Cognizant Digital Nurture 4.0 – Week 2:

M. Aditya Naidu  
4992389

# Topic: PL SQL, Unit Testing

## JUnit and Mockito Equivalent for NUnit and Moq (Java Version)

1. Calculator and Unit Test (JUnit equivalent)

Calculator.java

package calc;  
  
public class Calculator {  
 public int add(int a, int b) {  
 return a + b;  
 }  
}

Output: Returns the sum of two integers.

CalculatorTest.java

package calc;  
  
import org.junit.jupiter.api.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
  
class CalculatorTest {  
 private Calculator calculator;  
  
 @BeforeEach  
 void setup() {  
 calculator = new Calculator();  
 }  
  
 @AfterEach  
 void teardown() {  
 calculator = null;  
 }  
  
 @Test  
 @DisplayName("Add: Valid inputs")  
 void testAdd() {  
 assertEquals(5, calculator.add(2, 3));  
 assertEquals(0, calculator.add(-1, 1));  
 assertEquals(0, calculator.add(0, 0));  
 assertEquals(300, calculator.add(100, 200));  
 }  
  
 @Disabled("Subtraction not implemented yet")  
 @Test  
 void testSubtract() {  
 }  
}

Output: All test cases pass validating the add method.

2. Mocking Example using Mockito

IMailSender.java

package customer;  
  
public interface IMailSender {  
 boolean sendMail(String toAddress, String message);  
}

Output: Interface defining the email sending contract.

CustomerCommTest.java

package customer;  
  
import org.junit.jupiter.api.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
import static org.mockito.Mockito.\*;  
  
class CustomerCommTest {  
  
 private IMailSender mockSender;  
 private CustomerComm customerComm;  
  
 @BeforeEach  
 void setup() {  
 mockSender = mock(IMailSender.class);  
 when(mockSender.sendMail(anyString(), anyString())).thenReturn(true);  
 customerComm = new CustomerComm(mockSender);  
 }  
  
 @Test  
 void testSendMailToCustomer\_ReturnsTrue() {  
 assertTrue(customerComm.sendMailToCustomer());  
 }  
}

Output: True (Mail sent successfully via mocked service).

## PL/SQL Examples

EX-1 Ranking and Window Functions

Step 1: Use ROW\_NUMBER() to assign a unique rank

SELECT \*  
FROM (SELECT  
 ProductID,  
 ProductName,  
 Category,  
 Price,  
 ROW\_NUMBER() OVER (PARTITION BY Category ORDER BY Price DESC) AS RowNum  
 FROM Products  
) AS Ranked  
WHERE RowNum <= 3;

Output: Top 3 products per category by price using ROW\_NUMBER.

Step 2: Use RANK() to handle ties

SELECT \*  
FROM (  
 SELECT  
 ProductID,  
 ProductName,  
 Category,  
 Price,  
 RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS RankNum  
 FROM Products  
) AS Ranked  
WHERE RankNum <= 3;

Output: Top 3 products including ties using RANK.

Step 3: Use DENSE\_RANK() to handle ties

SELECT \*  
FROM (  
 SELECT  
 ProductID,  
 ProductName,  
 Category,  
 Price,  
 DENSE\_RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS DenseRankNum  
 FROM Products  
) AS Ranked  
WHERE DenseRankNum <= 3;

Output: Top 3 products with continuous ranking using DENSE\_RANK.