**PL/SQL Programming**

**Exercise 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.

**Code Implementation (Oracle):**

SET SERVEROUTPUT ON;

BEGIN

FOR cust IN (SELECT CustomerID, Age FROM Customers) LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Scenario 1: Updated Loan Interest Rates for Customers above 60:');

FOR loan IN (SELECT LoanID, CustomerID, InterestRate FROM Loans) LOOP

DBMS\_OUTPUT.PUT\_LINE('LoanID: ' || loan.LoanID ||

', CustomerID: ' || loan.CustomerID ||

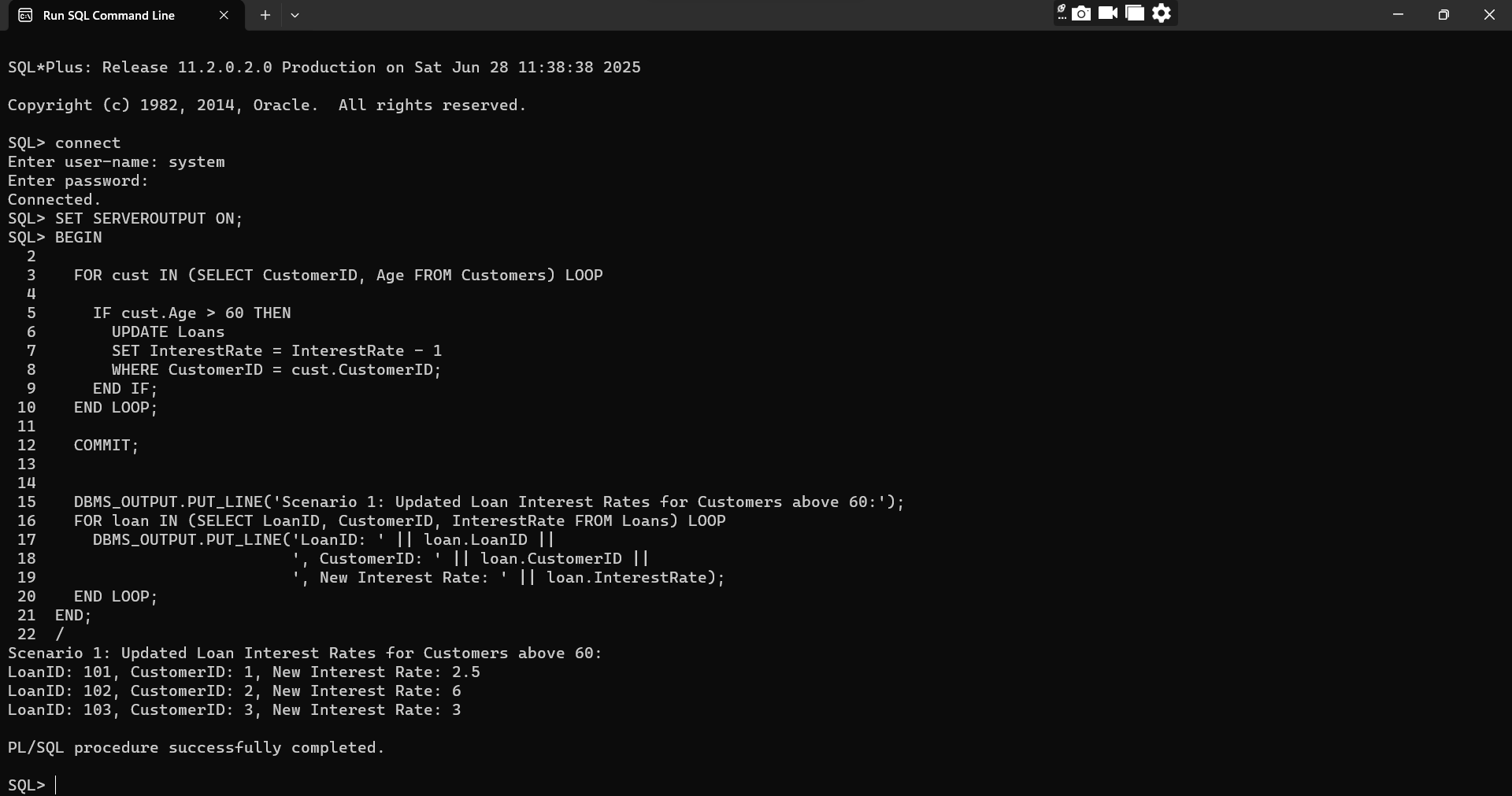
', New Interest Rate: ' || loan.InterestRate);

END LOOP;

END;

/

**Output:**



**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.

**Code Implementation:**

BEGIN

FOR cust IN (SELECT CustomerID, Balance FROM Customers) LOOP

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'Scenario 2: VIP Customers:');

FOR vip IN (SELECT CustomerID, Name, Balance FROM Customers WHERE IsVIP = 'TRUE') LOOP

DBMS\_OUTPUT.PUT\_LINE('CustomerID: ' || vip.CustomerID ||

', Name: ' || vip.Name ||

', Balance: ' || vip.Balance ||

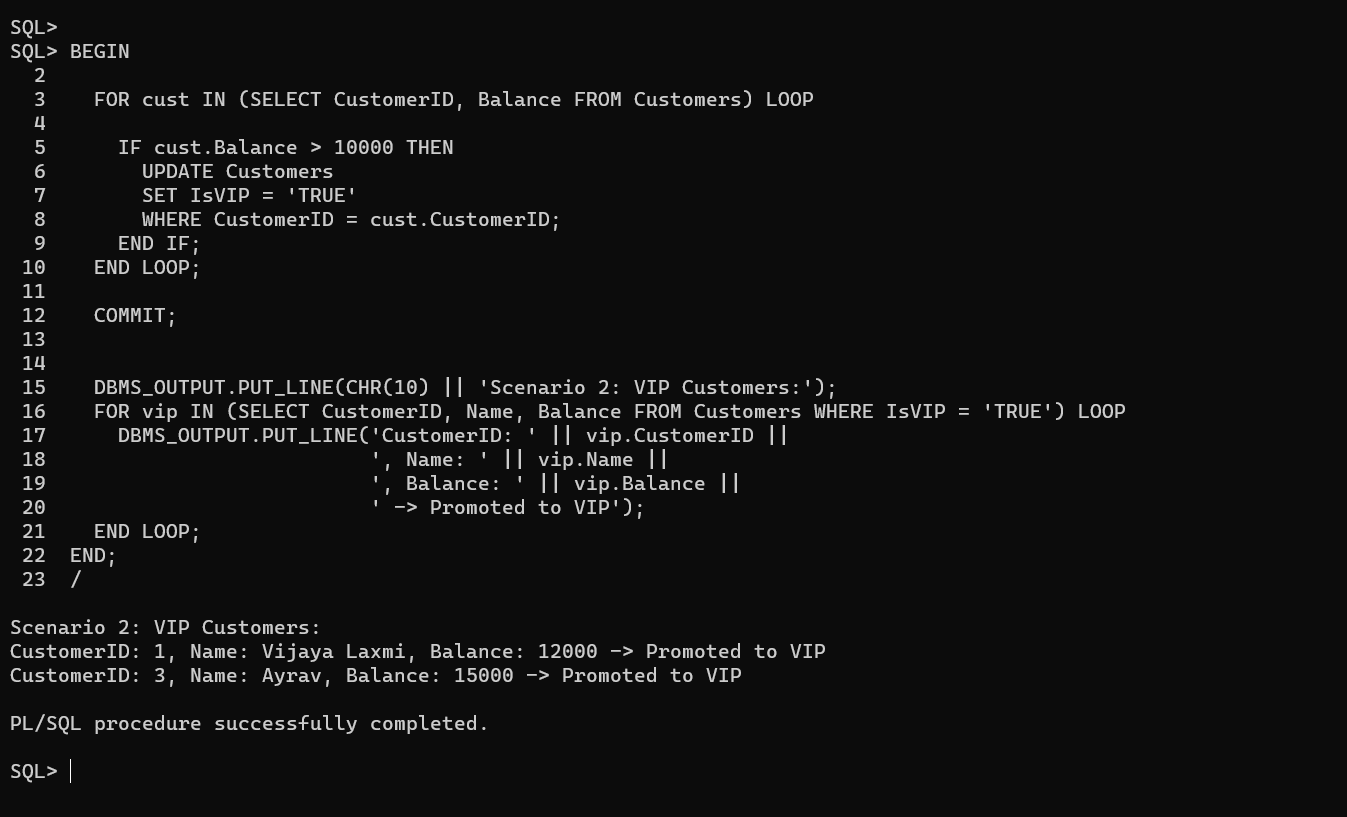
' -> Promoted to VIP');

END LOOP;

END;

/

**OUTPUT:**

****

**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.**

**Code Implementation:**

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'Scenario 3: Loan Due Reminders (within 30 days):');

FOR loan IN (

SELECT LoanID, CustomerID, TO\_CHAR(DueDate, 'DD-MON-YYYY') AS DueDate

FROM Loans

WHERE DueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan.LoanID ||

' for Customer ID ' || loan.CustomerID ||

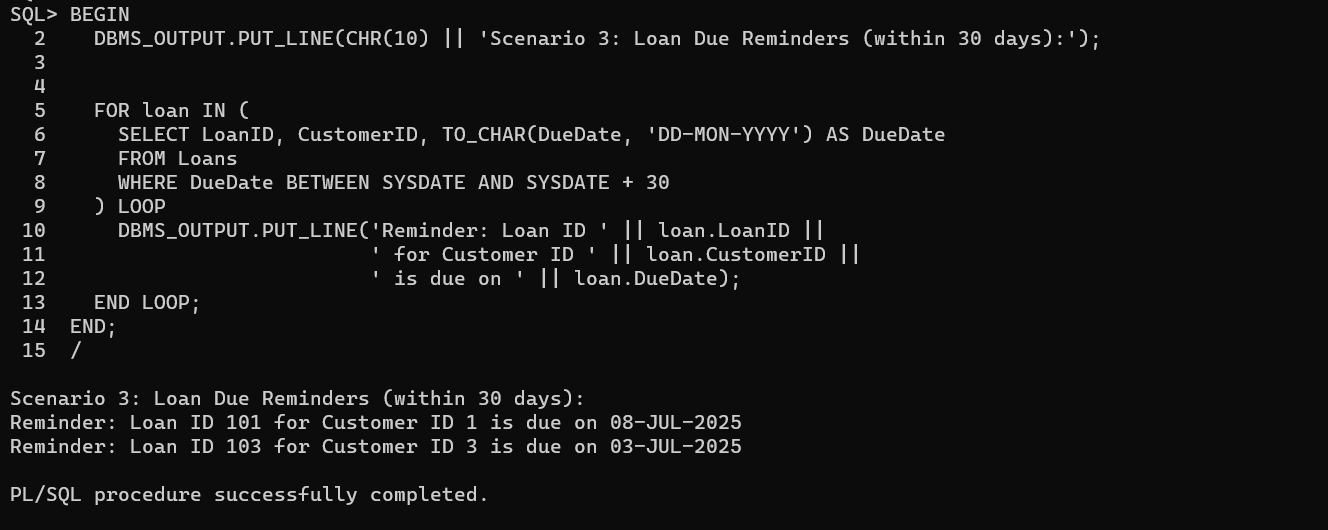
' is due on ' || loan.DueDate);

END LOOP;

END;

/

**OUTPUT:**

****

**Exercise 3: 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.

**Scenario 1 Code Implementation:**

**creating** the Accounts table:

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER(10, 2),

AccountType VARCHAR2(20)

);

**Insert some sample savings accounts:**

INSERT INTO Accounts VALUES (1001, 1, 5000.00, 'Savings');

INSERT INTO Accounts VALUES (1002, 2, 10000.00, 'Savings');

INSERT INTO Accounts VALUES (1003, 3, 7500.00, 'Current'); -- Won’t be updated

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

) LOOP

UPDATE Accounts

SET Balance = acc.Balance + (acc.Balance \* 0.01)

WHERE AccountID = acc.AccountID;

END LOOP;

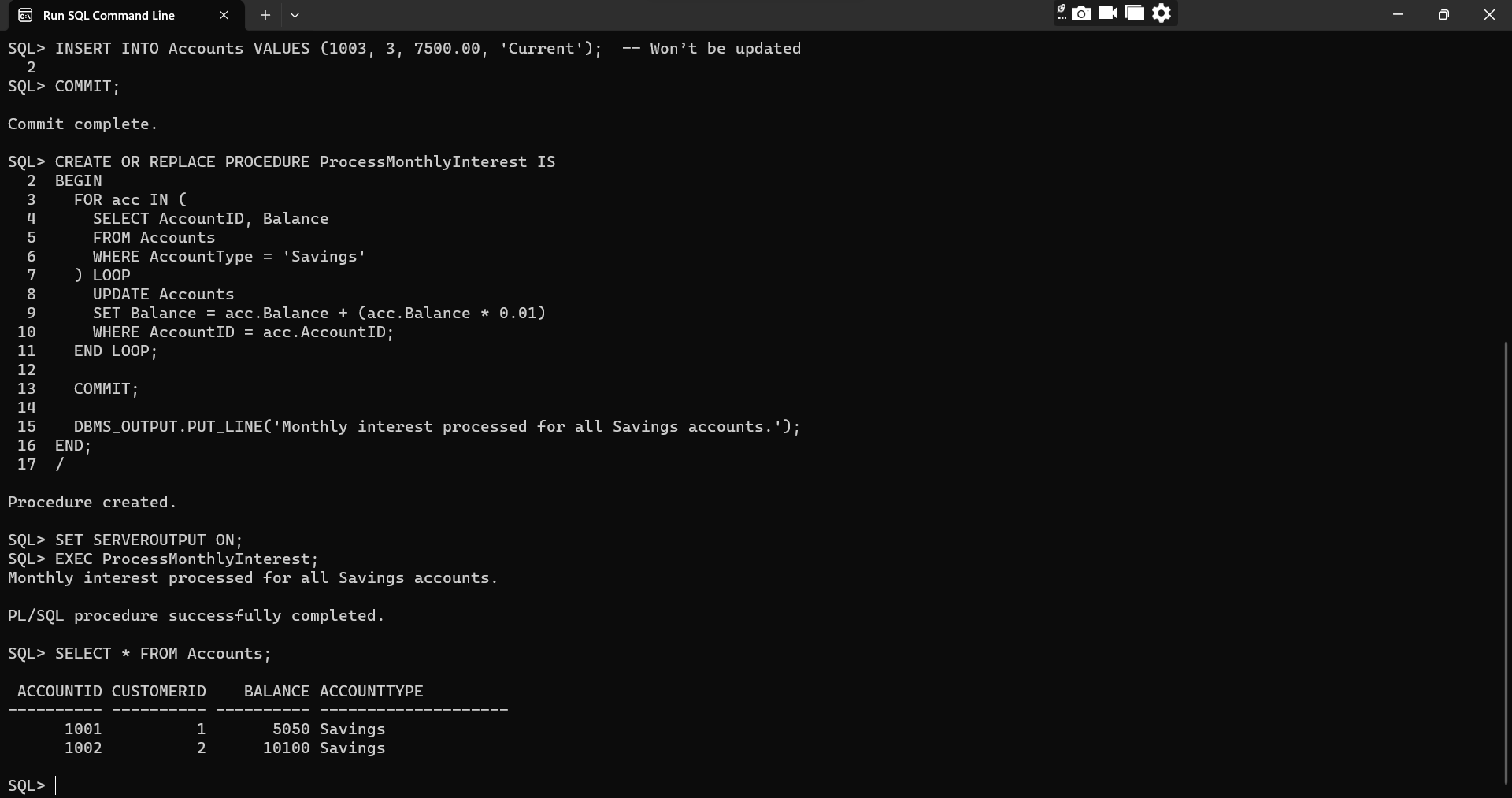
COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest processed for all Savings accounts.');

END;

/

**OUTPUT:**



**Scenario 2:**

**Code Implementation:**

**Create the Employees table**

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DepartmentID NUMBER,

Salary NUMBER(10, 2)

);

**Insert some sample data**

INSERT INTO Employees VALUES (1, 'Arjun', 101, 50000);

INSERT INTO Employees VALUES (2, 'Vanya', 101, 60000);

INSERT INTO Employees VALUES (3, 'Ayrav', 102, 55000);

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_dept\_id IN NUMBER,

p\_bonus\_pct IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE DepartmentID = p\_dept\_id;

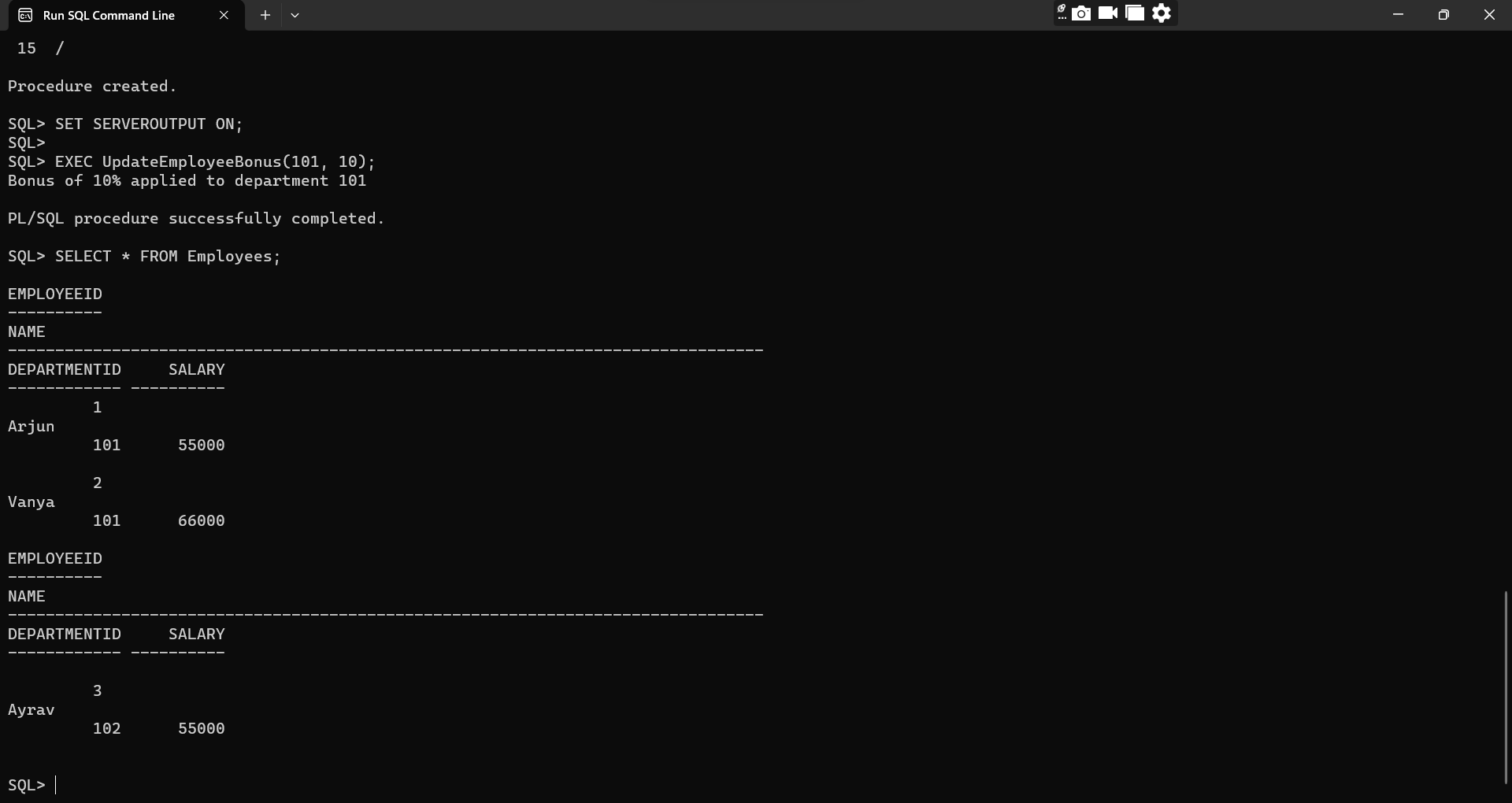
COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_bonus\_pct || '% applied to department ' || p\_dept\_id);

END;

/

**OUTPUT:**

****

**Scenario 3:**

**Code Implementation:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_acc IN NUMBER,

p\_to\_acc IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_acc

FOR UPDATE;

IF v\_balance < p\_amount THEN

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

ELSE

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_acc;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_acc;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Rs' || p\_amount || ' transferred from Account ' || p\_from\_acc || ' to Account ' || p\_to\_acc);

END IF;

EXCEPTION

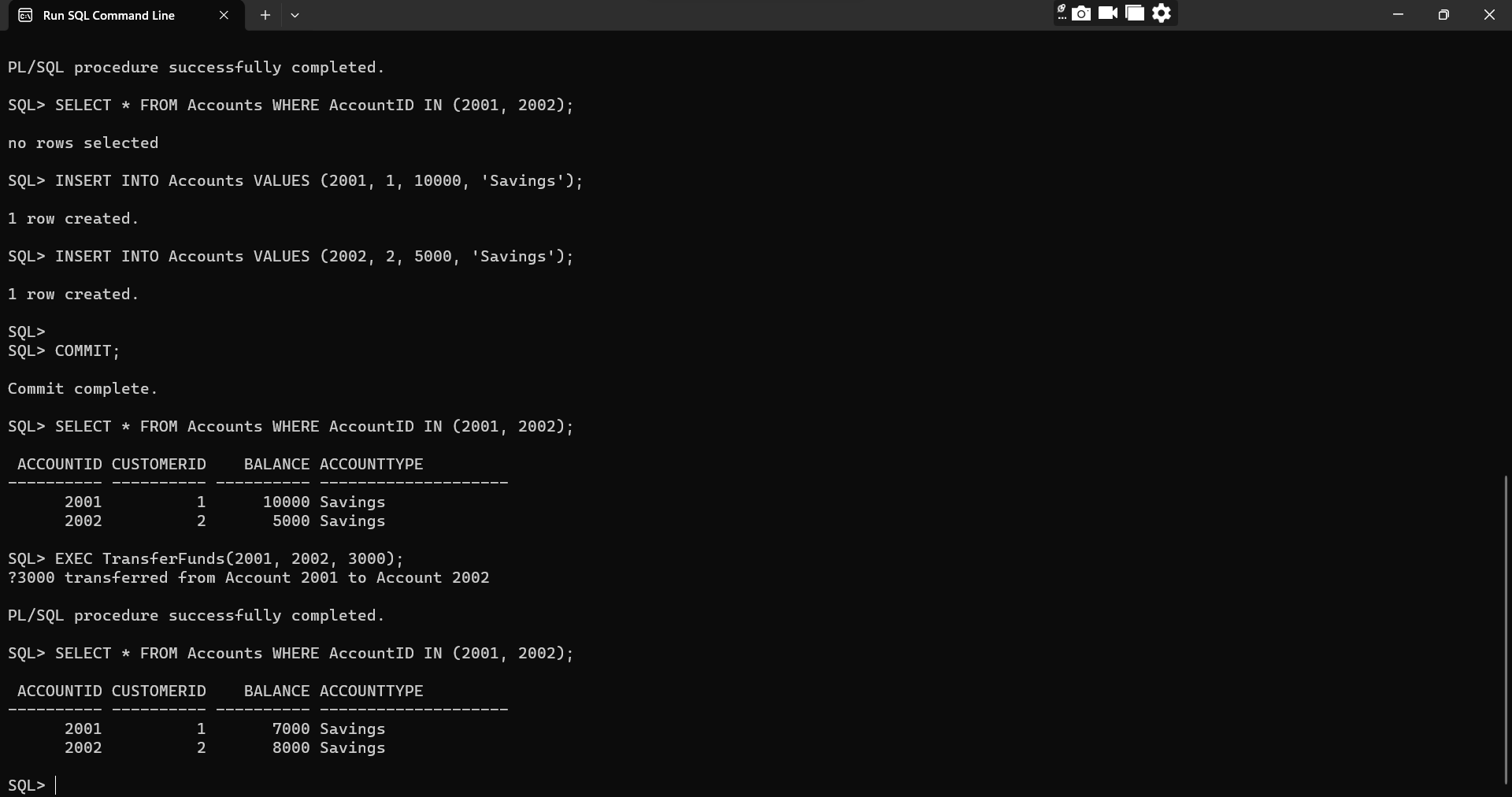
WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('ransfer failed: One or both accounts not found.');

END;

/

**Output:**

****

**TDD Using Junit5 and Mockito**

**Junit Basic Testing Exercises:**

**Exercise 1:** Setting Up JUnit

**Scenario:**

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

Steps:

1. Create a new Java project in your IDE.

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

pom.xml:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

3). Create a new test class in your project.

**Code Implementation: (Without using Maven)**

***Calculator.java***

public class Calculator {

    public int add(int a, int b) {

        return a + b;

    }

}

***CalculatorTest.java***

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class CalculatorTest {

    @Test

    public void testAddition() {

        Calculator calc = new Calculator();

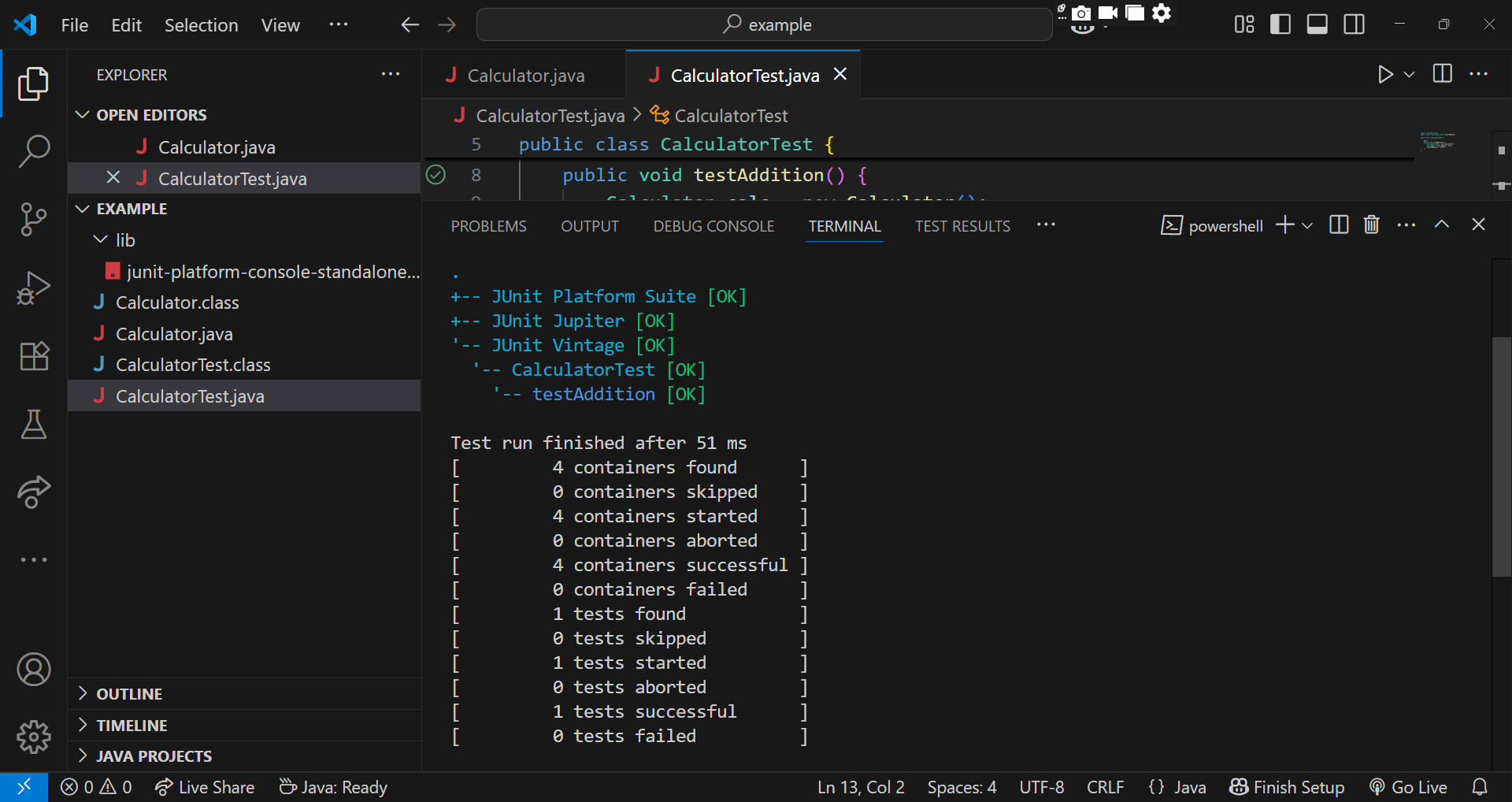
        int result = calc.add(3, 4);

        assertEquals(7, result);

    }

}

**OUTPUT:**



**Exercise 3:** Assertions in JUnit

**Scenario:**

You need to use different assertions in JUnit to validate your test results.

**Code Implementation:**

***AssertionsTest.java***

import org.junit.Test;

import static org.junit.Assert.\*;

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);

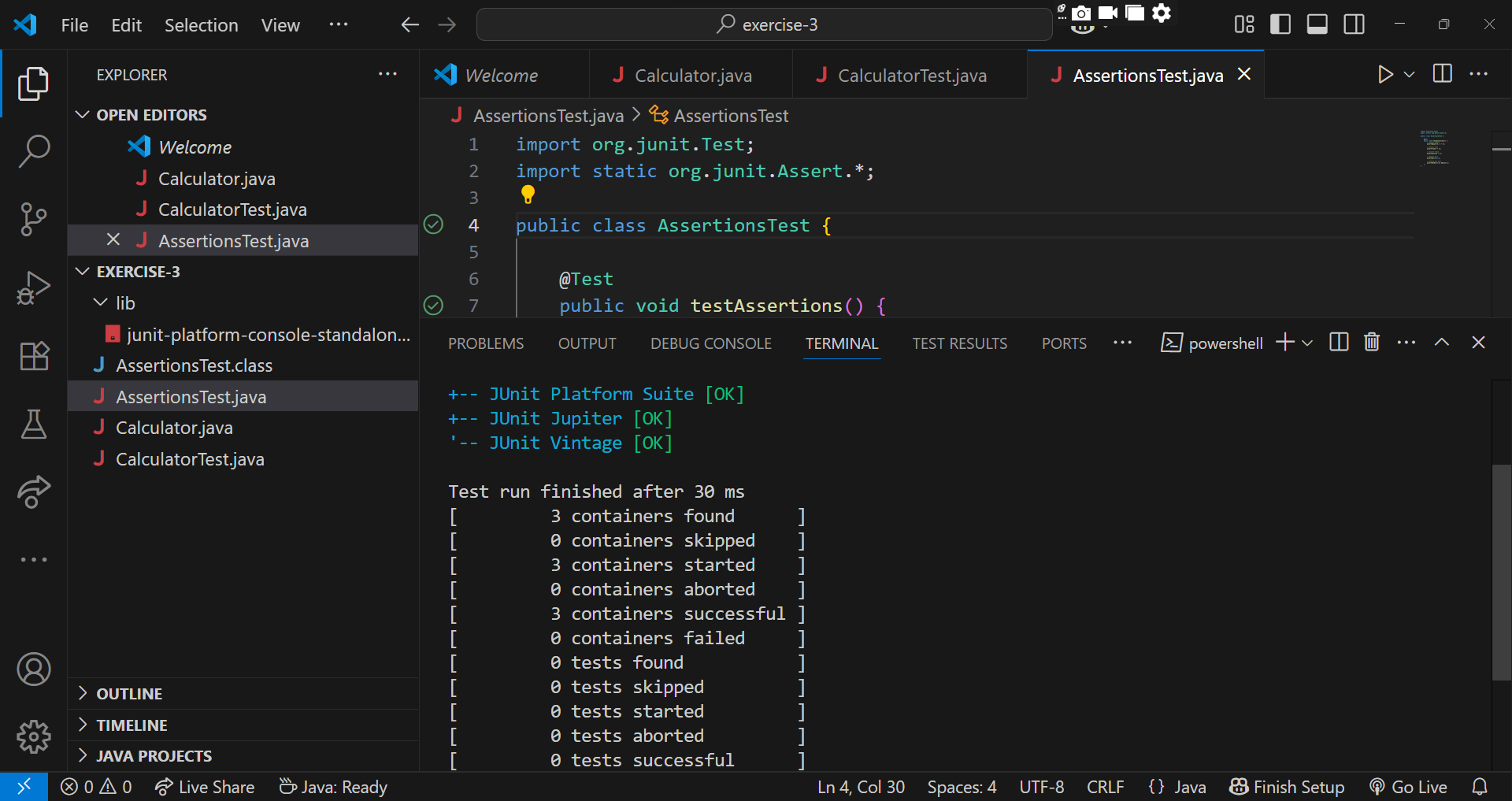
        // Assert not null

        assertNotNull(new Object());

    }

}

**OUTPUT:**



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

**Teardown Methods in JUnit**

Code Implementation:

*CalculatorAaaTest.java*

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorAaaTest {

    private Calculator calc;

    @Before

    public void setUp() {

        calc = new Calculator();

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

    }

    @After

    public void tearDown() {

        System.out.println("Teardown: Test completed\n");

    }

    @Test

    public void testAddition() {

        int result = calc.add(3, 4);

        assertEquals(7, result);

    }

    @Test

    public void testAdditionWithNegativeNumbers() {

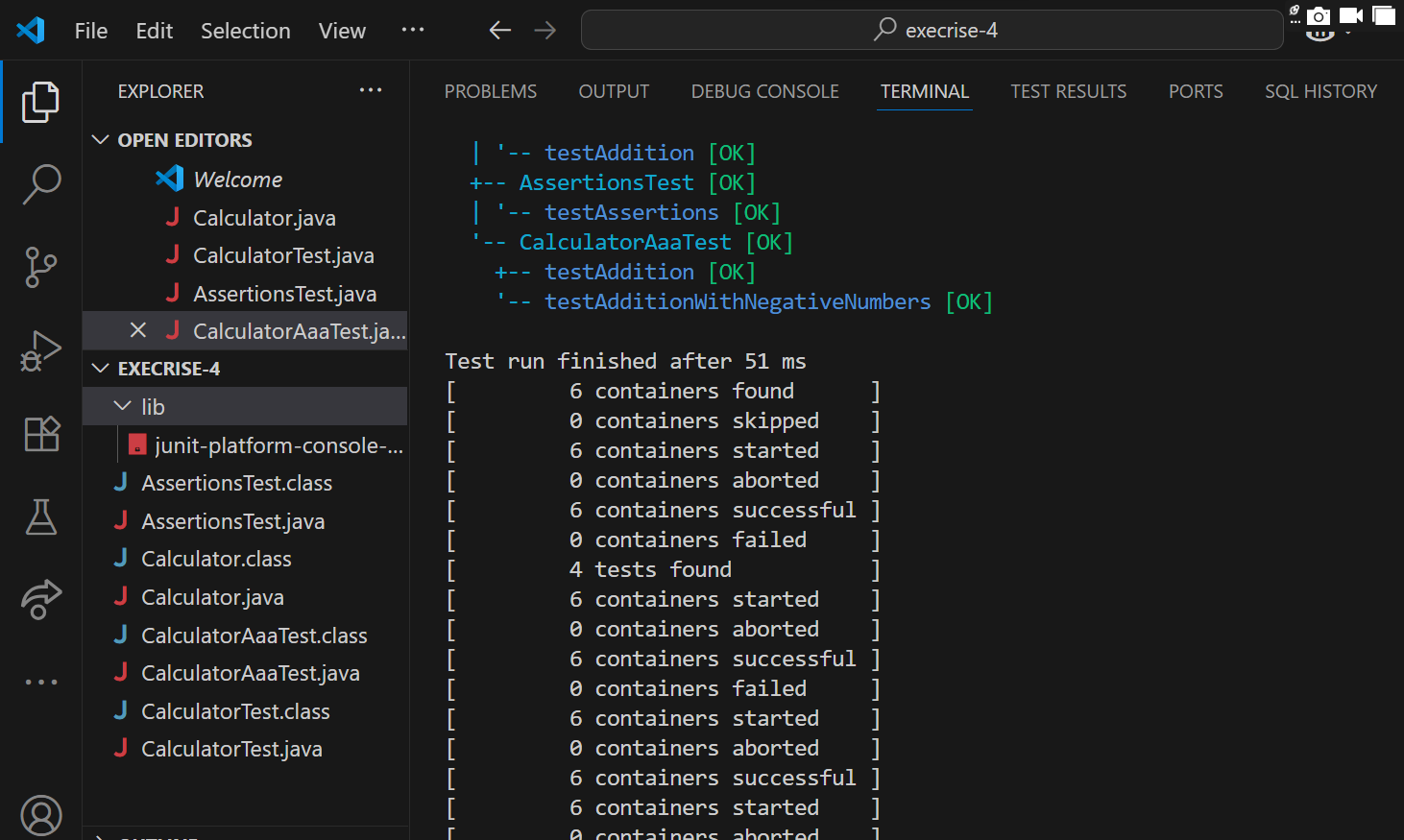
        int result = calc.add(-2, -3);

        assertEquals(-5, result);

    }

}

**OUTPUT:**



**Mockito Exercises**

**Exercise 1: Mocking and Stubbing**

**Scenario:**  You need to test a service that depends on an external API. Use Mockito to mock the

external API and stub its methods.

**Code Implementation:**

**ExteranalApi.java**

public interface ExternalApi {

    String getData();

}

**MyService.java**

public class MyService {

    private final ExternalApi api;

    public MyService(ExternalApi api) {

        this.api = api;

    }

    public String fetchData() {

        return api.getData();

    }

}

**MyServiceTest.java**

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

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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);

    }

}

**Pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<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.prac</groupId>

  <artifactId>demo</artifactId>

  <version>1.0-SNAPSHOT</version>

  <properties>

    <maven.compiler.source>17</maven.compiler.source>

    <maven.compiler.target>17</maven.compiler.target>

    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

  </properties>

  <dependencies>

    <dependency>

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

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

      <version>5.8.2</version>

      <scope>test</scope>

    </dependency>

    <dependency>

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

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

      <version>5.8.2</version>

      <scope>test</scope>

    </dependency>

    <dependency>

      <groupId>org.mockito</groupId>

      <artifactId>mockito-core</artifactId>

      <version>4.0.0</version>

      <scope>test</scope>

    </dependency>

  </dependencies>

  <build>

    <plugins>

      <plugin>

        <artifactId>maven-compiler-plugin</artifactId>

        <version>3.8.0</version>

        <configuration>

          <source>17</source>

          <target>17</target>

        </configuration>

      </plugin>

      <plugin>

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

        <version>2.22.2</version>

        <configuration>

          <includes>

            <include>\*\*/\*Test.java</include>

          </includes>

        </configuration>

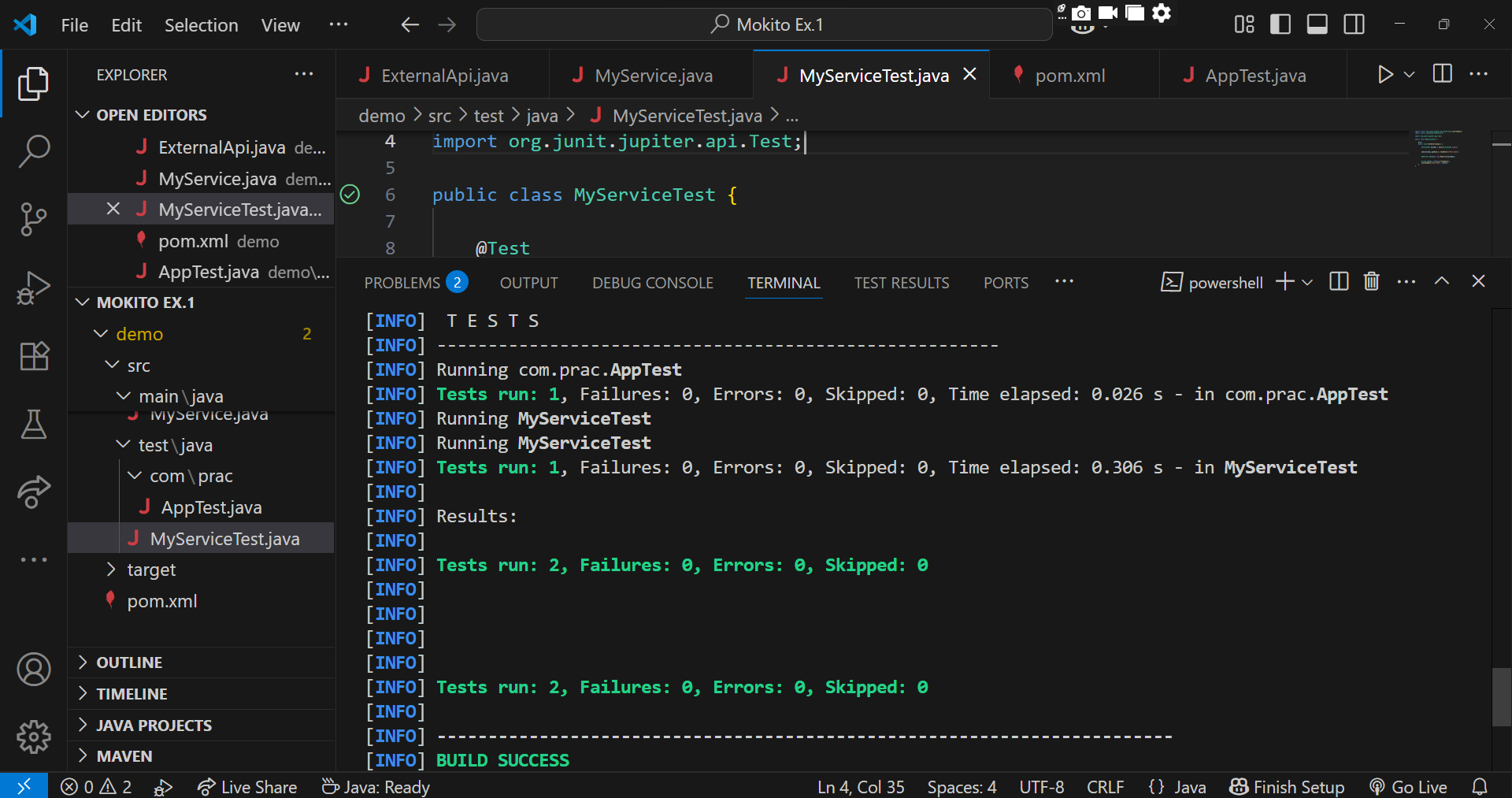
      </plugin>

    </plugins>

  </build>

</project>

**OUTPUT:**



**Exercise 2: Verifying Interactions**

**Scenario:** You need to ensure that a method is called with specific arguments.

**Steps:**

1. Create a mock object.

2. Call the method with specific arguments.

3. Verify the interaction.

***ExternalApi.java***

public interface ExternalApi {

    String getData();

}

**MyService.java**

public class MyService {

    private final 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);

        MyService service = new MyService(mockApi);

        service.fetchData();

        verify(mockApi).getData();

    }

}

**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>demo</groupId>

  <artifactId>demo</artifactId>

  <version>1.0</version>

  <properties>

    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <maven.compiler.source>17</maven.compiler.source>

    <maven.compiler.target>17</maven.compiler.target>

  </properties>

  <dependencies>

    <dependency>

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

      <artifactId>junit-jupiter</artifactId>

      <version>5.8.2</version>

      <scope>test</scope>

    </dependency>

    <dependency>

      <groupId>org.mockito</groupId>

      <artifactId>mockito-core</artifactId>

      <version>4.0.0</version>

      <scope>test</scope>

    </dependency>

  </dependencies>

  <build>

    <plugins>

      <plugin>

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

        <version>2.22.2</version>

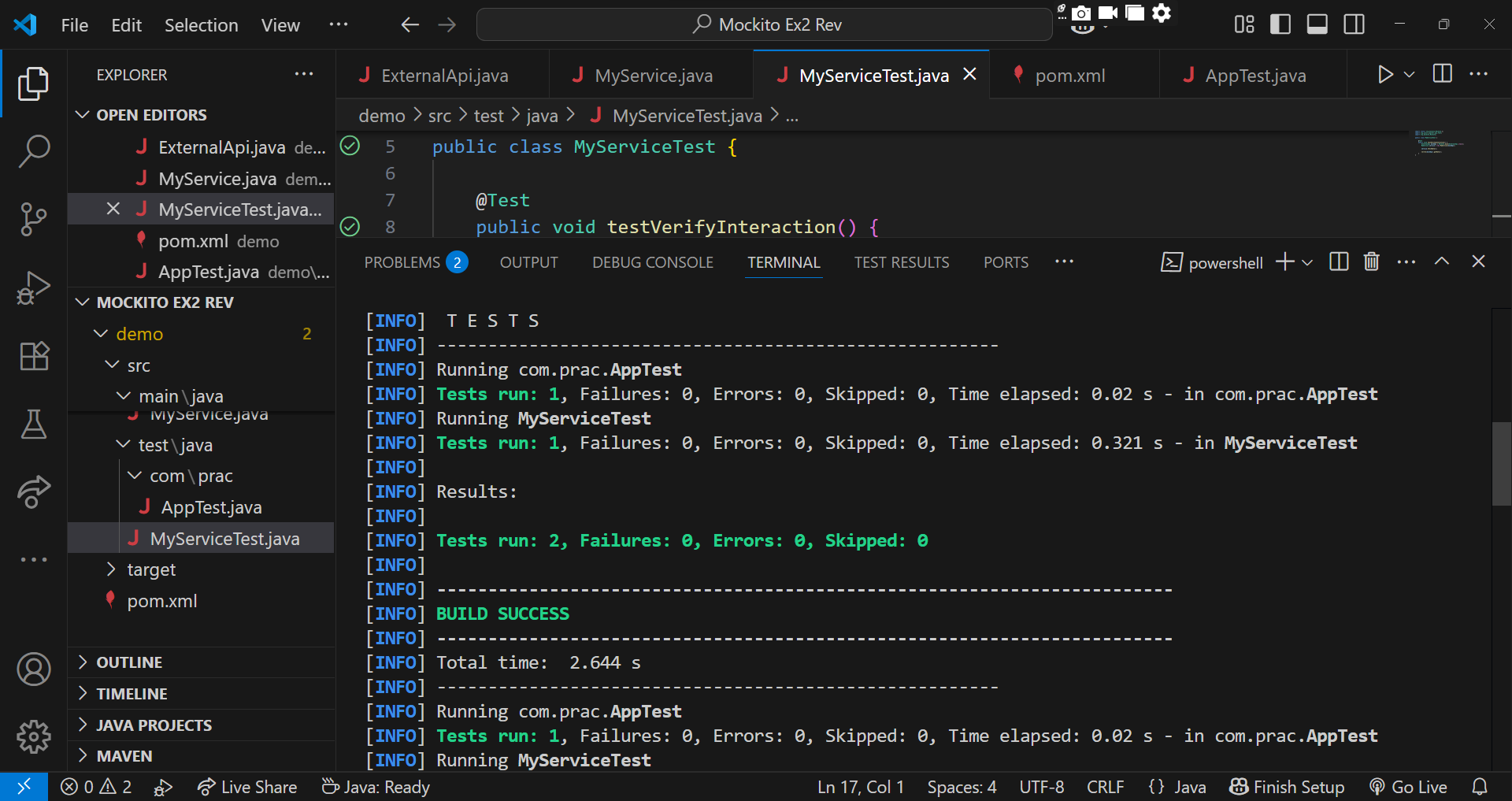
      </plugin>

    </plugins>

  </build>

</project>

**OUTPUT:**

****

**SLF4J Logging Framework**

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

**Task:** Write a Java application that demonstrates logging error messages and warning levels

using SLF4J.

**Code Implementation:**

***Pom.xml***

<?xml version="1.0" encoding="UTF-8"?>

<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>demo</artifactId>

  <version>1.0-SNAPSHOT</version>

  <properties>

    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <maven.compiler.source>17</maven.compiler.source>

    <maven.compiler.target>17</maven.compiler.target>

  </properties>

  <dependencies>

    <!-- SLF4J API -->

    <dependency>

      <groupId>org.slf4j</groupId>

      <artifactId>slf4j-api</artifactId>

      <version>1.7.30</version>

    </dependency>

    <!-- Logback implementation for SLF4J -->

    <dependency>

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

      <artifactId>logback-classic</artifactId>

      <version>1.2.3</version>

    </dependency>

    <!-- Optional: JUnit for testing -->

    <dependency>

      <groupId>junit</groupId>

      <artifactId>junit</artifactId>

      <version>4.11</version>

      <scope>test</scope>

    </dependency>

  </dependencies>

  <build>

    <plugins>

      <!-- Plugin to run the main class using mvn exec:java -->

      <plugin>

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

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

        <version>3.1.0</version>

        <configuration>

          <mainClass>LoggingExample</mainClass>

        </configuration>

      </plugin>

      <!-- Plugin to compile Java code -->

      <plugin>

        <artifactId>maven-compiler-plugin</artifactId>

        <version>3.8.0</version>

        <configuration>

          <source>17</source>

          <target>17</target>

        </configuration>

      </plugin>

    </plugins>

  </build>

</project>

***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");

        logger.info("This is an info message (optional)");

    }

}

**OUTPUT:**

