**WEEK 2 – ASSIGNMENT –MANDATORY**

**Exercise 1: Control Structures**

**Scenario 1: Apply Discount to Loan Interest for Customers Above 60**

BEGIN

FOR cust IN (SELECT c.CustomerID, l.LoanID, l.InterestRate, c.DOB

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID) LOOP

IF MONTHS\_BETWEEN(SYSDATE, cust.DOB) / 12 > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = cust.LoanID;

END IF;

END LOOP;

COMMIT;

END;

**Scenario 2: Promote Customers to VIP Based on Balance**

ALTER TABLE Customers ADD IsVIP CHAR(1);

BEGIN

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

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

**Scenario 3: Send Reminders for Loans Due in Next 30 Days**

BEGIN

FOR loan IN (SELECT l.LoanID, l.CustomerID, c.Name, l.EndDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30) LOOP

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

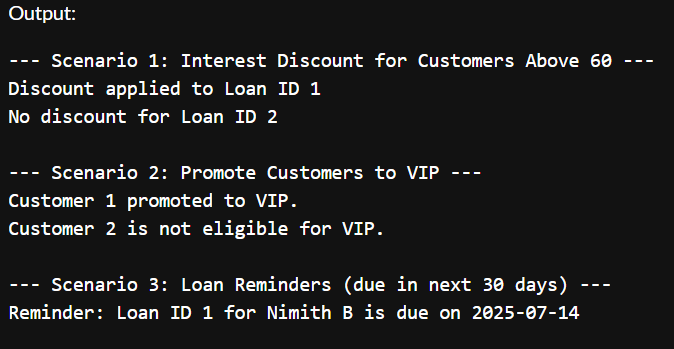
' for customer ' || loan.Name ||

' is due on ' || TO\_CHAR(loan.EndDate, 'YYYY-MM-DD'));

END LOOP;

END;

**OUTPUT:**

****

**Exercise 3: Stored Procedures**

**Scenario 1: ProcessMonthlyInterest**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings') LOOP

UPDATE Accounts

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

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID ' || acc.AccountID);

END LOOP;

COMMIT;

END;

**Scenario 2: UpdateEmployeeBonus**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

FOR emp IN (SELECT EmployeeID, Salary FROM Employees WHERE Department = p\_department) LOOP

UPDATE Employees

SET Salary = Salary + (emp.Salary \* p\_bonus\_percent / 100)

WHERE EmployeeID = emp.EmployeeID;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to Employee ID ' || emp.EmployeeID);

END LOOP;

COMMIT;

END;

**Scenario 3: TransferFunds**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_account;

IF v\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_amount || ' from Account ' || p\_from\_account || ' to Account ' || p\_to\_account);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('One of the accounts does not exist.');

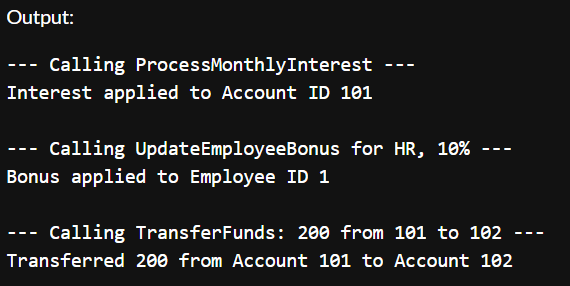
WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);

END;

**OUTPUT :**

****

**TDD Using JUNIT**

**Exercise 1 Setting up JUNIT:**

**pom.xml**

<dependencies>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

</dependencies>

**NOTE: I added test class called calculator here to test if junit is setup**

**Calculator.java**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**Exercise 3: Assertions in Junit**

**//Here assertions and test is implemented**

**CalculatorTest.java**

import org.junit.jupiter.api.Test;

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

public class CalculatorTest {

@Test

void testAdd() {

Calculator calc = new Calculator();

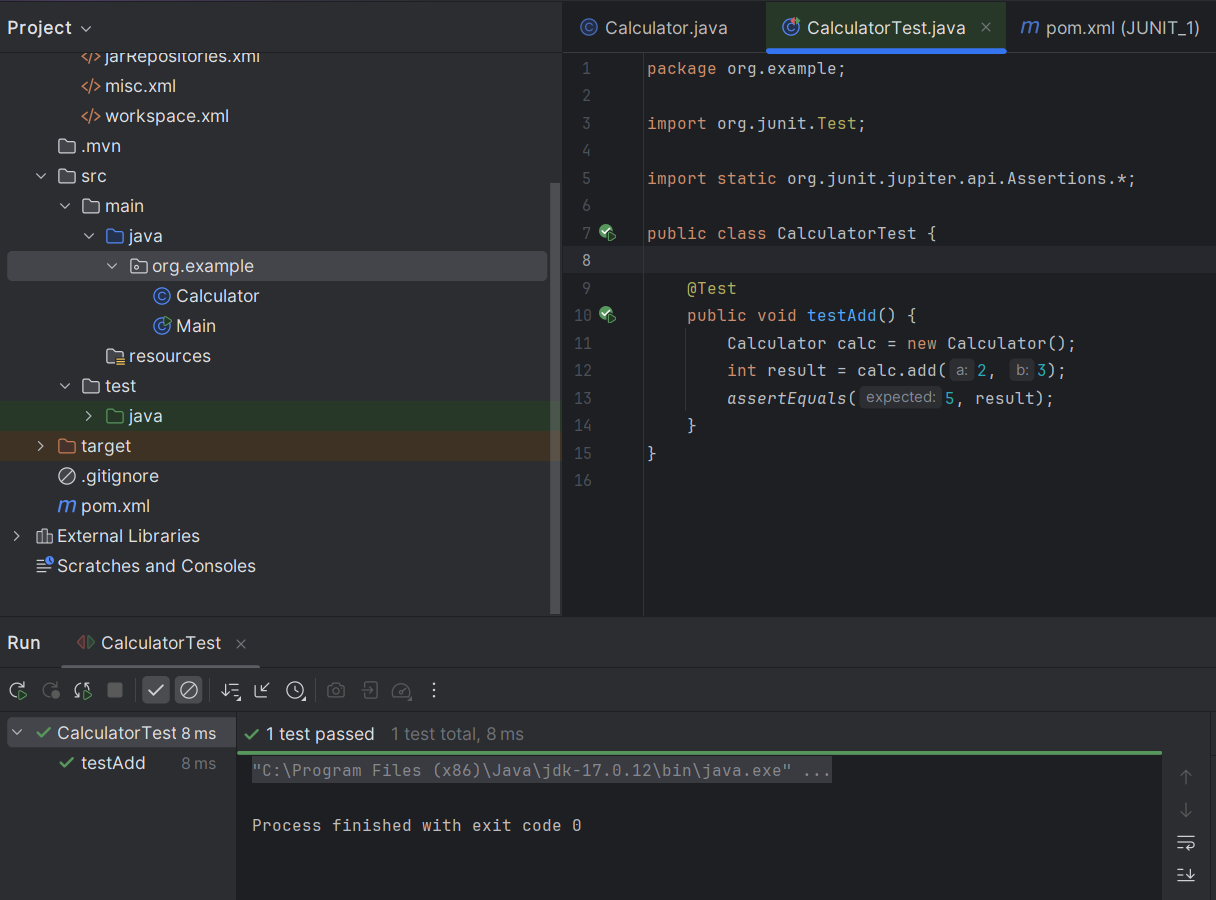
int result = calc.add(2, 3);

assertEquals(5, result);

}

}

**OUTPUT:**

****

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

**Calculator.java**

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.jupiter.api.\*;

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

public class CalculatorTest {

private Calculator calculator;

@BeforeEach

void setUp() {

calculator = new Calculator(); // Arrange

System.out.println("Setup done.");

}

@AfterEach

void tearDown() { **// TEARDOWN**

System.out.println("Test complete.\n");

}

@Test

void testAdd() {

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

assertEquals(5, result); // Assert

}

@Test

void testSubtract() {

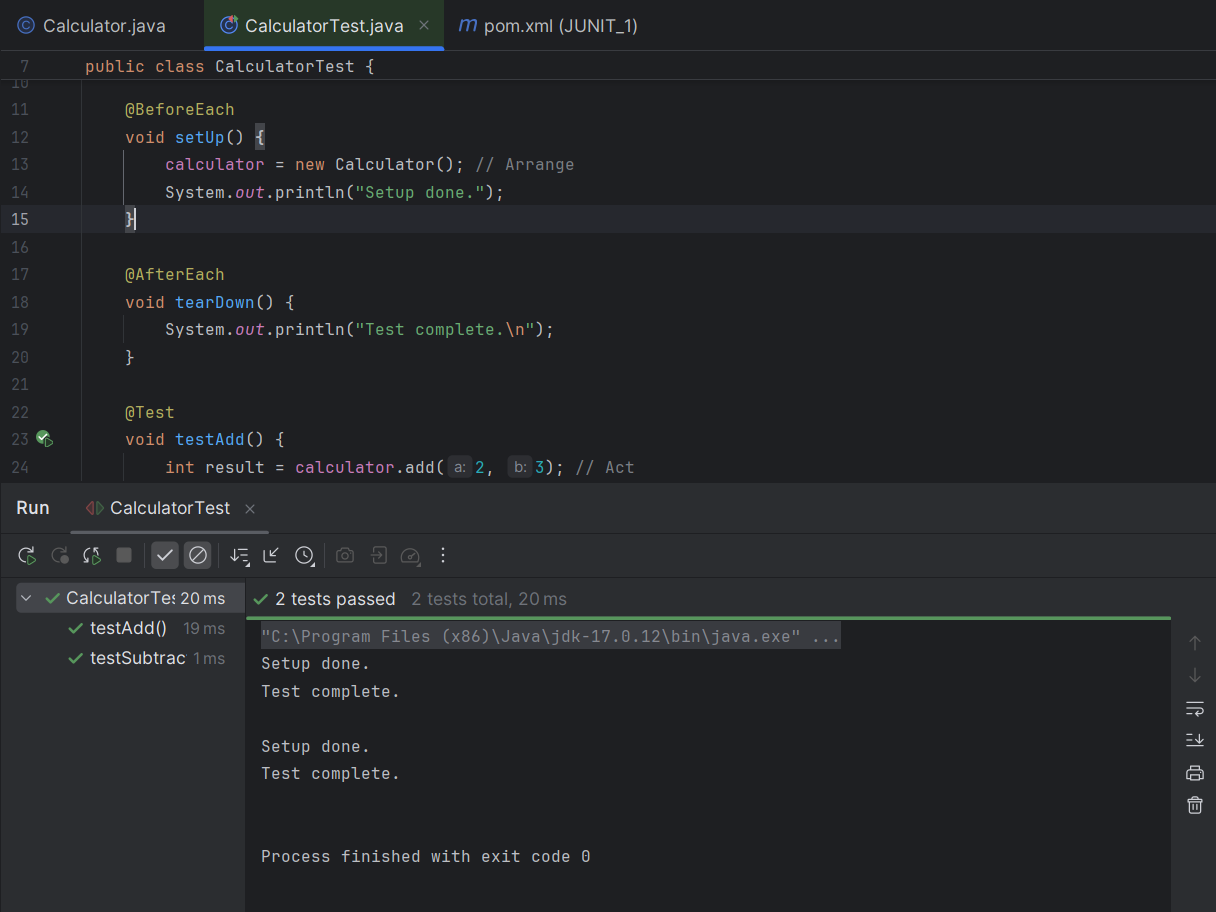
int result = calculator.subtract(10, 4); // Act

assertEquals(6, result); // Assert

}

}

**OUTPUT:**

****

**Exercise 1: Mocking and Stubbing:**

**ExternalApi.java**

public class ExternalApi {

public String getData() {

return "Real Data";

}

}

**MyService.java**

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest**

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

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

public class MyServiceTest {

@Test

void testExternalApi() {

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

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

MyService service = new MyService(mockApi);

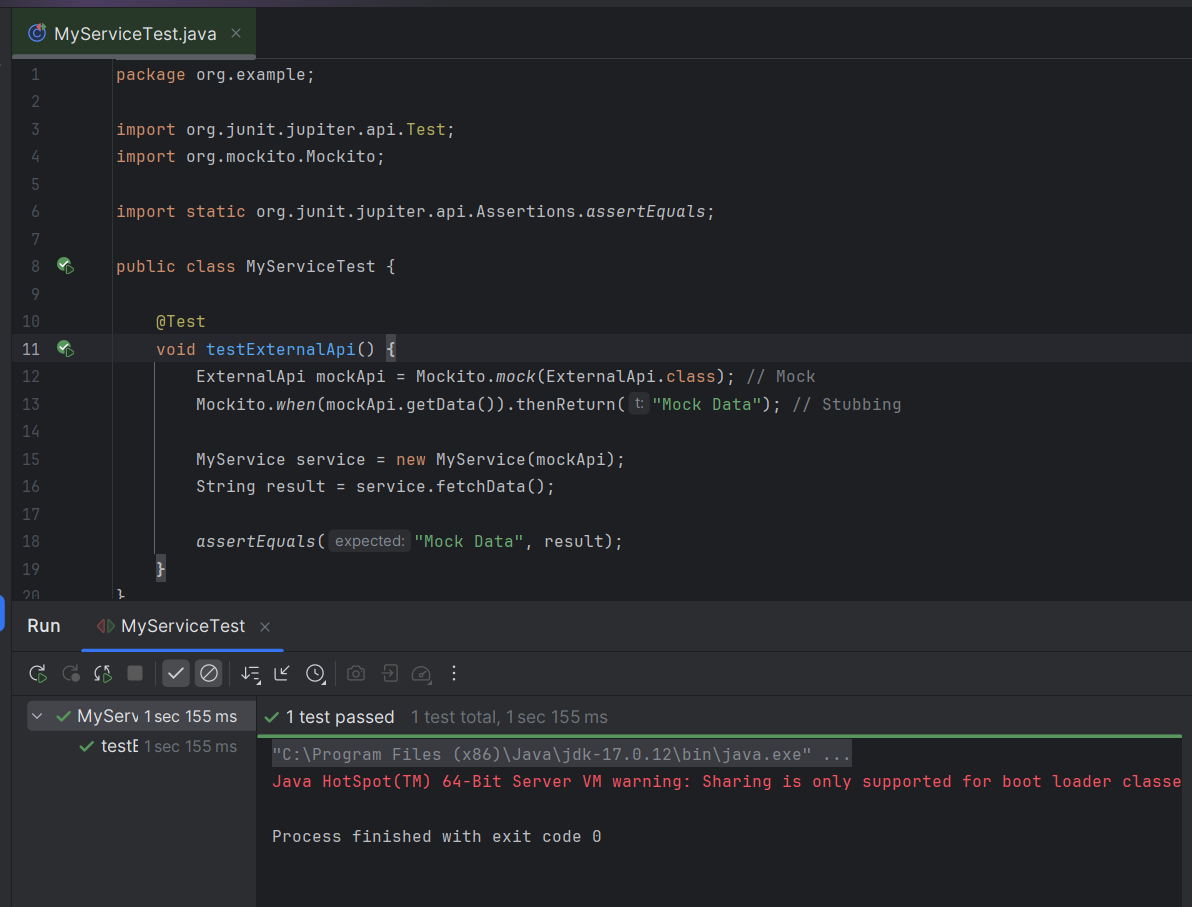
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**OUTPUT:**



**Exercise 2: Verifying Interactions**

**ExternalApi.java**

public class ExternalApi {

public String getData() {

return "Real Data";

}

}

**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 org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

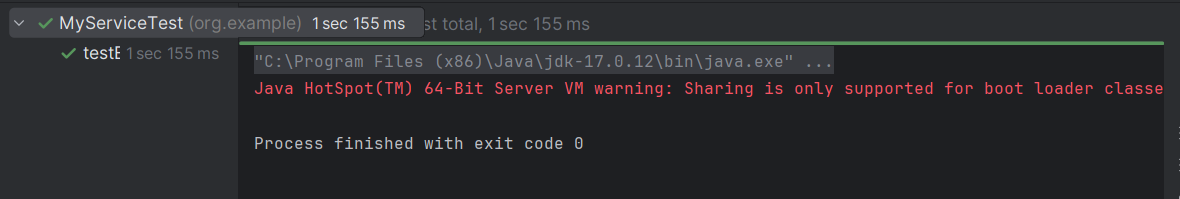
service.fetchData();

verify(mockApi).getData(); // Verifying interaction

}

}

**OUTPUT:**



**SLF4J logging framework**

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

**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) {

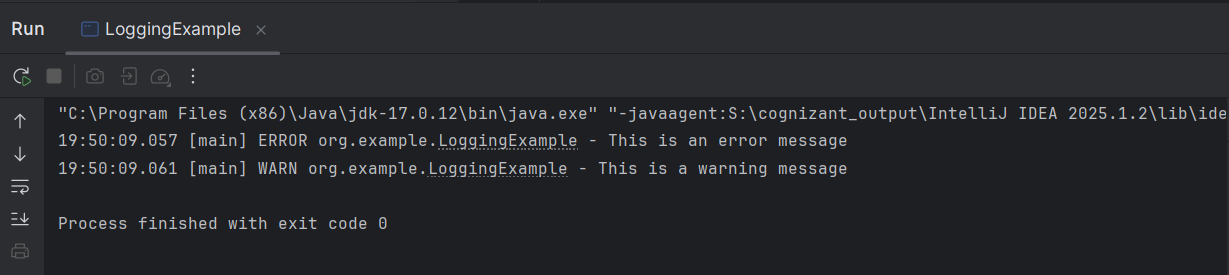
logger.error("This is an error message");

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

}

}

**OUTPUT:**

****