

WEEK 2:

PLSQL Exercises

Exercise 1: Control Structures

Scenario 1: The bank wants to apply a discount to loan interest rates for customers above 60 years old.

Question: Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

SOLUTION:

```
CREATE TABLE Customers (
    CustomerID NUMBER,
    Name VARCHAR2(50),
    Age NUMBER,
    Balance NUMBER,
    IsVIP VARCHAR2(5)
);

CREATE TABLE Loans (
    LoanID NUMBER,
    CustomerID NUMBER,
    InterestRate NUMBER,
    DueDate DATE
);

INSERT INTO Customers VALUES (1, 'Alice', 65, 12000, 'FALSE');
INSERT INTO Customers VALUES (2, 'Bob', 58, 9000, 'FALSE');
INSERT INTO Customers VALUES (3, 'Charlie', 70, 15000, 'FALSE');

INSERT INTO Loans VALUES (101, 1, 8.5, SYSDATE + 10);
INSERT INTO Loans VALUES (102, 2, 9.0, SYSDATE + 40);
INSERT INTO Loans VALUES (103, 3, 7.5, SYSDATE + 5);

COMMIT;

BEGIN
    FOR customer_rec IN (
        SELECT CustomerID
        FROM Customers
        WHERE Age > 60
    ) LOOP

        UPDATE Loans
        SET InterestRate = InterestRate - 1
        WHERE CustomerID = customer_rec.CustomerID;

        DBMS_OUTPUT.PUT_LINE('Discount applied for Customer ID: ' || customer_rec.CustomerID);
    END LOOP;
    COMMIT;
END;
```

The screenshot shows the Oracle Live SQL interface. In the Navigator panel, 'CUSTOMERS' and 'LOANS' are selected. The SQL Worksheet contains the following PL/SQL code:

```

29   FROM Customers
30   WHERE Age > 60
31   ) LOOP
32
33   UPDATE Loans
34   SET InterestRate = InterestRate - 1
35   WHERE CustomerID = customer_rec.CustomerID;
36
37   DBMS_OUTPUT.PUT_LINE('Discount applied for Customer ID: ' || customer_rec.CustomerID);
38
39   END LOOP;
40   COMMIT;
41
42 /

```

The Script output tab shows the results of the execution:

```

Discount applied for Customer ID: 1
Discount applied for Customer ID: 3

PL/SQL procedure successfully completed.
Elapsed: 00:00:00.088

```

Scenario 2: A customer can be promoted to VIP status based on their balance.

Question: Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over \$10,000.

```

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

  DBMS_OUTPUT.PUT_LINE('Customer ' || vip_rec.CustomerID || ' is now a VIP.');
END LOOP;
COMMIT;
END;

```

The screenshot shows the Oracle Live SQL interface. In the Navigator panel, 'CUSTOMERS' and 'LOANS' are selected. The SQL Worksheet contains the following PL/SQL code:

```

47   SELECT CustomerID
48   FROM Customers
49   WHERE Balance > 10000
50   ) LOOP
51   UPDATE Customers
52   SET IsVIP = 'TRUE'
53   WHERE CustomerID = vip_rec.CustomerID;
54
55   DBMS_OUTPUT.PUT_LINE('Customer ' || vip_rec.CustomerID || ' is now a VIP.');
56
57   END LOOP;
58   COMMIT;
59
60 /

```

The Script output tab shows the results of the execution:

```

Customer 1 is now a VIP.
Customer 3 is now a VIP.

PL/SQL procedure successfully completed.
Elapsed: 00:00:00.097

```

- **Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

Question: Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

```

BEGIN
FOR loan_rec IN (
  SELECT LoanID, CustomerID, DueDate
  FROM Loans

```

```

        WHERE DueDate <= SYSDATE + 30
    ) LOOP
        DBMS_OUTPUT.PUT_LINE(
            'Reminder: Loan ID ' || loan_rec.LoanID ||
            ' for Customer ID ' || loan_rec.CustomerID ||
            ' is due on ' || TO_CHAR(loan_rec.DueDate, 'DD-MON-YYYY')
        );
    END LOOP;
END;

```

The screenshot shows the Oracle Live SQL interface. On the left, the Navigator pane displays 'My Schema' with 'Tables' expanded, showing 'CUSTOMERS' and 'LOANS'. The main area is a SQL Worksheet containing the following PL/SQL code:

```

62   FOR loan_rec IN (
63     SELECT LoanID, CustomerID, DueDate
64     FROM Loans
65     WHERE DueDate <= SYSDATE + 30
66   ) LOOP
67     DBMS_OUTPUT.PUT_LINE(
68       'Reminder: Loan ID ' || loan_rec.LoanID ||
69       ' for Customer ID ' || loan_rec.CustomerID ||
70       ' is due on ' || TO_CHAR(loan_rec.DueDate, 'DD-MON-YYYY')
71     );
72   END LOOP;
73
74 /
75
76

```

Below the code, the 'Script output' tab is selected, showing the results of the execution:

```

Reminder: Loan ID 103 for Customer ID 3 is due on 30-JUN-2025
Reminder: Loan ID 101 for Customer ID 1 is due on 05-JUL-2025

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.013

```

Exercise 3: Stored Procedures

Scenario 1: The bank needs to process monthly interest for all savings accounts.

Question: Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

```

CREATE TABLE Accounts (
    AccountID NUMBER PRIMARY KEY,
    AccountHolder VARCHAR2(100),
    AccountType VARCHAR2(20), -- e.g., 'Savings', 'Current'
    Balance NUMBER(15, 2)
);

```

```

CREATE TABLE Employees (
    EmployeeID NUMBER PRIMARY KEY,
    Name VARCHAR2(100),
    DepartmentID NUMBER,
    Salary NUMBER(10, 2)
);

```

```

INSERT INTO Accounts VALUES (101, 'A', 'Savings', 10000);
INSERT INTO Accounts VALUES (102, 'B', 'Current', 8000);
INSERT INTO Accounts VALUES (103, 'C', 'Savings', 15000);

```

```

INSERT INTO Employees VALUES (1, 'June', 10, 50000);
INSERT INTO Employees VALUES (2, 'July', 20, 60000);
INSERT INTO Employees VALUES (3, 'Jan', 10, 55000);

```

```

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
    UPDATE Accounts

```

```
SET Balance = Balance + (Balance * 0.01)
```

```
WHERE AccountType = 'Savings';
```

```
COMMIT;
```

```
END;
```

The screenshot shows the Oracle Live SQL interface. On the left, the Navigator pane displays the schema structure with tables: ACCOUNTS, CUSTOMERS, EMPLOYEES, and LOANS. The central area is the SQL Worksheet, containing the following PL/SQL code:

```
36    bonus_pct IN NUMBER
37 ) IS
38 BEGIN
39   UPDATE Employees
40   SET Salary = Salary + (Salary * bonus_pct)
41   WHERE DepartmentID = dept_id;
42
43   COMMIT;
44
45
46
```

The Query result tab shows the output of the executed code, which includes the UPDATE statement and a message indicating the procedure was compiled. The elapsed time is shown as 00:00:00.019.

Scenario 2: The bank wants to implement a bonus scheme for employees based on their performance.

Question: Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
dept_id IN NUMBER,
bonus_pct IN NUMBER
) IS
BEGIN
UPDATE Employees
SET Salary = Salary + (Salary * bonus_pct)
WHERE DepartmentID = dept_id;
```

```
COMMIT;
```

```
END;
```

The screenshot shows the Oracle Live SQL interface. On the left, the Navigator pane displays the schema structure with tables: ACCOUNTS, CUSTOMERS, EMPLOYEES, and LOANS. The central area is the SQL Worksheet, containing the following PL/SQL code:

```
29 WHERE AccountType = 'Savings';
30
31 COMMIT;
32
33
34 CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
35   dept_id IN NUMBER,
36   bonus_pct IN NUMBER
37 ) IS
38 BEGIN
39   UPDATE Employees
40   SET Salary = Salary + (Salary * bonus_pct)
41   WHERE DepartmentID = dept_id;
42
43   COMMIT;
44
45
46
```

The Query result tab shows the output of the executed code, which includes the CREATE OR REPLACE PROCEDURE statement and a message indicating the procedure was compiled. The elapsed time is shown as 00:00:00.019.

Scenario 3: Customers should be able to transfer funds between their accounts.

Question: Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

```

CREATE OR REPLACE PROCEDURE TransferFunds (
from_acc_id IN NUMBER,
to_acc_id IN NUMBER,
amount IN NUMBER
) IS
insufficient_balance EXCEPTION;
BEGIN
-- Check balance
DECLARE
current_balance NUMBER;
BEGIN
SELECT Balance INTO current_balance
FROM Accounts
WHERE AccountID = from_acc_id;

IF current_balance < amount THEN
RAISE insufficient_balance;
END IF;
END;

-- Perform transfer
UPDATE Accounts
SET Balance = Balance - amount
WHERE AccountID = from_acc_id;

UPDATE Accounts
SET Balance = Balance + amount
WHERE AccountID = to_acc_id;

COMMIT;

EXCEPTION
WHEN insufficient_balance THEN
ROLLBACK;
DBMS_OUTPUT.PUT_LINE('X Error: Insufficient balance.');
WHEN OTHERS THEN
ROLLBACK;
DBMS_OUTPUT.PUT_LINE('X Unexpected error: ' || SQLERRM);
END;

```

The screenshot shows the Oracle SQL Developer interface with the 'SQL Worksheet' tab active. The code area displays the creation of the 'TransferFunds' procedure, including its logic for checking account balances and performing transfers, along with exception handling for insufficient funds and other errors. The 'Script output' tab below the code shows the successful compilation of the procedure.

```

77  EXCEPTION
78  WHEN insufficient_balance THEN
79  :::: ROLLBACK;
80  :::: DBMS_OUTPUT.PUT_LINE('X Error: Insufficient balance.');
81  WHEN OTHERS THEN
82  :::: ROLLBACK;
83  :::: DBMS_OUTPUT.PUT_LINE('X Unexpected error: ' || SQLERRM);
84  END;
85
86
Procedure UPDATEREEMPLOYEEBONUS compiled
Elapsed: 00:00:00.019
SQL> CREATE OR REPLACE PROCEDURE TransferFunds (
  from_acc_id IN NUMBER,
  to_acc_id IN NUMBER,
  amount IN NUMBER...
Show more...
Procedure TRANSFERFUNDS compiled
Elapsed: 00:00:00.020

```

JUnit Testing Exercises :

Exercise 1: Setting Up JUnit

Scenario:

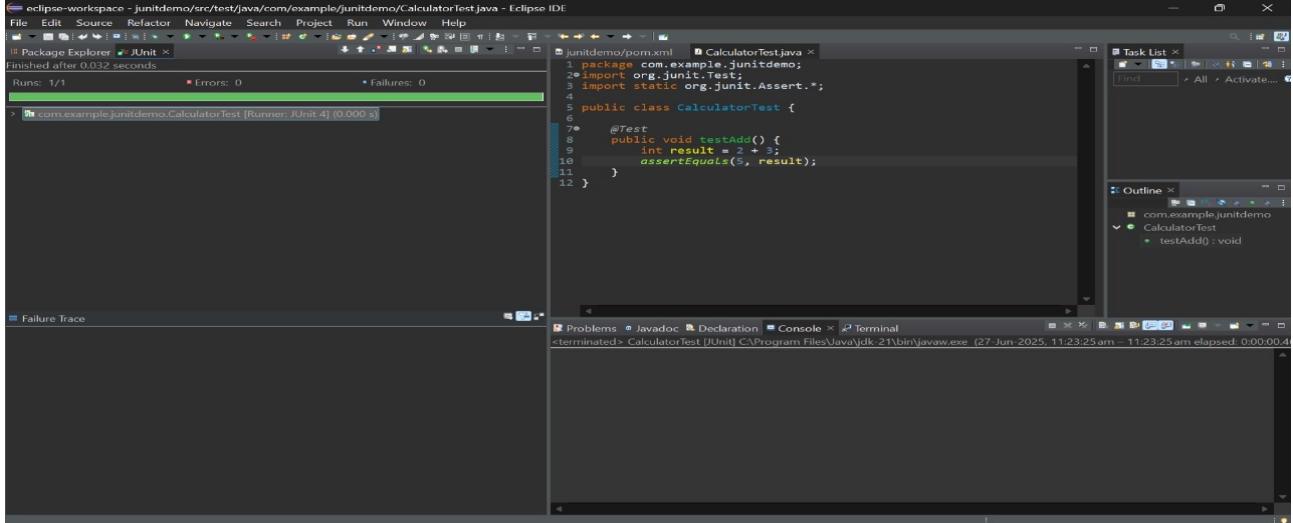
You need to set up JUnit in your Java project to start writing unit tests.

Steps:

1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).
2. Add JUnit dependency to your project. If you are using Maven, add the following to your pom.xml:

```
<dependency> <groupId>junit</groupId> <artifactId>junit</artifactId> <version>4.13.2</version>
<scope>test</scope>
</dependency>
```

3. Create a new test class in your project.



Exercise 3: Assertions in JUnit

Scenario:

You need to use different assertions in JUnit to validate your test results. Steps:

1. Write tests using various JUnit assertions. Solution Code:

```
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()); }

}

```

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** eclipse-workspace - junitdemo/src/test/java/com/example/junitdemo/AssertionsTest.java - Eclipse IDE
- Toolbar:** File Edit Source Refactor Navigate Project Run Window Help
- Package Explorer:** Shows the project structure with JUnit selected.
- Console:** Displays the test results: Runs: 1/1 Errors: 0 Failures: 0. Finished after 0.03 seconds.
- Code Editor:** Contains the `AssertionsTest.java` code, which includes assertions for equality, truth, falsehood, and nullity.
- Outline View:** Shows the class `AssertionsTest` and its method `testAssertions()`.
- Problems View:** Shows no errors or warnings.
- Terminal:** Shows the command-line output of the test execution.

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

Scenario:

You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.

Steps:

1. Write tests using the AAA pattern.
2. Use `@Before` and `@After` annotations for setup and teardown methods.

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** eclipse-workspace - junitdemo/src/test/java/com/example/junitdemo/CalculatorTest.java - Eclipse IDE
- Toolbar:** File Edit Source Refactor Navigate Project Run Window Help
- Package Explorer:** Shows the project structure with JUnit selected.
- Console:** Displays the test results: Runs: 2/2 Errors: 0 Failures: 0. Finished after 0.032 seconds.
- Code Editor:** Contains the `CalculatorTest.java` code, which includes an `@Before` annotation for setup and `@After` annotations for teardown.
- Outline View:** Shows the class `CalculatorTest` and its methods `setUp()`, `tearDown()`, and `testAdd()`.
- Problems View:** Shows no errors or warnings.
- Terminal:** Shows the command-line output of the test execution, including the setup and teardown logs.

Mockito Hands-On Exercises

Exercise 1: Mocking and Stubbing

Scenario:

You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.

Steps:

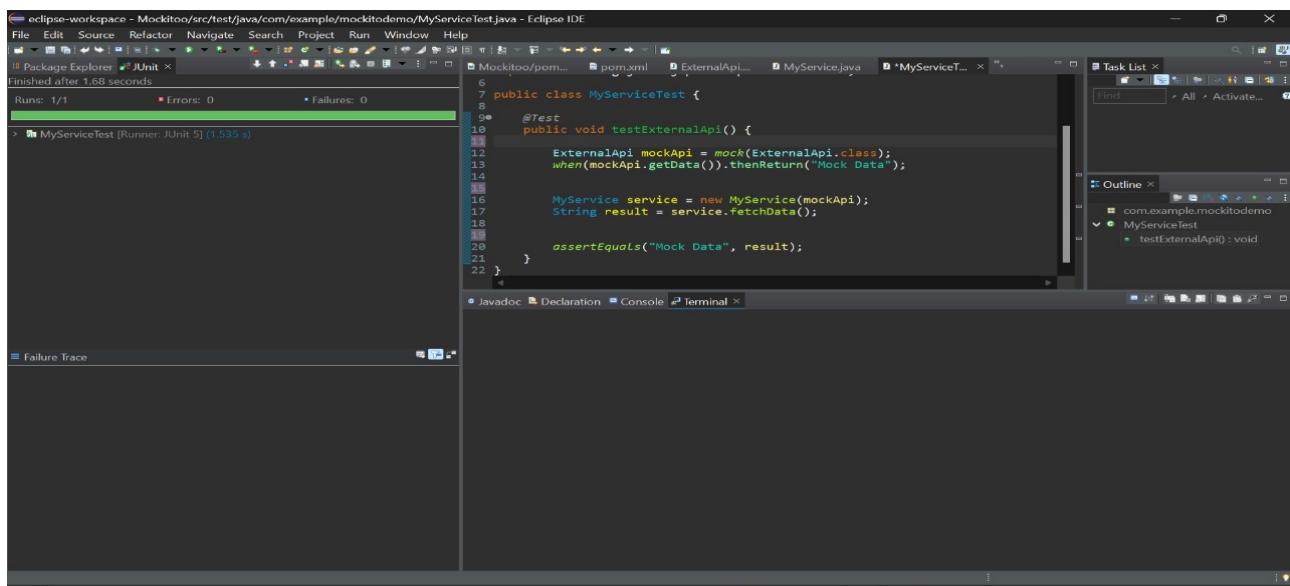
1. Create a mock object for the external API.
2. Stub the methods to return predefined values.
3. Write a test case that uses the mock object.

Solution Code:

```
import static org.mockito.Mockito.*; 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);
    }
}
```



Exercise 2: Verifying Interactions

Scenario:

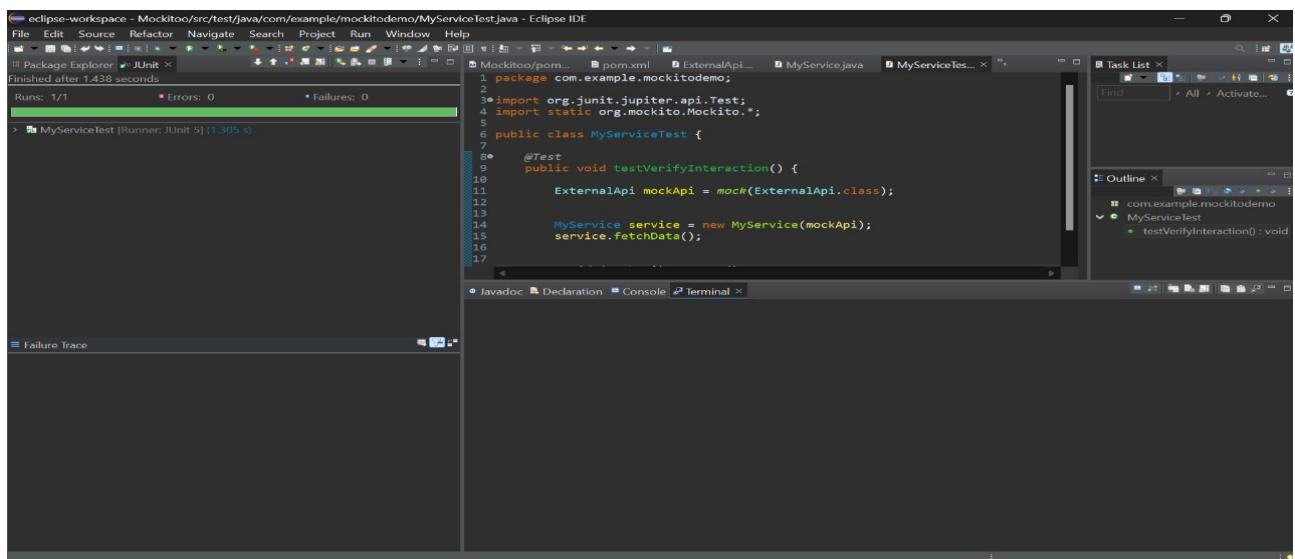
You need to ensure that a method is called with specific arguments.

Steps:

1. Create a mock object.
2. Call the method with specific arguments.
3. Verify the interaction.

Solution Code:

```
import static org.mockito.Mockito.*;  
  
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();  
    } }
```



Logging using SLF4J

Exercise 1: Logging Error Messages and Warning Levels

Task: Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

Step-by-Step Solution:

1. Add SLF4J and Logback dependencies to your `pom.xml` file:

```

<dependency> <groupId>org.slf4j</groupId> <artifactId>slf4j-api</artifactId>
<version>1.7.30</version>

</dependency> <dependency>

<groupId>ch.qos.logback</groupId> <artifactId>logback-classic</artifactId>
<version>1.2.3</version>

</dependency>

```

2. Create a Java class that uses SLF4J for logging:

```

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");
}
}

```

