**Digital Nurture 4.0 Deep Skilling**

**Week 2**

**Exercise 1: Control Structures**

**Code for creating the table :**A screenshot of a computer program

Description automatically generated

**Code for Inserting data into the table :**

A group of numbers and values

Description automatically generated

**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.**

**Code :**

A computer screen shot of a code

Description automatically generated

**Output:**

A white background with black text

Description automatically generated

**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.**

**Code :**

A screenshot of a computer

Description automatically generated

**Output :**

A close-up of a text

Description automatically generated

**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.**

**Code :** A computer screen shot of a computer code

Description automatically generated

**Output :**

A close-up of a logo

Description automatically generated

**Exercise 3: Stored Procedures**

**Code for creating the table :**

A screenshot of a computer program

Description automatically generated

**Code for Inserting data into the table :**

A computer code with text

Description automatically generated with medium confidence

**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.**

**Code :**

A screenshot of a computer

Description automatically generated

**Output :**

A black text on a white background

Description automatically generated

**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.**

**Code :**

A screenshot of a computer

Description automatically generated

**Output :**

A black text on a white background

Description automatically generated

**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.**

**Code :**

A computer screen shot of a code

Description automatically generated

**Output :**

A close up of a text

Description automatically generated

**JUnit Basic Testing**

**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

**Code :**

**Calculator.java**

A screenshot of a computer program

Description automatically generated

**CalculatorTest.java**

A screen shot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Output :**

A screenshot of a computer

Description automatically generated

**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());

}

**Code :**

A screenshot of a computer program

Description automatically generated

**Output :**

A screenshot of a computer program

Description automatically generated

**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.

**Code :**

A screen shot of a computer code

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

**Output :**

A screenshot of a computer

Description automatically generated

**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);

}

**Code :**

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

**Output :**

A screenshot of a computer

Description automatically generated

**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();

**Code :**

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A computer screen shot of a program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Output :**

A screenshot of a computer

Description automatically generated

**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");

}

}

**Code :**

A computer screen shot of a program code

Description automatically generated

**Output :**

A screen shot of a computer

Description automatically generated