**MANDATORY HANDS-ON**

**WEEK-2**

**PL/SQL PROGRAMMING**

**Exercise 1: Control Structures**

**Scenario 1:**

**Sql code:**

BEGIN

FOR rec IN (

SELECT c.CustomerID, c.DOB, l.LoanID, l.InterestRate

FROM Customers c JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE c.DOB IS NOT NULL

) LOOP

IF MONTHS\_BETWEEN(SYSDATE, rec.DOB) / 12 > 60 AND rec.InterestRate > 1 THEN

UPDATE Loans SET InterestRate = InterestRate - 1 WHERE LoanID = rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('1% discount applied to Customer ID: ' || rec.CustomerID);

END IF;

END LOOP;

COMMIT;

END;

/

**Scenario 2:**

**Sql code:**

BEGIN

FOR rec IN (SELECT CustomerID, Balance FROM Customers) LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers SET IsVIP = 'TRUE' WHERE CustomerID = rec.CustomerID;

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

ELSE

UPDATE Customers SET IsVIP = 'FALSE' WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

**Scenario 3:**

**Sql code:**

BEGIN

UPDATE Loans SET EndDate = SYSDATE + 10 WHERE LoanID = 1;

COMMIT;

FOR rec IN (

SELECT c.Name, l.LoanID, l.EndDate

FROM Loans l JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.LoanID || ' for customer ' || rec.Name || ' is due on ' || TO\_CHAR(rec.EndDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

**Outputs:**



**Exercise 3: Stored Procedures**

**Scenario 1:**

**Sql code:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

) LOOP

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest added to Account ID ' || acc.AccountID || '. New Balance: ' || (acc.Balance + (acc.Balance \* 0.01)));

END LOOP;

COMMIT;

END;

/

BEGIN

ProcessMonthlyInterest;

END;

/

**Scenario 2:**

**Sql code:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

dept IN VARCHAR2,

bonus\_pct IN NUMBER

) IS

BEGIN

FOR emp IN (

SELECT EmployeeID, Salary FROM Employees WHERE Department = dept

) LOOP

UPDATE Employees

SET Salary = Salary + (Salary \* bonus\_pct / 100)

WHERE EmployeeID = emp.EmployeeID;

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

END LOOP;

COMMIT;

END;

/

BEGIN

UpdateEmployeeBonus('IT', 10);

END;

/

**Scenario 3:**

**Sql code:**

CREATE OR REPLACE PROCEDURE TransferFunds(

fromAcc IN NUMBER,

toAcc IN NUMBER,

amount IN NUMBER

) IS

fromBal NUMBER;

BEGIN

SELECT Balance INTO fromBal FROM Accounts WHERE AccountID = fromAcc;

IF fromBal >= amount THEN

UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = fromAcc;

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful from ' || fromAcc || ' to ' || toAcc || ' for amount ' || amount);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in account ' || fromAcc);

END IF;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

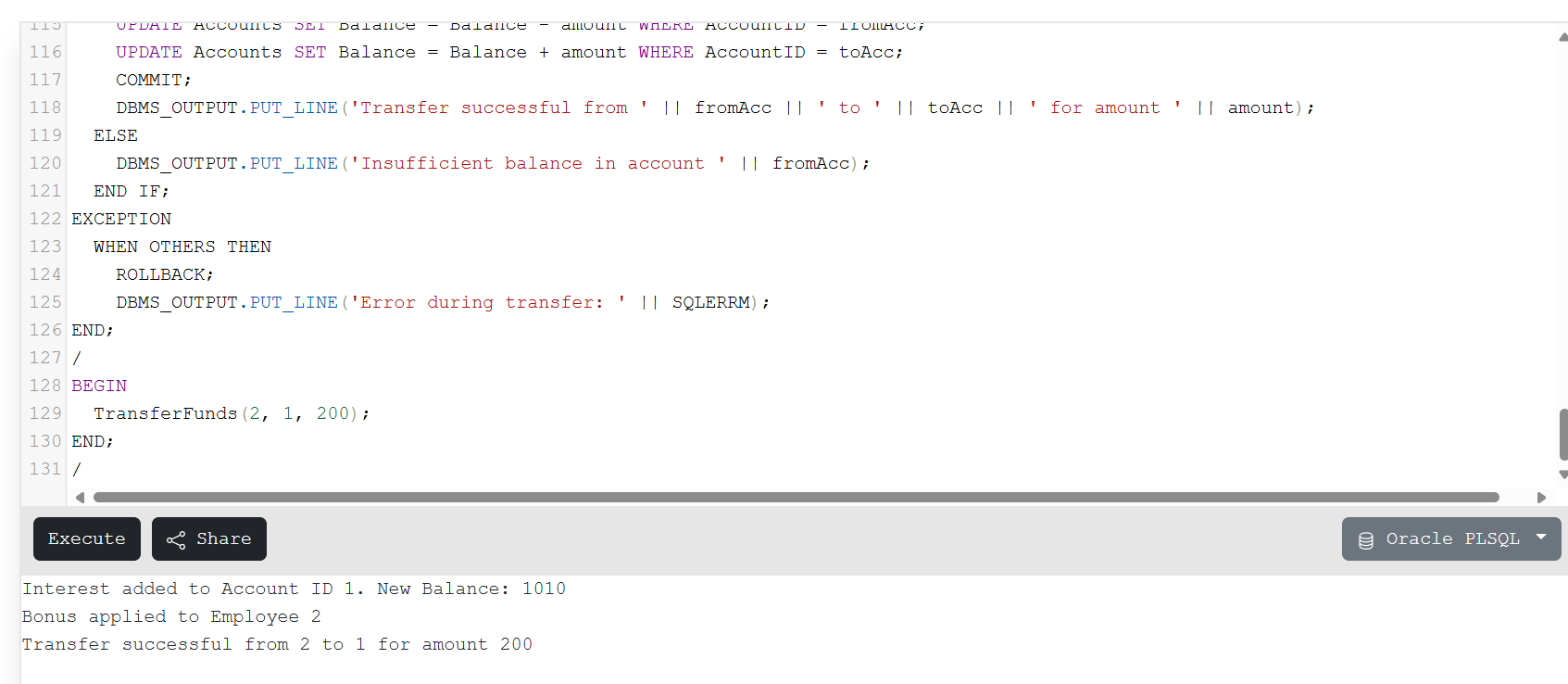
BEGIN

TransferFunds(2, 1, 200);

END;

/

**Outputs:**

****

**PL/SQL PROGRAMMING**

**(Other Hands-on exercises of pl/sql programming)**

**Exercise 2: Error Handling**

**Scenario 1:**

**Sql code:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

fromAcc IN NUMBER,

toAcc IN NUMBER,

amount IN NUMBER

) IS

fromBal NUMBER;

BEGIN

SELECT Balance INTO fromBal FROM Accounts WHERE AccountID = fromAcc;

IF fromBal < amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds');

END IF;

UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = fromAcc;

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: One of the account IDs not found.');

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

BEGIN

SafeTransferFunds(1, 2, 500);

END;

/

**Scenario 2:**

**Sql code:**

CREATE OR REPLACE PROCEDURE UpdateSalary(

empID IN NUMBER,

percent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* percent / 100)

WHERE EmployeeID = empID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID ' || empID || ' not found.');

ELSE

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated for Employee ID: ' || empID);

END IF;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

BEGIN

UpdateSalary(2, 10);

END;

/

**Scenario 3:**

**Sql code:**

CREATE OR REPLACE PROCEDURE AddNewCustomer(

id IN NUMBER,

name IN VARCHAR2,

dob IN DATE,

balance IN NUMBER

) IS

BEGIN

INSERT INTO Customers(CustomerID, Name, DOB, Balance, LastModified, IsVIP)

VALUES (id, name, dob, balance, SYSDATE, 'FALSE');

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully.');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID already exists.');

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

BEGIN

AddNewCustomer(3, 'Ravi Kumar', TO\_DATE('1990-01-01', 'YYYY-MM-DD'), 4000);

END;

/

**Outputs:**



**Exercise 4: Functions**

**Scenario 1:**

**Sql code:**

CREATE OR REPLACE FUNCTION CalculateAge(dob DATE) RETURN NUMBER IS

BEGIN

RETURN TRUNC(MONTHS\_BETWEEN(SYSDATE, dob)/12);

END;

/

SELECT CalculateAge(TO\_DATE('1985-05-15','YYYY-MM-DD')) AS Age FROM dual;

**Scenario 2:**

**Sql code:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

loanAmount IN NUMBER,

interestRate IN NUMBER,

years IN NUMBER

) RETURN NUMBER IS

monthlyRate NUMBER := interestRate / 1200;

n NUMBER := years \* 12;

BEGIN

RETURN loanAmount \* monthlyRate / (1 - POWER(1 + monthlyRate, -n));

END;

/

SELECT CalculateMonthlyInstallment(10000, 6, 2) AS EMI FROM dual;

**Scenario 3:**

**Sql code:**

CREATE OR REPLACE FUNCTION HasSufficientBalance(

accID IN NUMBER,

amount IN NUMBER

) RETURN BOOLEAN IS

accBal NUMBER;

BEGIN

SELECT Balance INTO accBal FROM Accounts WHERE AccountID = accID;

RETURN accBal >= amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN RETURN FALSE;

END;

/

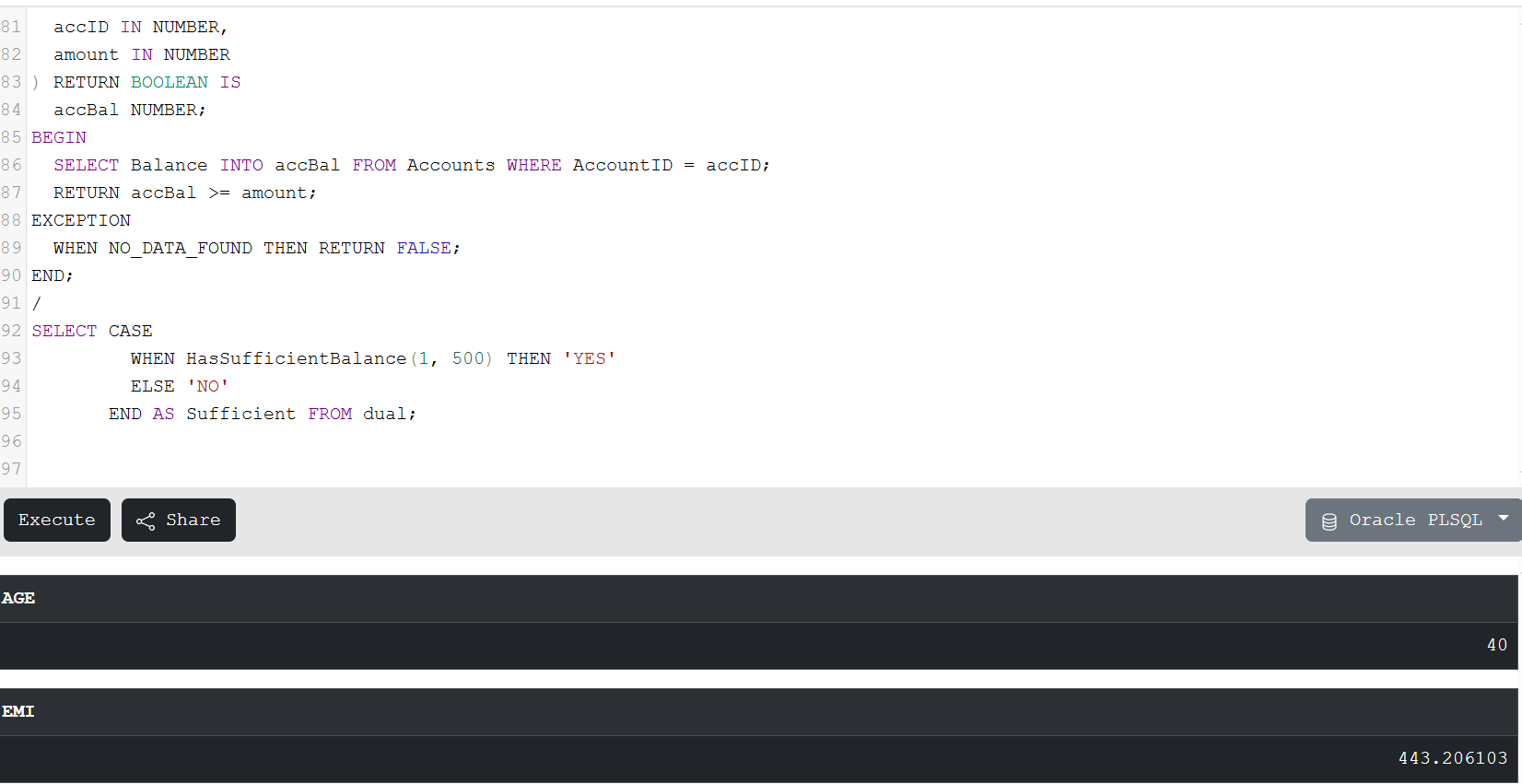
SELECT CASE

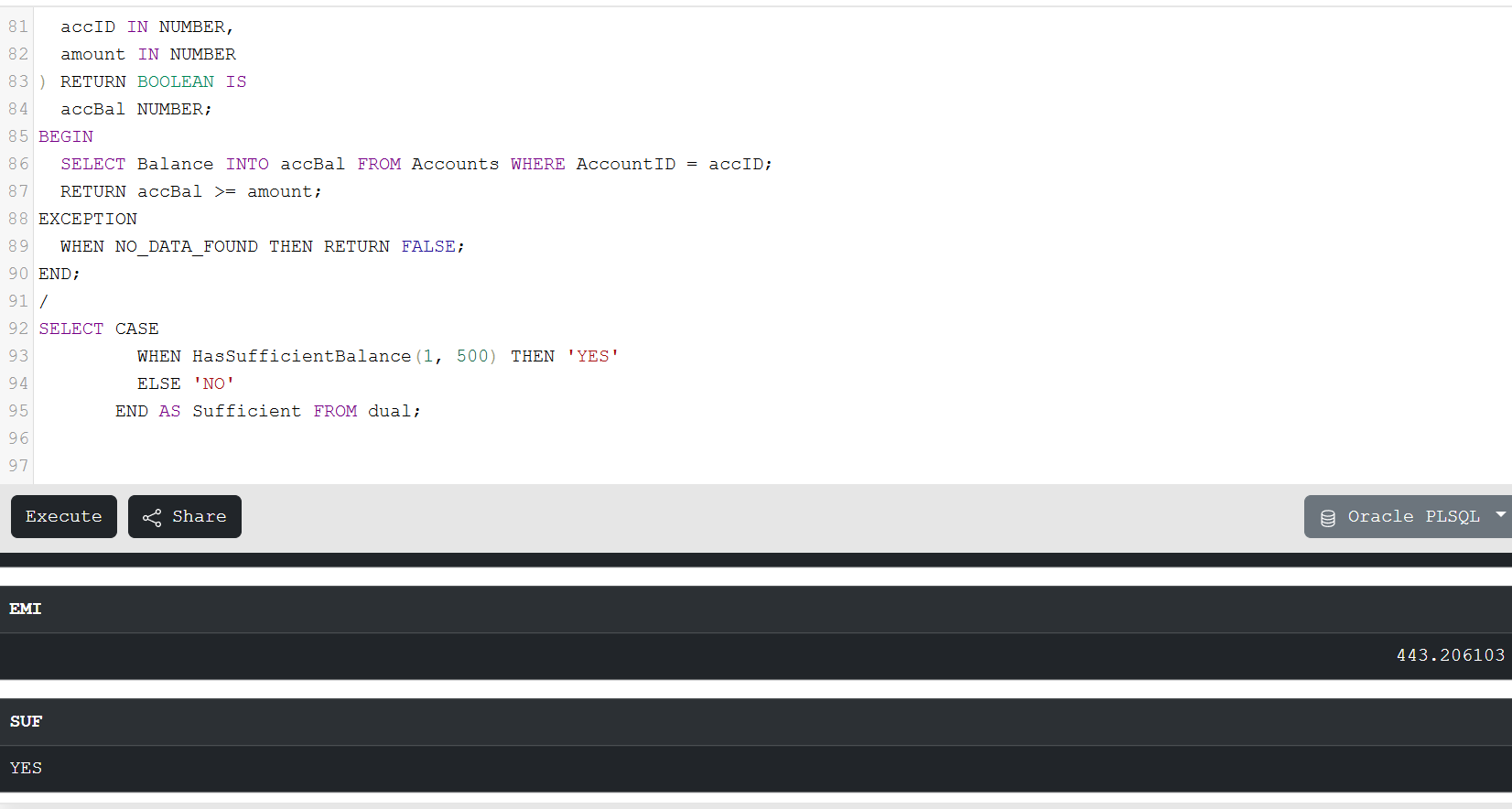
WHEN HasSufficientBalance(1, 500) THEN 'YES'

ELSE 'NO'

END AS Sufficient FROM dual;

**Outputs:**





**Exercise 5: Triggers**

**Scenario 1:**

**Sql code:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

SELECT Name, TO\_CHAR(LastModified, 'DD-MON-YYYY HH24:MI:SS') AS LastModified FROM Customers WHERE CustomerID = 1;

UPDATE Customers SET Name = 'John D Updated' WHERE CustomerID = 1;

COMMIT;

SELECT Name, TO\_CHAR(LastModified, 'DD-MON-YYYY HH24:MI:SS') AS LastModified FROM Customers WHERE CustomerID = 1;

**Scenario 2:**

**Sql code:**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (Action, ActionDate)

VALUES ('Transaction added: ' || :NEW.TransactionID, SYSDATE);

END;

/

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (3, 1, SYSDATE, 250, 'Deposit');

COMMIT;

SELECT \* FROM AuditLog WHERE Action LIKE 'Transaction added%';

**Scenario 3:**

**Sql code:**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

accBal NUMBER;

BEGIN

SELECT Balance INTO accBal FROM Accounts WHERE AccountID = :NEW.AccountID;

IF :NEW.TransactionType = 'Withdrawal' THEN

IF :NEW.Amount > accBal THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance for withdrawal');

ELSIF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Withdrawal amount must be positive');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Deposit amount must be greater than zero');

END IF;

ELSE

RAISE\_APPLICATION\_ERROR(-20004, 'Invalid transaction type');

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20005, 'Account does not exist');

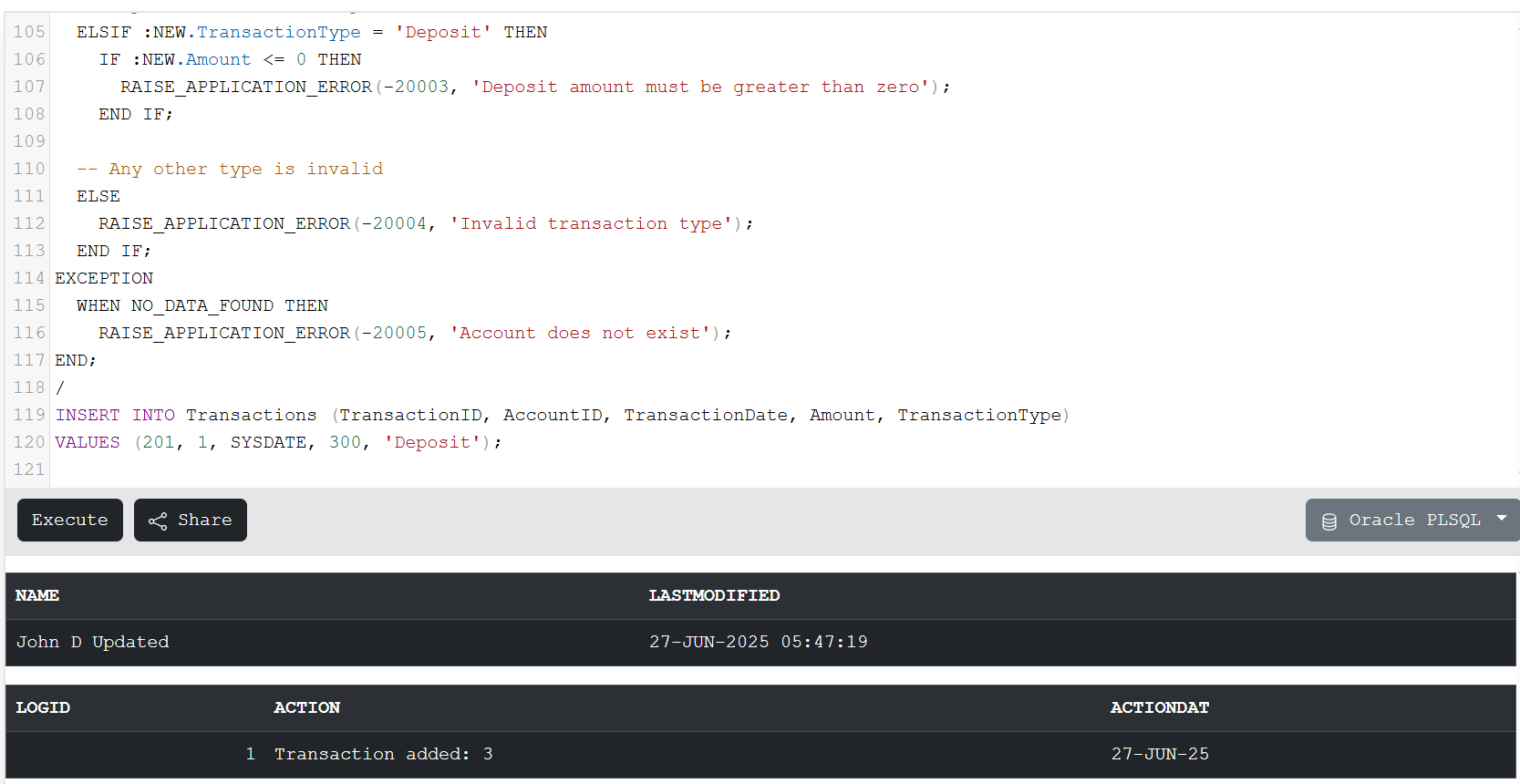
END;

/

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (201, 1, SYSDATE, 300, 'Deposit');

**Outputs:**



**Exercise 6: Cursors**

**Scenario 1:**

**Sql code:**

DECLARE

CURSOR txn\_cursor IS

SELECT c.Name, t.TransactionDate, t.Amount, t.TransactionType

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

JOIN Customers c ON a.CustomerID = c.CustomerID

WHERE TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');

BEGIN

FOR rec IN txn\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Customer: ' || rec.Name ||

', Date: ' || TO\_CHAR(rec.TransactionDate, 'DD-MON-YYYY') ||

', Amount: ' || rec.Amount ||

', Type: ' || rec.TransactionType

);

END LOOP;

END;

/

**Scenario 2:**

**Sql code:**

DECLARE

CURSOR acc\_cursor IS

SELECT AccountID FROM Accounts;

BEGIN

FOR rec IN acc\_cursor LOOP

UPDATE Accounts

SET Balance = Balance - 100

WHERE AccountID = rec.AccountID;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Annual maintenance fee applied to all accounts.');

END;

/

**Scenario 3:**

**Sql code:**

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID FROM Loans;

BEGIN

FOR rec IN loan\_cursor LOOP

UPDATE Loans

SET InterestRate = InterestRate + 0.5

WHERE LoanID = rec.LoanID;

END LOOP;

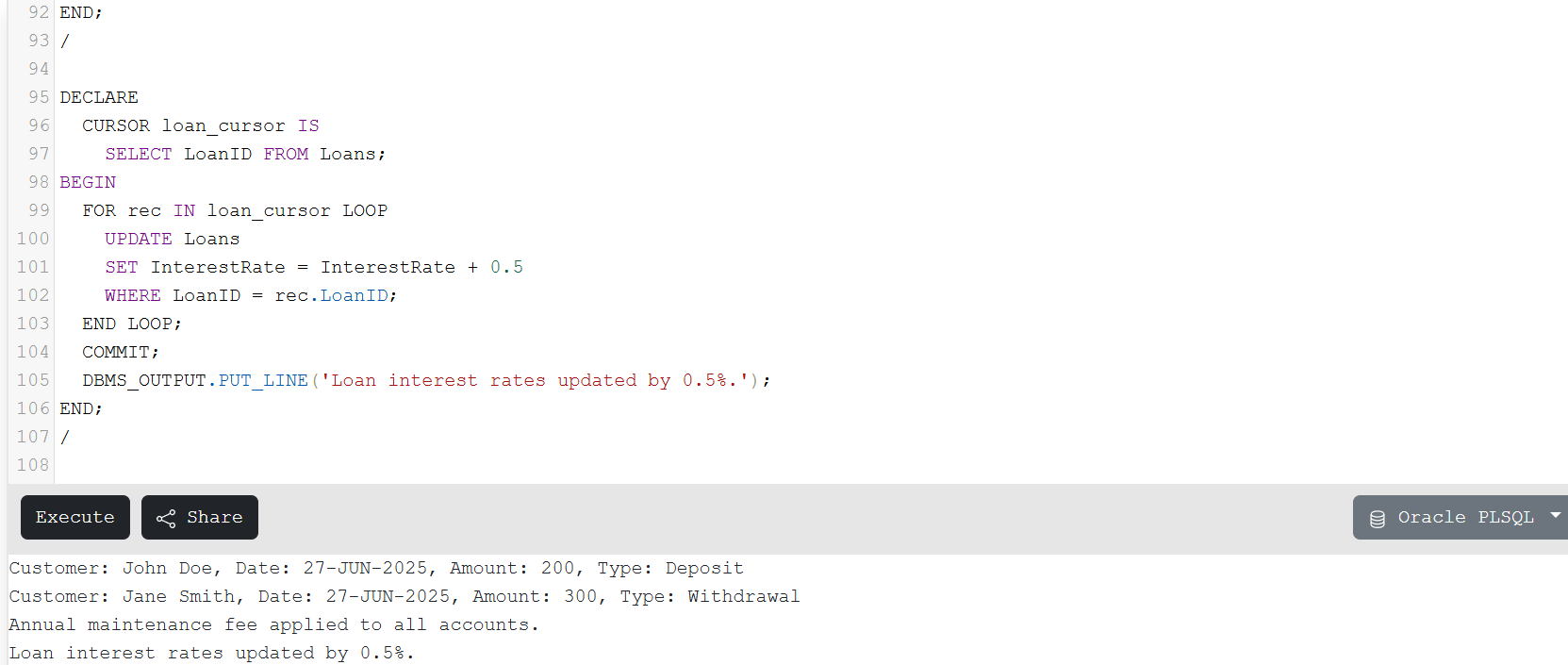
COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Loan interest rates updated by 0.5%.');

END;

/

**Outputs:**



**Exercise 7: Packages**

**Scenario 1:**

**Sql code:**

CREATE OR REPLACE PACKAGE CustomerManagement IS

PROCEDURE AddCustomer(id NUMBER, name VARCHAR2, dob DATE, balance NUMBER);

PROCEDURE UpdateCustomer(id NUMBER, name VARCHAR2);

FUNCTION GetCustomerBalance(id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement IS

PROCEDURE AddCustomer(id NUMBER, name VARCHAR2, dob DATE, balance NUMBER) IS

BEGIN

INSERT INTO Customers VALUES (id, name, dob, balance, SYSDATE, 'FALSE');

DBMS\_OUTPUT.PUT\_LINE('Customer added: ' || name);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Customer ID already exists.');

END;

PROCEDURE UpdateCustomer(id NUMBER, name VARCHAR2) IS

BEGIN

UPDATE Customers SET Name = name WHERE CustomerID = id;

DBMS\_OUTPUT.PUT\_LINE('Customer updated: ' || name);

END;

FUNCTION GetCustomerBalance(id NUMBER) RETURN NUMBER IS

bal NUMBER;

BEGIN

SELECT Balance INTO bal FROM Customers WHERE CustomerID = id;

RETURN bal;

END;

END CustomerManagement;

/

BEGIN

CustomerManagement.AddCustomer(3, 'Ravi Kumar', TO\_DATE('1988-10-10','YYYY-MM-DD'), 9000);

CustomerManagement.UpdateCustomer(3, 'Ravi Krishna');

DBMS\_OUTPUT.PUT\_LINE('Balance: ' || CustomerManagement.GetCustomerBalance(3));

END;

/

**Scenario 2:**

**Sql code:**

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(id NUMBER, name VARCHAR2, position VARCHAR2, salary NUMBER, dept VARCHAR2);

PROCEDURE UpdateEmployee(id NUMBER, salary NUMBER);

FUNCTION GetAnnualSalary(id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

PROCEDURE HireEmployee(id NUMBER, name VARCHAR2, position VARCHAR2, salary NUMBER, dept VARCHAR2) IS

BEGIN

INSERT INTO Employees VALUES (id, name, position, salary, dept, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Employee hired: ' || name);

END;

PROCEDURE UpdateEmployee(id NUMBER, salary NUMBER) IS

BEGIN

UPDATE Employees SET Salary = salary WHERE EmployeeID = id;

DBMS\_OUTPUT.PUT\_LINE('Employee salary updated: ' || id);

END;

FUNCTION GetAnnualSalary(id NUMBER) RETURN NUMBER IS

sal NUMBER;

BEGIN

SELECT Salary INTO sal FROM Employees WHERE EmployeeID = id;

RETURN sal \* 12;

END;

END EmployeeManagement;

/

BEGIN

EmployeeManagement.HireEmployee(3, 'Krishna', 'Tester', 30000, 'QA');

EmployeeManagement.UpdateEmployee(3, 35000);

DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || EmployeeManagement.GetAnnualSalary(3));

END;

/

**Scenario 3:**

**Sql code:**

CREATE OR REPLACE PACKAGE AccountOperations IS

PROCEDURE OpenAccount(id NUMBER, custID NUMBER, accType VARCHAR2, balance NUMBER);

PROCEDURE CloseAccount(id NUMBER);

FUNCTION GetTotalBalance(custID NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

PROCEDURE OpenAccount(id NUMBER, custID NUMBER, accType VARCHAR2, balance NUMBER) IS

BEGIN

INSERT INTO Accounts VALUES (id, custID, accType, balance, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Account opened: ' || id);

END;

PROCEDURE CloseAccount(id NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = id;

DBMS\_OUTPUT.PUT\_LINE('Account closed: ' || id);

END;

FUNCTION GetTotalBalance(custID NUMBER) RETURN NUMBER IS

total NUMBER;

BEGIN

SELECT NVL(SUM(Balance), 0) INTO total FROM Accounts WHERE CustomerID = custID;

RETURN total;

END;

END AccountOperations;

/

BEGIN

AccountOperations.OpenAccount(5, 3, 'Savings', 4000);

AccountOperations.CloseAccount(5);

DBMS\_OUTPUT.PUT\_LINE('Total Balance for Customer 3: ' || AccountOperations.GetTotalBalance(3));

END;

/

**Outputs:**



**MANDATORY HANDS-ON**

**TDD USING JUNIT5 AND MOCKITO**

**Exercise 1: Setting Up Junit**

**Java project : JUnitExample**

**pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>JUnitExample</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**Calculator.java (Main class)**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorTest.java (Test class)**

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

int result = calc.add(3, 4);

assertEquals(7, result);

}

}

**Exercise 3: Assertions in Junit**

**AssertionsTest.java**

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals("Sum should be 5", 5, 2 + 3);

assertTrue("10 should be greater than 5", 10 > 5);

assertFalse("2 is not greater than 10", 2 > 10);

String name = null;

assertNull("Name should be null", name);

Object obj = new Object();

assertNotNull("Object should not be null", obj);

String str1 = "JUnit";

String str2 = str1;

assertSame("Both references point to the same object", str1, str2);

String str3 = new String("JUnit");

String str4 = new String("JUnit");

assertNotSame("Different objects in memory", str3, str4);

}

}

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

**BankAccount.java**

public class BankAccount {

private int balance;

public void deposit(int amount) {

balance += amount;

}

public void withdraw(int amount) {

balance -= amount;

}

public int getBalance() {

return balance;

}

public void reset() {

balance = 0;

}

}

**BankAccountTest.java**

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class BankAccountTest {

private BankAccount account;

@Before

public void setUp() {

account = new BankAccount();

account.deposit(100); // Initial balance for tests

}

@After

public void tearDown() {

account = null;

}

@Test

public void testDeposit() {

account.deposit(50);

assertEquals(150, account.getBalance());

}

public void testWithdraw() {

account.withdraw(40);

assertEquals(60, account.getBalance());

}

public void testBalanceAfterReset() {

account.reset();

assertEquals(0, account.getBalance());

}

}

**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

**ExternalApi.java**

public interface ExternalApi {

String getData();

}

**MyService.java**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

import static org.mockito.Mockito.\*;

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

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**

**MyService.java**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

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();

}

}

**pom.xml**

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.11.0</version>

<scope>test</scope>

</dependency>

</dependencies>

Logging using SLF4J

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

**pom.xml**

<dependencies>

<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>

</dependencies>

**LoggingExample.java**

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");

}

}

JUnit Testing Exercises

**(Other Hands-on of junit testing exercises)**

**Exercise 2: Writing Basic JUnit Tests**

**Calculator.java**

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) {

return a / b;

}

}

**CalculatorTest.java**

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

Calculator calc = new Calculator();

@Test

public void testAdd() {

assertEquals(10, calc.add(6, 4));

}

@Test

public void testSubtract() {

assertEquals(2, calc.subtract(7, 5));

}

@Test

public void testMultiply() {

assertEquals(15, calc.multiply(3, 5));

}

@Test

public void testDivide() {

assertEquals(5, calc.divide(10, 2));

}

}

Advanced JUnit Testing Exercises

(Hands-on exercises)

**Exercise 1: Parameterized Tests**

**EvenChecker.java**

public class EvenChecker {

public boolean isEven(int number) {

return number % 2 == 0;

}

}

**EvenCheckerTest.java**

import static org.junit.jupiter.api.Assertions.assertTrue;

import static org.junit.jupiter.api.Assertions.assertFalse;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.ValueSource;

public class EvenCheckerTest {

EvenChecker checker = new EvenChecker();

@ParameterizedTest

@ValueSource(ints = {2, 4, 6, 8, 10})

void testEvenNumbers(int number) {

assertTrue(checker.isEven(number));

}

@ParameterizedTest

@ValueSource(ints = {1, 3, 5, 7, 9})

void testOddNumbers(int number) {

assertFalse(checker.isEven(number));

}

}

**Exercise 2: Test Suites and Categories**

**FirstTest.java**

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class FirstTest {

@Test

void testOne() {

assertEquals(2, 1 + 1);

}

}

**SecondTest.java**

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertTrue;

public class SecondTest {

@Test

void testTwo() {

assertTrue("hello".startsWith("h"));

}

}

**AllTests.java**

import org.junit.platform.suite.api.SelectClasses;

import org.junit.platform.suite.api.Suite;

@Suite

@SelectClasses({FirstTest.class, SecondTest.class})

public class AllTests {

}

**Exercise 3: Test Execution Order**

**OrderedTests.java**

import org.junit.jupiter.api.MethodOrderer.OrderAnnotation;

import org.junit.jupiter.api.Order;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestMethodOrder;

import static org.junit.jupiter.api.Assertions.assertEquals;

@TestMethodOrder(OrderAnnotation.class)

public class OrderedTests {

@Test

@Order(2)

void secondTest() {

assertEquals(4, 2 \* 2);

}

@Test

@Order(1)

void firstTest() {

assertEquals(2, 1 + 1);

}

@Test

@Order(3)

void thirdTest() {

assertEquals(8, 4 + 4);

}

}

**Exercise 4: Exception Testing**

**ExceptionThrower.java**

public class ExceptionThrower {

public void throwException() {

throw new IllegalArgumentException("Invalid argument");

}

}

**ExceptionThrowerTest.java**

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertThrows;

public class ExceptionThrowerTest {

ExceptionThrower thrower = new ExceptionThrower();

@Test

void testThrowsException() {

assertThrows(IllegalArgumentException.class, () -> {

thrower.throwException();

});

}

}

**Exercise 5: Timeout and Performance Testing**

**PerformanceTester.java**

public class PerformanceTester {

public void performTask() throws InterruptedException {

Thread.sleep(100);

}

}

**PerformanceTesterTest.java**

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertTimeout;

import java.time.Duration;

public class PerformanceTesterTest {

PerformanceTester tester = new PerformanceTester();

@Test

void testTaskCompletesInTime() {

assertTimeout(Duration.ofMillis(200), () -> {

tester.performTask();

});

}

}

Mockito Hands-On Exercises

**(Other Hands-on exercises of Mockito hands-on exercises**

**Exercise 3: Argument Matching**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

class Calculator {

public int add(int a, int b) {

return a + b;

}

}

class CalculatorTest {

@Test

void testArgumentMatching() {

Calculator mockCalc = mock(Calculator.class);

mockCalc.add(10, 20);

verify(mockCalc).add(eq(10), eq(20));

}

}

**Exercise 4: Handling Void Methods**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

class Notifier {

public void sendEmail(String message) {

System.out.println("Email sent: " + message);

}

}

class NotifierTest {

@Test

void testVoidMethod() {

Notifier mockNotifier = mock(Notifier.class);

doNothing().when(mockNotifier).sendEmail(anyString());

mockNotifier.sendEmail("Hello User!");

verify(mockNotifier).sendEmail("Hello User!");

}

}

**Exercise 5: Mocking and Stubbing with Multiple Returns**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

interface ExternalAPI {

String fetchData();

}

class APIClient {

ExternalAPI api;

APIClient(ExternalAPI api) { this.api = api; }

String[] fetchTwice() {

return new String[]{api.fetchData(), api.fetchData()};

}

}

class APIClientTest {

@Test

void testMultipleReturns() {

ExternalAPI mockApi = mock(ExternalAPI.class);

when(mockApi.fetchData()).thenReturn("First").thenReturn("Second");

APIClient client = new APIClient(mockApi);

String[] results = client.fetchTwice();

assertArrayEquals(new String[]{"First", "Second"}, results);

}

}

**Exercise 6: Verifying Interaction Order**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

class Service {

void stepOne() {}

void stepTwo() {}

}

class ServiceTest {

@Test

void testInteractionOrder() {

Service mockService = mock(Service.class);

mockService.stepOne();

mockService.stepTwo();

InOrder inOrder = inOrder(mockService);

inOrder.verify(mockService).stepOne();

inOrder.verify(mockService).stepTwo();

}

}

**Exercise 7: Handling Void Methods with Exceptions**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

class Logger {

void log(String msg) {

}

}

class LoggerTest {

@Test

void testVoidMethodThrowsException() {

Logger mockLogger = mock(Logger.class);

doThrow(new RuntimeException("Log failure")).when(mockLogger).log("Error");

assertThrows(RuntimeException.class, () -> mockLogger.log("Error"));

verify(mockLogger).log("Error");

}

}

Spring Testing Exercises

**Exercise 1: Basic Unit Test for a Service Method**

// CalculatorService.java

@Service

public class CalculatorService {

public int add(int a, int b) {

return a + b;

}

}

// CalculatorServiceTest.java

@ExtendWith(MockitoExtension.class)

public class CalculatorServiceTest {

CalculatorService calculatorService = new CalculatorService();

@Test

void testAdd() {

int result = calculatorService.add(2, 3);

assertEquals(5, result);

}

}

**Exercise 2: Mocking a Repository in a Service Test**

// UserServiceTest.java

@ExtendWith(MockitoExtension.class)

public class UserServiceTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

@Test

void testGetUserById() {

User user = new User();

user.setId(1L);

user.setName("John");

when(userRepository.findById(1L)).thenReturn(Optional.of(user));

User result = userService.getUserById(1L);

assertEquals("John", result.getName());

}

}

**Exercise 3: Testing a REST Controller with MockMvc**

// UserControllerTest.java

@WebMvcTest(UserController.class)

public class UserControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

void testGetUser() throws Exception {

User user = new User();

user.setId(1L);

user.setName("Alice");

when(userService.getUserById(1L)).thenReturn(user);

mockMvc.perform(get("/users/1"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Alice"));

}

}

**Exercise 4: Integration Test with Spring Boot**

// UserIntegrationTest.java

@SpringBootTest

@AutoConfigureMockMvc

public class UserIntegrationTest {

@Autowired

private MockMvc mockMvc;

@Autowired

private UserRepository userRepository;

@BeforeEach

void setUp() {

userRepository.save(new User(1L, "IntegrationUser"));

}

@Test

void testFullFlow() throws Exception {

mockMvc.perform(get("/users/1"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("IntegrationUser"));

}

}

**Exercise 5: Test Controller POST Endpoint**

// UserControllerPostTest.java

@WebMvcTest(UserController.class)

public class UserControllerPostTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

void testCreateUser() throws Exception {

User user = new User(1L, "NewUser");

when(userService.saveUser(any(User.class))).thenReturn(user);

mockMvc.perform(post("/users")

.contentType(MediaType.APPLICATION\_JSON)

.content("{\"id\":1,\"name\":\"NewUser\"}"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("NewUser"));

}

}

**Exercise 6: Test Service Exception Handling**

// UserServiceExceptionTest.java

@ExtendWith(MockitoExtension.class)

public class UserServiceExceptionTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

@Test

void testUserNotFound() {

when(userRepository.findById(1L)).thenReturn(Optional.empty());

User result = userService.getUserById(1L);

assertNull(result);

}

}

**Exercise 7: Test Custom Repository Query**

// UserRepositoryTest.java

@DataJpaTest

public class UserRepositoryTest {

@Autowired

private UserRepository userRepository;

@Test

void testFindByName() {

User user = new User(1L, "CustomName");

userRepository.save(user);

List<User> users = userRepository.findByName("CustomName");

assertEquals(1, users.size());

}

}

**Exercise 8: Test Controller Exception Handling**

// UserControllerAdviceTest.java

@WebMvcTest(UserController.class)

public class UserControllerAdviceTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

void testHandleNotFound() throws Exception {

when(userService.getUserById(99L)).thenThrow(new NoSuchElementException());

mockMvc.perform(get("/users/99"))

.andExpect(status().isNotFound())

.andExpect(content().string("User not found"));

}

}

**Exercise 9: Parameterized Test with Junit**

// ParameterizedAddTest.java

@ExtendWith(MockitoExtension.class)

public class ParameterizedAddTest {

CalculatorService calculatorService = new CalculatorService();

@ParameterizedTest

@CsvSource({

"1, 2, 3",

"5, 3, 8",

"10, -5, 5"

})

void testAdd(int a, int b, int expected) {

assertEquals(expected, calculatorService.add(a, b));

}

}

Mocking Dependencies in Spring Tests using Mockito

**Exercise 1: Mocking a Service Dependency in a Controller Test**

**UserControllerTest.java**

import com.example.demo.controller.UserController;

import com.example.demo.entity.User;

import com.example.demo.service.UserService;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.boot.test.mock.mockito.MockBean;

import org.springframework.http.MediaType;

import org.springframework.test.web.servlet.MockMvc;

import static org.mockito.Mockito.when;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@WebMvcTest(UserController.class)

public class UserControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

public void testGetUser() throws Exception {

User user = new User();

user.setId(1L);

user.setName("John");

when(userService.getUserById(1L)).thenReturn(user);

mockMvc.perform(get("/users/1")

.accept(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.id").value(1))

.andExpect(jsonPath("$.name").value("John"));

}

}

**Exercise 2: Mocking a Repository in a Service Test**

**UserServiceTest.java**

import com.example.demo.entity.User;

import com.example.demo.repository.UserRepository;

import com.example.demo.service.UserService;

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.junit.jupiter.MockitoExtension;

import java.util.Optional;

import static org.mockito.Mockito.when;

import static org.junit.jupiter.api.Assertions.assertEquals;

import static org.junit.jupiter.api.Assertions.assertNull;

import org.junit.jupiter.api.extension.ExtendWith;

@ExtendWith(MockitoExtension.class)

public class UserServiceTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

@Test

public void testGetUserById\_UserExists() {

User user = new User();

user.setId(1L);

user.setName("Alice");

when(userRepository.findById(1L)).thenReturn(Optional.of(user));

User result = userService.getUserById(1L);

assertEquals("Alice", result.getName());

}

@Test

public void testGetUserById\_UserNotFound() {

when(userRepository.findById(2L)).thenReturn(Optional.empty());

User result = userService.getUserById(2L);

assertNull(result);

}

}

**Exercise 3: Mocking a Service Dependency in an Integration Test**

**UserIntegrationTest.java**

import com.example.demo.entity.User;

import com.example.demo.service.UserService;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.boot.test.mock.mockito.MockBean;

import org.springframework.http.MediaType;

import org.springframework.test.web.servlet.MockMvc;

import static org.mockito.Mockito.when;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@SpringBootTest

@AutoConfigureMockMvc

public class UserIntegrationTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

public void testGetUserEndpoint() throws Exception {

User user = new User();

user.setId(10L);

user.setName("Jane");

when(userService.getUserById(10L)).thenReturn(user);

mockMvc.perform(get("/users/10")

.accept(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.id").value(10))

.andExpect(jsonPath("$.name").value("Jane"));

}

}

Logging using SLF4J

**Exercise 2: Parameterized Logging**

**pom.xml**

<dependencies>

<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>

</dependencies>

Java Class with Parameterized Logging:

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class ParameterizedLoggingDemo {

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

public static void main(String[] args) {

String user = "John";

int items = 5;

logger.info("User {} has added {} items to the cart.", user, items);

logger.debug("Debug info: processing user {}, item count {}", user, items);

logger.warn("Warning: user {} might exceed limit with {} items.", user, items);

logger.error("Error: failed to process order for user {} with {} items.", user, items);

}

}

**Exercise 3: Using Different Appenders**

**logback.xml**

<configuration>

<appender name="console" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<appender name="file" class="ch.qos.logback.core.FileAppender">

<file>logs/app.log</file>

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<root level="debug">

<appender-ref ref="console" />

<appender-ref ref="file" />

</root>

</configuration>

Java Class to Use Logging with Appenders:

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingWithAppendersDemo {

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

public static void main(String[] args) {

logger.info("This is an info message - logged to console and file.");

logger.debug("This is a debug message - logged to console and file.");

logger.error("This is an error message - logged to console and file.");

}

}