1. **PL/SQL programming**

**Exercise 1: Control Structures**

**DDL OPERATIONS**

CREATE TABLE customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

age NUMBER,

balance NUMBER,

is\_vip CHAR(1) DEFAULT 'N'

);

CREATE TABLE loans (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

interest\_rate NUMBER(5,2),

due\_date DATE,

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

**DML OPERATIONS**

INSERT INTO customers (customer\_id, name, age, balance) VALUES (1, 'John Smith', 65, 15000);

INSERT INTO customers (customer\_id, name, age, balance) VALUES (2, 'Alice Johnson', 45, 8000);

INSERT INTO customers (customer\_id, name, age, balance) VALUES (3, 'Robert Brown', 70, 12000);

INSERT INTO customers (customer\_id, name, age, balance) VALUES (4, 'Diana Green', 30, 25000);

INSERT INTO customers (customer\_id, name, age, balance) VALUES (5, 'Michael Lee', 62, 9500);

INSERT INTO customers (customer\_id, name, age, balance) VALUES (6, 'Emily White', 58, 10500);

-- Customer 1: due in 10 days

INSERT INTO loans (loan\_id, customer\_id, interest\_rate, due\_date)

VALUES (101, 1, 8.50, SYSDATE + 10);

-- Customer 2: due in 40 days (not within 30 days)

INSERT INTO loans (loan\_id, customer\_id, interest\_rate, due\_date)

VALUES (102, 2, 7.25, SYSDATE + 40);

-- Customer 3: due in 5 days

INSERT INTO loans (loan\_id, customer\_id, interest\_rate, due\_date)

VALUES (103, 3, 9.00, SYSDATE + 5);

-- Customer 4: due in 25 days

INSERT INTO loans (loan\_id, customer\_id, interest\_rate, due\_date)

VALUES (104, 4, 6.75, SYSDATE + 25);

-- Customer 5: due in 90 days

INSERT INTO loans (loan\_id, customer\_id, interest\_rate, due\_date)

VALUES (105, 5, 8.00, SYSDATE + 90);

-- Customer 6: due in 15 days

INSERT INTO loans (loan\_id, customer\_id, interest\_rate, due\_date)

VALUES (106, 6, 7.80, SYSDATE + 15);

COMMIT;

**QUERY**

BEGIN

FOR rec IN (SELECT customer\_id FROM customers WHERE age > 60) LOOP

UPDATE loans

SET interest\_rate = interest\_rate - 1

WHERE customer\_id = rec.customer\_id;

END LOOP;

COMMIT;

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

END;

/

BEGIN

FOR rec IN (SELECT customer\_id FROM customers WHERE balance > 10000) LOOP

UPDATE customers

SET is\_vip = 'Y'

WHERE customer\_id = rec.customer\_id;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('VIP status updated for eligible customers.');

END;

/

BEGIN

FOR rec IN (

SELECT l.loan\_id, l.customer\_id, l.due\_date, c.name

FROM loans l

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

WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.loan\_id || ' for customer ' ||

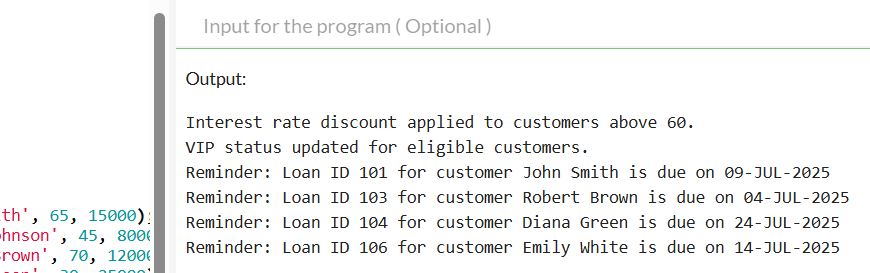
rec.name || ' is due on ' || TO\_CHAR(rec.due\_date, 'DD-MON-YYYY'));

END LOOP;

END;

/

**Output:**

****

**Exercise 2: Stored Procedures**

**DDL OPERATIONS**

SET SERVEROUTPUT ON;

-- Savings Accounts table

CREATE TABLE savings\_accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

balance NUMBER

);

-- Employees table

CREATE TABLE employees (

employee\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

department\_id NUMBER,

salary NUMBER

);

-- Generic Accounts table for transfer functionality

CREATE TABLE accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

balance NUMBER

);

**DML OPERATIONS**

INSERT INTO savings\_accounts (account\_id, customer\_id, balance) VALUES (201, 1, 5000);

INSERT INTO savings\_accounts (account\_id, customer\_id, balance) VALUES (202, 2, 10000);

INSERT INTO savings\_accounts (account\_id, customer\_id, balance) VALUES (203, 3, 7500);

INSERT INTO savings\_accounts (account\_id, customer\_id, balance) VALUES (204, 4, 2000);

INSERT INTO savings\_accounts (account\_id, customer\_id, balance) VALUES (205, 5, 12000);

INSERT INTO employees (employee\_id, name, department\_id, salary) VALUES (301, 'Alice', 10, 50000);

INSERT INTO employees (employee\_id, name, department\_id, salary) VALUES (302, 'Bob', 20, 60000);

INSERT INTO employees (employee\_id, name, department\_id, salary) VALUES (303, 'Charlie', 10, 55000);

INSERT INTO employees (employee\_id, name, department\_id, salary) VALUES (304, 'David', 30, 45000);

INSERT INTO employees (employee\_id, name, department\_id, salary) VALUES (305, 'Eva', 20, 62000);

INSERT INTO accounts (account\_id, customer\_id, balance) VALUES (401, 1, 7000);

INSERT INTO accounts (account\_id, customer\_id, balance) VALUES (402, 2, 3000);

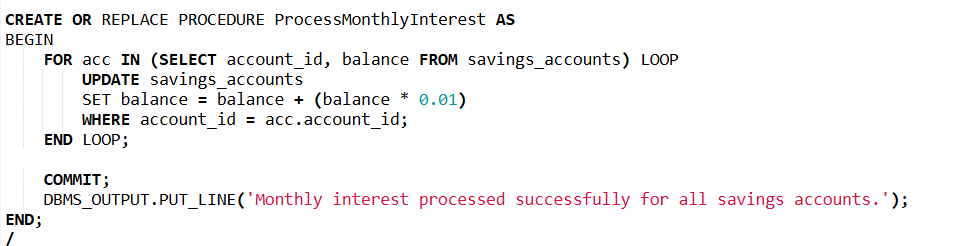
INSERT INTO accounts (account\_id, customer\_id, balance) VALUES (403, 3, 15000);

INSERT INTO accounts (account\_id, customer\_id, balance) VALUES (404, 4, 4000);

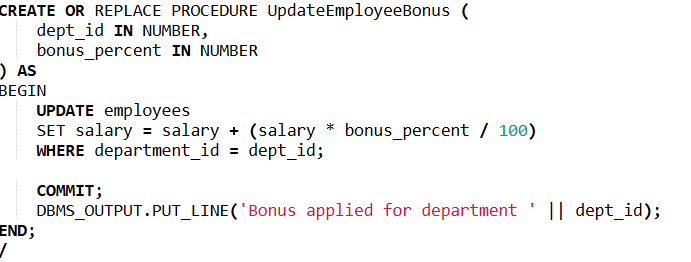
INSERT INTO accounts (account\_id, customer\_id, balance) VALUES (405, 5, 9000);

COMMIT;

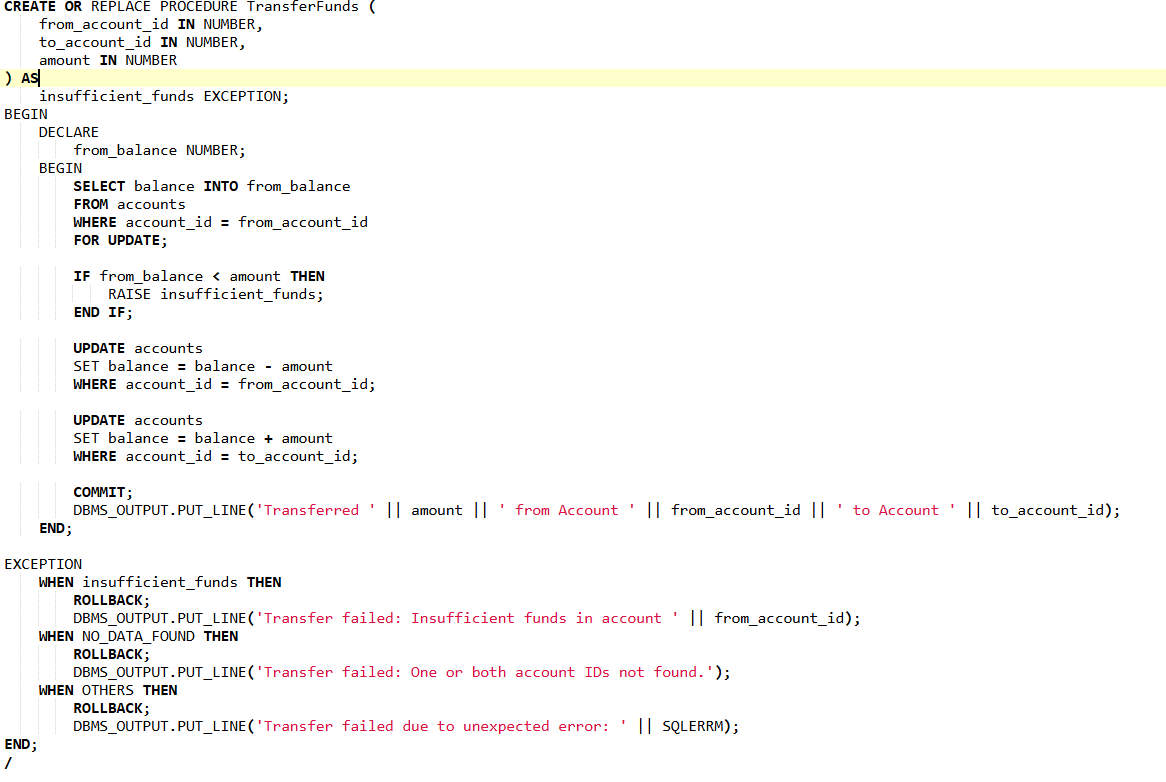
**QUERY**

****

1. ProcessMonthlyInterest

****

1. UpdateEmployeeBonus

****

1. TransferFunds
2. **TDD using JUnit5 and Mockito**

**Exercise 1: Setting Up Junit**

**Calculator.java**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

}

**CalculatorTest.java**

package main;

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

public class TestCalculator {

@Test

public void testAdd() {

Calculator calc = new Calculator();

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

assertEquals(8, result);

}

@Test

public void testMultiply() {

Calculator calc = new Calculator();

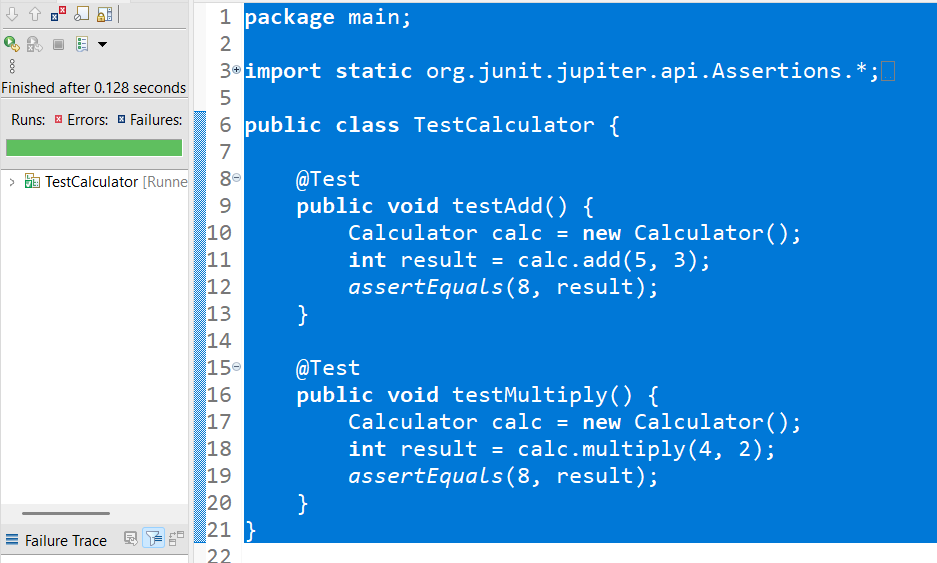
int result = calc.multiply(4, 2);

assertEquals(8, result);

}

}

**Output**



**Exercise 2: Assertions in Junit**

**AssertionsTest.java**

package main;

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

import org.junit.jupiter.api.Test;

public class AssertionsTest {

@Test

public void testAssertions() {

// Assert equals

System.out.println("Test running...");

assertEquals(5, 2 + 3, "2 + 3 should equal 5");

// Assert true

assertTrue(5 > 3, "5 is greater than 3");

// Assert false

assertFalse(5 < 3, "5 is not less than 3");

// Assert null

assertNull(null, "Expected null value");

// Assert not null

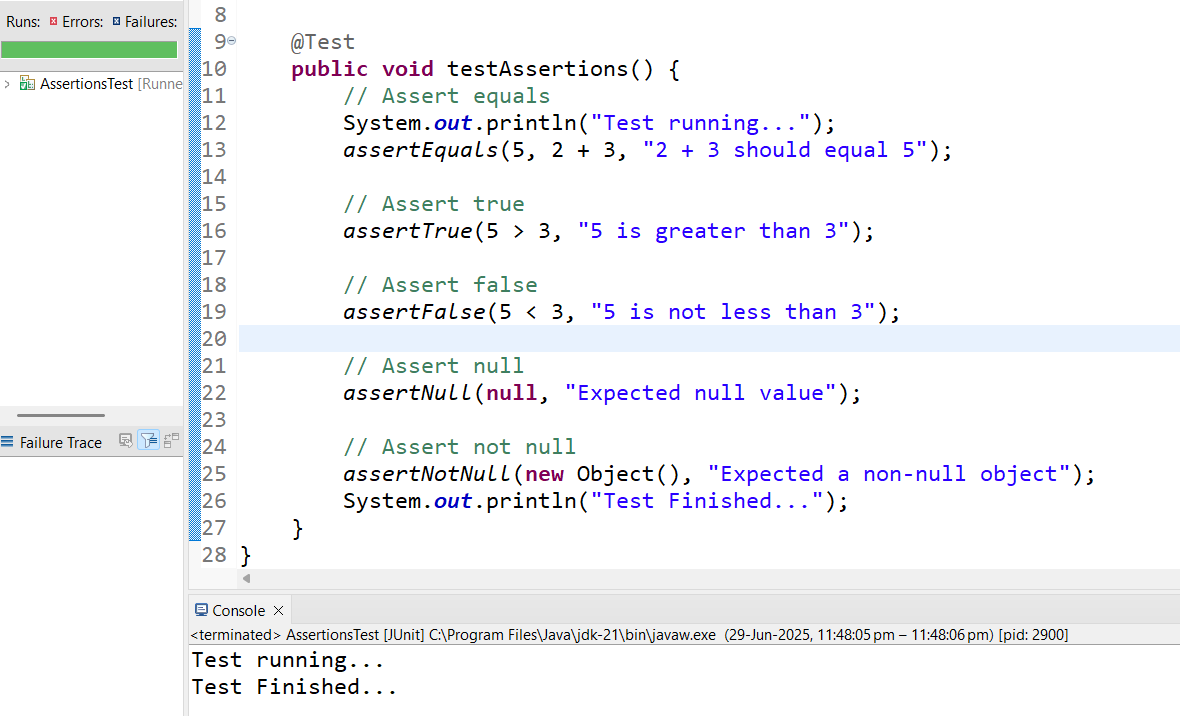
assertNotNull(new Object(), "Expected a non-null object");

System.out.println("Test Finished...");

}

}

**Output**



**Exercise 3: 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:**

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

@Before

public void setUp() {

calculator = new Calculator(); // Arrange: Create test object

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

}

@After

public void tearDown() {

calculator = null; // Clean up

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

}

@Test

public void testAdd() {

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

assertEquals("Addition should return 15", 15, result);

}

@Test

public void testSubtract() {

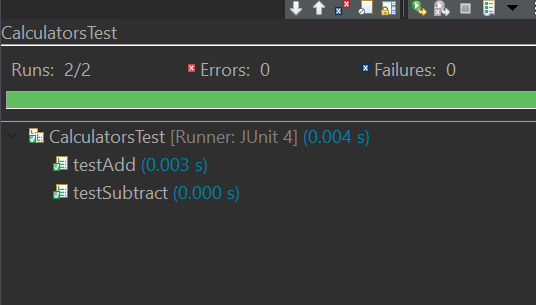
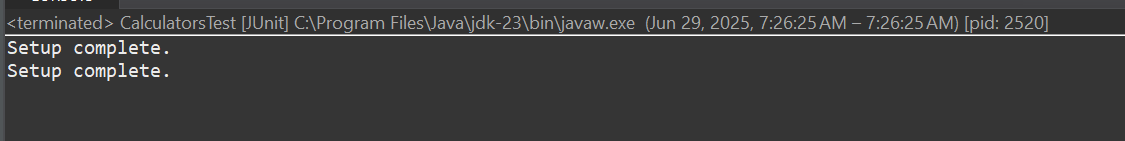
int result = calculator.subtract(10, 5);

assertEquals("Subtraction should return 5", 5, result);

}

}

**Output**

**Exercise 4: Mocking and Stubbing**

**1: External API Interface**

public interface ExternalApi {

String getData();

}

**2: Service That Uses the External API**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**3: Test Class Using Mockito**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testExternalApi() {

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

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

MyService service = new MyService(mockApi);

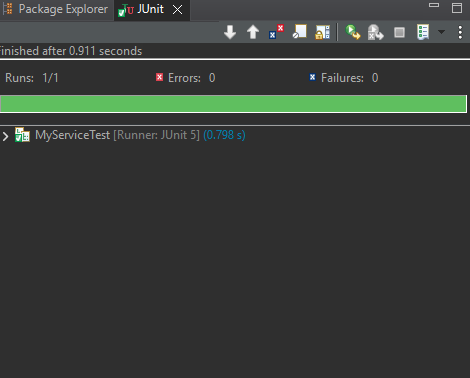
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**OUTPUT**

****

**Exercise 5: Verifying Interactions**

**1: External API Interface**

public interface ExternalApi {

String getData();

}

**2: Service That Uses the External API**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**3. Test Class with Mockito Verification:**

import static org.mockito.Mockito.\*;

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

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);

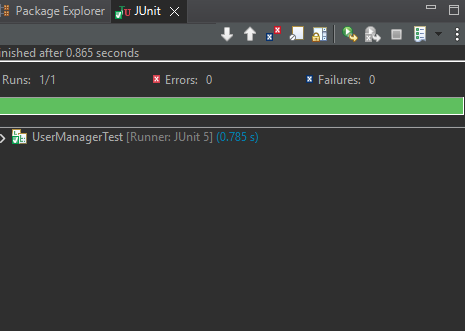
service.fetchData();

verify(mockApi).getData();

}

}

**OUTPUT:**

****

**3)TDD using JUnit5 and Mockito**

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

package com. dilip. maven. maven\_handson;

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) {

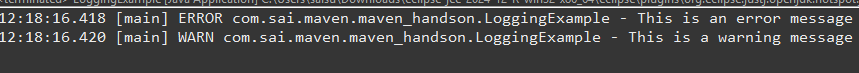
Logger. error ("This is an error message");

Logger warn ("This is a warning message");

}

}

**OUTPUT:**

****