Joseph Lerman's MPP Midterm January 26, 2019

Please make sure that you write **VERY CLEARLY**! If I can not read your writing you will **Not get credit** for the question.

For All code below, if you need to create another helper method that is fine. Just write the code for the helper method.

1) What is your name?

Note that there is a mistake in the class diagram below. An OrderLine has only one IProduct inside it, Not a list of IProduct.

2) (16 points). **Inside the ACustomer class**, write a method called 'averageOrderPrice' which returns the average order price of all the orders for that customer. **The class diagram is below**.

The first line of the method is : double averageOrderPrice ()

---

3) (16 points). **Inside the ACustomer class**, write a method called 'smallestPriceInsideAnOrderline' which returns the data member 'price' inside an orderLine object, which has the smallest value for that customer, **considering All of the Orders** and **All of the OrderLines for that customer**.

The first line of the method is : double smallestPriceInsideAnOrderline ()

---

4) (16 points). Add the following to the class diagram below. A MainApp class that has main in it. The MainApp class has a List of 'ICustomer' objects inside it. (Draw this new class in the class diagram below to make things clearer for yourself.)

Let's add another class, a **child** to the 'ACustomer' class called MilitaryCustomer. A MilitaryCustomer is for example, 'The Navy'.

A MilitaryCustomer object has a data member inside of it called 'maximum_That_Can_Spend_Per_Month', which is a double that tells how much money that military customer can spend in a month.

Write a method inside the MainApp class called 'budget_For_Each_Military_Customer' that prints out the name of the military customer, and the 'maximum_That_Can_Spend_Per_Month' for each of the MilitaryCustomer objects.

Sample output could look like :

The Navy $1,200,000
The Army $1.300,000
Air Force $1,400,000

5) (12 points). Add the following to the class diagram below. A MainApp class that has main in it. The MainApp class has a List of 'ICustomer' objects inside it. Write a method inside the MainApp class called 'customerSmallestPoints'. This method prints out the name of the customer, and, looking at All of the Orders for that customer, and All of the OrderLines for that customer, it prints out the smallest 'points', which is a data member in OrderLine.

Sample output could look like :

John Jones 1
Tom Smith 0.5
Helen Jones 2

---

6) (18 points). Design a UML class diagram for a small system that computes order prices for a greeting card company. This company sells packets of greeting cards. There are three general categories: thankyou cards, birthday cards, and holiday cards. There are three types of holiday cards: Christmas cards, Father's Day cards, and Mother's Day cards. Holiday cards are sometimes discounted (e.g., after a holiday has passed), but not thankyou cards or birthday cards.

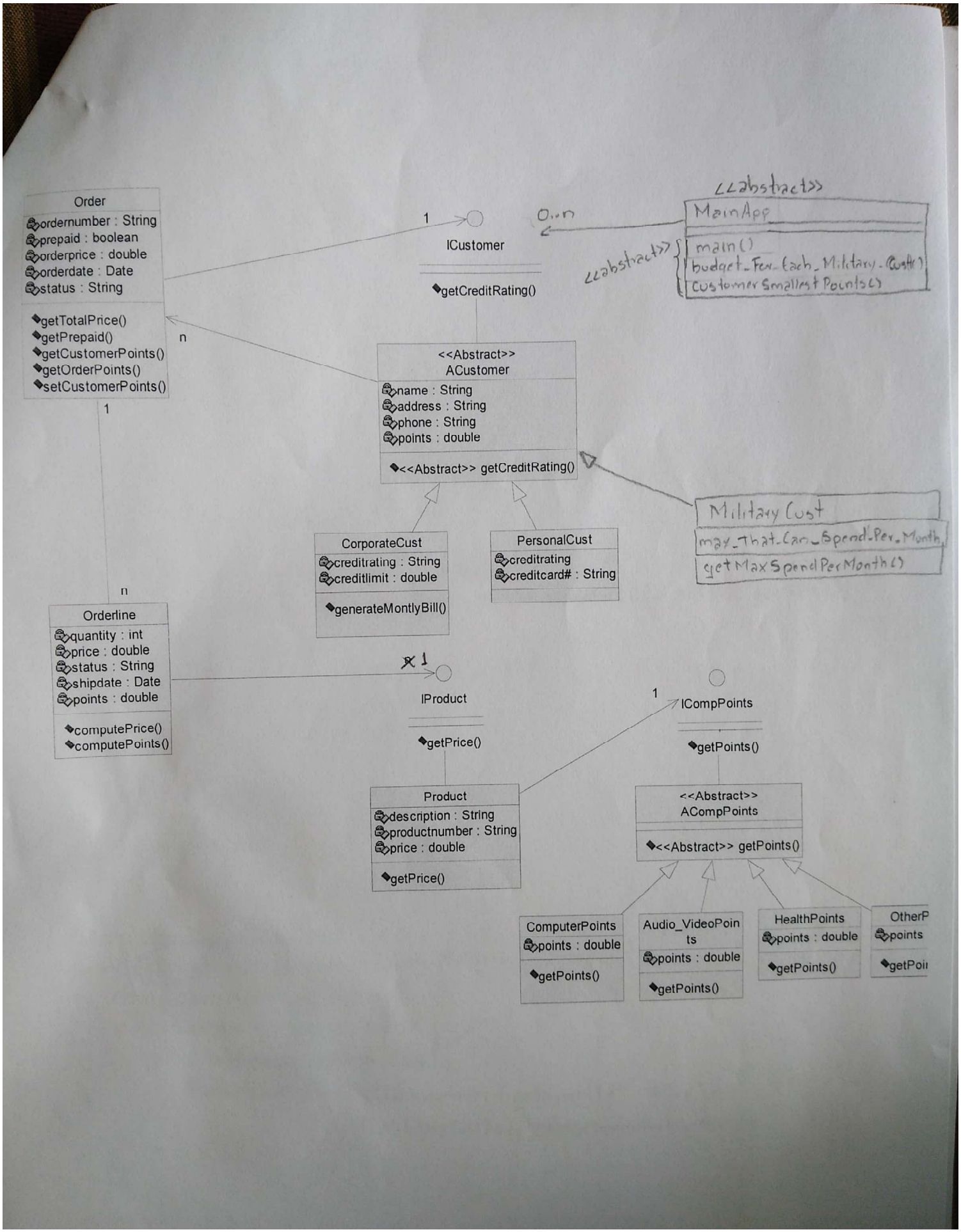
A customer can buy one or more packages of cards, and can also subscribe to receive an annual selection of Christmas cards. Clients who have a subscription receive a 10% discount on any purchases. This application will only take orders for purchases, but needs to give discounts to subscription customers. The company's managers have indicated that they may want to have other discount categories in the future.

For every client we record their name, address, and age. For every order we record the date of the order, the names of the card packages and the base price. A receipt will need to be printed that shows the date, card package names, base prices, any seasonal discounts, and any customer discounts.

Make sure you specify cardinalities (multiplicities), relationship types and direction, attributes and methods, interfaces and abstract classes. You do not have to use interfaces unless they are required, but make sure to indicate classes that should be abstract. You do not need to show getter and setter methods. You do not need to show attribute types, but you should show method signatures. Your design should follow the open-closed principle—in particular, allow for polymorphism where appropriate.

7) (3 points). Write a paragraph or two relating SCI to the 'Class' diagram, which is used in UML.

Good Luck!



2) Inside the ACustomer class:

```
public double averageOrderPrice(){
    double sum = 0.0;
    double count = 0.0;
    for(Order o : orderList){
        count++;
        sum += o.getTotalPrice();
    }
    return (sum/count);
}
```

3) Inside the ACustomer class:

```
public double smallestPriceInsideAnOrderline(){
    double smallest = orderList.get(0).getOrderlineList().get(0).getPrice();
    for(Order o : orderList){
        for(Orderline ol : o.getOrderlineList()){
            if(smallest > ol.getPrice()){
                smallest = ol.getPrice();
            }
        }
    }
    return smallest;
}
```

4) Inside the MainApp class:

```
public abstract void budget_For_Each_Military_Customer(){
    String output = ""; // Not used
    for(ICustomer c : customerList){
        if(c instanceof MilitaryCust){
            System.out.printf(((ACustomer)c).getName()+" $%.f\n",
                              ((MilitaryCust)c).getMaxSpendPerMonth());
        }
    }
}
```

15 1/2 / 16

5) Inside the MainApp class:

```
public abstract void customerSmallestPoints() {
    for(ICustomer c: customerList){
        double smallestPoints = ((ACustomer)c).getOrderList().get(0).getOrderlineList().get(0).
                                getPoints();
        for(Order o : c.getOrderList()){
            for(Orderline ol: o.getOrderline()) {
                if(smallestPoints > ol.getPoints()){
                    smallestPoints = ol.getPoints();
                }
            }
        }
        System.out.println(((ACustomer)c).getName() + " " + smallestPoints);
    }
}
```

very Good!

7) Let's remember one of the principles of STC: "The whole is more than the sum of the parts", that's exactly what happen with the Class Diagram, you can have many classes, interfaces, objects in general, but without the relation between them, they're just alone parts, no meaning, but when you associate them each other and make a whole, then you have meaning, and expression for something bigger than every single part place together.

Excellent!