MSc. in Applied Software Engineering

PC#2 Mockito Assessment 2022 – Friday 11th November

Time : 11.15 – 13.25 – At 13.25 upload your Test class to moodle. Moodle will close automatically at 13.30

Notes on marking scheme:-

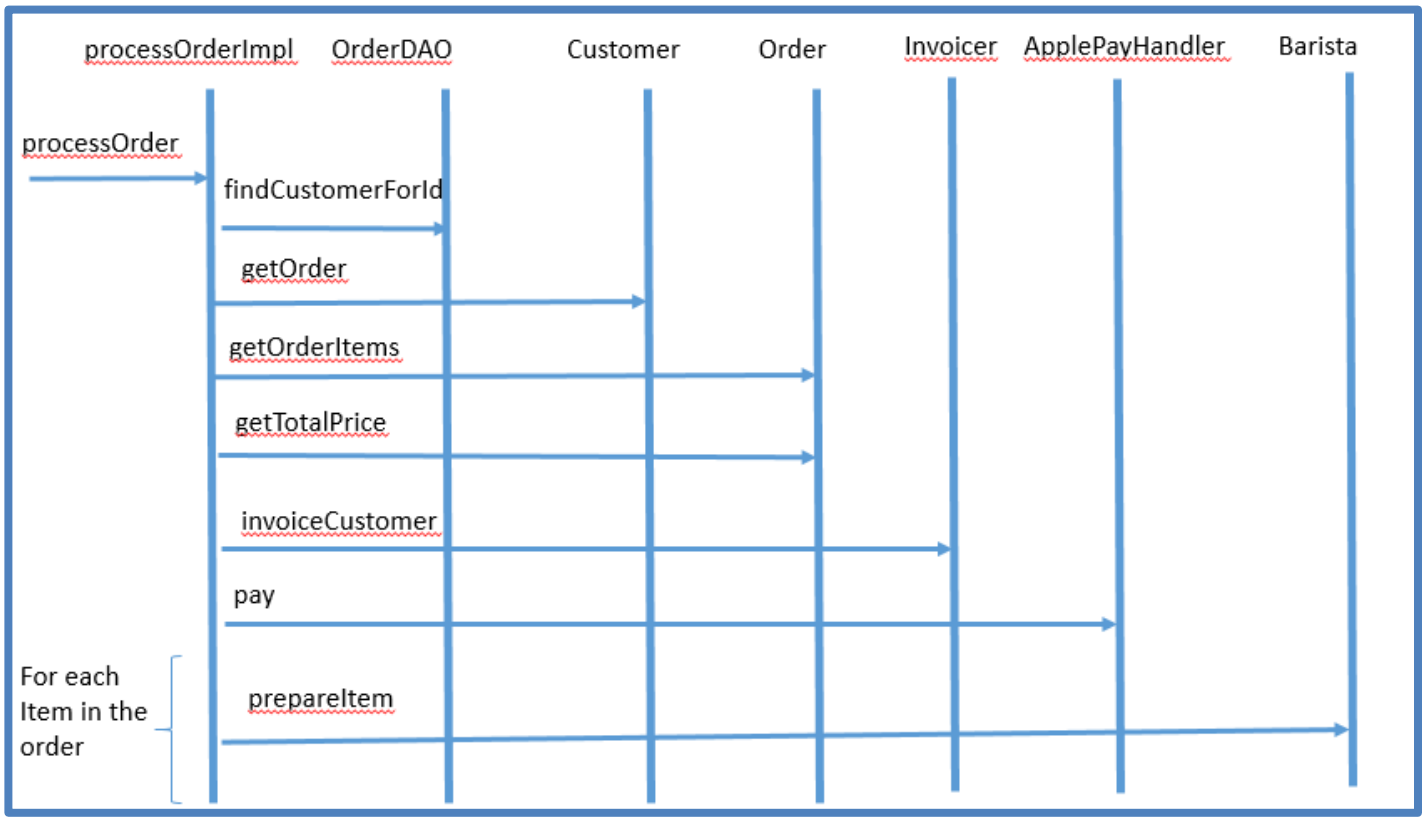
1. **Your test code must compile. Automatic zero for non-compiling code.**
2. **Do not modify any of the code you have been given. Complete the class ProcessOrderImplTest.java**
3. **Set up the project with the packages as given.**
4. **You test should be passing before it will be marked. No marks for partially completed/failing tests. Tests should include asserts and verifys as specified in the test descriptions below. Name the tests as specified below.**
5. **What to submit :- 1. Your java test class ProcessOrderImplTest.java**
6. **Upload a plagiarism form for the assessment.**

Coffee Corner is a Coffee Shop with on online ordering facility. Using an app, customers can create an order by adding products (e.g. tea and coffee etc.) to the order. When a customer has created their order, the click the "Complete Order" option. This triggers the method `processOrder` to be called on an instance of `ProcessOrderImpl` which takes care of processing the order in the coffee shop (payment, invoice customer, notify the barista to prepare the drink/product).

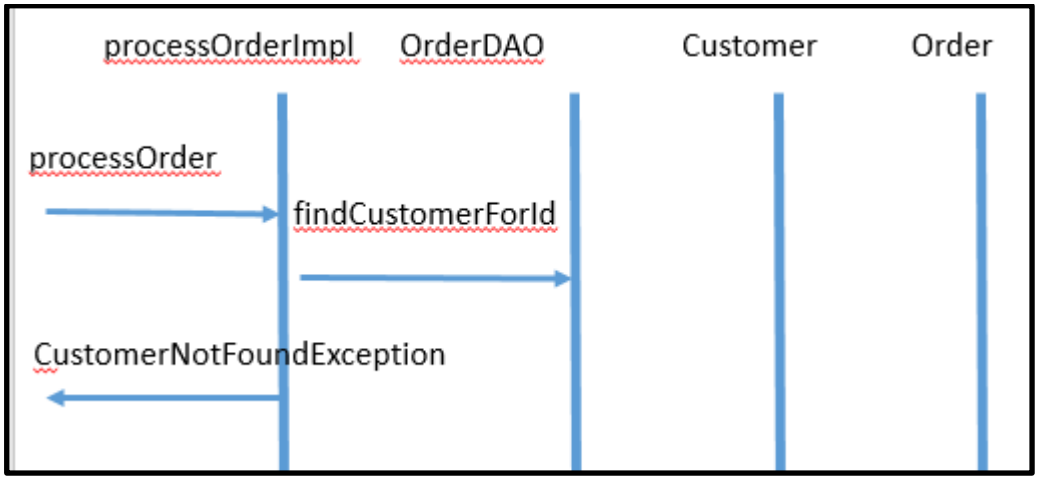
When method `processOrder` is called `ProcessOrderImpl.java` has to do the following:-

1. Retrieve customer object from the DAO based on the Customer id.
2. Retrieve the `order` object reference from the `customer` object.
3. Retrieve the order list (ArrayList of OrderItems) from the order object.
4. Retrieve the orderTotal (total cost of order) from the order object.
5. Trigger the sending of an invoice to the customer by calling the method invoiceCustomer on Invoicer. An invoice number is returned.
6. Trigger the payment via ApplePay by calling the pay method on ApplePayHandler.
7. For each item in the order, notify the barista to prepare the order by calling prepareItem on Barista

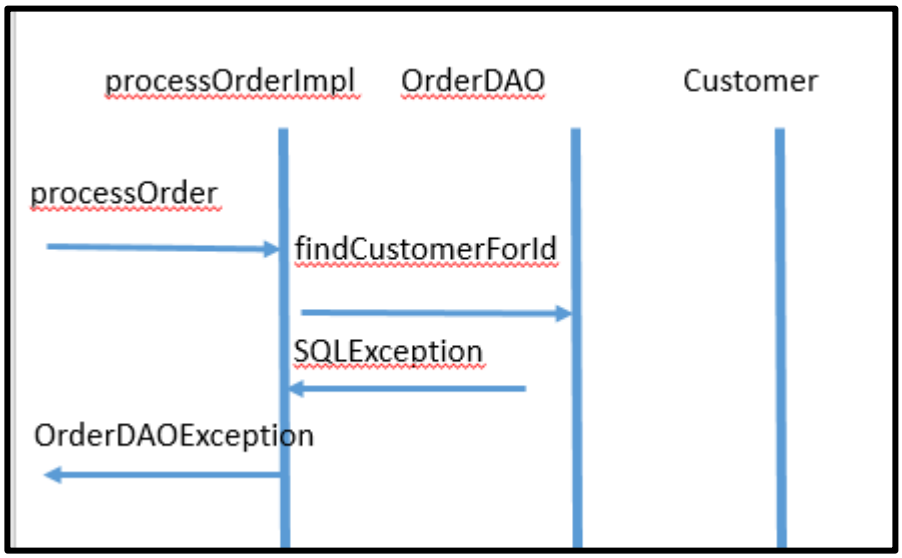
The diagram below shows the sequence of messages when the processOrder method is called on ProcessOrderImpl.java



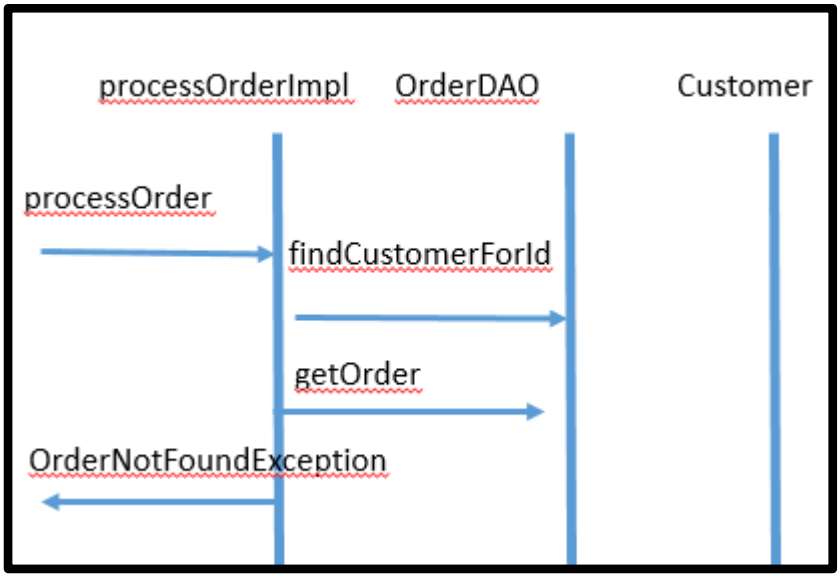
Scenario 1 – Positive Flow



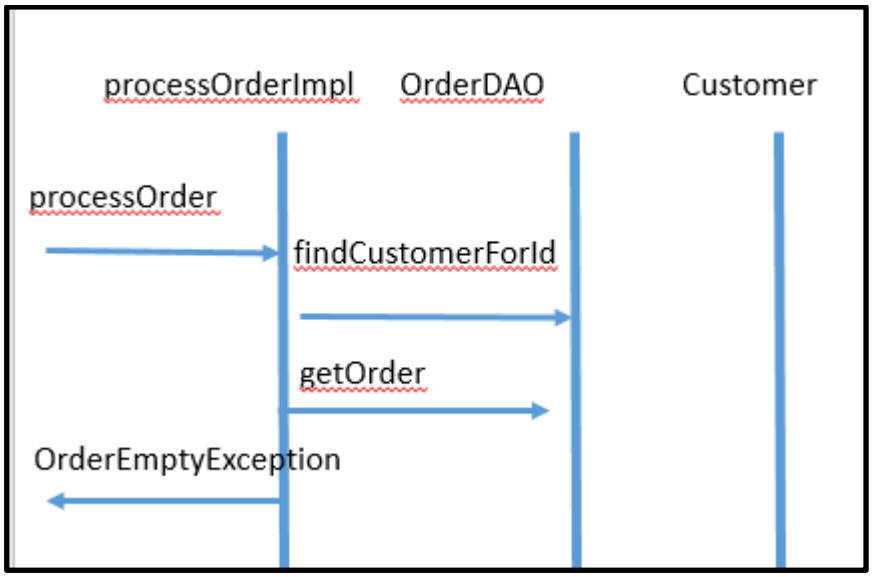
Scenario 2 – CustomerNotFoundException



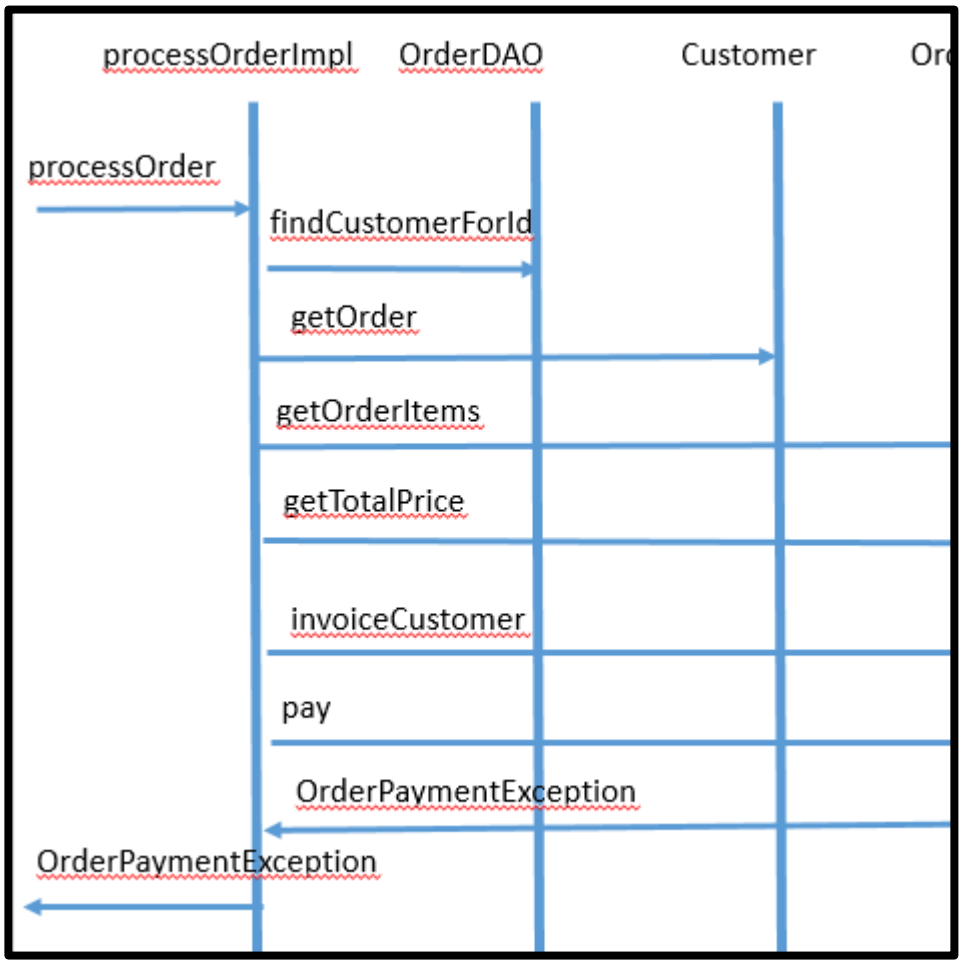
Scenario 3 – OrderDAOException



Scenario 4 – OrderNotFoundException

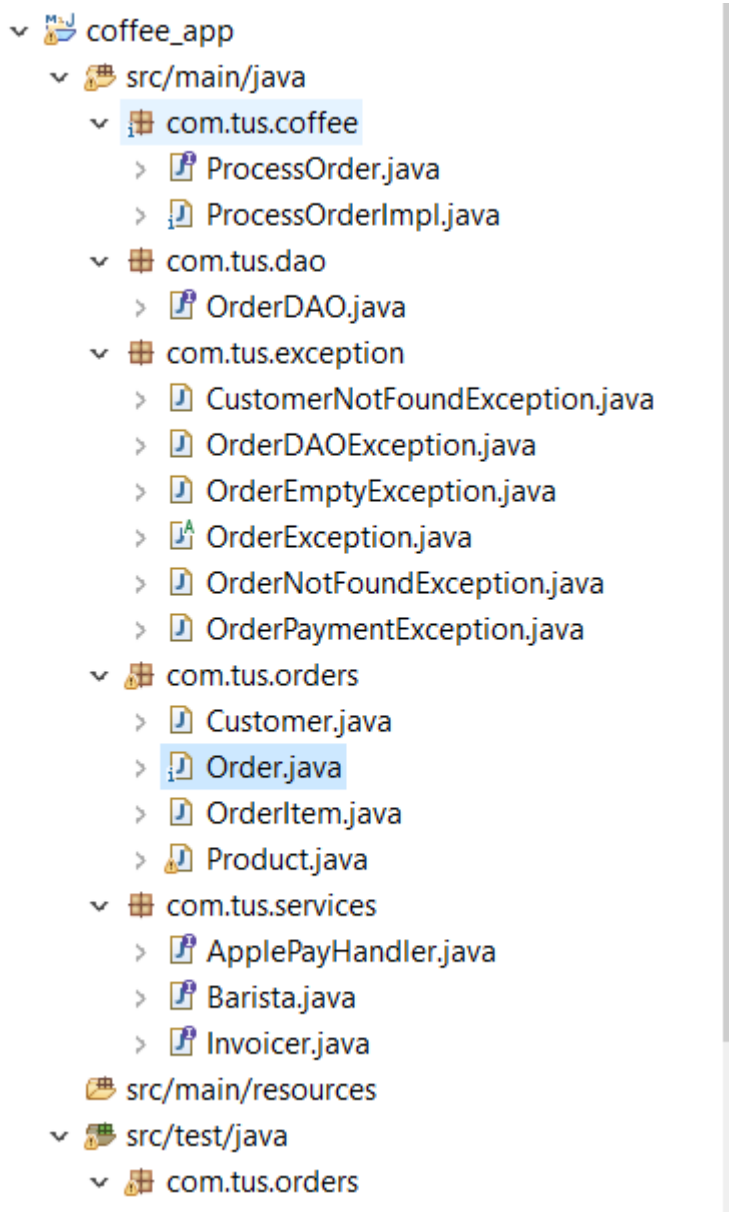


Scenario 5 – OrderEmptyException



Scenario 6 – OrderPaymentException

You are given the classes and interfaces below.



Complete the test class called ProcessOrderImplTest.java that will test ProcessOrderImpl.java.

Test 1 **testCustomerNotFoundException**

Write a test that checks that CustomerNotFoundException is thrown when customer object received from OrderDAO is null. Test that the correct message is received in the exception.

Verify that there is no interaction with the mocks for ApplePayHandler, invoicer or Barista.

Test 2 **testOrderDAOSQLException**

Write a test that checks that OrderDAOException is thrown when OrderDAO throws an SQLException. Check that the correct message is provided in the exception. Verify that there is no interaction with the mocks for ApplePayHandler, invoicer or Barista.

Test 3 **testOrderNotFoundException**

Write a test that checks that OrderNotFoundException is thrown when getOrder returns a null value. Check that the correct message is provided in the exception. Verify that there is no interaction with the mocks for ApplePayHandler, invoicer or Barista.

Test 4 **testOrderEmptyException**

Write a test that checks that OrderEmptyException is thrown when the ArrayList of OrderItems is empty. Check that the correct message is provided in the exception. Verify that there is no interaction with the mocks for ApplePayHandler, Invoicer or Barista.

Test 5 **testOneItemInOrder**

Write a test that checks the correct interactions with the mocks ApplePayHandler, Invoicer or Barista (including actual values) when there is one item (with a quantity of one) in the list of orders.

Test 6 testTwoItemsInOrder

Write a test that checks the correct interactions with the mocks ApplePayHandler, Invoicer or Barista (including actual values) when there are two items (each with a quantity of one) in the list of orders.

Test 7 testOneTeaTwoCoffeeItemsInOrder

Write a test that checks the correct interactions with the mocks ApplePayHandler, Invoicer or Barista (including actual values) when there are two items (one with a quantity of one and the other with a quantity of two) in the list of orders.

Test 8 testOrderPaymentException

Write a test that checks the correct interactions with the mocks ApplePayHandler, Invoicer or Barista when ApplePayHandler throws OrderPaymentException. Check that OrderPaymentException is received with the correct values.

Because ApplePayHandler doesn't return a value the specifying the behaviour of the mock is as follows:

```
doThrow(OrderPaymentException.class).when(applePayHandler).pay(anyString(), anyString(), anyDouble());
```