**WEEK 2:**

**Module 3 - PL/SQL Programming**

**Exercise 1: Control Structures**

Write a PL/SQL block to check if a number is positive, negative, or zero using IF-ELSIF-ELSE, and if positive, print numbers from 1 to that number using a LOOP.

PL/SQL Query:

DECLARE

num NUMBER := 4;

i NUMBER := 1;

BEGIN

IF num > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Positive Number');

LOOP

EXIT WHEN i > num;

DBMS\_OUTPUT.PUT\_LINE('Number: ' || i);

i := i + 1;

END LOOP;

ELSIF num < 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Negative Number');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Number is Zero');

END IF;

END;

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 3: Stored Procedures (PL/SQL)**

Write a stored procedure in PL/SQL that accepts an employee's name and salary, and displays whether the employee is eligible for a bonus (salary ≥ 50000).

PL/SQL Query:

CREATE OR REPLACE PROCEDURE check\_bonus (

emp\_name IN VARCHAR2,

emp\_salary IN NUMBER

) AS

BEGIN

IF emp\_salary >= 50000 THEN

DBMS\_OUTPUT.PUT\_LINE(emp\_name || ' is eligible for a bonus.');

ELSE

DBMS\_OUTPUT.PUT\_LINE(emp\_name || ' is NOT eligible for a bonus.');

END IF;

END;

/

BEGIN

check\_bonus('John Doe', 55000);

END;

/

**OUTPUT:**

A black text on a white background

AI-generated content may be incorrect.

**Module 4 – Test driven development and Logging framework**

**Exercise 1: Setting Up JUnit**

Add Junit jar files:

A screenshot of a computer

AI-generated content may be incorrect.

**Code:**

package com.example.junit;

public class PrimeChecker {

public boolean isPrime(int number) {

if (number <= 1) return false;

for (int i = 2; i <= Math.sqrt(number); i++) {

if (number % i == 0) return false;

}

return true;

}

}

package com.example.junit;

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

import org.junit.jupiter.api.Test;

public class PrimeCheckerTest {

@Test

public void testIsPrime() {

PrimeChecker checker = new PrimeChecker();

assertTrue(checker.isPrime(2));

assertTrue(checker.isPrime(3));

assertTrue(checker.isPrime(13));

assertFalse(checker.isPrime(1));

assertFalse(checker.isPrime(0));

assertFalse(checker.isPrime(4));

assertFalse(checker.isPrime(100));

}

}

**OUTPUT:**

A screenshot of a computer error message

AI-generated content may be incorrect.

**Exercise 3: Assertions in Junit**

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;

}

public int multiply(int a, int b) {

return a \* b;

}

public int divide(int a, int b) {

if (b == 0) throw new IllegalArgumentException("Cannot divide by zero");

return a / b;

}

}

package com.example.junit;

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

import org.junit.jupiter.api.Test;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

assertEquals(5, calc.add(2, 3));

assertNotEquals(6, calc.add(2, 3));

}

@Test

public void testSubtract() {

Calculator calc = new Calculator();

assertEquals(1, calc.subtract(3, 2));

}

@Test

public void testMultiply() {

Calculator calc = new Calculator();

assertEquals(6, calc.multiply(2, 3));

}

@Test

public void testDivide() {

Calculator calc = new Calculator();

assertEquals(2, calc.divide(6, 3));

assertThrows(IllegalArgumentException.class, () -> calc.divide(5, 0));

}

}

**OUTPUT:**

A screenshot of a computer error message

AI-generated content may be incorrect.

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

Code:

package com.example.junit;

public class BankAccount {

private double balance;

public BankAccount(double initialBalance) {

this.balance = initialBalance;

}

public void deposit(double amount) {

if (amount > 0) balance += amount;

}

public void withdraw(double amount) {

if (amount > 0 && amount <= balance) balance -= amount;

}

public double getBalance() {

return balance;

}

}

package com.example.junit;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.Test;

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

public class BankAccountTest {

private BankAccount account;

@BeforeEach

public void setUp() {

account = new BankAccount(1000.0); // Arrange

System.out.println("BeforeEach - account created with 1000.0 balance");

}

@AfterEach

public void tearDown() {

account = null;

System.out.println("AfterEach - account set to null");

}

@Test

public void testDeposit() {

account.deposit(500.0);

assertEquals(1500.0, account.getBalance());

}

@Test

public void testWithdraw() {

account.withdraw(300.0);

assertEquals(700.0, account.getBalance());

}

@Test

public void testOverdrawNotAllowed() {

account.withdraw(2000.0);

assertEquals(1000.0, account.getBalance(), "Overdrawing should not be allowed");

}

}

**OUTPUT:**

A screenshot of a computer error message

AI-generated content may be incorrect.

**Exercise 1: Mocking and Stubbing**  
Code:

package com.example.junit;

public interface UserRepository {

String getUserNameById(int id);

}

package com.example.junit;

public class UserService {

private UserRepository repo;

public UserService(UserRepository repo) {

this.repo = repo;

}

public String getWelcomeMessage(int id) {

String name = repo.getUserNameById(id);

return "Welcome, " + name + "!";

}

}

package com.example.junit;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

public class UserServiceTest {

@Test

public void testWelcomeMessage() {

UserRepository mockRepo = mock(UserRepository.class);

when(mockRepo.getUserNameById(101)).thenReturn("Alice");

UserService service = new UserService(mockRepo);

String result = service.getWelcomeMessage(101);

assertEquals("Welcome, Alice!", result);

verify(mockRepo).getUserNameById(101);

System.out.println("Result = " + result);

}

}

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer error message

AI-generated content may be incorrect.

**Exercise 2: Verifying Interactions**  
Code:

package com.example.junit;

public interface Inventory {

boolean checkStock(String item);

void removeItem(String item);

}

package com.example.junit;

public class OrderService {

private Inventory inventory;

public OrderService(Inventory inventory) {

this.inventory = inventory;

}

public boolean placeOrder(String item) {

if (inventory.checkStock(item)) {

inventory.removeItem(item);

return true;

}

return false;

}

}

package com.example.junit;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

public class OrderServiceTest {

@Test

public void testPlaceOrderInteraction() {

Inventory mockInventory = mock(Inventory.class);

when(mockInventory.checkStock("apple")).thenReturn(true);

OrderService service = new OrderService(mockInventory);

boolean result = service.placeOrder("apple");

assertTrue(result);

verify(mockInventory).checkStock("apple"); // was called once

verify(mockInventory).removeItem("apple"); // was called once

verify(mockInventory, times(1)).removeItem("apple"); // explicit

verify(mockInventory, never()).removeItem("banana"); // never called

}

}

**OUTPUT:**

A screenshot of a computer error message

AI-generated content may be incorrect.