Java and PL/SQL Assignment with Unit Testing, Mockito, and Logging

# Exercise 1: Control Structures

# Scenario 1: Apply Discount to Loan Interest Rates

BEGIN  
 FOR cust\_rec IN (SELECT customer\_id FROM customers WHERE age > 60) LOOP  
 UPDATE loans  
 SET interest\_rate = interest\_rate - 1  
 WHERE customer\_id = cust\_rec.customer\_id;  
 END LOOP;  
 COMMIT;  
END;

## Output

Loan interest reduced by 1% for all customers above age 60.

# Scenario 2: Promote to VIP Based on Balance

BEGIN  
 FOR cust\_rec IN (SELECT customer\_id FROM customers WHERE balance > 10000) LOOP  
 UPDATE customers  
 SET isvip = 'TRUE'  
 WHERE customer\_id = cust\_rec.customer\_id;  
 END LOOP;  
 COMMIT;  
END;

## Output

Customers with balance > $10,000 flagged as VIP.

# Scenario 3: Send Reminders for Loan Dues

DECLARE  
 CURSOR loan\_cursor IS  
 SELECT l.loan\_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 SYSDATE + 30;  
BEGIN  
 FOR rec IN loan\_cursor LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.loan\_id || ' for ' || rec.name || ' due on ' || TO\_CHAR(rec.due\_date, 'DD-MON-YYYY'));  
 END LOOP;  
END;

## Output

Reminder message printed for each loan due in next 30 days.

# Exercise 3: Stored Procedures

# Scenario 1: Process Monthly Interest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS  
BEGIN  
 UPDATE accounts  
 SET balance = balance + (balance \* 0.01)  
 WHERE account\_type = 'SAVINGS';  
 COMMIT;  
END;

## Output

1% interest applied to all savings account balances.

# Scenario 2: Update Employee Bonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (  
 p\_department\_id IN NUMBER,  
 p\_bonus\_percent IN NUMBER  
) IS  
BEGIN  
 UPDATE employees  
 SET salary = salary + (salary \* p\_bonus\_percent / 100)  
 WHERE department\_id = p\_department\_id;  
 COMMIT;  
END;

## Output

Salary updated with bonus percentage for specified department.

# Scenario 3: Transfer Funds Between Accounts

CREATE OR REPLACE PROCEDURE TransferFunds (  
 p\_from\_account\_id IN NUMBER,  
 p\_to\_account\_id IN NUMBER,  
 p\_amount IN NUMBER  
) IS  
 v\_balance NUMBER;  
BEGIN  
 SELECT balance INTO v\_balance FROM accounts WHERE account\_id = p\_from\_account\_id FOR UPDATE;  
 IF v\_balance < p\_amount THEN  
 RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance.');  
 END IF;  
 UPDATE accounts SET balance = balance - p\_amount WHERE account\_id = p\_from\_account\_id;  
 UPDATE accounts SET balance = balance + p\_amount WHERE account\_id = p\_to\_account\_id;  
 COMMIT;  
END;

## Output

Funds transferred if source has sufficient balance.

# Unit Testing Exercise 1: Setting Up JUnit

<dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
</dependency>

## Output

JUnit added to project and ready for writing tests.

# Exercise 3: Assertions in JUnit

public class AssertionsTest {  
 @Test  
 public void testAssertions() {  
 assertEquals(5, 2 + 3);  
 assertTrue(5 > 3);  
 assertFalse(5 < 3);  
 assertNull(null);  
 assertNotNull(new Object());  
 }  
}

## Output

All assertions passed in unit test.

# Exercise 4: AAA Pattern and Setup/Teardown

public class ExampleTest {  
 private Calculator calc;  
  
 @Before  
 public void setUp() {  
 calc = new Calculator();  
 }  
  
 @After  
 public void tearDown() {  
 calc = null;  
 }  
  
 @Test  
 public void testAddition() {  
 int result = calc.add(2, 3);  
 assertEquals(5, result);  
 }  
}

## Output

Test organized using Arrange-Act-Assert pattern with setup and teardown.

# Mockito Exercise 1: Mocking and Stubbing

@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);  
}

## Output

Mock API returned 'Mock Data' successfully.

# Mockito Exercise 2: Verifying Interactions

@Test  
public void testVerifyInteraction() {  
 ExternalApi mockApi = Mockito.mock(ExternalApi.class);  
 MyService service = new MyService(mockApi);  
 service.fetchData();  
 verify(mockApi).getData();  
}

## Output

Verified that getData() was called on mock API.

# SLF4J Logging: Error and Warning

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");  
 }  
}

## Output

Error and warning messages logged using SLF4J.