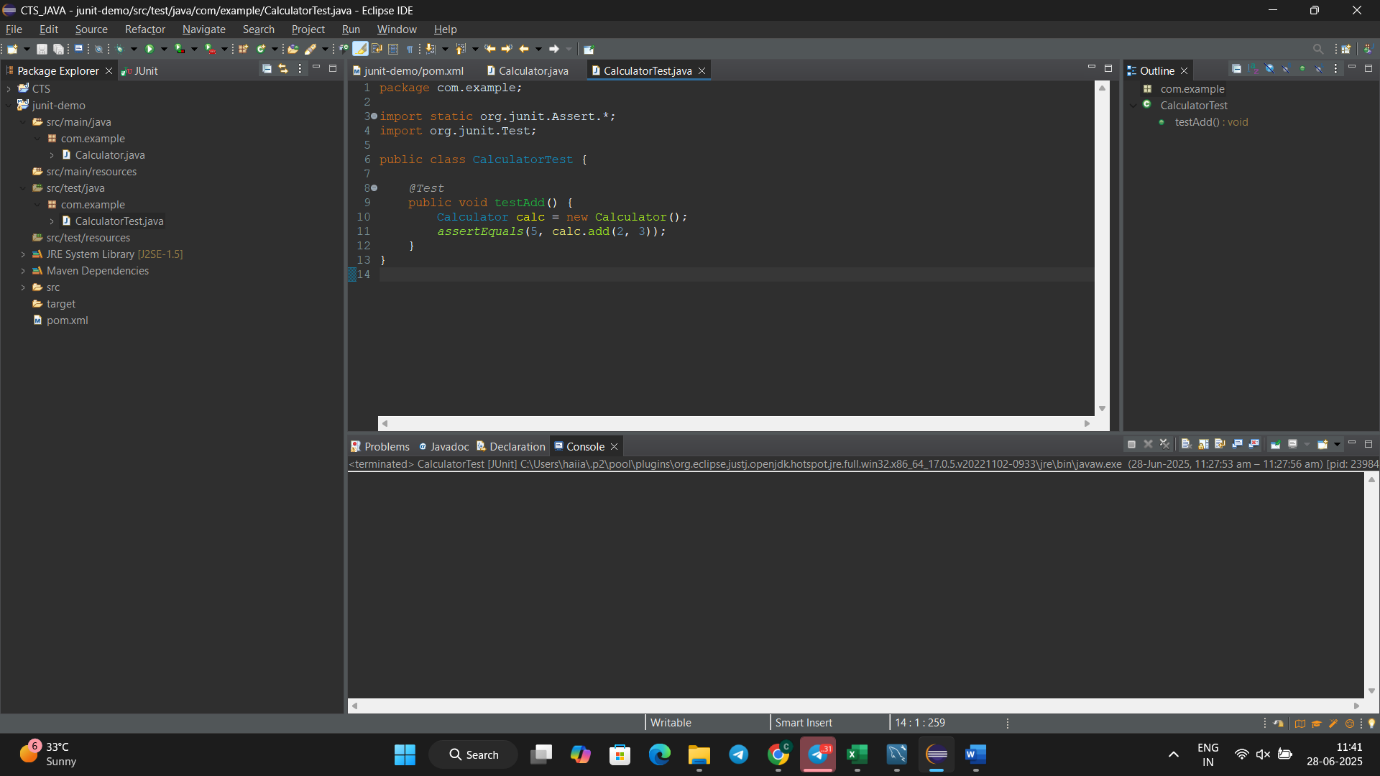
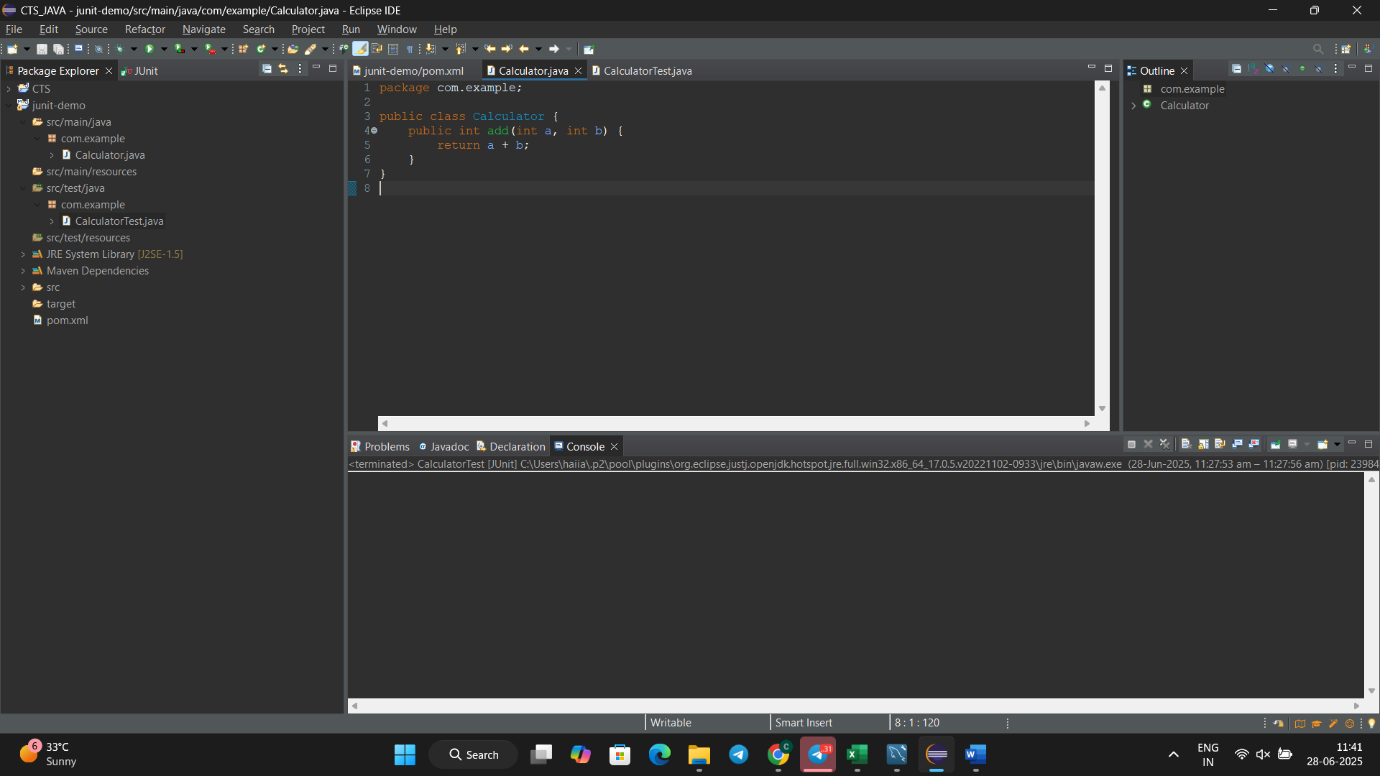
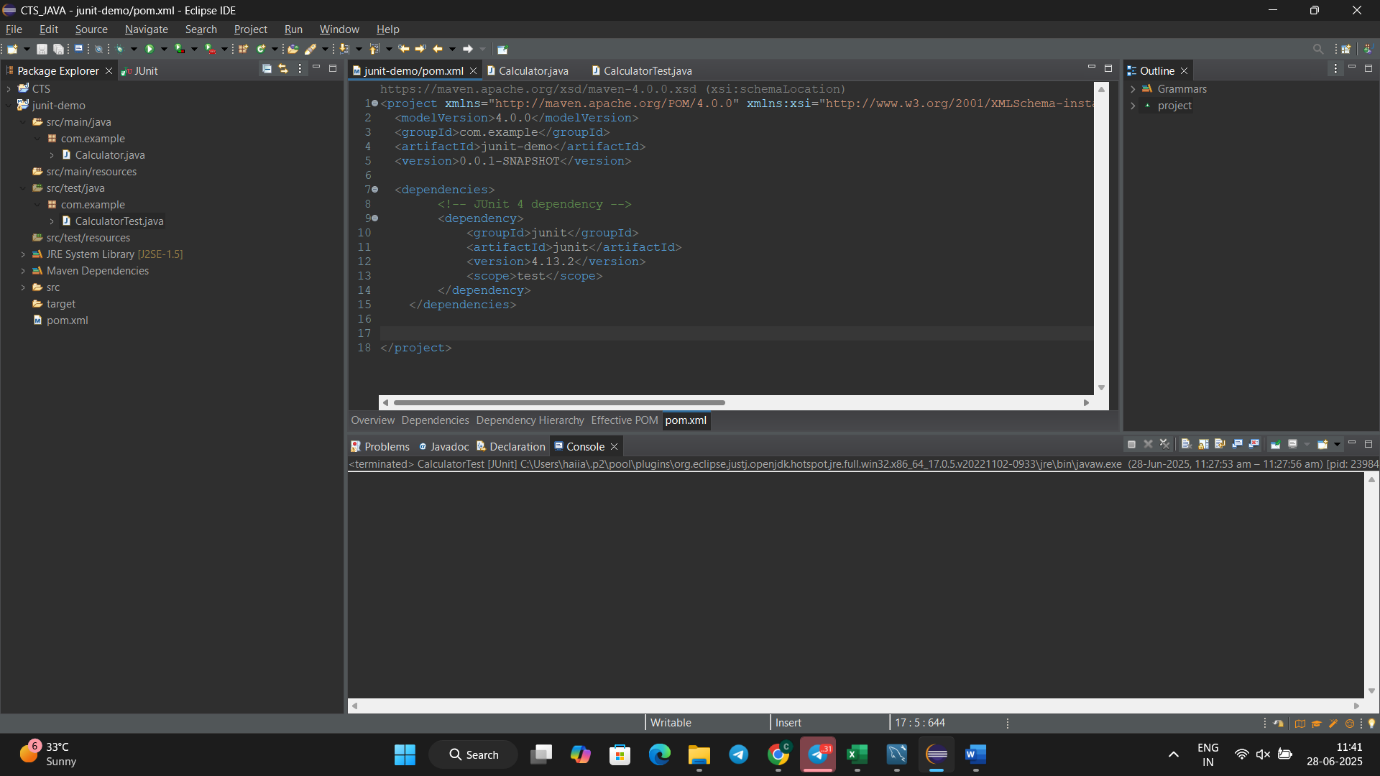
public void testAdd() {

Addition calc = new Addition();

assertEquals(5, calc.add(2, 3));

}

}

**Output:- **

**4. Assertions in Junit:-**

**Step-1:-** Create a test file named with **AssertionsTest.java.**

**Under** src/test/java/com/example/

package com.example;

import static org.junit.Assert.\*;

import org.junit.Test;

public class AssertionsTest {

@Test

public void testAssertions() {

// Assert equals

assertEquals(5, 2 + 3);

// Assert true

assertTrue(5 > 3);

// Assert false

assertFalse(5 < 3);

// Assert null

assertNull(null);

// Assert not null

assertNotNull(new Object());

}

}

**Output:-**

A computer screen shot of a program

AI-generated content may be incorrect.

**5. Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit**

**Step-1:-** Create a test file named with **CalculatorTestWithSetup.java**

**Under** src/test/java/com/example/

package com.example;

import static org.junit.Assert.\*;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

public class CalculatorTestWithSetup {

private Calculator calculator;

// Setup method (runs before each test)

*@Before*

public void setUp() {

calculator = new Calculator(); // Arrange

System.***out***.println("Setup: Calculator initialized");

}

// Teardown method (runs after each test)

*@After*

public void tearDown() {

calculator = null;

System.***out***.println("Teardown: Calculator cleaned up");

}

*@Test*

public void testAddition() {

// Act

int result = calculator.add(10, 5);

// Assert

*assertEquals*(15, result);

}

*@Test*

public void testAdditionWithNegativeNumbers() {

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

*assertEquals*(-5, result);

}

}

**Output:-**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**TDD using JUnit5 and Mockito**

**6. Mocking and Stubbing**

**Add Dependencies:**

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.11.0</version> <!-- or latest -->

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version> <!-- or latest -->

<scope>test</scope>

</dependency>

**externalAPI interface:**

package com.cognizant.Week2;

public interface ExternalApi {

String getData();

}

**Created the MyService Class:**

package com.cognizant.Week2;

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**myServiceTest Class:**

package com.cognizant.Week2;

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() {

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

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

MyService service = new MyService(mockApi);

String result = service.fetchData();

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

}

}

**Outputs:-**

A screen shot of a computer

AI-generated content may be incorrect.A screen shot of a computer

AI-generated content may be incorrect.A screen shot of a computer

AI-generated content may be incorrect.

**7. Mocking and Stubbing with Multiple Returns:**

**ApiStub** interface:-

package com.example.Week2;

public interface ApiStub {

String getData();

}

**My Service.java:-**

package com.example.Week2;

public class MyServiceStub {

private ApiStub externalApiStub;

public MyServiceStub(ApiStub externalApiStub) {

this.externalApiStub = externalApiStub;}

public String fetchFirstCall() {

return externalApiStub.getData();

}

public String fetchSecondCall() {

return externalApiStub.getData();

}}

**MyServiceTest.java:-**

package com.example.Week2;

import static org.mockito.Mockito.\*;

import static org.junit.Assert.\*;

import org.junit.Test;

public class MyServiceTestStub {

*@Test*

public void testExternalApiMultipleReturns() {

ApiStub mockApi = *mock*(ApiStub.class);

*when*(mockApi.getData())

.thenReturn("First Response")

.thenReturn("Second Response");

MyServiceStub service = new MyServiceStub(mockApi);

String first = service.fetchFirstCall();

String second = service.fetchSecondCall();

System.***out***.println("First: " + first);

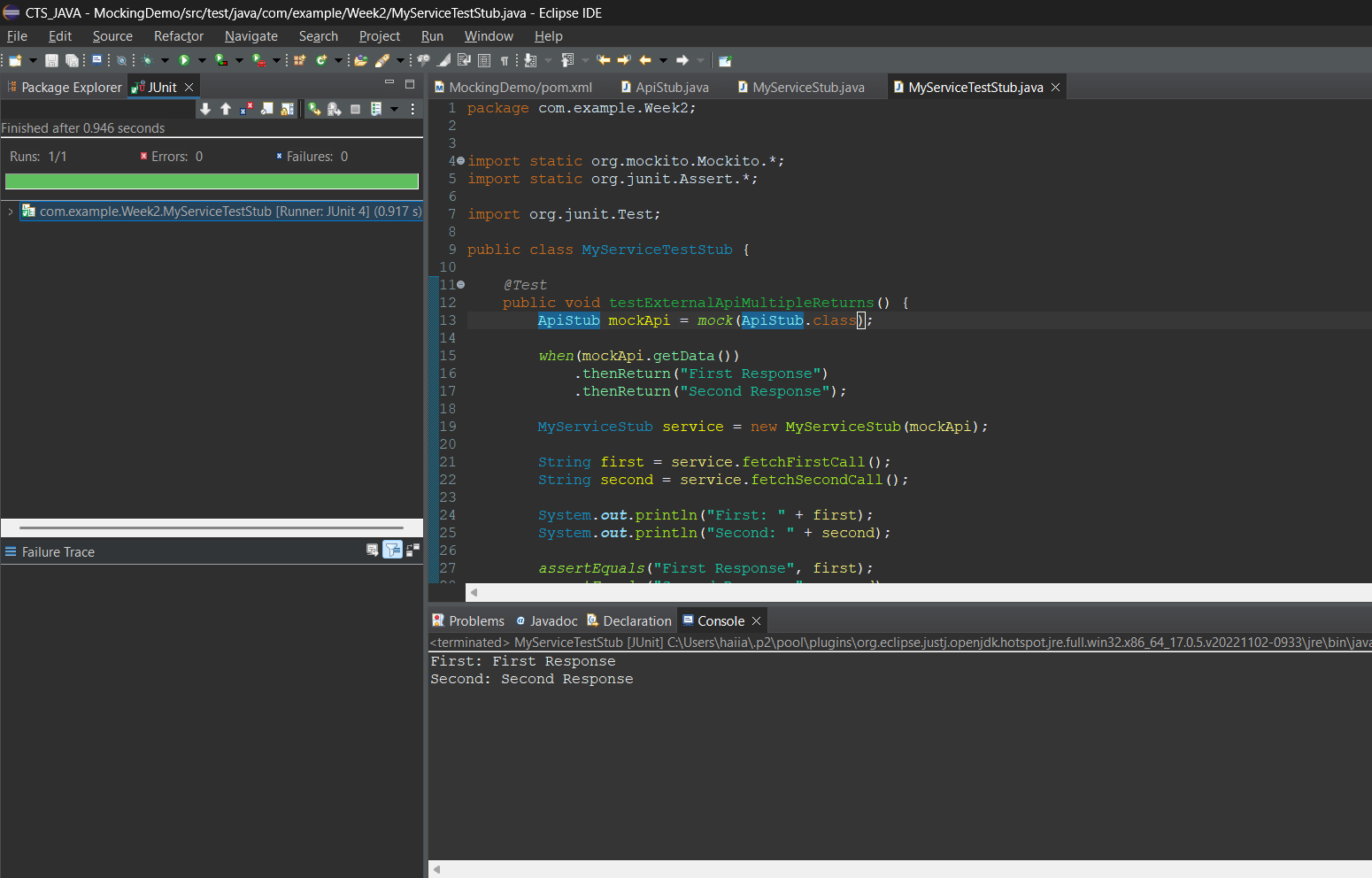
System.***out***.println("Second: " + second);

*assertEquals*("First Response", first);

*assertEquals*("Second Response", second);

}}

**Outputs :-**A screenshot of a computer

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.

**8. Verifying Interactions:**

**NotificationService:**

package com.cognizant.Week2;

public interface NotificationService {

void sendEmail();

void sendSMS();

void sendPush();

}

**UserNotifier:**

package com.cognizant.Week2;

public class UserNotifier {

private NotificationService service;

public UserNotifier(NotificationService service) {

this.service = service;

}

public void notifyUser() {

service.sendEmail();

service.sendSMS();

service.sendPush();

}

}

**UserNotifierTest:**

package com.cognizant.Week2;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

public class UserNotifierTest {

@Test

public void testNotificationOrder() {

NotificationService mockService = *mock*(NotificationService.class);

UserNotifier notifier = new UserNotifier(mockService);

notifier.notifyUser();

InOrder inOrder = *inOrder*(mockService);

inOrder.verify(mockService).sendEmail();

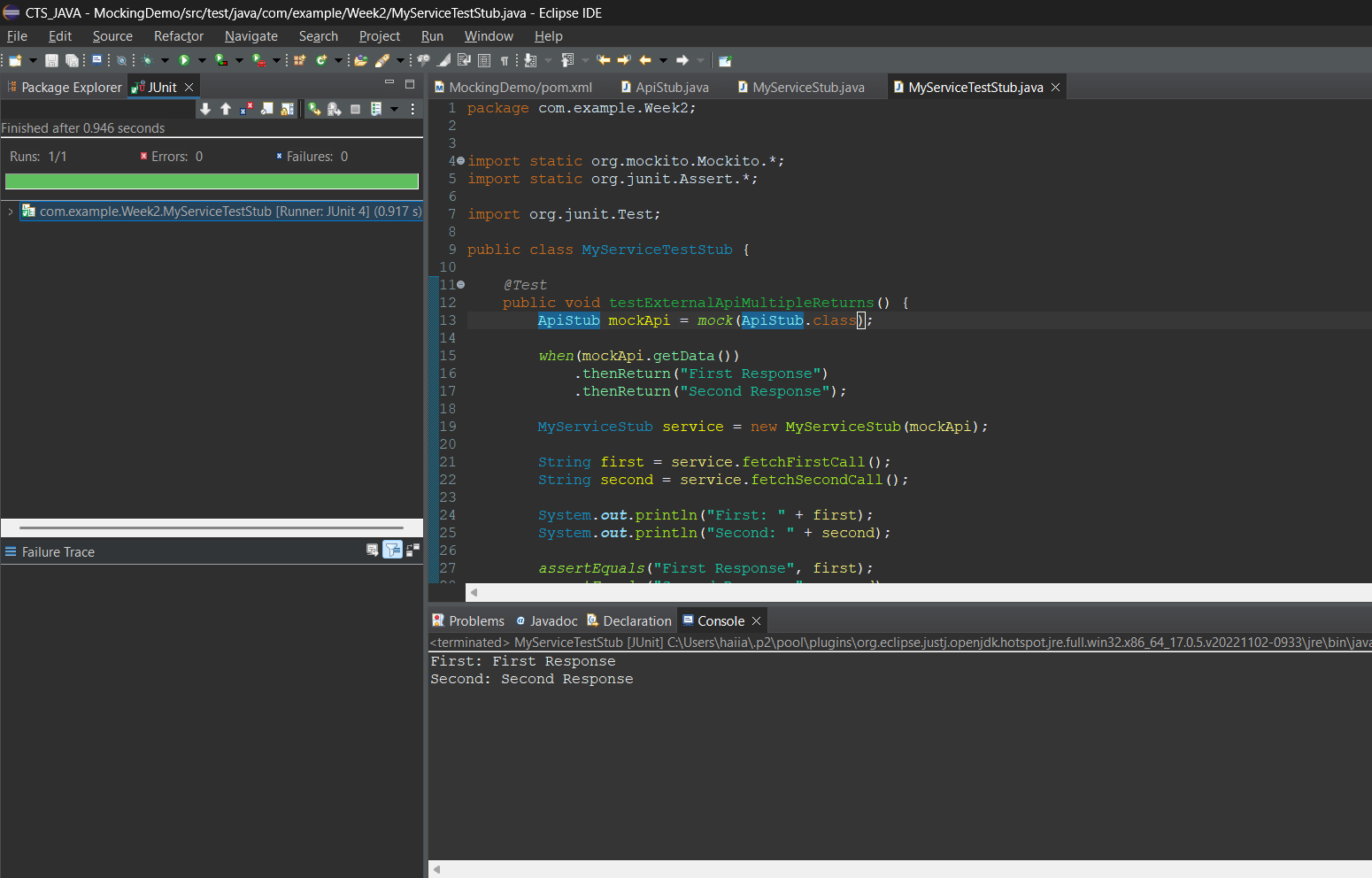
inOrder.verify(mockService).sendSMS();

inOrder.verify(mockService).sendPush();

}

}

**Output:**



**SLF4J logging framework**

**9. Logging Error Messages and Warning Levels:**

**Added Dependencies:**

<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>

**Logging.java:**

package com.cognizant.Week2;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class Logging {

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

public static void main(String[] args) {

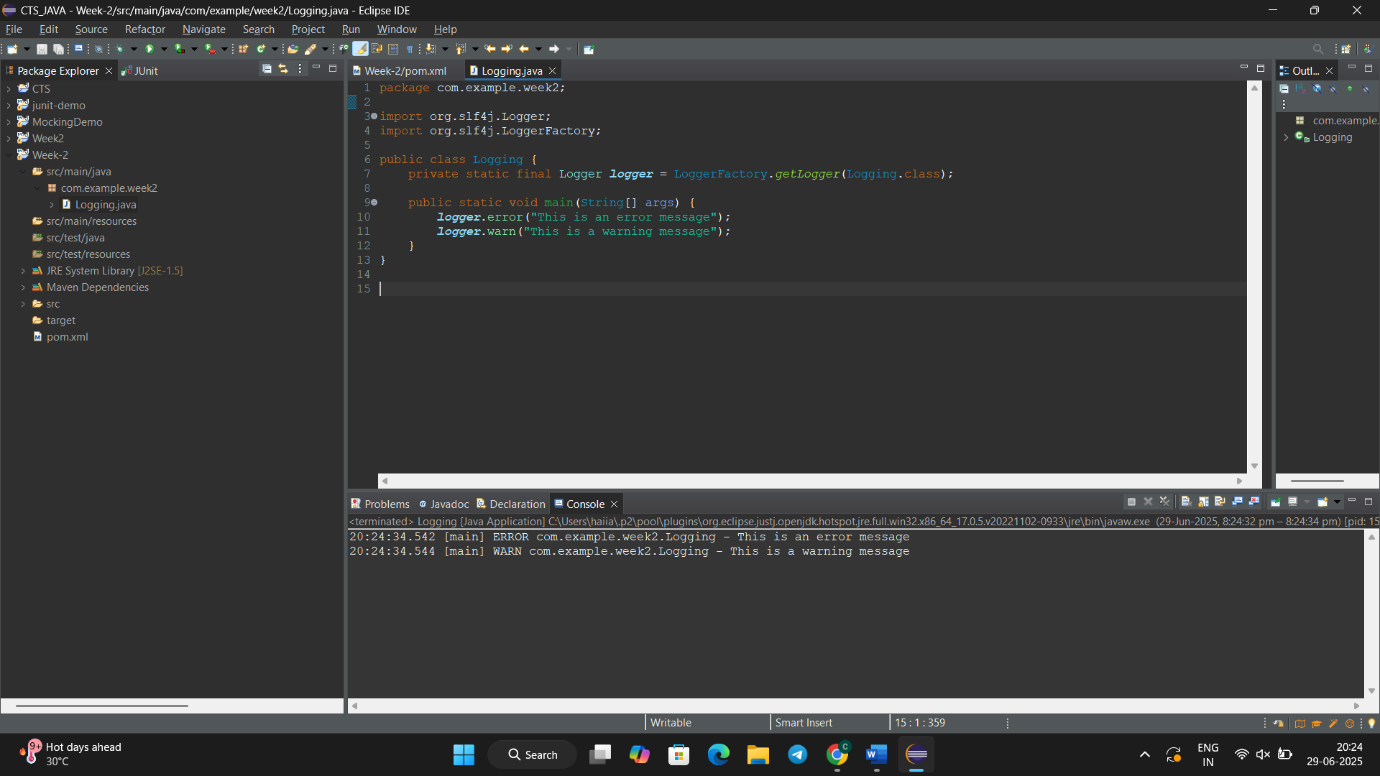
*logger*.error("This is an error message");

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

}

}

**Output:**

****