WEEK – 2

PL/SQL programming:

Schema to be Created

```
CREATE TABLE Customers (
   CustomerID NUMBER PRIMARY KEY,
   Name VARCHAR2(100),
   DOB DATE,
   Balance NUMBER,
   LastModified DATE
);

CREATE TABLE Accounts (
   AccountID NUMBER PRIMARY KEY,
   CustomerID NUMBER,
   AccountType VARCHAR2(20),
   Balance NUMBER,
   LastModified DATE,
   FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

CREATE TABLE Transactions (
   TransactionID NUMBER PRIMARY KEY,
   AccountID NUMBER,
   TransactionDate DATE,
   Amount NUMBER,
   TransactionType VARCHAR2(10),
   FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)
);

CREATE TABLE Loans (
   LoanID NUMBER PRIMARY KEY,
   CustomerID NUMBER,
   LoanAmount NUMBER,
   InterestRate NUMBER,
   StartDate DATE,
   EndDate DATE,
   FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

CREATE TABLE Employees (
   EmployeeID NUMBER PRIMARY KEY,
   Name VARCHAR2(100),
   Position VARCHAR2(50),
   Salary NUMBER,
   Department VARCHAR2(50),
  HireDate DATE
);
```

Example Scripts for Sample Data Insertion

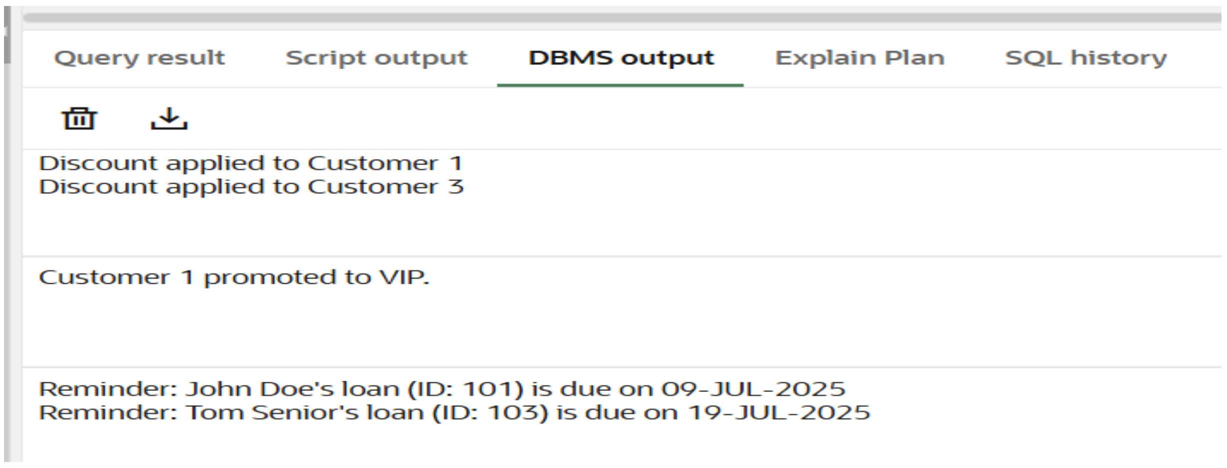
```
INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)
VALUES (1, 'John Doe', TO_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)
VALUES (2, 'Jane Smith', TO_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)
VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)
VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)
VALUES (1, 1, 5000, 5, SYSDATE, ADD_MONTHS(SYSDATE, 60));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)
VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)
VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO_DATE('2017-03-20', 'YYYY-MM-DD'));
```

Exercise 1: Control Structures :

CODE:

```
---------
BEGIN
 FOR c IN (
  SELECT c.CustomerID, c.DOB, l.LoanID, l.InterestRate
  FROM Customers c
  JOIN Loans l ON c.CustomerID = l.CustomerID
 ) LOOP
  IF MONTHS_BETWEEN(SYSDATE, c.DOB)/12 > 60 THEN
   UPDATE Loans
   SET InterestRate = InterestRate - 1
   WHERE LoanID = c.LoanID;

   DBMS_OUTPUT.PUT_LINE('Discount applied to Customer ' || c.CustomerID);
  END IF;
 END LOOP;

 COMMIT;
END;
/

BEGIN
 FOR c IN (SELECT CustomerID, Balance FROM Customers) LOOP
  IF c.Balance > 10000 THEN
   UPDATE Customers
   SET IsVIP = 'TRUE'
   WHERE CustomerID = c.CustomerID;

   DBMS_OUTPUT.PUT_LINE('Customer ' || c.CustomerID || ' promoted to VIP.');
  END IF;
 END LOOP;

 COMMIT;
END;
/
BEGIN
 FOR rec IN (
  SELECT l.LoanID, l.EndDate, c.Name
  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: ' || rec.Name || '''s loan (ID: ' || rec.LoanID || ') is due on ' ||
TO_CHAR(rec.EndDate, 'DD-MON-YYYY'));
 END LOOP;
END;
/
```

OUTPUT :



```
Discount applied to Customer 1
Discount applied to Customer 3

Customer 1 promoted to VIP.

Reminder: John Doe's loan (ID: 101) is due on 09-JUL-2025
Reminder: Tom Senior's loan (ID: 103) is due on 19-JUL-2025
```

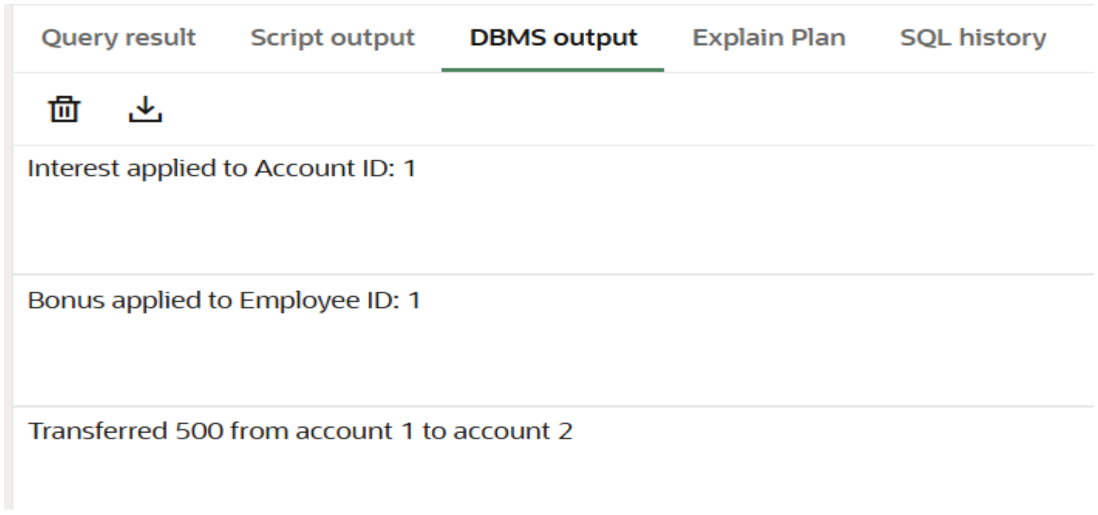
Exercise 3: Stored Procedures :

CODE:

```
---------
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;
/

BEGIN
  PROCESSMONTHLYINTEREST;
END;
/

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;
/

BEGIN
    UPDATEEMPLOYEEBONUS('HR', 10);
END;
/

CREATE OR REPLACE PROCEDURE TRANSFERFUNDS (
    P_FROM_ACCOUNT IN NUMBER,
    P_TO_ACCOUNT   IN NUMBER,
    P_AMOUNT       IN NUMBER
) IS
    V_BALANCE NUMBER;
BEGIN
  -- Get balance from source account
    SELECT
        BALANCE
    INTO V_BALANCE
    FROM
        ACCOUNTS
    WHERE
        ACCOUNTID = P_FROM_ACCOUNT
    FOR UPDATE;

    IF V_BALANCE < P_AMOUNT THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds in source account.');
    END IF;

  -- Deduct from source
    UPDATE ACCOUNTS
    SET
        BALANCE = BALANCE - P_AMOUNT,
        LASTMODIFIED = SYSDATE
    WHERE
        ACCOUNTID = P_FROM_ACCOUNT;

  -- Add to destination
  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('Error: One or both accounts not found.');
  WHEN OTHERS THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE('Transfer failed: ' || SQLERRM);
END;
/

BEGIN
  TRANSFERFUNDS(1, 2, 500); -- Transfer 500 from Account 1 to 2
END;
/
```

OUTPUT:



```
Interest applied to Account ID: 1

Bonus applied to Employee ID: 1

Transferred 500 from account 1 to account 2
```

1. JUnit Basic Testing Exercises:

Exercise 1: Setting Up Junit:

Exercise 3: Assertions in Junit

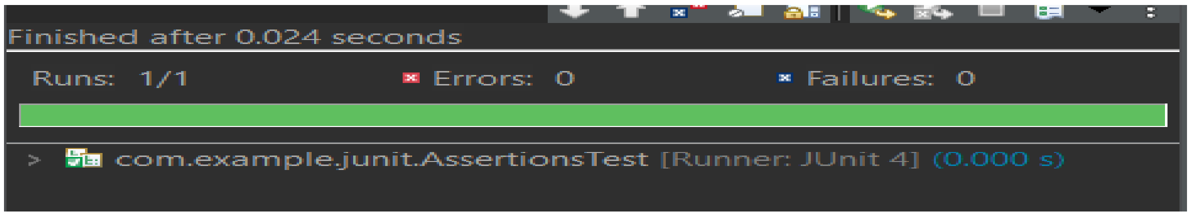
Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and

CODE:

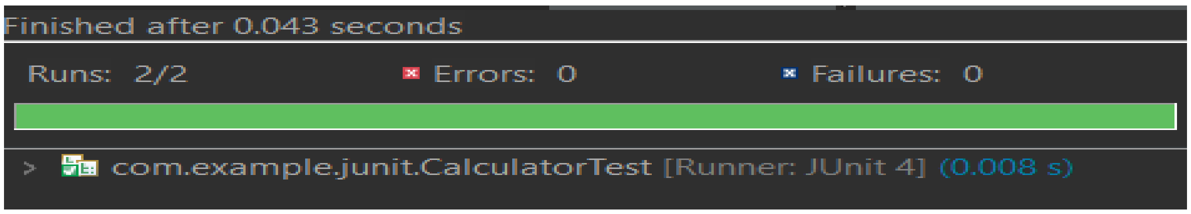
```
package com.example.junit;

public class Calculator {
   public int add(int a, int b) {
      return a + b;
   }

   public int subtract(int a, int b) {
      return a - b;
   }
}
----------------------------------------------------------------------------------------------------------
package com.example.junit;

import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.After;
import org.junit.Test;

public class CalculatorTest {

   private Calculator calc;

   @Before
   public void setUp() {
      calc = new Calculator();
   }

   @After
   public void tearDown() {
      calc = null;
   }
   @Test
   public void testAdd() {
      int result = calc.add(2, 3);
      System.out.println("testAdd result: " + result); // 👈 console output
      assertEquals(5, result);
   }

   @Test
   public void testSubtract() {
            int results = calc.subtract(5, 2);
            System.out.println("testsub results: " + results);
            assertEquals(3, results);
    }


}
```

```
package com.example.junit;

import static org.junit.Assert.*;
import org.junit.Test;

public class AssertionsTest {
    @Test
    public void testAssertions() {
        assertEquals(5, 2 + 3);
        assertTrue(5 > 3);
        assertFalse(5 < 3);
        assertNull(null);
        assertNotNull(new Object());
    }
}
```

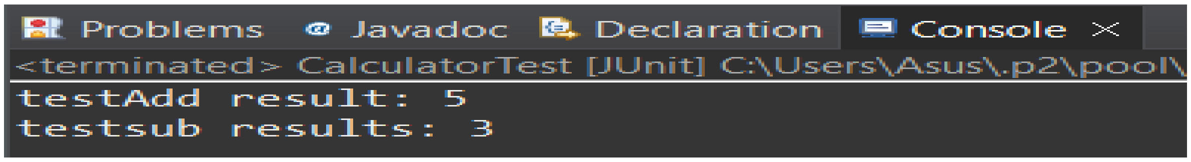
OUTPUTS:



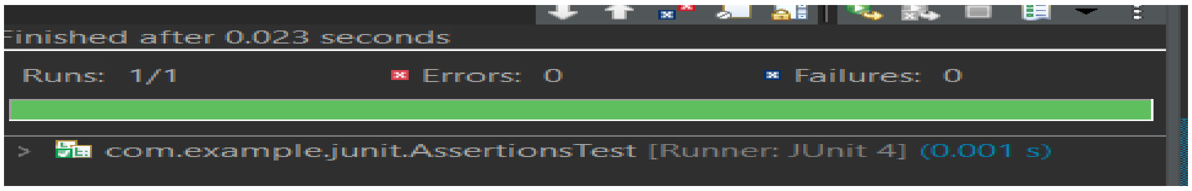
```
Finished after 0.024 seconds
Runs:  1/1          Errors:  0          Failures:  0
> com.example.junit.AssertionsTest [Runner: JUnit 4] (0.000 s)
```



```
Finished after 0.043 seconds
Runs:  2/2          Errors:  0          Failures:  0
> com.example.junit.CalculatorTest [Runner: JUnit 4] (0.008 s)
```



```
<terminated> CalculatorTest [JUnit] C:\Users\Asus\.p2\pool\
testAdd result: 5
testsub results: 3
```



```
Finished after 0.023 seconds
Runs:  1/1          Errors:  0          Failures:  0
> com.example.junit.AssertionsTest [Runner: JUnit 4] (0.001 s)
```

3. Mockito exercises:

Exercise 1: Mocking and Stubbing:

Exercise 2: Verifying Interactions:

CODE:

Dependency:

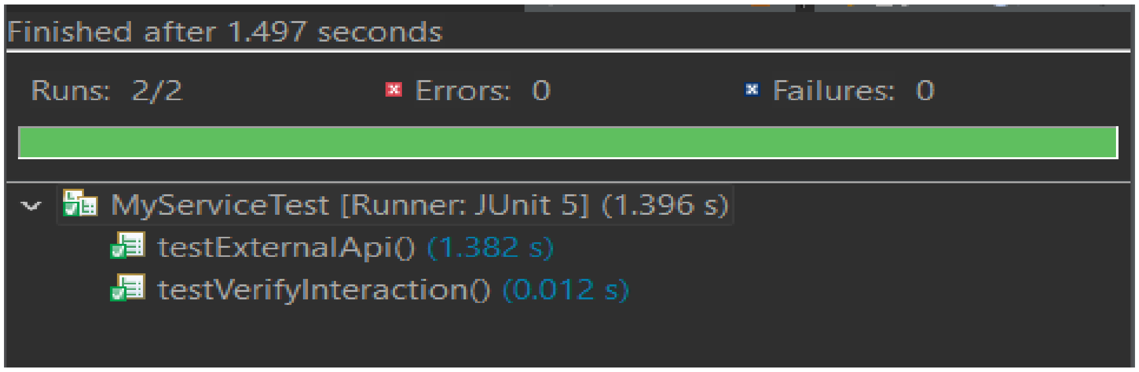
```
<dependencies>
  <dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>5.11.0</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.10.0</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

```
package com.example.mockito_demo;
public interface ExternalApi {
  String getData();
}
```

```
package com.example.mockito_demo;
public class MyService {
  private ExternalApi api;
  public MyService(ExternalApi api) {
    this.api = api;
  }
  public String fetchData() {
    return api.getData();
  }
}
```

```
package com.example.mockito_demo;

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);
  }
  @Test
  public void testVerifyInteraction() {
  ExternalApi mockApi = mock(ExternalApi.class);
     MyService service = new MyService(mockApi);
     service.fetchData();
     verify(mockApi).getData();
  }
}
```

OUTPUT:



```
Finished after 1.497 seconds
Runs: 2/2        Errors: 0        Failures: 0
MyServiceTest [Runner: JUnit 5] (1.396 s)
    testExternalApi() (1.382 s)
    testVerifyInteraction() (0.012 s)
```

6. SL4J Logging exercises:

Exercise 1: Logging Error Messages and Warning Levels:

CODE:

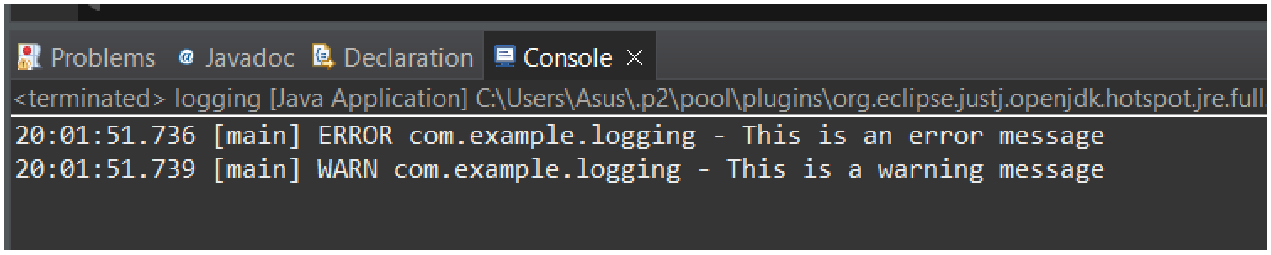
Dependency:

```
<dependencies>
```

```
  <!-- SLF4J API -->
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.30</version>
  </dependency>
  <!-- Logback for SLF4J implementation -->
  <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.2.3</version>
  </dependency>
</dependencies>
```

```
package com.example;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class logging{
  private static final Logger logger = LoggerFactory.getLogger(logging.class);
  public static void main(String[] args) {
    logger.error("This is an error message");
    logger.warn("This is a warning message");
  }
}
```

OUTPUT:



```
<terminated> logging [Java Application] C:\Users\Asus\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full
20:01:51.736 [main] ERROR com.example.logging - This is an error message
20:01:51.739 [main] WARN com.example.logging - This is a warning message
```