BIRZEIT UNIVERSITY

Computer Science Department

COMP2310 (2nd Semester 2022/2023)

Assignment Three ***Due Date: Thu. 08/07/2023 by 10:00 pm***

Notes:

1. The assignment should be submitted by the due date and time (Thu 08/07/2023 at 10:00 PM) ***(Late Assignments will not be accepted for any reason) on ITC by replying to the assignment message.***
2. The assignments are *individual* effort, and copying the assignment will be treated as a cheating attempt, which may lead to *FAILING* the course.

You have given the UML diagram below for an order management system. Your task is to write a complete Java program applying the OOP programming principles, mainly inheritance, polymorphism, and encapsulation, to implement the program.

Notes:

1- Order Class is an abstract class that has an abstract method, “createOrder”. There are two types of orders “MailOrder” and “PhoneOrder”. The mail order has a post date, the date an order has been sent. Phone Order has the name of the customer representative who takes the order over the phone call and the duration.

2- Order must implement the Comparable interface, and the orders are compared based on the order dates.

3- You must override toString methods as needed to return an appropriate string representation of the objects.

Write a driver class that allows the user to add a customer to the customer list, create new orders, and inquire about an order. Your driver class should do the following:

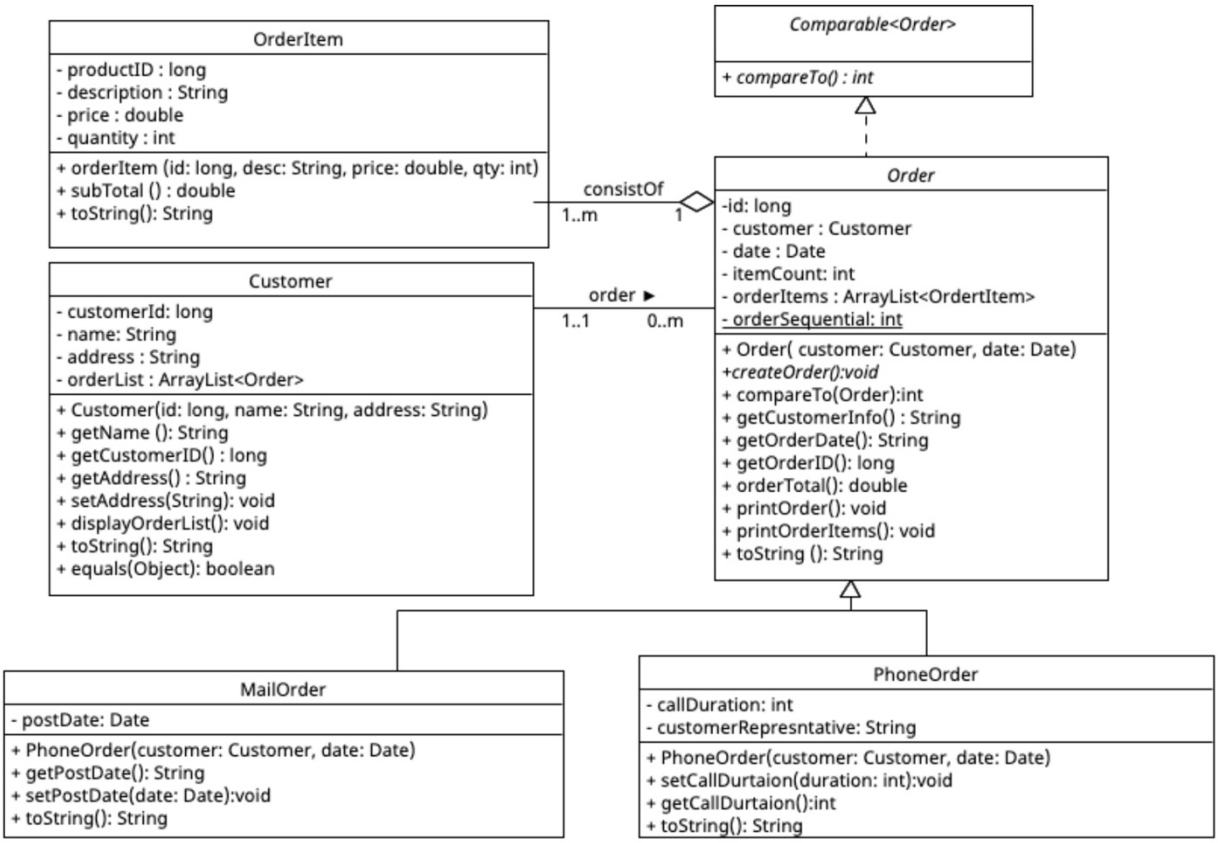
1) Prompt the user with the following menu choices:

1) Add a customer
2) Make a new Order
3) Display Customer’s Orders
4) Quit

2) Execute the user's choice and then continue till they quit.

What each menu choice should do:

1) Adding a new customer to the customer list: Ask the user for the customer's details as shown in the UML diagram, then add the customer to the customer list. Use ArrayList to store the customer list. You must make sure the customer is not on the list; customer objects are the same if both customers have the same customer ID. Therefore, **you must override the equals method**. If the customer exit, display an informative message to indicate that the customer is already on the list.
2) Make a new Order. The orders are created following the following steps:
    a. The system (your program) should ask the user to enter the customer ID, then the system should retrieve the customer from the customer list. If the customer has not been found, the system asks the user either to enter a different customer ID or create a new one. Also, the user should be given the option to quit.
    b. The system asks the user for the order type, either Mail order or Phone Order
    c. The order object is created using the constructors as shown in the UML. Notice that the order ID is assigned to the order by the constructor. The order ID is a sequential number, **you need to use static data members to keep track of the last value.**
    d. Then invoke the createOrder method to ask the user to enter order details. For MailOrder, the user must enter the post date. For Phone orders must enter the name of the customer representative. Then the user enters the order items as shown in the UML. The order can consist of many Order Items. The OrderItems objects are added to the items array list. The user can enter as many items as s/he wants. When the user finish entering the order item details, **only** for Phone orders, the system asks the user for the call duration.
    e. When the user finishes creating the order, the order must be added to the customer's orderList, and the order details must be displayed to the user by invoking printOrder method, which displays the following information about the order: order id, order date, customer name, customer address, *for Mail orders post date, for Phone orders order name of customer representative and call duration.* Then order items are displayed each item in a separate row. Each row has product ID, description, price, quantity, and subtotal. Finally, the order total amount.
3) Display Customer's Orders: ask the user for the customer ID; if it has been found, display Customer's Orders which must be sorted according to the Order Date. Otherwise, display an informative message indicating that the customer has not been found in the customer list.


OrderItem
- productID : long
- description : String
- price : double
- quantity : int
+ orderItem (id: long, desc: String, price: double, qty: int)
+ subTotal () : double
+ toString(): String
Comparable<Order>
+ compareTo() : int
Order
-id: long
- customer : Customer
- date : Date
- itemCount: int
- orderItems : ArrayList<OrdertItem>
- orderSequential: int
+ Order(customer: Customer, date: Date)
+createOrder():void
+ compareTo(Order):int
+ getCustomerInfo() : String
+ getOrderDate(): String
+ getOrderID(): long
+ orderTotal(): double
+ printOrder(): void
+ printOrderItems(): void
+ toString (): String
consistOf
1..m
1
Customer
- customerId: long
- name: String
- address : String
- orderList : ArrayList<Order>
+ Customer(id: long, name: String, address: String)
+ getName (): String
+ getCustomerID() : long
+ getAddress() : String
+ setAddress(String): void
+ displayOrderList(): void
+ toString(): String
+ equals(Object): boolean
order ►
1..1
0..m
MailOrder
- postDate: Date
+ PhoneOrder(customer: Customer, date: Date)
+ getPostDate(): String
+ setPostDate(date: Date):void
+ toString(): String
PhoneOrder
- callDuration: int
- customerRepresntative: String
+ PhoneOrder(customer: Customer, date: Date)
+ setCallDurtaion(duration: int):void
+ getCallDurtaion():int
+ toString(): String