WEEK2

# JUnit Exercise 1: Setting Up JUnit

Scenario: Set up JUnit in your Java project and create a test class to verify that JUnit works.  
Step 1: Add JUnit to your pom.xml:

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

public class SampleTest {  
 @Test  
 public void additionTest() {  
 int result = 2 + 3;  
 assertEquals(5, result);  
 }  
}

# JUnit Exercise 3: Assertions in JUnit

Scenario: Use different types of JUnit assertions to validate outcomes in your test cases.

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

# JUnit Exercise 4: AAA Pattern, Setup and Teardown

Scenario: Structure your tests using the Arrange-Act-Assert pattern and use @Before and @After methods.

public class CalculatorTest {  
 private Calculator calculator;  
  
 @Before  
 public void setUp() {  
 calculator = new Calculator();  
 }  
  
 @After  
 public void tearDown() {  
 calculator = null;  
 }  
  
 @Test  
 public void testAddition() {  
 int result = calculator.add(10, 5);  
 assertEquals(15, result);  
 }  
  
 @Test  
 public void testSubtraction() {  
 int result = calculator.subtract(10, 3);  
 assertEquals(7, result);  
 }  
}

public class Calculator {  
 public int add(int a, int b) {  
 return a + b;  
 }  
 public int subtract(int a, int b) {  
 return a - b;  
 }  
}

# Mockito Exercise 1: Mocking and Stubbing

Scenario: Test a service that depends on an external API using 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);  
 }  
}

public class MyService {  
 private ExternalApi api;  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
 public String fetchData() {  
 return api.getData();  
 }  
}

public interface ExternalApi {  
 String getData();  
}

# Mockito Exercise 2: Verifying Interactions

Scenario: Ensure that a method is called with specific arguments using Mockito.

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

# SLF4J Logging Exercise: Logging Error and Warning Messages

Scenario: Log error and warning messages using SLF4J with Logback.  
Step 1: Add SLF4J and Logback dependencies to pom.xml:

<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>

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");  
 }  
}

PL/SQL Exercise 1 & Exercise 3 with Output

# Exercise 1: Control Structures

## Scenario 1: Interest Discount for Customers Above 60

SET SERVEROUTPUT ON;  
  
DECLARE  
 CURSOR cur\_customers IS  
 SELECT customer\_id, interest\_rate  
 FROM loan  
 JOIN customer USING (customer\_id)  
 WHERE age > 60;  
  
BEGIN  
 FOR rec IN cur\_customers LOOP  
 UPDATE loan  
 SET interest\_rate = interest\_rate - 1  
 WHERE customer\_id = rec.customer\_id;  
  
 DBMS\_OUTPUT.PUT\_LINE('Applied 1% discount to Customer ID: ' || rec.customer\_id);  
 END LOOP;  
  
 COMMIT;  
END;  
/

Sample Output:  
Applied 1% discount to Customer ID: 101  
Applied 1% discount to Customer ID: 103

## Scenario 2: Promote Customers to VIP Based on Balance

SET SERVEROUTPUT ON;  
  
DECLARE  
 CURSOR cur\_customers IS  
 SELECT customer\_id  
 FROM customer  
 WHERE balance > 10000;  
  
BEGIN  
 FOR rec IN cur\_customers LOOP  
 UPDATE customer  
 SET IsVIP = 'TRUE'  
 WHERE customer\_id = rec.customer\_id;  
  
 DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || rec.customer\_id || ' promoted to VIP.');  
 END LOOP;  
  
 COMMIT;  
END;  
/

Sample Output:  
Customer ID 101 promoted to VIP.  
Customer ID 103 promoted to VIP.

## Scenario 3: Reminder for Loan Due in Next 30 Days

SET SERVEROUTPUT ON;  
  
DECLARE  
 CURSOR cur\_due\_loans IS  
 SELECT l.loan\_id, c.customer\_id, c.name, l.due\_date  
 FROM loan l  
 JOIN customer c ON l.customer\_id = c.customer\_id  
 WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30;  
  
BEGIN  
 FOR rec IN cur\_due\_loans LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.loan\_id ||   
 ' for Customer ' || rec.name ||  
 ' (ID: ' || rec.customer\_id ||   
 ') is due on ' || TO\_CHAR(rec.due\_date, 'DD-MON-YYYY'));  
 END LOOP;  
END;  
/

Sample Output:  
Reminder: Loan ID 201 for Customer Keerthika Setty (ID: 101) is due on 12-JUL-2025  
Reminder: Loan ID 203 for Customer Sunita Rani (ID: 103) is due on 07-JUL-2025

# Exercise 3: Stored Procedures

## Scenario 1: ProcessMonthlyInterest

SET SERVEROUTPUT ON;  
  
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS  
BEGIN  
 UPDATE accounts  
 SET balance = balance + (balance \* 0.01)  
 WHERE account\_type = 'SAVINGS';  
  
 DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to all savings accounts.');  
  
 COMMIT;  
END;  
/  
-- Procedure Call  
BEGIN  
 ProcessMonthlyInterest;  
END;  
/

Sample Output:  
Monthly interest applied to all savings accounts.

## Scenario 2: UpdateEmployeeBonus

SET SERVEROUTPUT ON;  
  
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (  
 dept\_id IN NUMBER,  
 bonus\_pct IN NUMBER  
) AS  
BEGIN  
 UPDATE employees  
 SET salary = salary + (salary \* bonus\_pct / 100)  
 WHERE department\_id = dept\_id;  
  
 DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || bonus\_pct || '% applied to department ' || dept\_id);  
  
 COMMIT;  
END;  
/  
-- Procedure Call  
BEGIN  
 UpdateEmployeeBonus(10, 5);  
END;  
/

Sample Output:  
Bonus of 5% applied to department 10

## Scenario 3: TransferFunds

SET SERVEROUTPUT ON;  
  
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  
 SELECT balance INTO current\_balance  
 FROM accounts  
 WHERE account\_id = from\_account\_id  
 FOR UPDATE;  
  
 IF current\_balance < amount THEN  
 RAISE insufficient\_balance;  
 END IF;  
  
 UPDATE accounts  
 SET balance = balance - amount  
 WHERE account\_id = from\_account\_id;  
  
 UPDATE accounts  
 SET balance = balance + amount  
 WHERE account\_id = to\_account\_id;  
  
 COMMIT;  
 DBMS\_OUTPUT.PUT\_LINE('Transferred ' || amount ||   
 ' from Account ' || from\_account\_id ||   
 ' to Account ' || to\_account\_id);  
  
EXCEPTION  
 WHEN insufficient\_balance THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient balance in account ' || from\_account\_id);  
 WHEN OTHERS THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);  
END;  
/  
-- Procedure Call  
BEGIN  
 TransferFunds(1001, 1002, 500);  
END;  
/

Sample Output:  
Transferred 500 from Account 1001 to Account 1002