**Superset ID:** **6403222**

**MODULE 2 SOLUTIONS**

**PL/SQL programming**

**Exercise 1: Control Structures**

DECLARE

v\_name VARCHAR2(50) := 'John';

v\_mark1 NUMBER := 85;

v\_mark2 NUMBER := 74;

v\_mark3 NUMBER := 92;

v\_total NUMBER;

v\_average NUMBER;

v\_grade CHAR(1);

BEGIN

v\_total := v\_mark1 + v\_mark2 + v\_mark3;

v\_average := v\_total / 3;

IF v\_average >= 90 THEN

v\_grade := 'A';

ELSIF v\_average >= 80 THEN

v\_grade := 'B';

ELSIF v\_average >= 70 THEN

v\_grade := 'C';

ELSIF v\_average >= 60 THEN

v\_grade := 'D';

ELSE

v\_grade := 'F';

END IF;

DBMS\_OUTPUT.PUT\_LINE('Student Name: ' || v\_name);

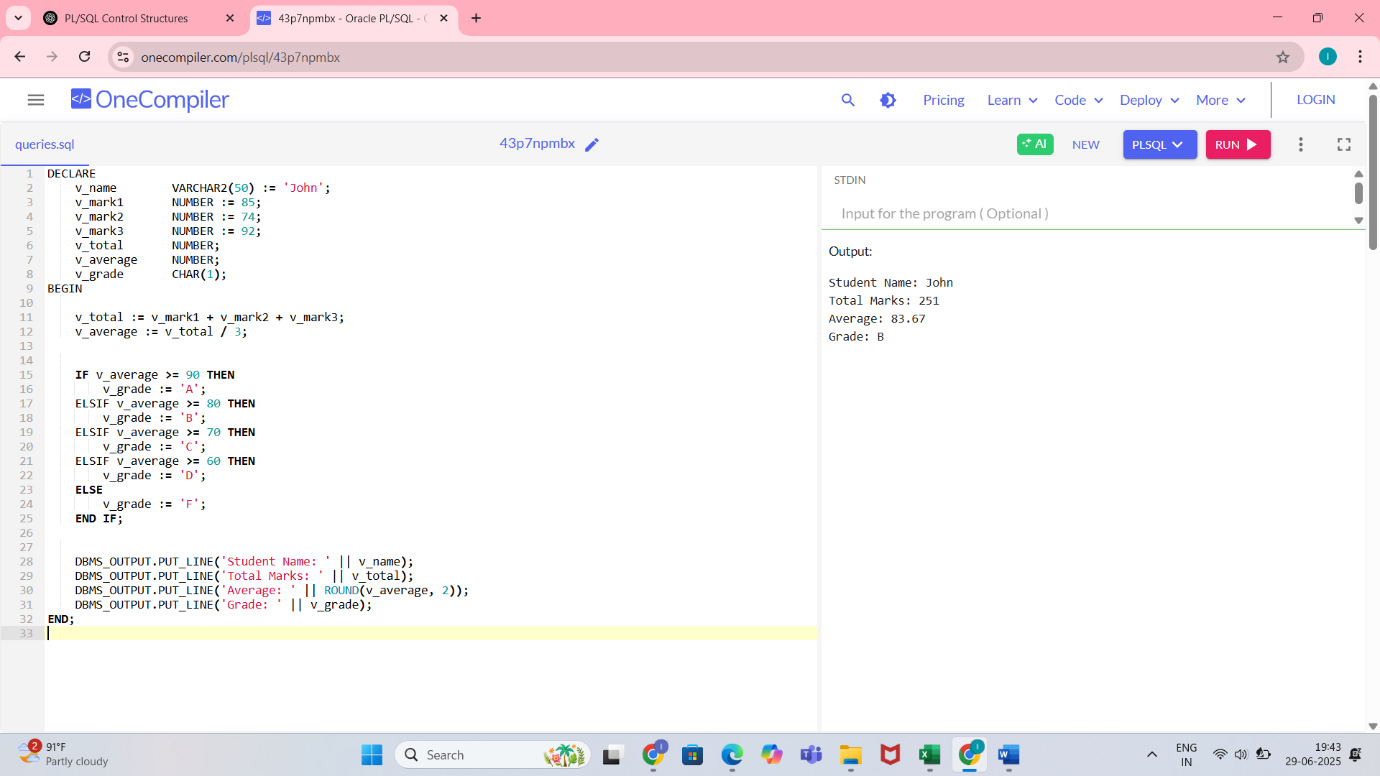
DBMS\_OUTPUT.PUT\_LINE('Total Marks: ' || v\_total);

DBMS\_OUTPUT.PUT\_LINE('Average: ' || ROUND(v\_average, 2));

DBMS\_OUTPUT.PUT\_LINE('Grade: ' || v\_grade);

END;

**OUTPUT:**



**Exercise 3: Stored Procedures**

CREATE TABLE employees (

emp\_id NUMBER PRIMARY KEY,

emp\_name VARCHAR2(100),

salary NUMBER

);

INSERT INTO employees VALUES (101, 'John', 50000);

COMMIT;

CREATE OR REPLACE PROCEDURE update\_salary\_with\_bonus (

p\_emp\_id IN NUMBER,

p\_bonus\_percent IN NUMBER,

p\_new\_salary OUT NUMBER

)

IS

v\_current\_salary NUMBER;

BEGIN

SELECT salary INTO v\_current\_salary

FROM employees

WHERE emp\_id = p\_emp\_id;

p\_new\_salary := v\_current\_salary + (v\_current\_salary \* p\_bonus\_percent / 100);

UPDATE employees

SET salary = p\_new\_salary

WHERE emp\_id = p\_emp\_id;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully. New salary: ' || p\_new\_salary);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Employee not found.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

/

DECLARE

v\_new\_salary NUMBER;

BEGIN

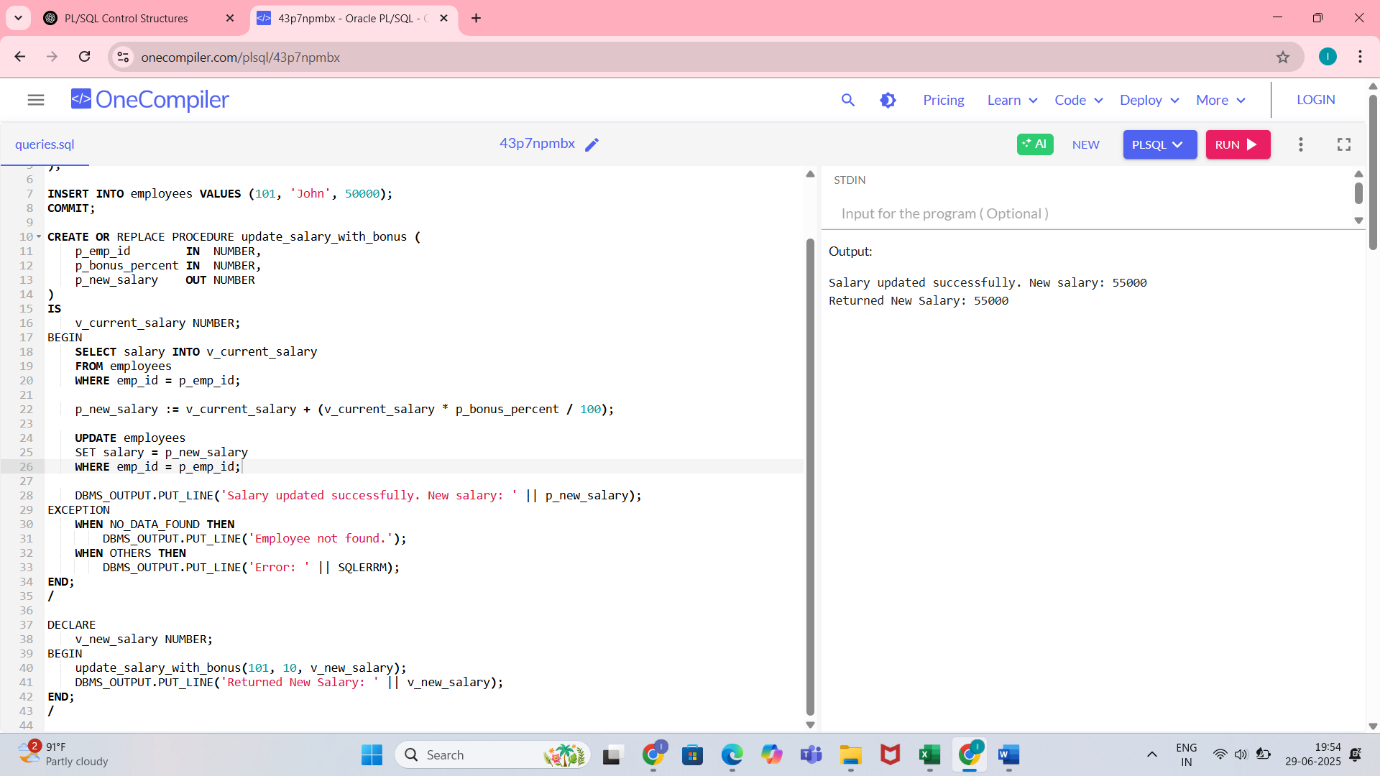
update\_salary\_with\_bonus(101, 10, v\_new\_salary);

DBMS\_OUTPUT.PUT\_LINE('Returned New Salary: ' || v\_new\_salary);

END;

/

**OUTPUT:**

****

**TDD using JUnit5 and Mockito  
  
Exercise 1: Setting Up JUnit**

public class MyMath {

public int add(int a, int b) {

return a + b;

}

public int max(int a, int b) {

return (a > b) ? a : b;

}

public boolean isEven(int n) {

return n % 2 == 0;

}

}

// MyMathTest.java

import org.junit.jupiter.api.Test;

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

public class MyMathTest {

@Test

public void testAdd() {

MyMath math = new MyMath();

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

}

@Test

public void testMax() {

MyMath math = new MyMath();

assertEquals(7, math.max(5, 7));

assertEquals(10, math.max(10, 1));

}

@Test

public void testIsEven() {

MyMath math = new MyMath();

assertTrue(math.isEven(4));

assertFalse(math.isEven(3));

}

}

**Exercise 3: Assertions in JUnit**

// File: Calculator.java

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public boolean isEven(int number) {

return number % 2 == 0;

}

public String greet(String name) {

return "Hello, " + name;

}

}

// File: CalculatorTest.java

import org.junit.jupiter.api.Test;

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

public class CalculatorTest {

Calculator calculator = new Calculator();

@Test

void testAdd() {

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

}

@Test

void testIsEvenTrue() {

assertTrue(calculator.isEven(4), "4 should be even");

}

@Test

void testIsEvenFalse() {

assertFalse(calculator.isEven(5), "5 should not be even");

}

@Test

void testGreet() {

assertEquals("Hello, Indhu", calculator.greet("Indhu"), "Greeting should be correct");

}

@Test

void testNotNullGreeting() {

assertNotNull(calculator.greet("Indhu"), "Greeting should not be null");

}

}

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

public class BankAccount {

private int balance;

public BankAccount(int initialBalance) {

this.balance = initialBalance;

}

public void deposit(int amount) {

if (amount > 0) {

balance += amount;

}

}

public int getBalance() {

return balance;

}

}

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;

// Setup before each test

@BeforeEach

void setUp() {

account = new BankAccount(100); // Arrange

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

}

// Teardown after each test

@AfterEach

void tearDown() {

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

}

@Test

void testDeposit() {

// Act

account.deposit(50);

// Assert

assertEquals(150, account.getBalance(), "Balance should be 150 after depositing 50");

}

@Test

void testInitialBalance() {

// Act

int initial = account.getBalance();

// Assert

assertEquals(100, initial, "Initial balance should be 100");

}

@Test

void testInvalidDeposit() {

// Act

account.deposit(-10);

// Assert

assertEquals(100, account.getBalance(), "Balance should not change on invalid deposit");

}

}

**3. Mockito exercises**

**Exercise 1: Mocking and Stubbing**

public class User {

private String username;

public User(String username) {

this.username = username;

}

public String getUsername() {

return username;

}

}

public interface UserRepository {

User findUserByUsername(String username);

}

public class UserService {

private UserRepository userRepository;

public UserService(UserRepository repo) {

this.userRepository = repo;

}

public boolean userExists(String username) {

return userRepository.findUserByUsername(username) != null;

}

}

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class UserServiceTest {

private UserRepository mockRepo;

private UserService userService;

@BeforeEach

void setup() {

mockRepo = Mockito.mock(UserRepository.class); // 🧪 Mocking

userService = new UserService(mockRepo);

}

@Test

void testUserExists\_whenUserFound() {

// 🧪 Stubbing

when(mockRepo.findUserByUsername("john")).thenReturn(new User("john"));

boolean exists = userService.userExists("john");

assertTrue(exists, "User should exist");

}

@Test

void testUserExists\_whenUserNotFound() {

when(mockRepo.findUserByUsername("jane")).thenReturn(null);

boolean exists = userService.userExists("jane");

assertFalse(exists, "User should not exist");

}

}

**Exercise 2: Verifying Interactions**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class UserServiceTest {

private UserRepository mockRepo;

private UserService userService;

@BeforeEach

void setup() {

mockRepo = Mockito.mock(UserRepository.class);

userService = new UserService(mockRepo);

}

@Test

void testUserExists\_VerifyMethodCalled() {

// Arrange

when(mockRepo.findUserByUsername("alice")).thenReturn(new User("alice"));

// Act

boolean result = userService.userExists("alice");

// Assert

assertTrue(result);

verify(mockRepo).findUserByUsername("alice"); // ✅ Verifying interaction

}

@Test

void testUserDoesNotExist\_VerifyMethodCalledOnce() {

when(mockRepo.findUserByUsername("bob")).thenReturn(null);

userService.userExists("bob");

verify(mockRepo, times(1)).findUserByUsername("bob"); // ✅ Exactly once

}

@Test

void testUserExists\_NeverCalledWithOtherName() {

when(mockRepo.findUserByUsername("charlie")).thenReturn(new User("charlie"));

userService.userExists("charlie");

verify(mockRepo, never()).findUserByUsername("wrongName");

}

}

**SLF4J logging framework**

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

<dependencies>

<!-- SLF4J API -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>2.0.9</version>

</dependency>

<!-- Logback (backend for SLF4J) -->

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.4.11</version>

</dependency>

</dependencies>

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingDemo {

// Create a Logger instance for this class

private static final Logger logger = LoggerFactory.getLogger(LoggingDemo.class);

public static void main(String[] args) {

int[] numbers = {1, 2, 3};

try {

int value = numbers[5]; // Out-of-bounds to trigger error

} catch (ArrayIndexOutOfBoundsException e) {

logger.error("An error occurred while accessing the array: {}", e.toString());

}

String filePath = "data/config.yaml";

boolean fileFound = false;

if (!fileFound) {

logger.warn("Configuration file not found at path: {}", filePath);

}

logger.info("Application is running...");

}

}

OUTPUT:

22:15:14.123 [main] ERROR LoggingDemo - An error occurred while accessing the array: java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3

22:15:14.126 [main] WARN LoggingDemo - Configuration file not found at path: data/config.yaml

22:15:14.127 [main] INFO LoggingDemo - Application is running...