**WEEK-2**

**Exercise 1: Control Structures:**

**Customers.sql:**

Create the Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

LoanInterestRate NUMBER(5,2),

Balance NUMBER(10,2),

IsVIP VARCHAR2(5)

);

-- Insert test data

INSERT INTO Customers VALUES (1, 'Alice', 65, 8.5, 12000, 'FALSE');

INSERT INTO Customers VALUES (2, 'Bob', 45, 9.2, 9500, 'FALSE');

INSERT INTO Customers VALUES (3, 'Carol', 70, 7.8, 15000, 'FALSE');

COMMIT;

SET SERVEROUTPUT ON;

BEGIN

FOR rec IN (

SELECT CustomerID, LoanInterestRate

FROM Customers

WHERE Age > 60

)

LOOP

UPDATE Customers

SET LoanInterestRate = LoanInterestRate - 1

WHERE CustomerID = rec.CustomerID;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('1% discount applied to customers above 60.');

END;

To run:

SELECT \* FROM Customers;

**Scenario1:**

SET SERVEROUTPUT ON;

BEGIN

FOR rec IN (

SELECT CustomerID, LoanInterestRate

FROM Customers

WHERE Age > 60

)

LOOP

UPDATE Customers

SET LoanInterestRate = LoanInterestRate - 1

WHERE CustomerID = rec.CustomerID;

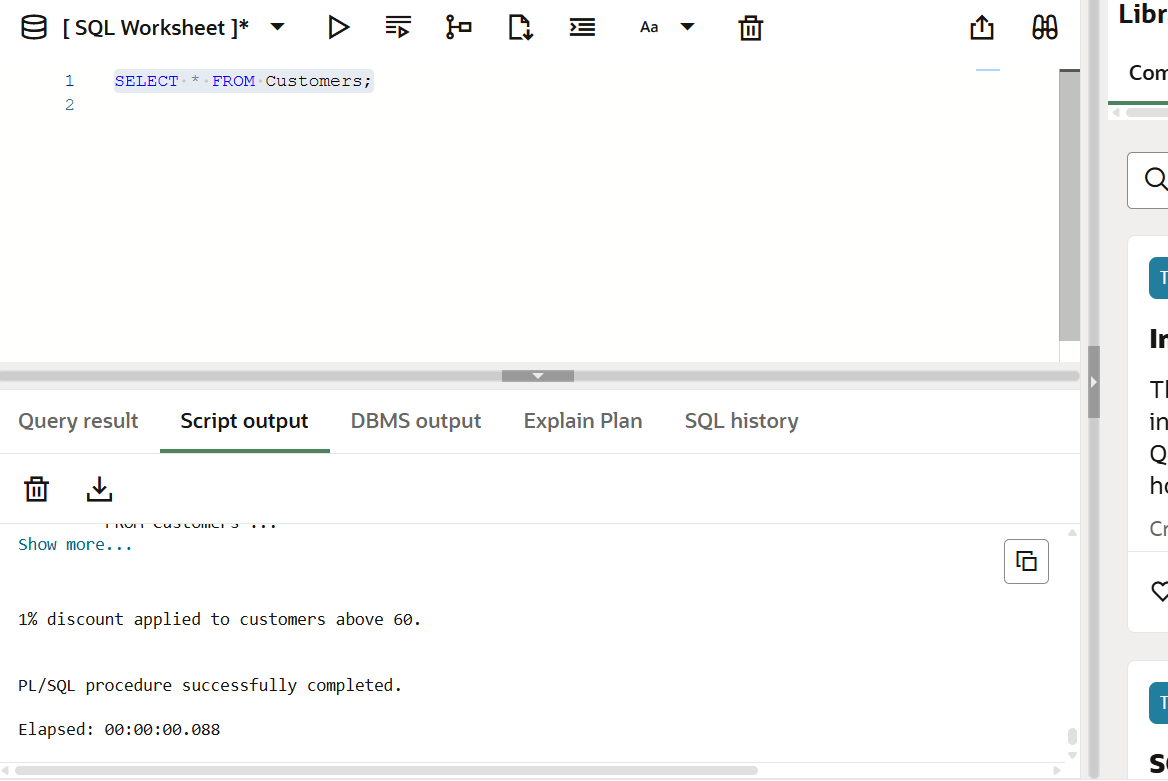
END LOOP;

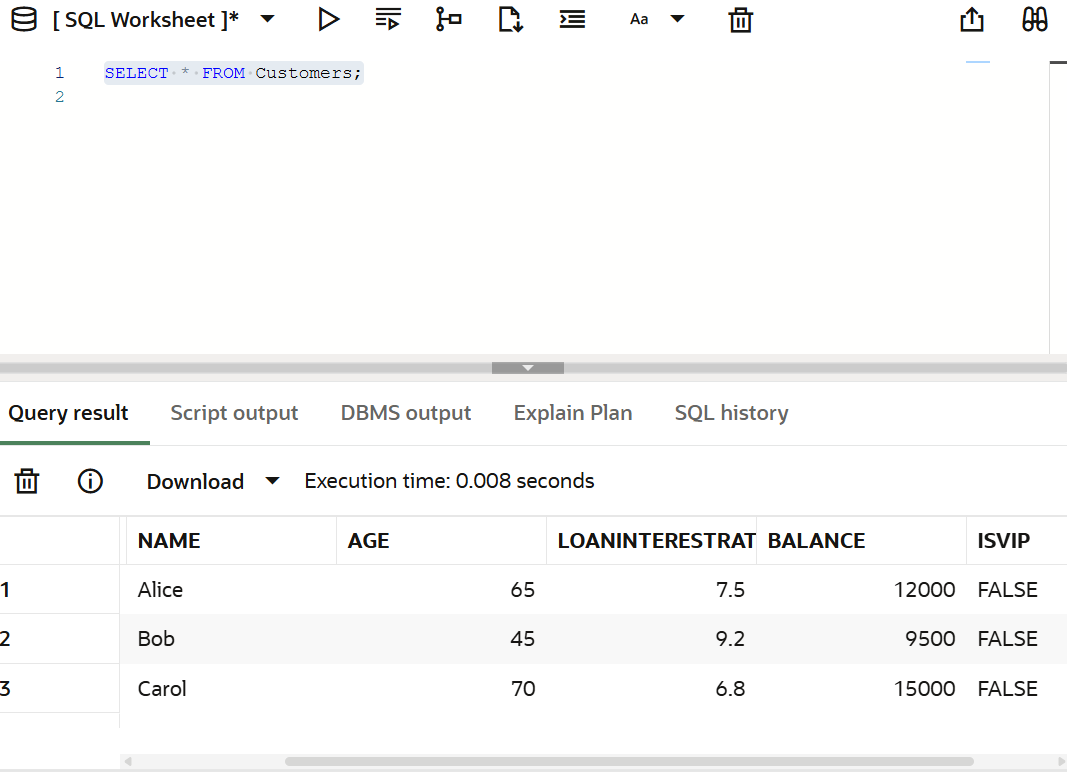
COMMIT;

DBMS\_OUTPUT.PUT\_LINE('1% discount applied to customers above 60.');

END;

**Output:**

****

****

**Scenario2:**

BEGIN

FOR rec IN (

SELECT CustomerID

FROM Customers

WHERE Balance > 10000

)

LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

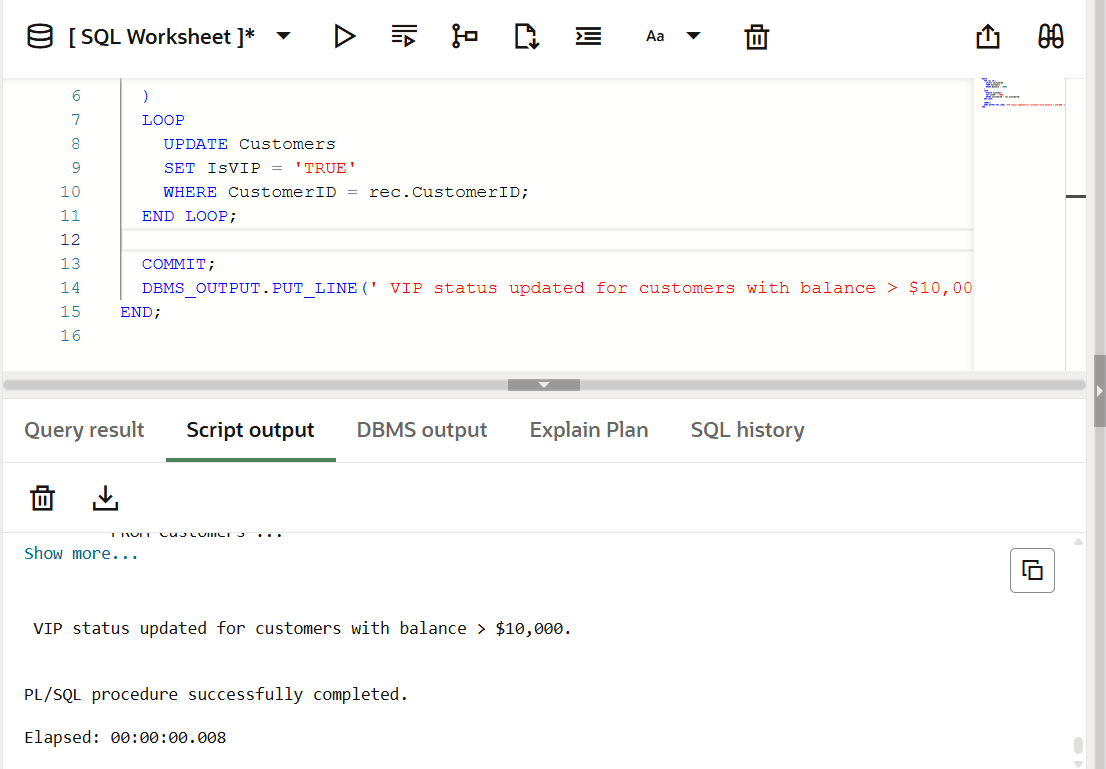
END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('✅ VIP status updated for customers with balance > $10,000.');

END;

**Output:**

****

**Scenario3:**

BEGIN

FOR rec IN (

SELECT L.CustomerID, C.Name, L.DueDate

FROM Loans L

JOIN Customers C ON L.CustomerID = C.CustomerID

WHERE L.DueDate BETWEEN SYSDATE AND SYSDATE + 30

)

LOOP

DBMS\_OUTPUT.PUT\_LINE('🔔 Reminder: Loan due on ' || TO\_CHAR(rec.DueDate, 'DD-MON-YYYY') ||

' for customer ' || rec.Name || '.');

END LOOP;

END;

**Output:**

Reminder: Loan due on 06-JUL-2025 for customer Alice.

Reminder: Loan due on 16-JUL-2025 for customer Carol.

**Exercise 3: Stored Procedures:**

### 1. SavingsAccounts Table

CREATE TABLE SavingsAccounts (

AccountID NUMBER PRIMARY KEY,

CustomerName VARCHAR2(100),

Balance NUMBER(10,2)

);

### 2. Employees Table

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Department VARCHAR2(100),

Salary NUMBER(10,2)

);

### 3. BankAccounts Table

CREATE TABLE BankAccounts (

AccountID NUMBER PRIMARY KEY,

CustomerName VARCHAR2(100),

Balance NUMBER(10,2)

);

**-**- Savings Accounts

INSERT INTO SavingsAccounts VALUES (1, 'Alice', 10000);

INSERT INTO SavingsAccounts VALUES (2, 'Bob', 15000);

-- Employees

INSERT INTO Employees VALUES (101, 'John', 'Finance', 50000);

INSERT INTO Employees VALUES (102, 'Jane', 'IT', 60000);

INSERT INTO Employees VALUES (103, 'Mike', 'Finance', 55000);

-- Bank Accounts

INSERT INTO BankAccounts VALUES (1, 'Alice', 20000);

INSERT INTO BankAccounts VALUES (2, 'Bob', 10000);

COMMIT;

**Scenario1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

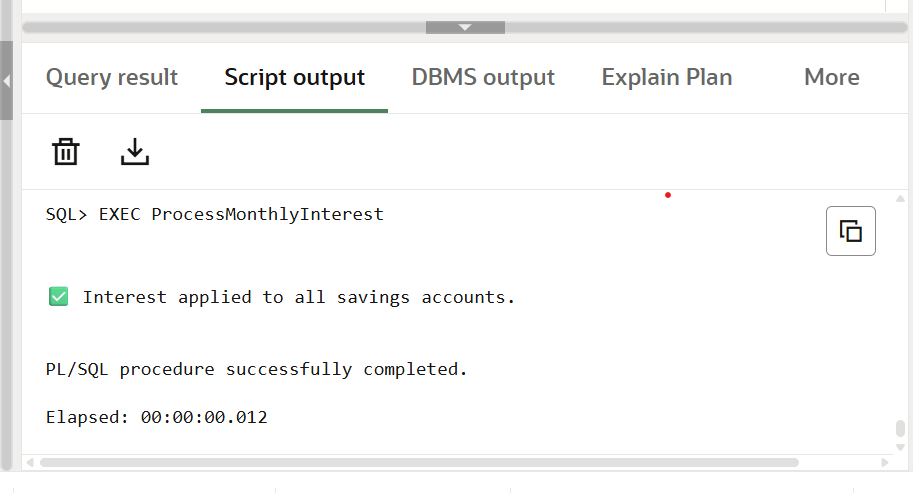
UPDATE SavingsAccounts

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

DBMS\_OUTPUT.PUT\_LINE('✅ Interest applied to all savings accounts.');

END;

**Output:**



**Scenario2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

deptName IN VARCHAR2,

bonusPercent IN NUMBER

) IS

BEGIN

UPDATE Employees

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

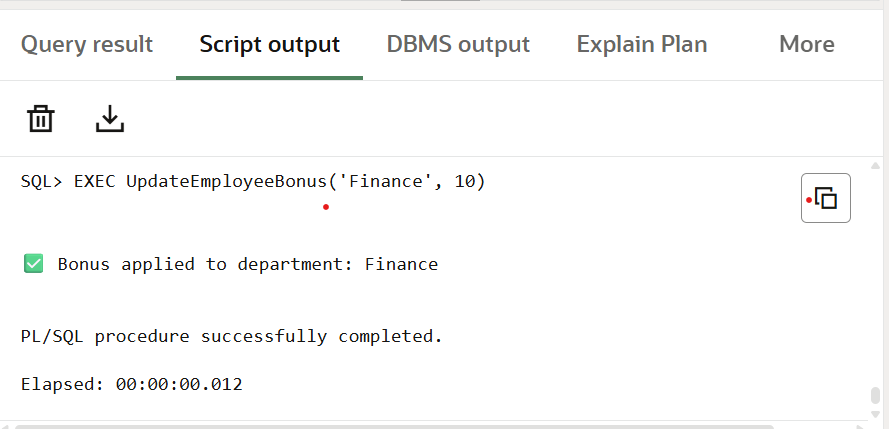
WHERE Department = deptName;

DBMS\_OUTPUT.PUT\_LINE('✅ Bonus applied to department: ' || deptName);

END;

EXEC UpdateEmployeeBonus('Finance', 10);

**Output:**



**Scenario3:**

DECLARE

v\_from NUMBER := 1;

v\_to NUMBER := 2;

v\_amount NUMBER := 5000;

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM BankAccounts

WHERE AccountID = v\_from

FOR UPDATE;

IF v\_balance < v\_amount THEN

DBMS\_OUTPUT.PUT\_LINE(' Transfer failed: Insufficient balance.');

ELSE

UPDATE BankAccounts

SET Balance = Balance - v\_amount

WHERE AccountID = v\_from;

UPDATE BankAccounts

SET Balance = Balance + v\_amount

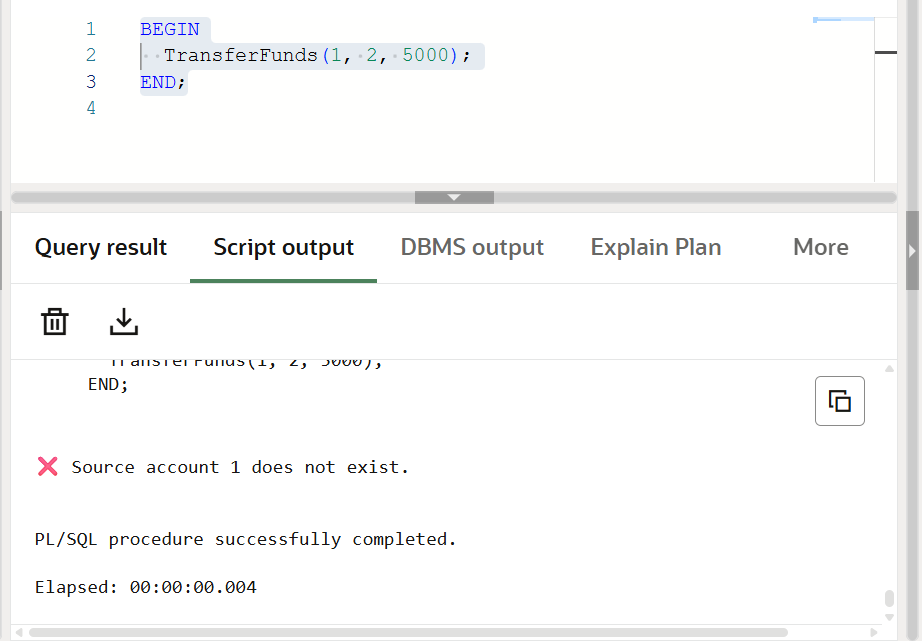
WHERE AccountID = v\_to;

DBMS\_OUTPUT.PUT\_LINE(' ₹' || v\_amount || ' transferred from Account ' || v\_from || ' to Account ' || v\_to);

END IF;

END;

**Output:**



**JUnit Testing Exercises :**

**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>org.example</groupId>

<artifactId>CognizantWeek2</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<name>CognizantWeek2</name>

<url>http://maven.apache.org</url>

<properties>

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

<maven.compiler.source>1.8</maven.compiler.source>

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

</properties>

<dependencies>

<!-- JUnit 4 for unit testing -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

<!-- Hamcrest is needed by JUnit for assertions -->

<dependency>

<groupId>org.hamcrest</groupId>

<artifactId>hamcrest-core</artifactId>

<version>1.3</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**Calculator.java:**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorTest.java:**

package com.example;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

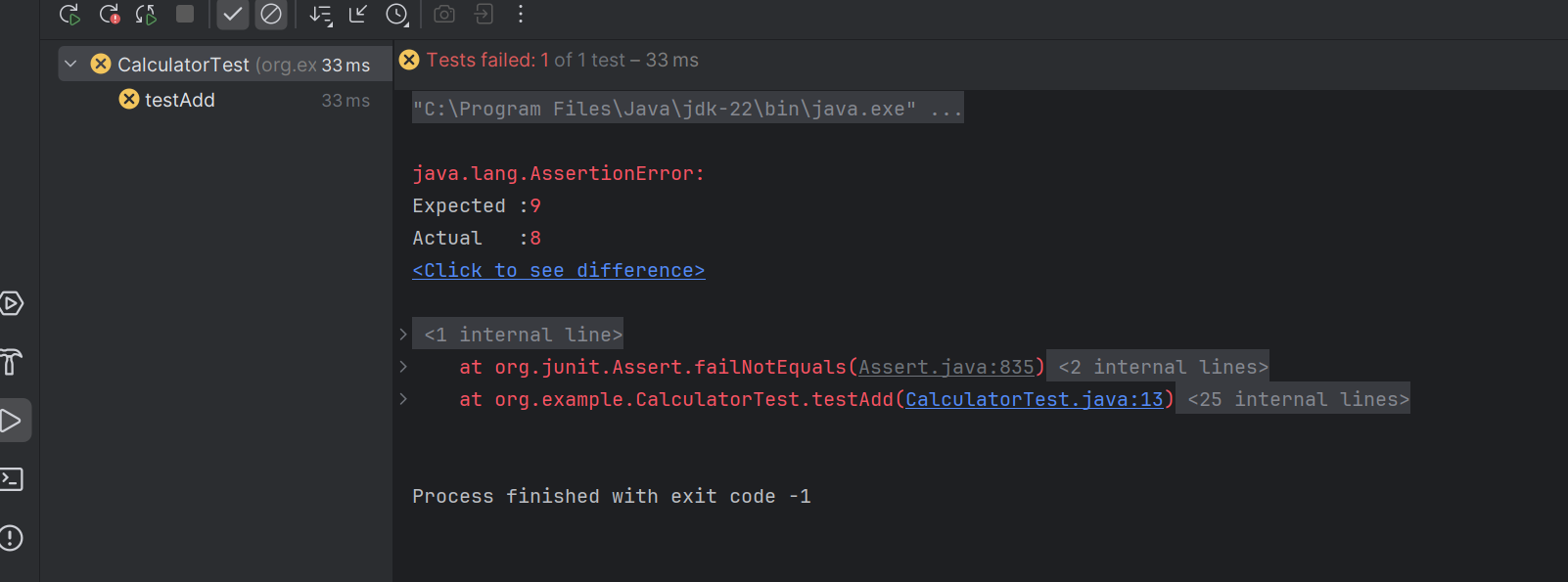
int result = calc.add(5, 3);

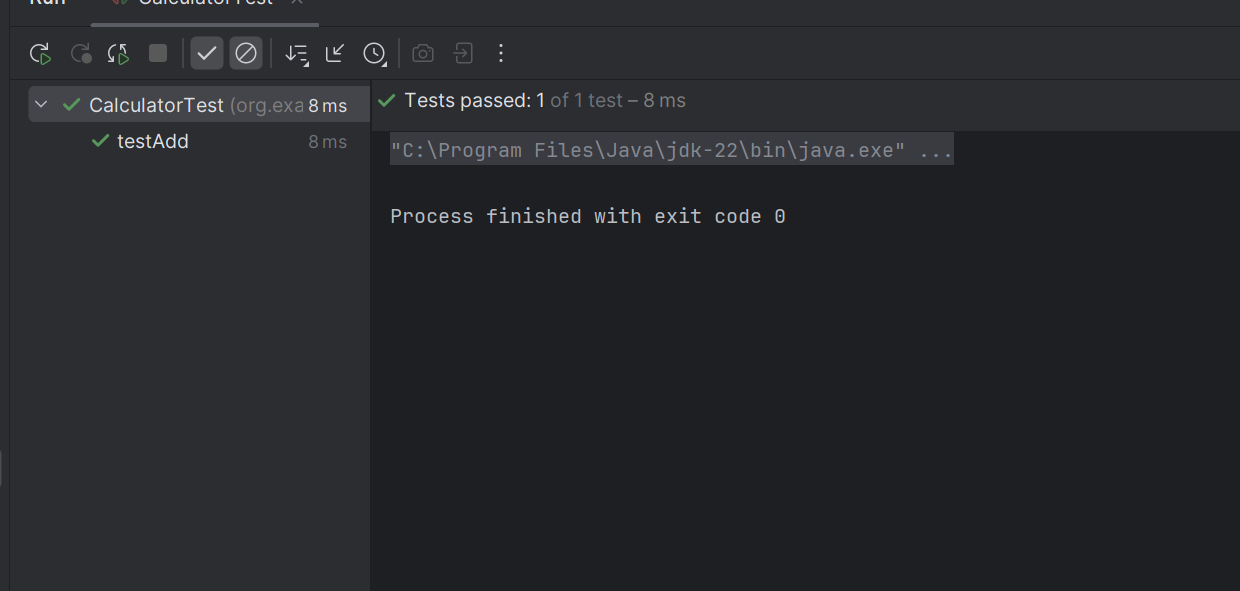
assertEquals(8, result);

}

}

**Output:**





**Exercise 3: Assertions in JUnit:**

**AssertionTest.java:**

package org.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionTest {

@Test

public void testAssertions() {

System.out.println(" Starting Assertions Test...");

// Assert equals

System.out.println(" Checking assertEquals...");

assertEquals("Sum should be 5", 5, 2 + 3);

// Assert true

System.out.println(" Checking assertTrue...");

assertTrue("Expected condition to be true", 5 > 3);

// Assert false

System.out.println(" Checking assertFalse...");

assertFalse("Expected condition to be false", 5 < 3);

// Assert null

System.out.println(" Checking assertNull...");

assertNull("Expected value to be null", null);

// Assert not null

System.out.println(" Checking assertNotNull...");

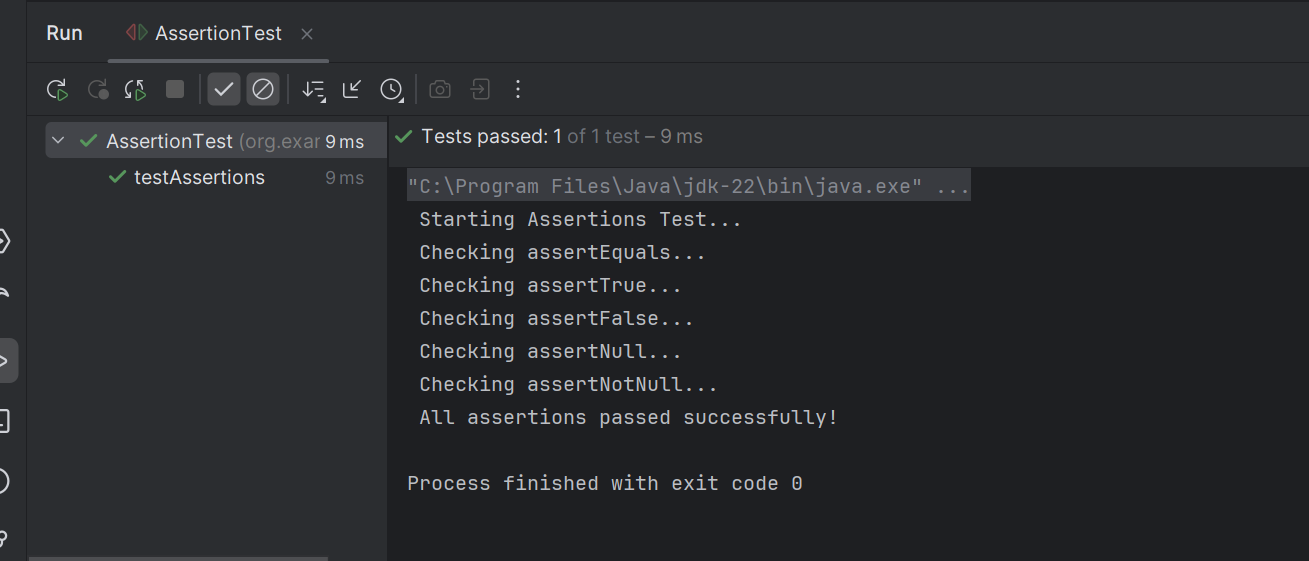
assertNotNull("Expected object to be not null", new Object());

System.out.println(" All assertions passed successfully!");

}

}

**Output:**



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

**Calculator.java:**

package org.example;

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 org.example;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

// Setup method – runs before every test

@Before

public void setUp() {

System.out.println(" Setting up Calculator instance...");

calculator = new Calculator(); // Arrange

}

// Teardown method – runs after every test

@After

public void tearDown() {

System.out.println(" Cleaning up Calculator instance...");

calculator = null;

}

// Test for addition using AAA pattern

@Test

public void testAddition() {

// Act

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

// Assert

System.out.println(" Testing addition...");

assertEquals(15, result);

}

// Test for subtraction using AAA pattern

@Test

public void testSubtraction() {

// Act

int result = calculator.subtract(10, 3);

// Assert

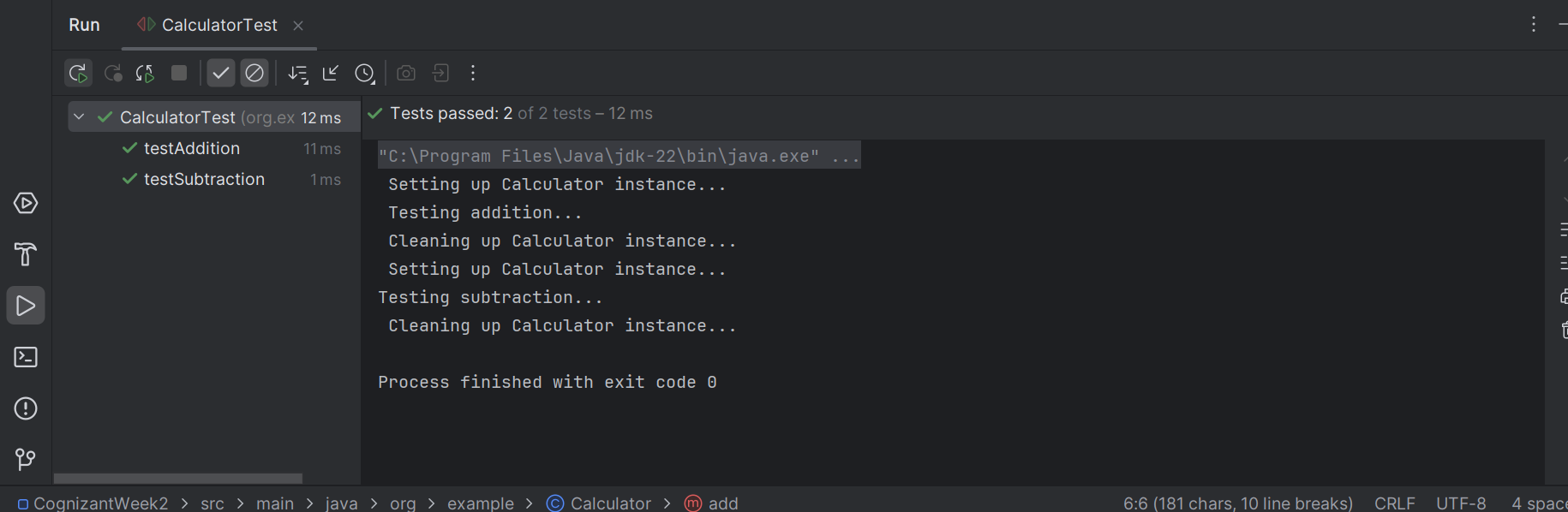
System.out.println("Testing subtraction...");

assertEquals(7, result);

}

}

**OUTPUT:**



**Mockito Hands-On Exercises:**

#### **ExternalApi.java:**

package org.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 org.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class MyServiceTest {

@Test

public void testExternalApi() {

// Step 1: Create mock

ExternalApi mockApi = mock(ExternalApi.class);

// Step 2: Stub method

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

// Step 3: Use mock in service

MyService service = new MyService(mockApi);

String result = service.fetchData();

// Step 4: Assert result

assertEquals("Mock Data", result);

// Optional: Verify interaction

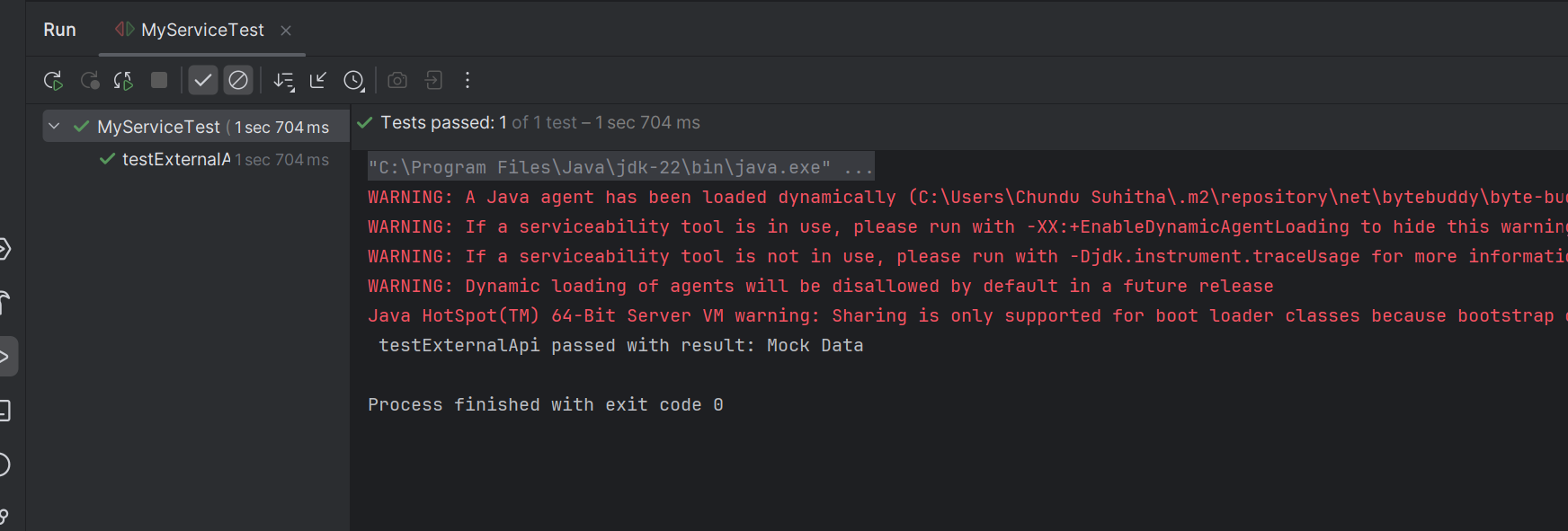
verify(mockApi).getData();

System.out.println(" testExternalApi passed with result: " + result);

}

}

**Output:**



**Exercise 2: Verifying Interactions :**

**ExternalApi.java:**

package org.example;

public interface ExternalApi {

String getData();

}

#### MyService.java

package org.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 org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// Step 1: Create a mock object

ExternalApi mockApi = mock(ExternalApi.class);

// Step 2: Create service and call the method

MyService service = new MyService(mockApi);

service.fetchData();

// Step 3: Verify that getData() was called exactly once

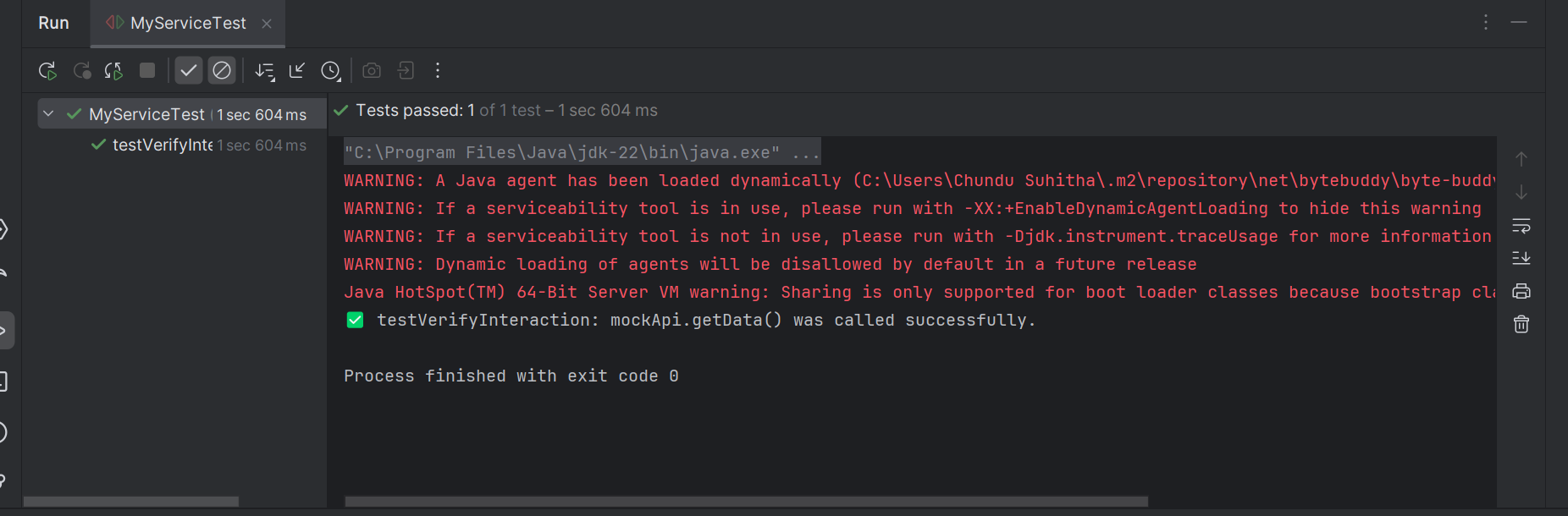
verify(mockApi).getData();

System.out.println("✅ testVerifyInteraction: mockApi.getData() was called successfully.");

}

}

**OUTPUT:**



**Logging using SLF4J :**

**LoggingExample.java:**

package org.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");

logger.info("This is an info message (won't show by default if root level is WARN)");

}

}

**OUTPUT:**

