**PL/SQL Programming**

**Exercise 1 : Control Structures**

**SCENARIO – 1 :**

BEGIN

FOR cust\_rec IN (

SELECT CustomerID

FROM Customers

WHERE MONTHS\_BETWEEN(SYSDATE, DOB)/12 > 60

)

LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust\_rec.CustomerID;

END LOOP;

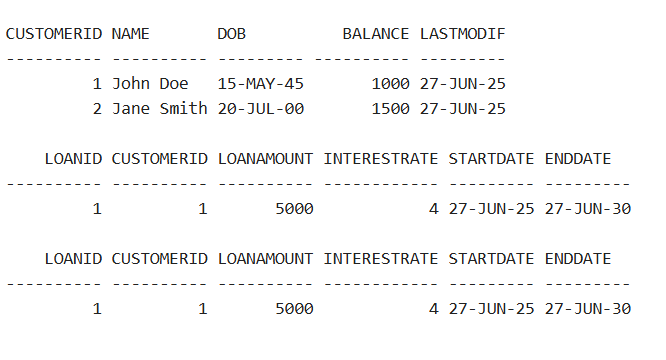
COMMIT;

END;

/

SELECT\*FROM Customers;

**OUTPUT SCREENSHOTS:**



**SCENARIO – 2 :**

BEGIN

FOR cust\_rec IN (SELECT CustomerID, Balance FROM Customers)

LOOP

IF cust\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || cust\_rec.CustomerID || 'promoted to VIP.');

END IF;

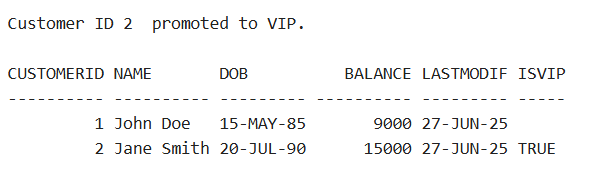
END LOOP;

END;

/

SELECT\*FROM Customers;

**OUTPUT SCREENSHOTS :**



**SCENARIO – 3 :**

BEGIN

FOR loan\_rec IN (SELECT l.LoanID, l.CustomerID, l.DueDate, c.Name 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: Dear ' || loan\_rec.Name ||

', your loan ID: ' || loan\_rec.LoanID ||

' is due on ' || TO\_CHAR(loan\_rec.DueDate, 'DD-MON-YYYY')

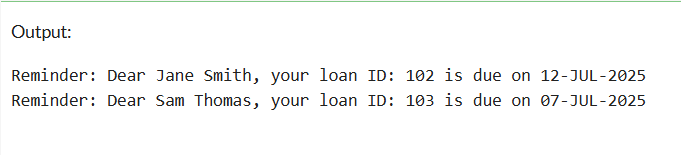
);

END LOOP;

END;

/

**OUTPUT SCREENSHOTS:**

****

**Exercise 3: Stored Procedures**

**SCENARIO 1:**

CREATE OR REPLACE PROCEDURE **ProcessMonthlyInterest** IS

BEGIN

FOR acc IN (SELECT AccountID, Balance FROM Accounts)

LOOP

UPDATE Accounts

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

WHERE AccountID = acc.AccountID;

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

END LOOP;

COMMIT;

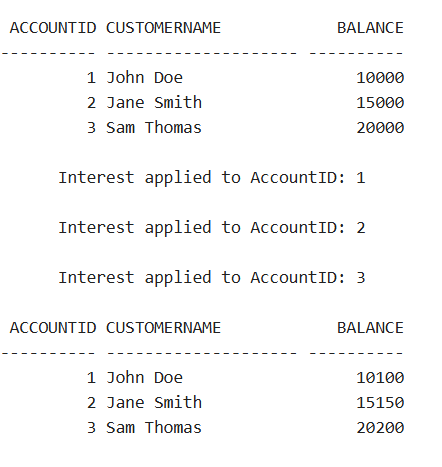
END;

/

SELECT\*FROM Accounts;

EXEC ProcessMonthlyInterest;

**OUTPUT SCREENSHOTS :**

****

**SCENARIO 2 :**

CREATE OR REPLACE PROCEDURE **UpdateEmployeeBonus**(

dept\_name IN VARCHAR2,

bonus\_percent IN NUMBER) IS

BEGIN

FOR emp IN (SELECT EmployeeID, Salary FROM Employees

WHERE Department = dept\_name)

LOOP

UPDATE Employees

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

WHERE EmployeeID = emp.EmployeeID;

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

END LOOP;

END;

/

SELECT\*FROM Employees;

EXEC UpdateEmployeeBonus('IT', 10);

**OUTPUT SCREENSHOTS :**

****

**SCENARIO – 3 :**

CREATE OR REPLACE PROCEDURE TransferFunds(

from\_account IN NUMBER,

to\_account IN NUMBER,

amount IN NUMBER) IS

insufficient\_balance EXCEPTION;

BEGIN

DECLARE

from\_balance NUMBER;

BEGIN

SELECT Balance INTO from\_balance FROM Accounts WHERE AccountID = from\_account;

IF from\_balance < amount THEN

RAISE insufficient\_balance;

END IF;

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountID = from\_account;

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = to\_account;

DBMS\_OUTPUT.PUT\_LINE('

Transferred ' || amount || ' from ' || from\_account || ' to ' || to\_account);

COMMIT;

END;

EXCEPTION

WHEN insufficient\_balance THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient balance in AccountID ' || from\_account);

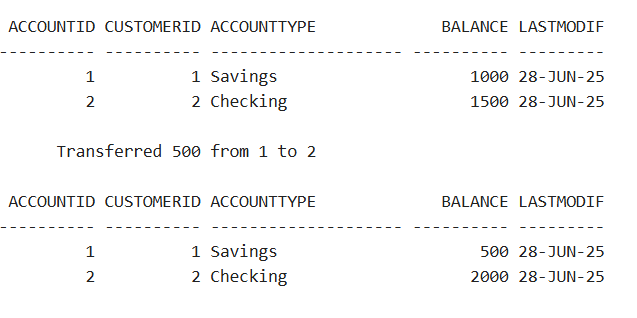
END;

/

SELECT\*FROM Accounts;

EXEC TransferFunds(1, 2, 500);

**OUTPUT SCREENSHOTS :**

****

**TDD and Logging Frameworks**

**Exercise 1: Setting Up Junit**

* Updated pom.xml file by adding the dependency tag.

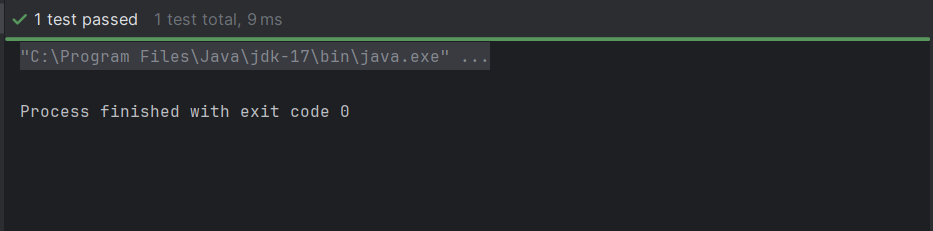
**FILENAME:** Calculator.java

package com.task2;  
  
public class Calculator {  
 public int add(int a, int b) {  
 return a + b;  
 }  
}

**FILENAME :** CalculatorTest.java

package com.task2;  
  
import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class CalculatorTest {  
 @Test  
 public void testAddition() {  
 Calculator calc = new Calculator();  
 int result = calc.add(2, 3);  
 *assertEquals*(5, result);  
 }  
}

**OUTPUT SCREENSHOTS :**



**Exercise 3: Assertions In Junit**

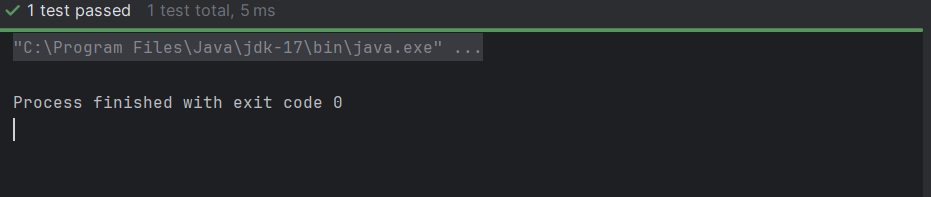
**FILENAME** : AssertionsTest

package com.task2;

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

public class AssertionsTest {  
  
 @Test  
 public void testAssertions() {  
  
 // Check if two values are equal  
 *assertEquals*(5, 2 + 3);  
  
 // Check if a condition is true  
 *assertTrue*(5 > 3);  
  
 // Check if a condition is false  
 *assertFalse*(5 < 3);  
  
 // Check if an object is null  
 *assertNull*(null);  
  
 // Check if an object is not null  
 *assertNotNull*(new Object());  
 }  
}

**OUTPUT SCREENSHOTS :**



**Exercise 4: Arrange-Act-Assert (Aaa) Pattern, Test Fixtures, Setup And Teardown Methods In Junit**

* Updated pom.xml file by adding the dependency tag.

**FILENAME** : Calculator.java

package com.task2;  
  
public class Calculator {  
 public int add(int a, int b) {  
 return a + b;  
 }  
  
 public int subtract(int a, int b) {  
 return a - b;  
 }  
}

**FILENAME** : CalculatorTest.java

package com.task2;

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

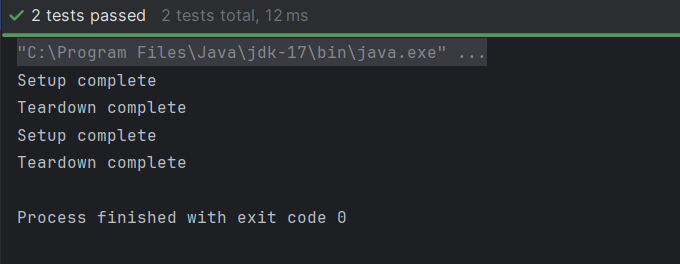
public class CalculatorTest {  
 private Calculator calculator;  
 @Before  
 public void setUp() {  
 calculator = new Calculator();  
 System.*out*.println("Setup complete");  
 }

@After  
 public void tearDown() {  
 calculator = null;  
 System.*out*.println("Teardown complete");  
 }

@Test  
 public void testAddition() {  
 int result = calculator.add(2, 3);  
 *assertEquals*(5, result);  
 }

@Test  
 public void testSubtraction() {  
 int result = calculator.subtract(5, 2);  
 *assertEquals*(3, result);  
 }  
}

**OUTPUT SCREENSHOTS :**

****

**MOCKITO EXERCISES**

**Exercise 1: Mocking And Stubbing**

* Updated pom.xml by adding dependency in it as below;

**FILENAME :** ExternalApi.java

package com.task2;  
public interface ExternalApi {  
 String getData();  
}

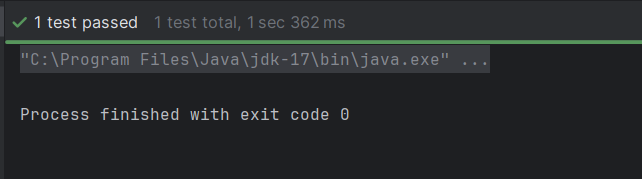
**FILENAME :** MyServiceTest.java

package com.task2;

import org.junit.Test;   
import static org.junit.Assert.\*;   
import static org.mockito.Mockito.\*;

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);   
 }  
}

**OUTPUT SCREENSHOTS :**

****

**Exercise 2: Verifying Interactions**

* Updated pom.xml by adding Dependency tag.

**FILENAME**: pom.xml(**Added tags)**

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>4.11.0</version>

<scope>test</scope>

</dependency>

</dependencies>

**FILENAME :** ExternalApi.java

package com.task2;

public interface ExternalApi

{

String getData();  
}

**FILENAME :** MyService.java

package com.task2;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {  
 this.api = api;  
 }

public String fetchData() {  
 return api.getData(); // calling the external API  
 }  
}

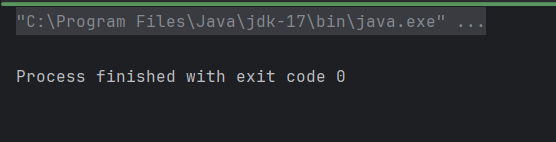
**FILENAME :** MyServiceTest.java

package com.task2;  
import org.junit.Test;   
import static org.mockito.Mockito.\*;   
public class MyServiceTest {  
 @Test  
 public void testExternalApi() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 *when*(mockApi.getData()).thenReturn("Mock Data");  
 MyService service = new MyService(mockApi);  
 Service.fetchData();  
 *verify*(mockApi).getData();

*verify*(mockApi, *times*(1)).getData();

}  
}

**OUTPUT SCREENSHOTS :**

****

**SL4J Logging Exercise**

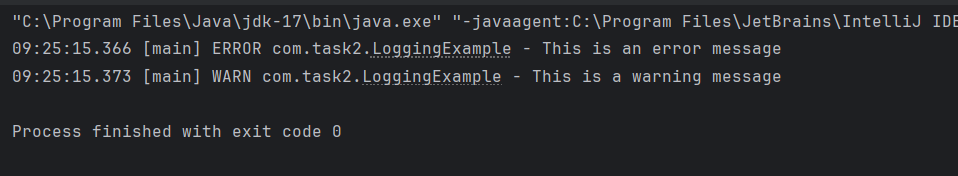
**Exercise – 1 : Logging error Messages and Warning Levels**

**FILENAME :** LoggingExample.java

package com.task2;  
  
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 SCREENSHOTS**

****