**Exercise 1: Mocking and Stubbing**

1. Project Dependencies (pom.xml)

<dependencies>

<!-- JUnit 5 -->

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.9.3</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>4.11.0</version>

<scope>test</scope>

</dependency>

</dependencies>

2. External API Interface (ExternalApi.java)

public interface ExternalApi {

String getData();

}

3. Service Class That Uses the API (MyService.java)

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

4. Unit Test Using Mockito (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**

1. ExternalApi.java

public interface ExternalApi {

String getData();

}

2. MyService.java

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

3. 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();

}

}

**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 1: pom.xml (Add the SLF4J and Logback dependencies)**

<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>

**Step 2: LoggingExample.java**

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

// Create logger instance

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

public static void main(String[] args) {

// Logging error and warning messages

logger.error("This is an error message");

logger.warn("This is a warning message");

}

}

**PL-SQL:  
  
Exercise 1: Control Structures**

**Scenario 1: Apply 1% discount for customers over 60**

**BEGIN**

**FOR customer\_rec IN (**

**SELECT customer\_id, age**

**FROM Customers**

**) LOOP**

**IF customer\_rec.age > 60 THEN**

**UPDATE Loans**

**SET interest\_rate = interest\_rate - 1**

**WHERE customer\_id = customer\_rec.customer\_id;**

**END IF;**

**END LOOP;**

**COMMIT;**

**END;**

**/**

**Scenario 2: Promote to VIP if balance > $10,000**

**BEGIN**

**FOR customer\_rec IN (**

**SELECT customer\_id, balance**

**FROM Customers**

**) LOOP**

**IF customer\_rec.balance > 10000 THEN**

**UPDATE Customers**

**SET IsVIP = 'TRUE'**

**WHERE customer\_id = customer\_rec.customer\_id;**

**END IF;**

**END LOOP;**

**COMMIT;**

**END;**

**/**

**Scenario 3: Send loan due reminders within next 30 days**

**DECLARE**

**v\_due\_date DATE := SYSDATE + 30;**

**BEGIN**

**FOR loan\_rec IN (**

**SELECT l.loan\_id, l.customer\_id, c.name, l.due\_date**

**FROM Loans l**

**JOIN Customers c ON l.customer\_id = c.customer\_id**

**WHERE l.due\_date BETWEEN SYSDATE AND v\_due\_date**

**) LOOP**

**DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan\_rec.loan\_id ||**

**' for Customer ' || loan\_rec.name ||**

**' is due on ' || TO\_CHAR(loan\_rec.due\_date, 'DD-MON-YYYY'));**

**END LOOP;**

**END;**

**/**

**Exercise 3: Stored Procedures**

**Scenario 1: Process Monthly Interest on Savings Accounts**

**CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS**

**BEGIN**

**UPDATE Accounts**

**SET balance = balance + (balance \* 0.01)**

**WHERE account\_type = 'SAVINGS';**

**COMMIT;**

**END;**

**/**

**Scenario 2: Update Bonus for Employees**

**CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (**

**dept\_id IN NUMBER,**

**bonus\_percent IN NUMBER**

**) AS**

**BEGIN**

**UPDATE Employees**

**SET salary = salary + (salary \* bonus\_percent / 100)**

**WHERE department\_id = dept\_id;**

**COMMIT;**

**END;**

**/**

**Scenario 3: Transfer Funds Between Accounts**

**CREATE OR REPLACE PROCEDURE TransferFunds (**

**from\_account\_id IN NUMBER,**

**to\_account\_id IN NUMBER,**

**amount IN NUMBER**

**) AS**

**insufficient\_balance EXCEPTION;**

**current\_balance NUMBER;**

**BEGIN**

**-- Get balance of source account**

**SELECT balance INTO current\_balance**

**FROM Accounts**

**WHERE account\_id = from\_account\_id**

**FOR UPDATE;**

**-- Check for sufficient funds**

**IF current\_balance < amount THEN**

**RAISE insufficient\_balance;**

**END IF;**

**-- Deduct from source**

**UPDATE Accounts**

**SET balance = balance - amount**

**WHERE account\_id = from\_account\_id;**

**-- Add to destination**

**UPDATE Accounts**

**SET balance = balance + amount**

**WHERE account\_id = to\_account\_id;**

**COMMIT;**

**EXCEPTION**

**WHEN insufficient\_balance THEN**

**ROLLBACK;**

**DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient balance.');**

**WHEN OTHERS THEN**

**ROLLBACK;**

**DBMS\_OUTPUT.PUT\_LINE('Transfer failed due to unexpected error: ' || SQLERRM);**

**END;**

**/**