# WEEK-2

## **Module-3:PL/SQL Programming**

## **Exercise -1: Control structures**

**Code: Creating A Table and inserting values to it**

**CREATE TABLE customers (**

**customer\_id NUMBER PRIMARY KEY,**

**name VARCHAR2(100),**

**age NUMBER,**

**balance NUMBER(10, 2),**

**is\_vip VARCHAR2(5) DEFAULT 'FALSE'**

**);**

**CREATE TABLE loans (**

**loan\_id NUMBER PRIMARY KEY,**

**customer\_id NUMBER REFERENCES customers(customer\_id),**

**interest\_rate NUMBER(5,2),**

**due\_date DATE**

**);**

**INSERT INTO customers VALUES (1, 'Alice', 65, 12000, 'FALSE');**

**INSERT INTO customers VALUES (2, 'Bob', 45, 8000, 'FALSE');**

**INSERT INTO customers VALUES (3, 'Carol', 70, 9500, 'FALSE');**

**INSERT INTO loans VALUES (101, 1, 7.5, SYSDATE + 10);**

**INSERT INTO loans VALUES (102, 2, 6.0, SYSDATE + 45);**

**INSERT INTO loans VALUES (103, 3, 8.0, SYSDATE + 5);**

**COMMIT;**

**Apply Discount on Above 60yr old:**

**BEGIN**

**FOR rec IN (**

**SELECT l.loan\_id**

**FROM loans l**

**JOIN customers c ON l.customer\_id = c.customer\_id**

**WHERE c.age > 60**

**) LOOP**

**UPDATE loans**

**SET interest\_rate = interest\_rate - 1**

**WHERE loan\_id = rec.loan\_id;**

**END LOOP;**

**COMMIT;**

**END;**

**/**

**mark\_vip\_customers.sql**

**BEGIN**

**FOR rec IN (**

**SELECT customer\_id**

**FROM customers**

**WHERE balance > 10000**

**) LOOP**

**UPDATE customers**

**SET is\_vip = 'TRUE'**

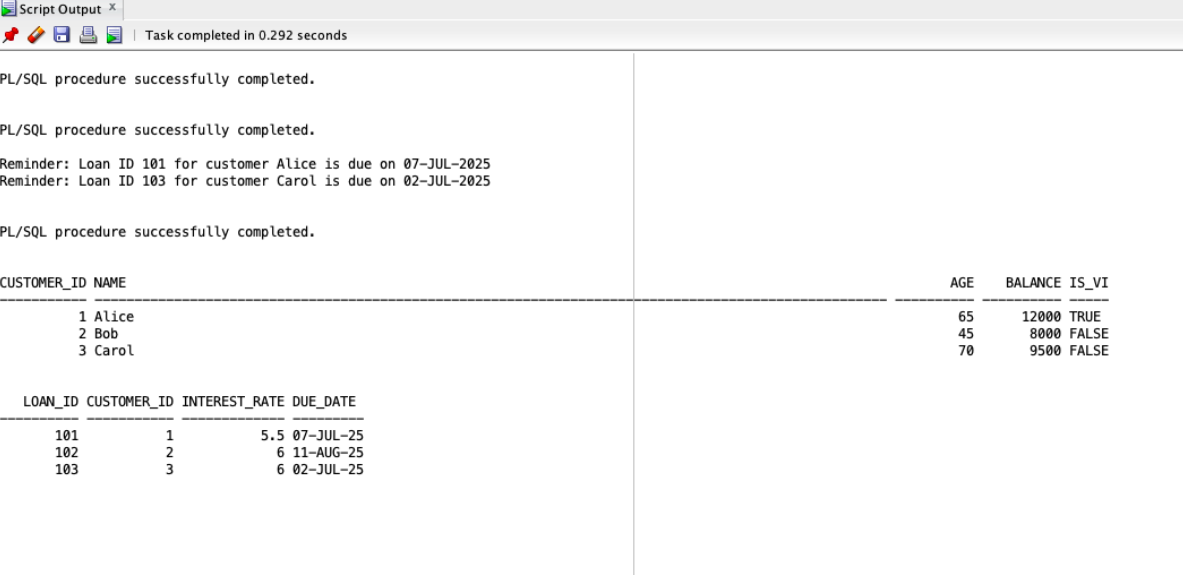
**WHERE customer\_id = rec.customer\_id;**

**END LOOP;**

**COMMIT;**

**END;**

**/**

**OUTPUT:**

**Exercise 3 - Stored Procedures**

**CREATING TABLE AND INSERTING VALUES IN IT**

**CREATE TABLE customers (**

**customer\_id NUMBER PRIMARY KEY,**

**name VARCHAR2(100),**

**age NUMBER,**

**balance NUMBER(15, 2),**

**is\_vip VARCHAR2(5) DEFAULT 'FALSE'**

**);**

**CREATE TABLE accounts (**

**account\_id NUMBER PRIMARY KEY,**

**customer\_id NUMBER REFERENCES customers(customer\_id),**

**balance NUMBER(15, 2),**

**account\_type VARCHAR2(20)**

**);**

**CREATE TABLE employees (**

**employee\_id NUMBER PRIMARY KEY,**

**department VARCHAR2(50),**

**salary NUMBER(15, 2)**

**);**

**INSERT ALL**

**INTO customers VALUES (1101, 'Alice', 65, 12000, 'FALSE')**

**INTO customers VALUES (1102, 'Bob', 45, 8000, 'FALSE')**

**INTO customers VALUES (1103, 'Carol', 70, 9500, 'FALSE')**

**SELECT \* FROM dual;**

**INSERT ALL**

**INTO accounts VALUES (101, 1, 15000, 'SAVINGS')**

**INTO accounts VALUES (102, 2, 300000, 'CHECKING')**

**SELECT \* FROM dual;**

**INSERT ALL**

**INTO employees (employee\_id, department, salary)**

**VALUES (101, 'IT', 15000)**

**INTO employees (employee\_id, department, salary)**

**VALUES (102, 'IT', 300000)**

**SELECT \* FROM dual;**

**COMMIT;**

**SELECT \* FROM customers;**

**SELECT \* FROM accounts;**

**SELECT \* FROM employees;**

**PROCESS MONTHLY INTEREST**

**CREATE OR REPLACE PROCEDURE Process Monthly Interest AS**

**BEGIN**

**UPDATE accounts**

**SET balance = balance \* 1.01**

**WHERE account\_type = 'SAVINGS';**

**COMMIT;**

**END;**

**/**

**Stored procedure test:**

BEGIN

ProcessMonthlyInterest;

END;

/

BEGIN

UpdateEmployeeBonus('IT', 5);

END;

/

BEGIN

TransferFunds(101, 102, 1000);

END;

/

SELECT \* FROM accounts;

SELECT \* FROM employees;

FUND TRANSFERS:

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_acct IN NUMBER,

p\_to\_acct IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

SELECT balance INTO v\_balance

FROM accounts

WHERE account\_id = p\_from\_acct

FOR UPDATE;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds: ' || v\_balance);

END IF;

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_acct;

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_acct;

COMMIT;

END;

/

**UpdateEmployeeBonus:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_dept IN VARCHAR2,

p\_bonus\_pct IN NUMBER

) AS

BEGIN

UPDATE employees

SET salary = salary \* (1 + p\_bonus\_pct / 100)

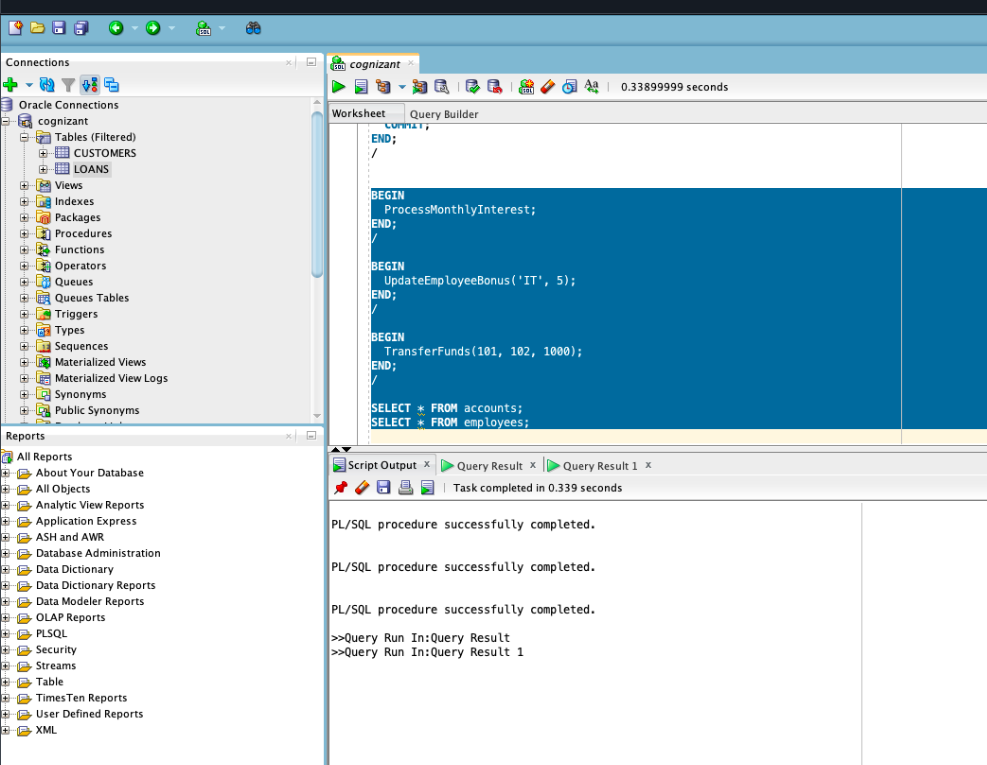
WHERE department = p\_dept;

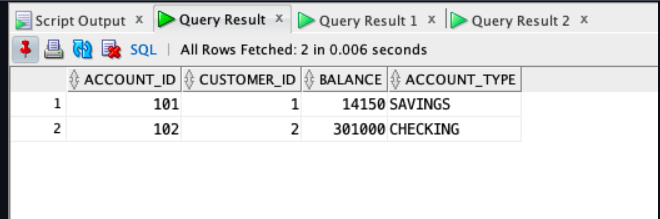
COMMIT;

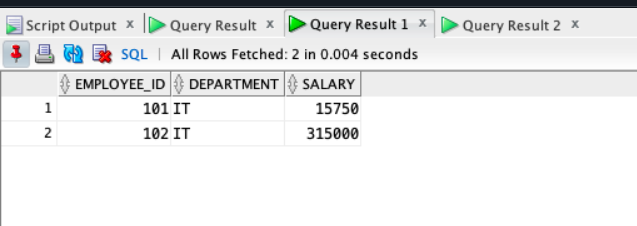
END;

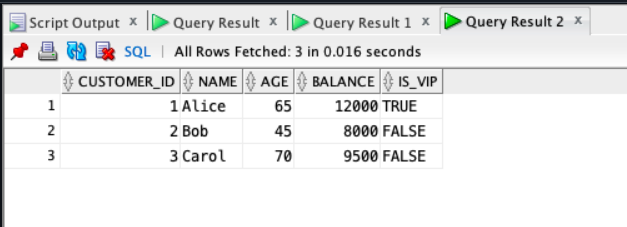
/

OUTPUT:









# **Module-4:Test driven development and Logging framework**

package com.example;

/\*\*

\* Hello world!

\*

\*/

public class App

{

public static void main( String[] args )

{

System.out.println( "Hello World!" );

}

}

***Calculator:***

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;

}

}

**J UNIT:**

**Exercise-01:SettingUpJunit:**

package com.example.JUnit.Exercise\_01\_Setting\_Up\_JUnit;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class TestClass {

@Test

public void testAddition() {

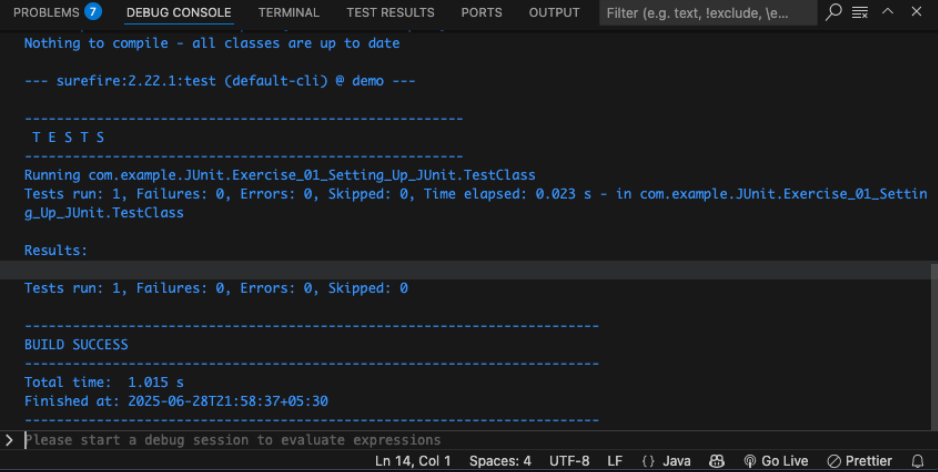
int result = 2 + 3;

assertEquals("Addition should be 5", 5, result);

}

}

OUTPUT:



**Exercise-03:Assertions\_in\_Junit:**

package com.example.JUnit.Exercise\_03\_Assertions\_in\_JUnit;

import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertFalse;

import static org.junit.Assert.assertNotNull;

import static org.junit.Assert.assertNull;

import static org.junit.Assert.assertTrue;

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);

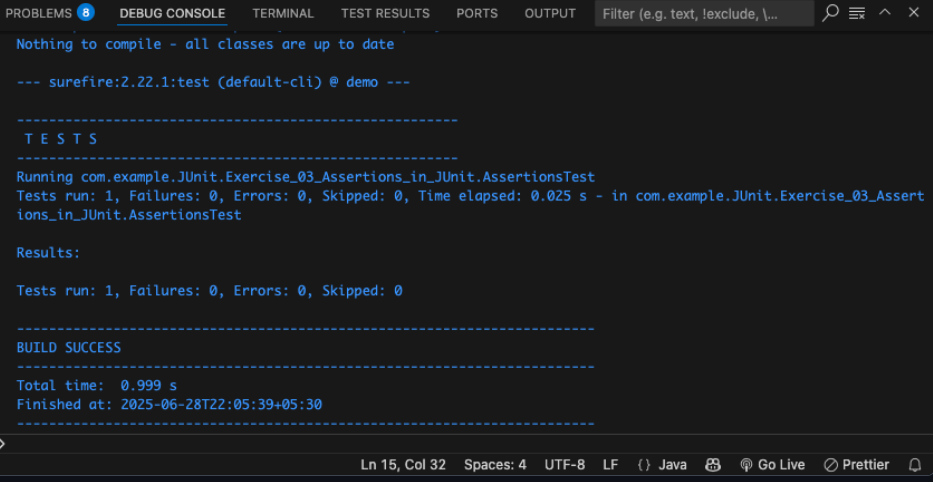
// Assert not null

assertNotNull(new Object());

}

}

OUTPUT:



# **Exercise\_04\_AAA\_and\_Fixtures:**

package com.example.JUnit.Exercise\_04\_AAA\_and\_Fixtures;

import org.junit.After;

import static org.junit.Assert.assertEquals;

import org.junit.Before;

import org.junit.Test;

import com.example.Calculator;

public class CalculatorTest {

private Calculator calculator;

@Before

public void setUp() {

calculator = new Calculator();

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

}

@After

public void tearDown() {

calculator = null;

System.out.println("Teardown complete.");

}

@Test

public void testAddition() {

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

assertEquals(8, result);

}

@Test

public void testSubtraction() {

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

assertEquals(6, result);

}

}

OUTPUT:



**Mockito:**

**Exercise\_01\_Mocking\_and\_Stubbing:**

package com.example.Mockito.Exercise\_01\_Mocking\_and\_Stubbing;

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

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.when;

import com.example.Mockito.ExternalApi;

import com.example.Mockito.MyService;

public class MyServiceTest {

@Test

public void testExternalApi() {

// Arrange

ExternalApi mockApi = mock(ExternalApi.class);

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

// Act

MyService service = new MyService(mockApi);

String result = service.fetchData();

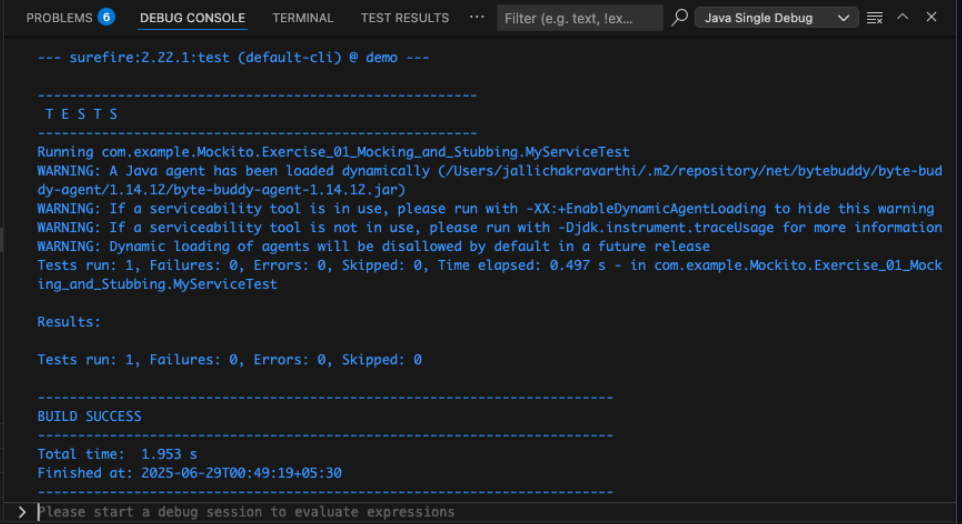
// Assert

assertEquals("Mock Data", result);

}

}

OUTPUT:



**Exercise\_02\_Verifying\_Interactions:**

package com.example.Mockito.Exercise\_02\_Verifying\_Interactions;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.verify;

import com.example.Mockito.ExternalApi;

import com.example.Mockito.MyService;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// Arrange

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

// Act

service.fetchData();

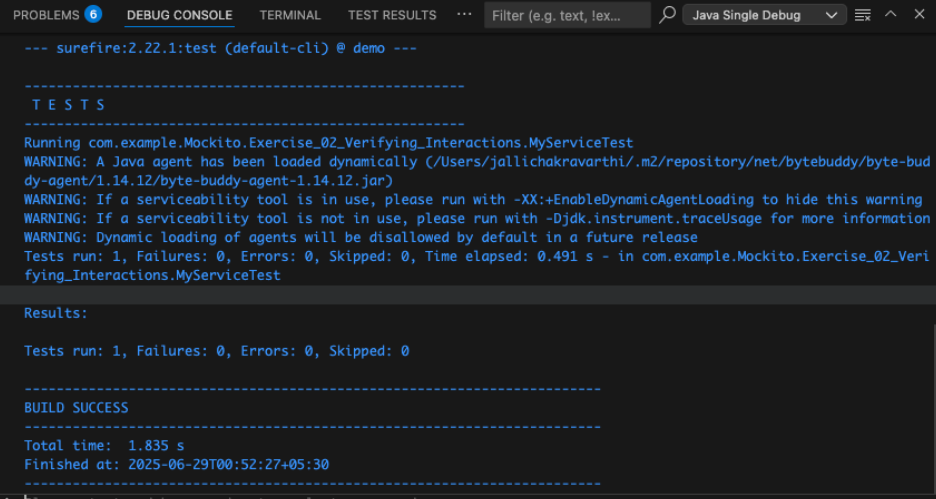
// Assert

verify(mockApi).getData();

}

}

OUTPUT:



ExternalApi:

package com.example.Mockito;

public interface ExternalApi {

String getData();

}

MY SERVICE :

package com.example.Mockito;

public class MyService {

private ExternalApi

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData()

}

}

[SLF4J\_Logging](https://github.com/jallichakravarthi/Digital-Nurture-4.0-6372771/tree/main/Week%2002-Programming%20Languages/Module%2004%20%E2%80%93%20Test%20driven%20development%20and%20Logging%20framework/src/test/java/com/example/SLF4J_Logging):

**Exercise\_01\_Logging\_Error\_Warning**

package com.example.SLF4J\_Logging.Exercise\_01\_Logging\_Error\_Warning;

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) {

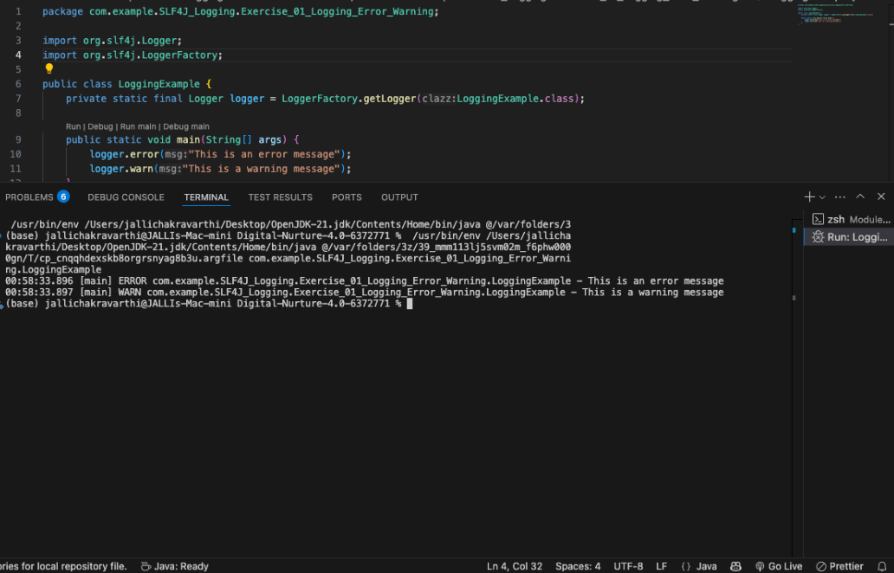
logger.error("This is an error message");

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

}

}

OUTPUT:



**AppTest:**

package com.example;

import static org.junit.Assert.\*;

import org.junit.Test;

/\*\*

\* Unit test for simple App.

\*/

public class AppTest

{

/\*\*

\* Rigorous Test :-)

\*/

@Test

public void shouldAnswerWithTrue()

{

assertTrue( true );

}

}