**WEEK-2**

**PLSQL**

**Exercise 1: Control Structures**

**MY CODE:**

CREATE TABLE CUSTOMER\_LOANS (

CUSTOMER\_ID NUMBER,

AGE NUMBER,

INTEREST\_RATE NUMBER(5,2)

);

INSERT INTO CUSTOMER\_LOANS (CUSTOMER\_ID, AGE, INTEREST\_RATE)

VALUES (101, 65, 7.5);

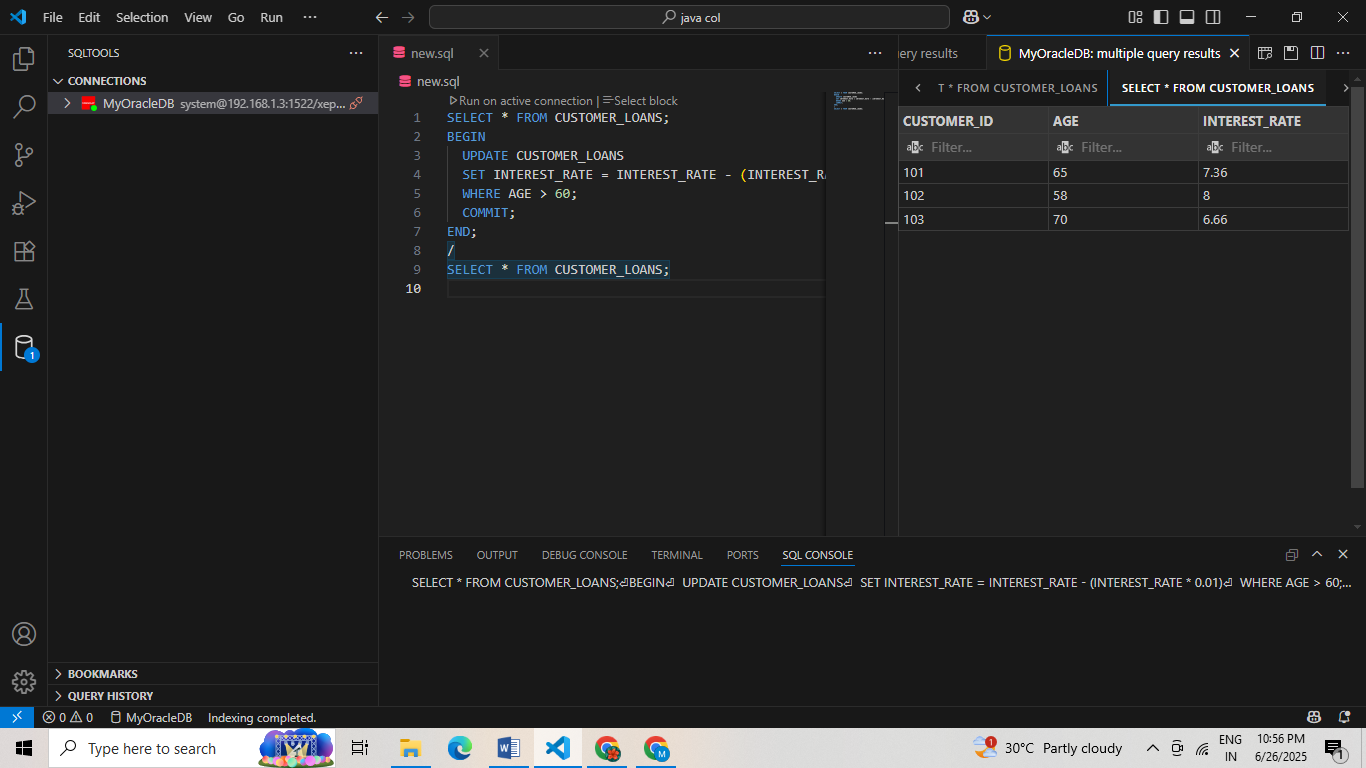
INSERT INTO CUSTOMER\_LOANS (CUSTOMER\_ID, AGE, INTEREST\_RATE)

VALUES (102, 58, 8.0);

INSERT INTO CUSTOMER\_LOANS (CUSTOMER\_ID, AGE, INTEREST\_RATE)

VALUES (103, 70, 6.8);

COMMIT;

**OUTPUT:**

**SCENARIO 1**

BEGIN

UPDATE CUSTOMER\_LOANS

SET INTEREST\_RATE = INTEREST\_RATE - (INTEREST\_RATE \* 0.01)

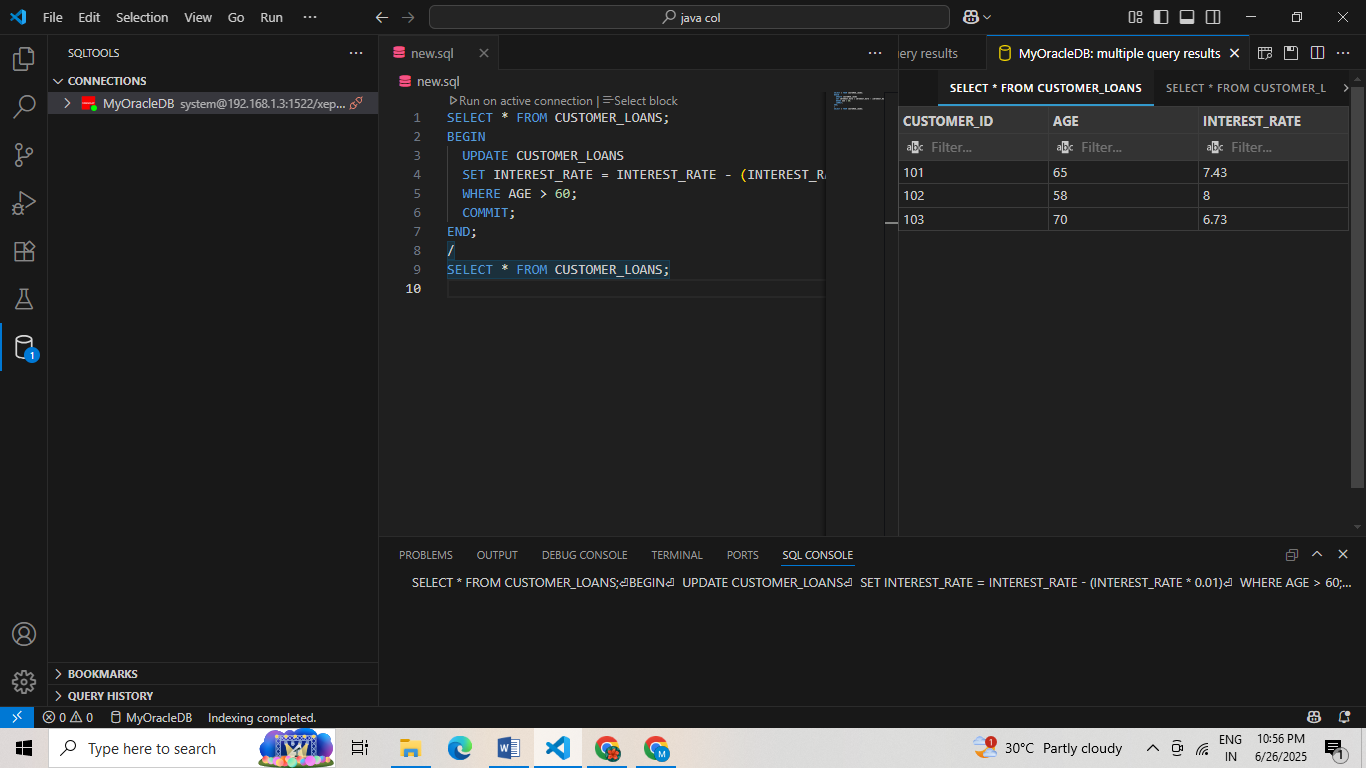
WHERE AGE > 60;

COMMIT;

END;

/

**OUTPUT:**

****

**SCENARIO 2:**

ALTER TABLE CUSTOMER\_LOANS ADD IS\_VIP VARCHAR2(1);

BEGIN

UPDATE CUSTOMERS

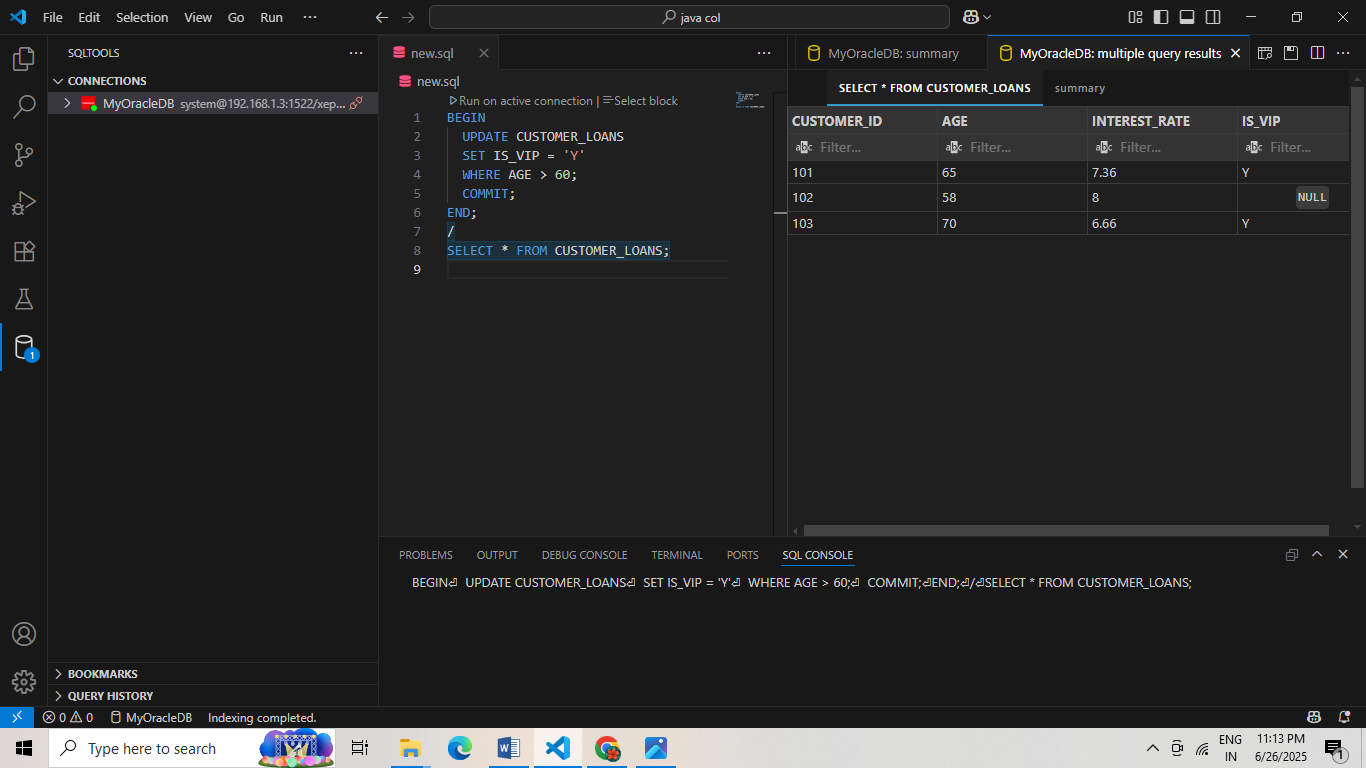
SET IS\_VIP = 'Y'

WHERE BALANCE > 10000;

COMMIT;

END;

**OUTPUT:**



**SCENARIO 3:**

SELECT COLUMN\_NAME

FROM USER\_TAB\_COLUMNS

WHERE TABLE\_NAME = 'CUSTOMER\_LOANS'

AND COLUMN\_NAME = 'DUE\_DATE';

ALTER TABLE CUSTOMER\_LOANS ADD DUE\_DATE DATE;

UPDATE CUSTOMER\_LOANS

SET DUE\_DATE = SYSDATE + 10

WHERE CUSTOMER\_ID = 101;

UPDATE CUSTOMER\_LOANS

SET DUE\_DATE = SYSDATE + 25

WHERE CUSTOMER\_ID = 102;

UPDATE CUSTOMER\_LOANS

SET DUE\_DATE = SYSDATE + 40

WHERE CUSTOMER\_ID = 103

COMMIT;

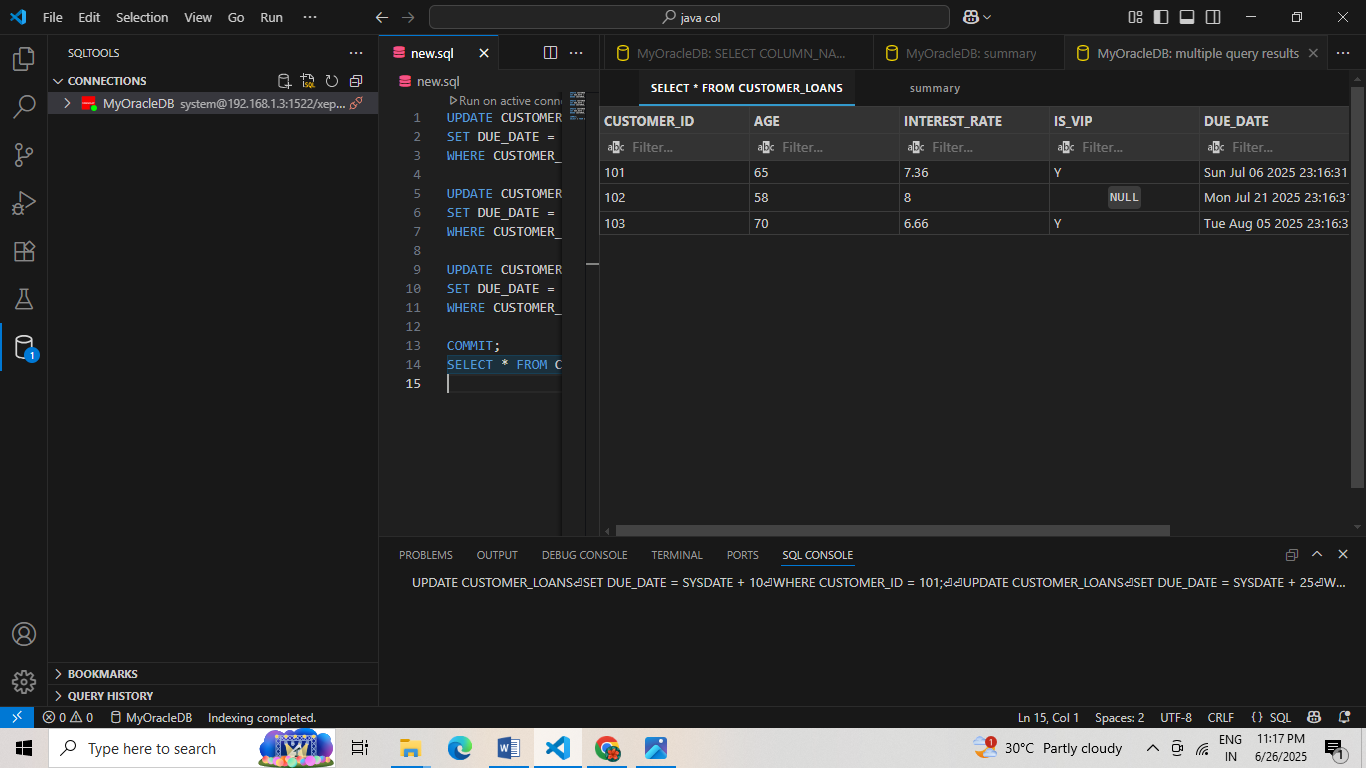
SELECT 'Reminder: Loan for customer ' || CUSTOMER\_ID ||

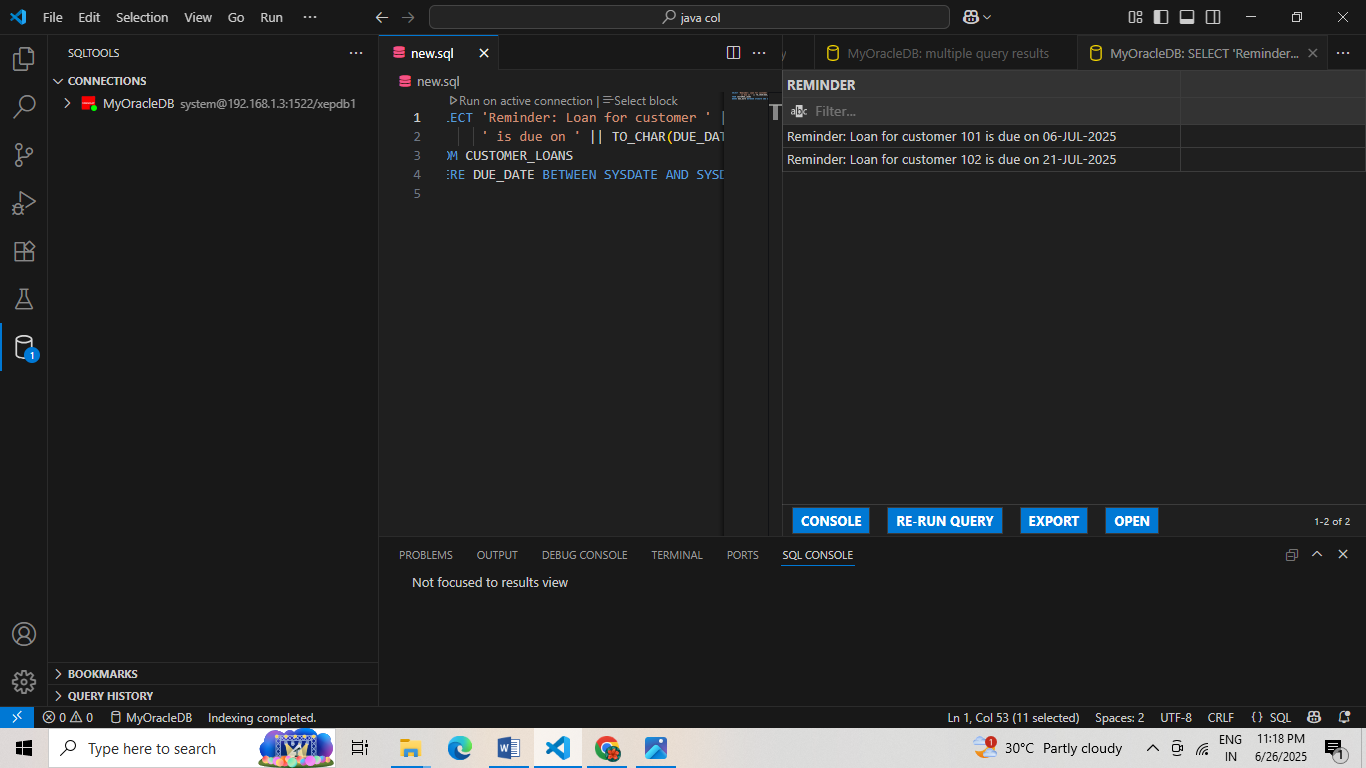
' is due on ' || TO\_CHAR(DUE\_DATE, 'DD-MON-YYYY') AS REMINDER

FROM CUSTOMER\_LOANS

WHERE DUE\_DATE BETWEEN SYSDATE AND SYSDATE + 30;

**OUTPUT:**





**Exercise 3: Stored Procedures**

**MY CODE:**

CREATE TABLE SAVINGS\_ACCOUNTS (

ACCOUNT\_ID NUMBER PRIMARY KEY,

BALANCE NUMBER

);

CREATE TABLE EMPLOYEES (

EMP\_ID NUMBER PRIMARY KEY,

DEPT\_ID NUMBER,

SALARY NUMBER

);

CREATE TABLE ACCOUNTS (

ACCOUNT\_ID NUMBER PRIMARY KEY,

BALANCE NUMBER

);

INSERT INTO SAVINGS\_ACCOUNTS VALUES (101, 5000);

INSERT INTO SAVINGS\_ACCOUNTS VALUES (102, 10000);

INSERT INTO EMPLOYEES VALUES (1, 5, 40000);

INSERT INTO EMPLOYEES VALUES (2, 5, 45000);

INSERT INTO ACCOUNTS VALUES (201, 7000);

INSERT INTO ACCOUNTS VALUES (202, 3000);

COMMIT;

**SCENARIO 1**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

UPDATE SAVINGS\_ACCOUNTS

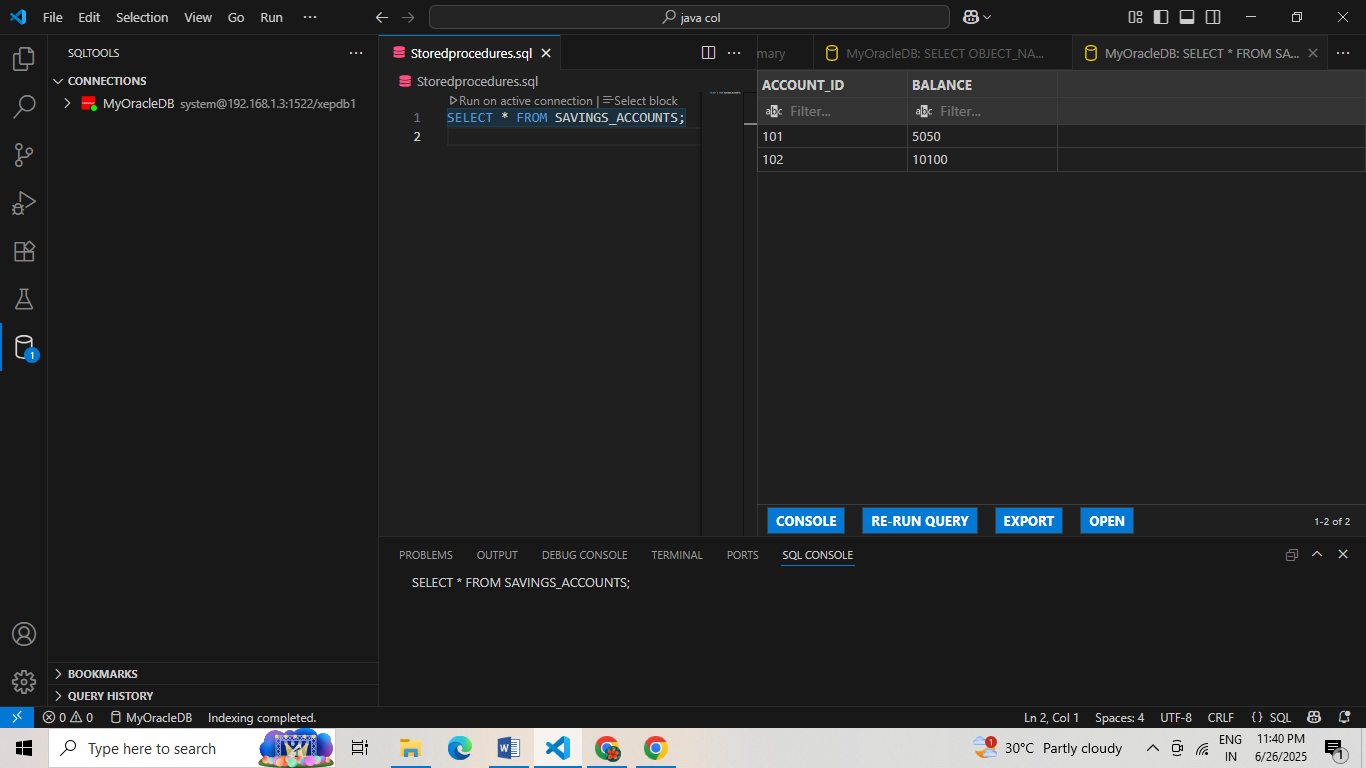
SET BALANCE = BALANCE + (BALANCE \* 0.01);

COMMIT;

END;

/

**OUTPUT:**

****

**SCENARIO 2**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_dept\_id IN NUMBER,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE EMPLOYEES

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

WHERE DEPT\_ID = p\_dept\_id;

COMMIT;

END;

/

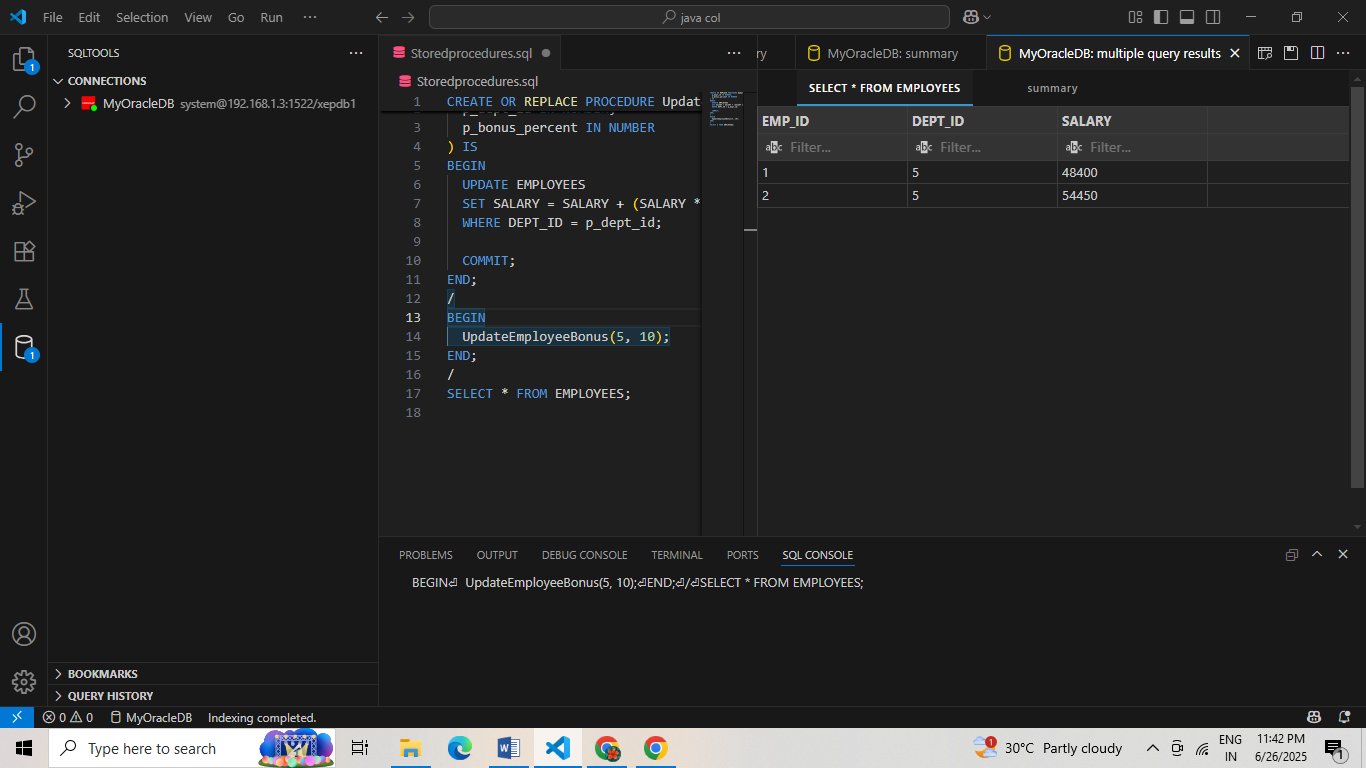
BEGIN

UpdateEmployeeBonus(5, 10);

END;

/

**OUTPUT:**

****

**SCENARIO 3**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

-- Check source account balance

SELECT BALANCE INTO v\_balance

FROM ACCOUNTS

WHERE ACCOUNT\_ID = p\_from\_account;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds');

END IF;

-- Deduct from source

UPDATE ACCOUNTS

SET BALANCE = BALANCE - p\_amount

WHERE ACCOUNT\_ID = p\_from\_account;

-- Credit to target

UPDATE ACCOUNTS

SET BALANCE = BALANCE + p\_amount

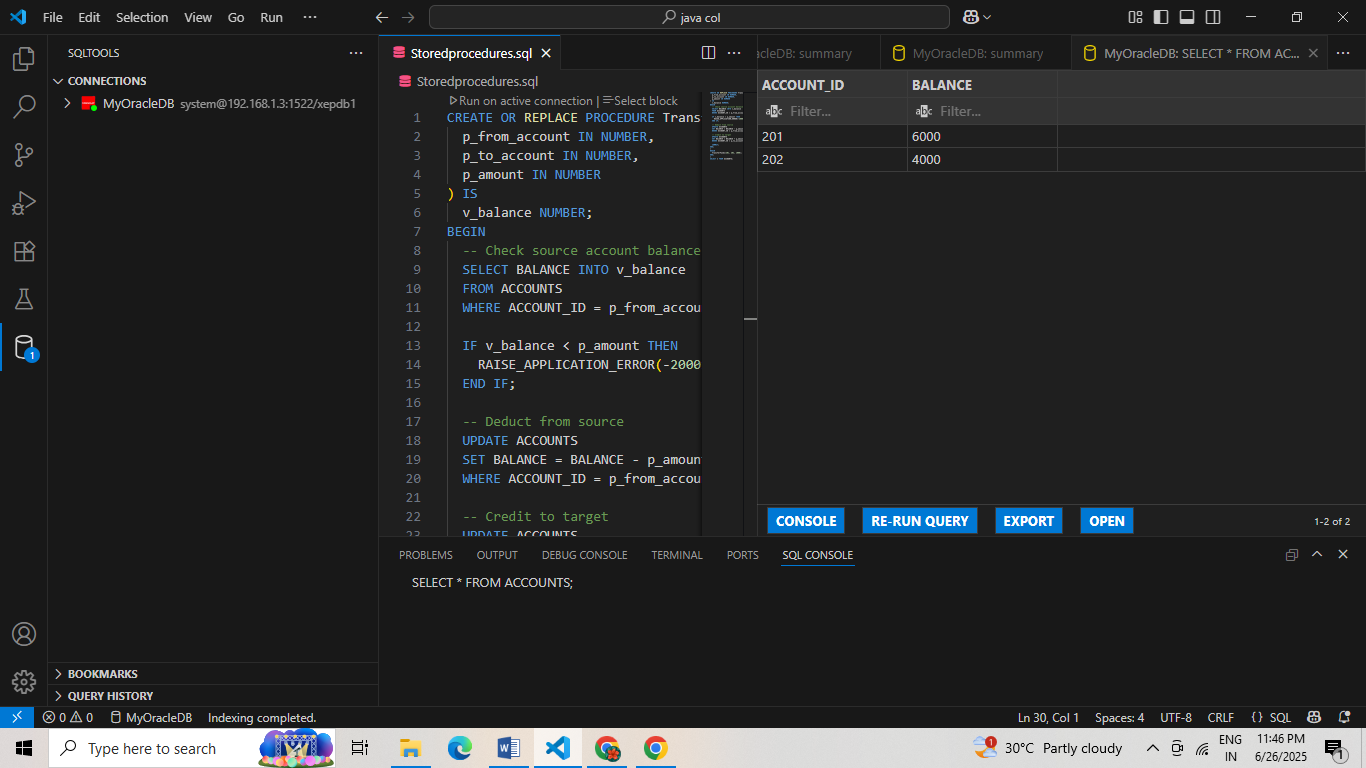
WHERE ACCOUNT\_ID = p\_to\_account;

COMMIT;

END;

/

**OUTPUT:**

****

**JUNIT TESTING**

**Exercise 1: Setting Up JUnit**

**MY CODE:**

**Calculator.java**

package com.example;

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;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

Calculator calc = new Calculator();

@Test

public void testAdd() {

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

}

@Test

public void testSubtract() {

assertEquals(1, calc.subtract(4, 3));

}

@Test

public void testMultiply() {

assertEquals(12, calc.multiply(3, 4));

}

@Test

public void testDivide() {

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

}

@Test(expected = IllegalArgumentException.class)

public void testDivideByZero() {

calc.divide(5, 0);

} ]

**Pom.xml**

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

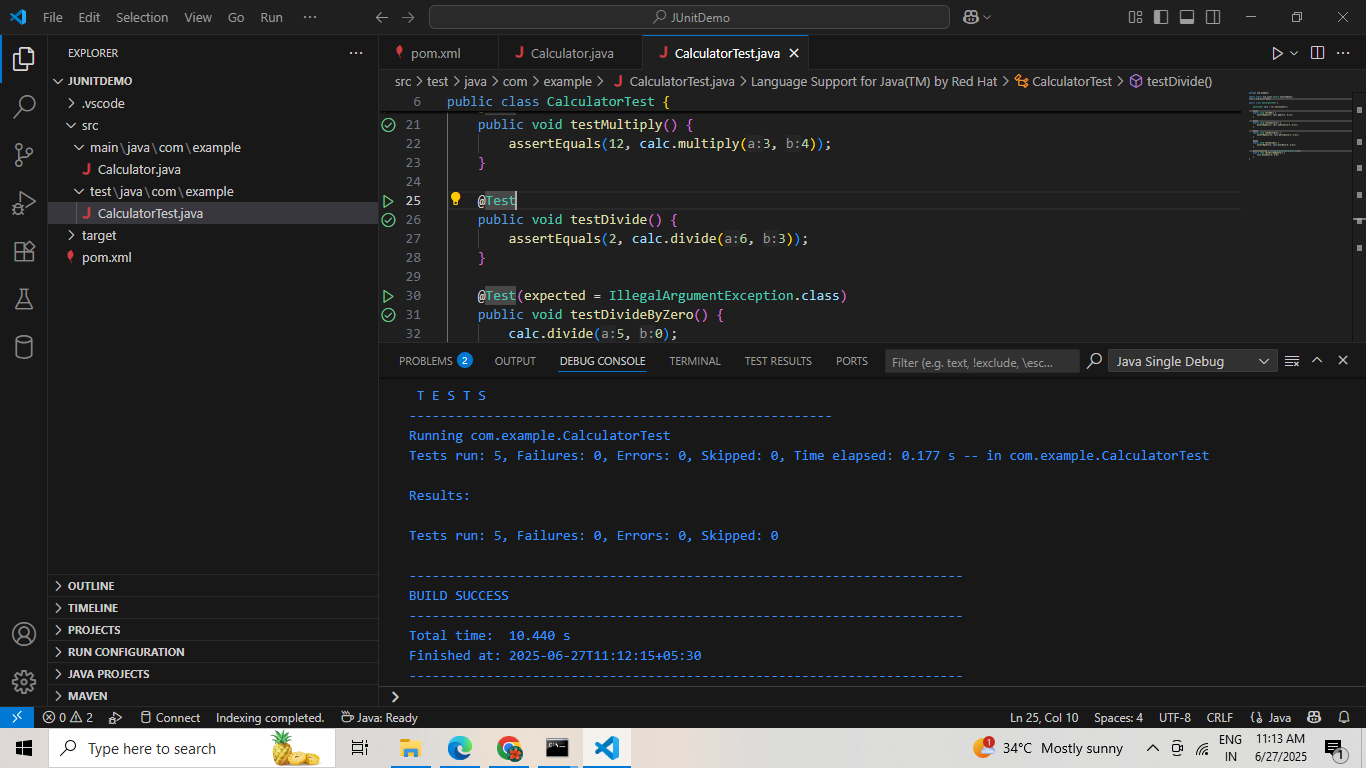
<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

**OUTPUT:**

****

**Exercise 3: Assertions in Junit**

**MY CODE:**

**AssertionsTest.java**

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals(5, 2 + 3);

assertTrue(5 > 3);

assertFalse(5 < 3);

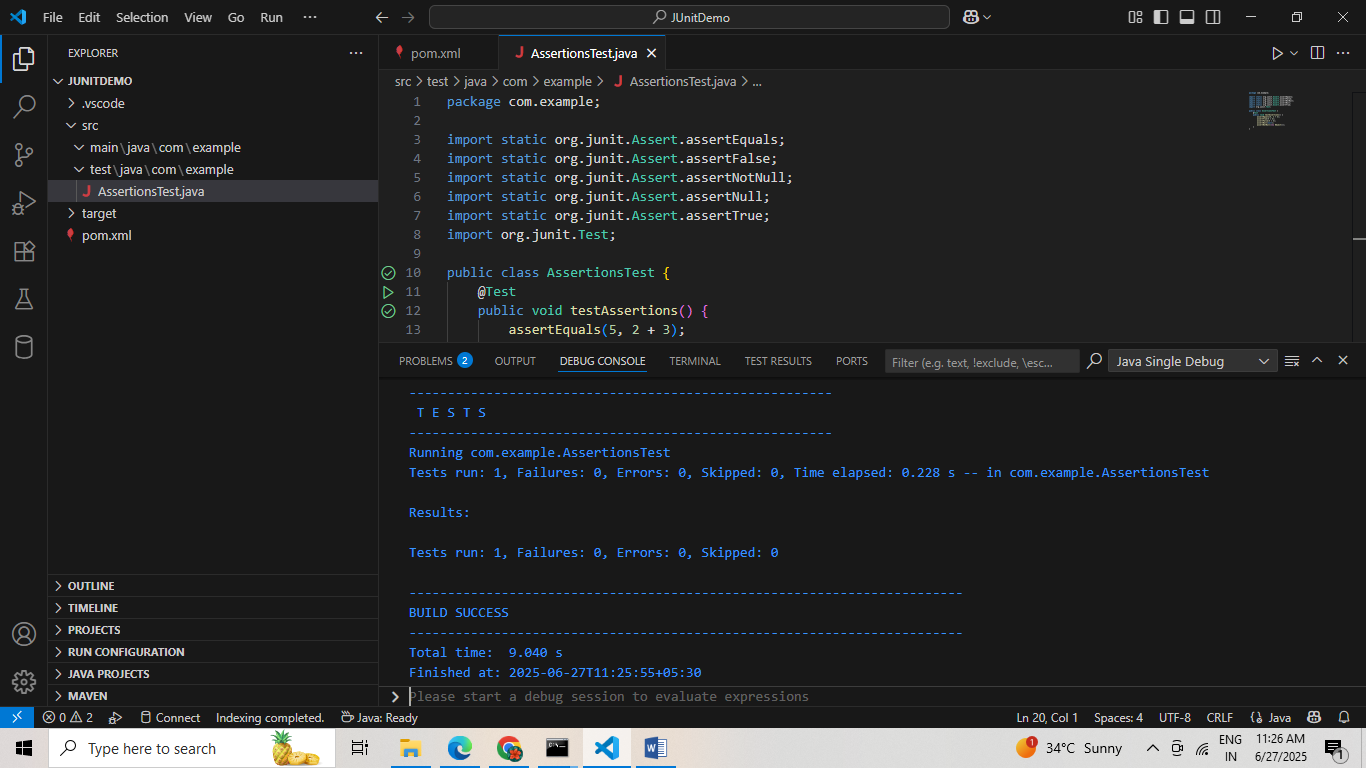
assertNull(null);

assertNotNull(new Object());

}

}

**OUTPUT:**

****

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

**MY CODE:**

**Calculator.java**

package com.example;

public class Calculator {

public int add(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;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calc;

@Before

public void setUp() {

calc = new Calculator();

System.out.println("Before - Calculator created");

}

@After

public void tearDown() {

System.out.println("After - Test completed");

}

@Test

public void testAddition()

int result = calc.add(4, 3);

assertEquals(7, result);

}

@Test

public void testDivision() {

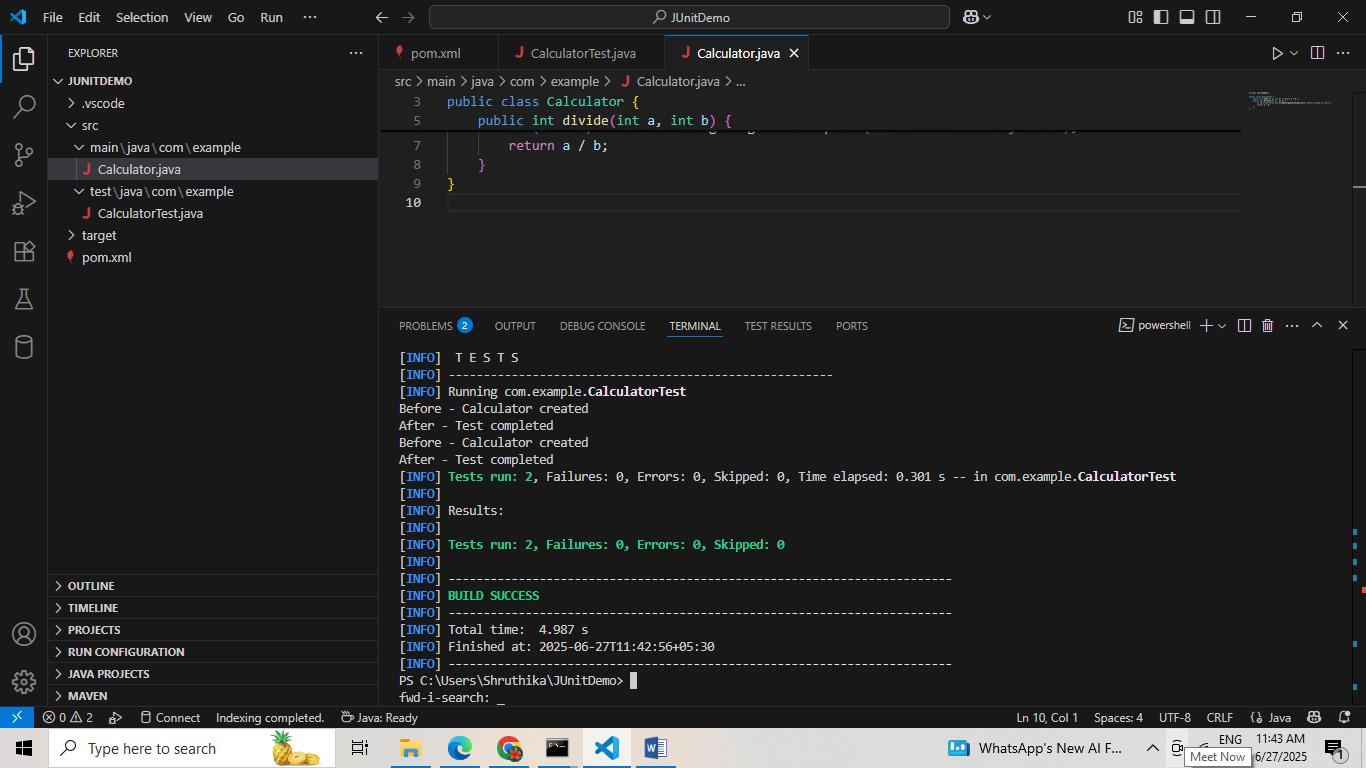
int result = calc.divide(10, 2);

assertEquals(5, result);

}

}

**OUTPUT:**

****

**MOCKITO**

**Exercise 1: Mocking and Stubbing**

**MY CODE:**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

}

### MyService.java

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

### MyServiceTest.java

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

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);

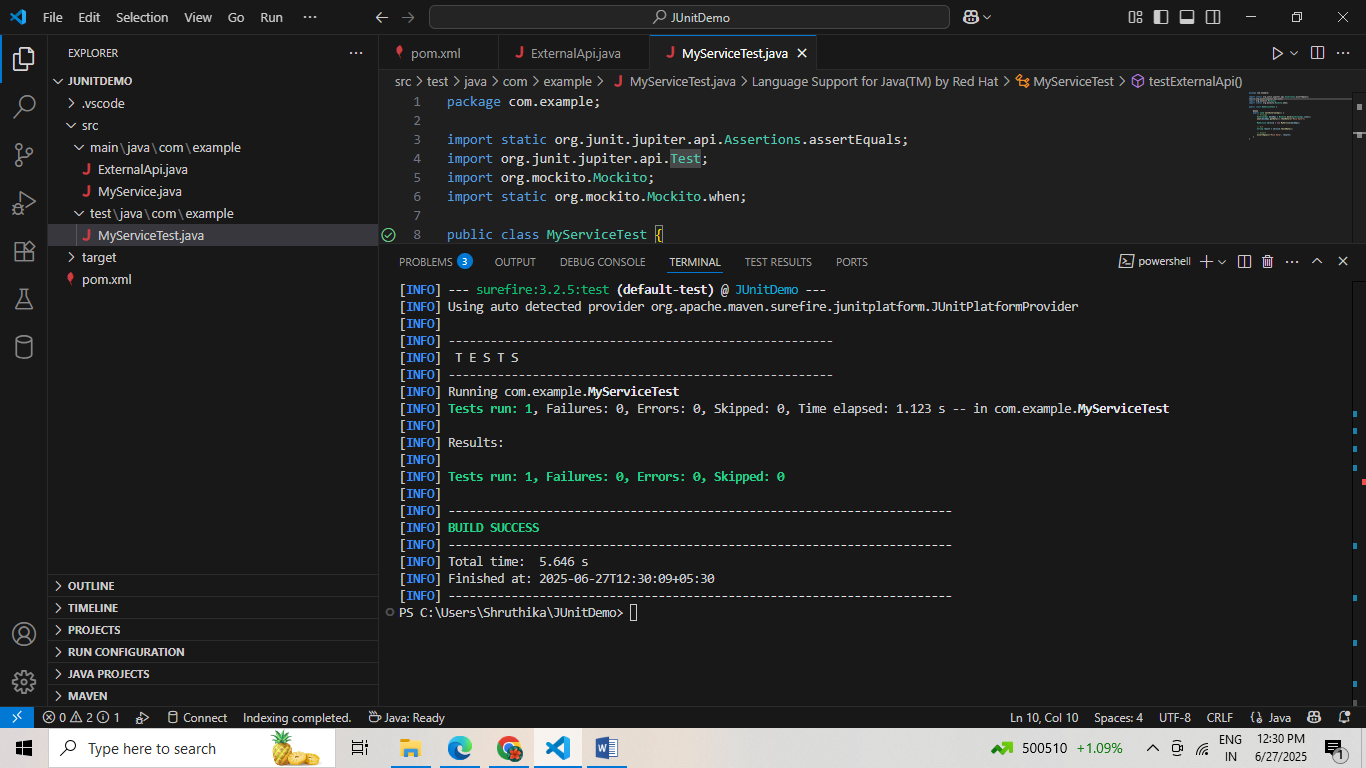
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**OUTPUT:**



**Exercise 2: Verifying Interactions**

**MY CODE:**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData(String id);

}

### MyService.java

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchById(String id) {

return api.getData(id);

}

}

### MyServiceTest.java

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

public class MyServiceTest {

@Test

public void testVerifyWithArgument() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

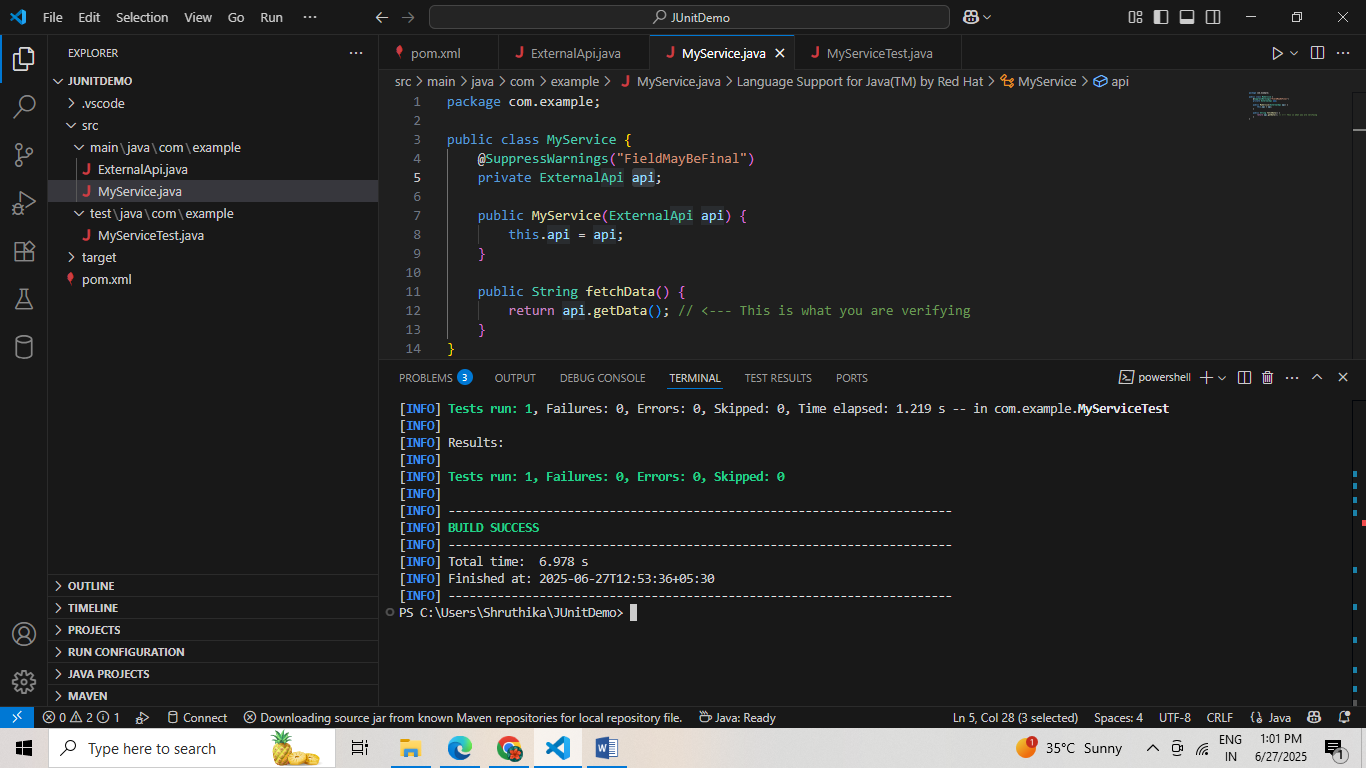
service.fetchById("abc"); // Act

verify(mockApi).getData("abc");

}

}

**OUTPUT:**

****

**LOGGING USING SLF4J**

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

**MY CODE:**

**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");

}}

**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/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>JUnitDemo</artifactId>

<packaging>jar</packaging>

<version>1.0-SNAPSHOT</version>

<name>JUnitDemo</name>

<dependencies>

<!-- JUnit 5 -->

<dependency>

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

<artifactId>junit-jupiter-api</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>4.11.0</version>

<scope>test</scope>

</dependency>

<!-- SLF4J + Logback -->

<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>

</dependencies>

<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>

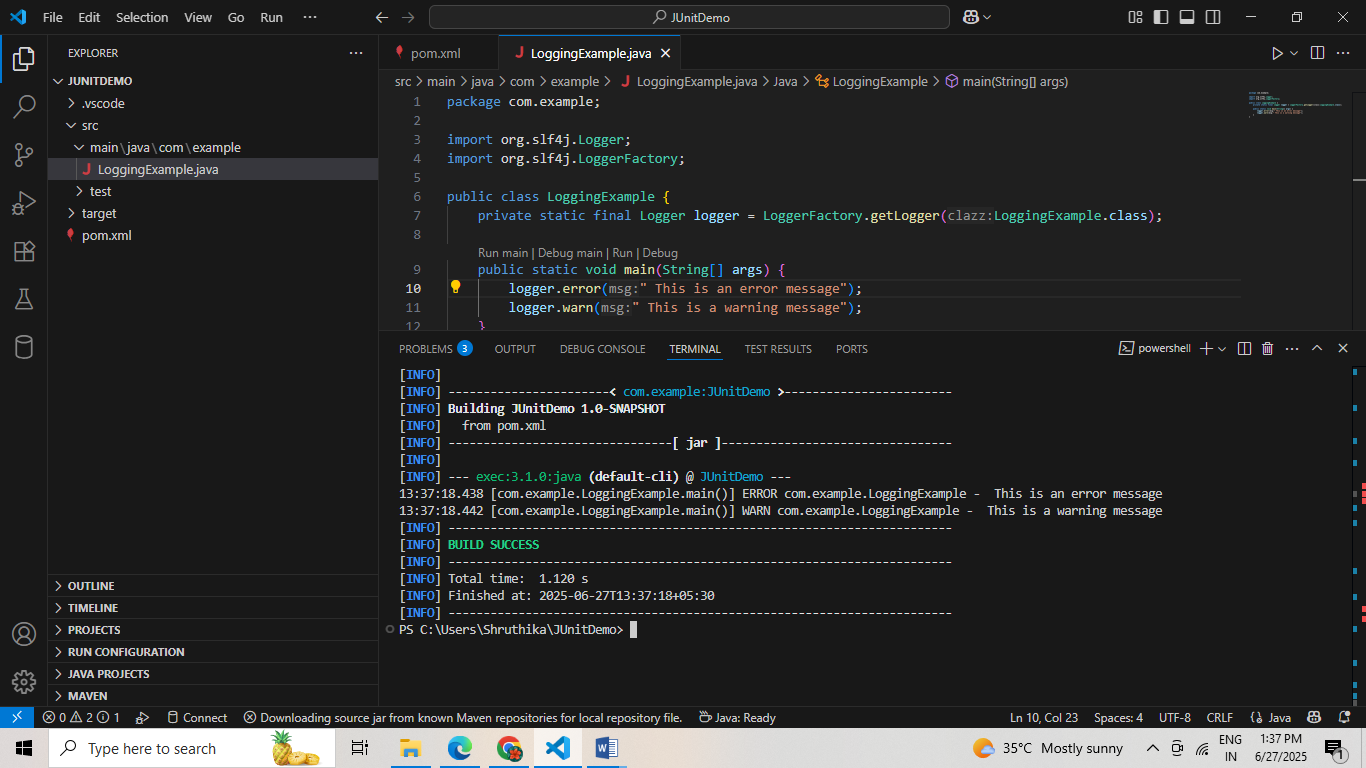
</plugin>

</plugins>

</build>

</project>

**OUTPUT:**

****