C212 Lab 7

Intro to Software Systems

Instructions:

- Review the requirements given below and complete the required work. Please compress all files into a zip file and submit it through Canvas. Do not include any binary/class files. ● The grading scheme is provided on Canvas

Objectives (100 points: 25+75+20)

- The goals of this lab are to review some very important OOP concepts that come up in job interviews: ○ Object oriented design ○ interfaces

Part1: Tic Tac Toe Player (25pt + Bonus 20pt)

Design the UML class diagram of a tic-tac-toe game that allows two human players to play against each other.

1. Design your own appropriate UML about the method, you can either submit the diagram as a picture or as a pdf exported in lucidchart.
    a. Helpful information:
        i. Use an interface containing all your methods that needs to be implemented

Here is an extra resource about UML instruction: https://www.lucidchart.com/pages/uml-class-diagram **Remember:** you **don't need** to pay for lucidchart, it's **free to use**.

2. **(Bonus 5pt)**
   Implement the classes you defined in your UML diagram. Below is an example of the running result. Don't need to do the same things as the example, for instance you can always design how you want to show users your board, your key instruction and so on. The purpose of this Part is to build your own program by following the rule of TicTacToe.

```
key instruction is shown below
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
|---|---|---|
Round for 'O'
key instruction is shown below
1
|---|---|---|
| O |   |   |
|   |   |   |
|   |   |   |
|---|---|---|
Round for 'X'
2
|---|---|---|
| O | X |   |
|   |   |   |
|   |   |   |
|---|---|---|
Round for 'O'
key instruction is shown below
3
```

```
key instruction is shown below
3
|---|---|---|
| O | X | O |
|   |   |   |
|   |   |   |
|---|---|---|
Round for 'X'
4
|---|---|---|
| O | X | O |
| X |   |   |
|   |   |   |
|---|---|---|
Round for 'O'
key instruction is shown below
5
|---|---|---|
| O | X | O |
| X | O |   |
|   |   |   |
|---|---|---|
Round for 'X'
```

```
6
|---|---|---|
| O | X | O |
| X | O | X |
|   |   |   |
|---|---|---|
Round for 'O'
key instruction is shown below
7
|---|---|---|
| O | X | O |
| X | O | X |
| O |   |   |
|---|---|---|
Player1 wins the game
```

3. **(Bonus 15 pt):**
    a. Add features with human vs computer, which computer will know how to defend if human is going to win the game, and also knows how to make an attack move when no need to defend.
    b. Remember also to update your UML, so that it matches your program.
    c. Here an example below:

```
key instruction is shown below
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
|---|---|---|
Round for 'O'
key instruction is shown below
1
|---|---|---|
| O |   |   |
|   |   |   |
|   |   |   |
|---|---|---|
Round for 'X'

2 ,3 ,4 ,7 ,5 ,9 ,
|---|---|---|
| O |   |   |
| X |   |   |
|   |   |   |
|---|---|---|
Round for 'O'
key instruction is shown below
```

Here is when computer defend,

```
Round for 'O'
key instruction is shown below
5
|---|---|---|
| O |   |   |
| X | O |   |
|   |   |   |
|---|---|---|
Round for 'X'
|---|---|---|
| O |   |   |
| X | O |   |
|   |   | X |
|---|---|---|
Round for 'O'
key instruction is shown below
8
|---|---|---|
| O |   |   |
| X | O |   |
|   | O | X |
|---|---|---|
Round for 'X'
|---|---|---|
| O | X |   |
| X | O |   |
|   | O | X |
|---|---|---|
```

```
Round for 'O'
key instruction is shown below
7
|---|---|---|
| O | X |   |
| X | O |   |
| O | O | X |
|---|---|---|
Round for 'X'
|---|---|---|
| O | X | X |
| X | O |   |
| O | O | X |
|---|---|---|
Round for 'O'
key instruction is shown below
6
|---|---|---|
| O | X | X |
| X | O | O |
| O | O | X |
|---|---|---|
Tie Game
```

Part2: Retail Store

Design a system that can stimulate customers who plan on going shopping during the COVID-19 pandemic. The stores within your system should only allow 5 customers, and customers should only be let in the store if they are wearing a mask. Customers should have different budgets for each store, and as they buy different items, their budgets should be updated accordingly. With that in mind, the customer should only be able to buy an item if they can afford it. Additionally, when a customer decides to check out of a store, they should be given a receipt with a description of the items, quantity, and the total amount they spent at the store. Your system should be centered around the *RetailDriver* class, which should contain your main menu, a list of all the stores, and a list of all the customers. A few other classes that your code will need to include are **Customer**, **Retail Store, Receipt**, and **Item**. You should follow the design process that was described in Chapter12. Be sure to include a ***UML Diagram.***

The *RetailDriver* class:

- Should have a list of all the Stores and Customers
- An interactive main menu where you can:
    - Create a New Store
        - Create a new Retail store instance with user provided name.
    - Create a New Customer and attempt to enter them in the store
        - Ask the user for the name of the Retail store and check to see if it is currently within the retail store list.
        - Create a new Customer instance with a given name, budget, and masks status, and add them to the given store **if the customer limit hasn't gone over five people**, and if they are wearing a mask.
        - Enter the Customer Menu
        - Prompt the name of the customer and see if their name appears in the customer list.
        - Prompt the name of the store and check to see if the store exists inside of the list of stores.
        - Enter the customer menu
            - Should get the instance of the customer and run a customer menu method. The method should already consider that they are in the perspective of being inside of a *specific* store. (Note that this menu is a nested menu within the RetailDriver menu – when you leave the Customer menu, you should return back to the RetailDriver)
    - Add new Item to a Store
        - Creates a new instance of an item and adds it to the list of items from the store. There are no limits to how many items are in the store.
    - End the program

The RetailStore class:

- Should maintain a list of customers that are currently in the store
- Maintain a list of stocked items and their quantities
- Hint: If you are not Familiar with Dictionaries/HashMap's in Java, you will have to use Parallel ArrayLists.
    - Example:
        - If StockedItems = ["Socks", "Gift Card", "Shoes"] and Quantity = [4,3,2]
        - Then Socks quantity is 4, Gift Card quantity is 3, etc. *The Customer* class:
- It should be able to store all the items in their cart currently
    - Additionally, you should be able to store the quantity of items similarly to how the stocked items and quantities are being stored in the Retail Class.
- In this class you will need to have a Customer Menu, where the customer can choose to:
    - Add to cart
        - Adding to cart should appropriately update the quantities in the store and your shopping cart.
        - Adding to cart should update customer budget as well.
    - Display Their Current Shopping Cart
    - Check out/ Leave Store
        - When you check out of the store, you will need to update the store customer list, and create a new instance of the receipt class.
            - Additionally, you will want to display your receipt out to the console as well.
    - Exit the Menu

The *Item* class:

- The item class only needs to store the name and price of the item

The *Receipt* class:

- Create a new receipt based upon the items and the shopping cart.
- You could create a displayRecepit() method here that could return a string representation of the Recepit.

Helpful Information:

- If it was not clear already, your system should always be prompting for the names (i.e store name, customer name, and item name).
  - Remember that ultimately, if you do your code a different way than as described in here, and it follows Chapter 12 Guidelines, then it will be considered a valid approach and you will receive full credit.
- Remember that if your system works and is done in a different way than as described on this PDF as long as it follows all of the practices described in Chapter 12, then your code will be considered valid, and you will receive full credit. Just be sure to create your UML diagram.
- It would be very wise to do your UML diagram first and then jump into code as you will be confused and overwhelmed if you just jump into creating the program.
- Your Driver class should be static and should include a main method. It is up to you how you want to organize what happens in there.
  - You can place the list of **all** the stores, and **all** the customers inside of the driver class.
- Your system does not need to handle duplicate customer names, store names, or item names.
  - Meaning if your code works with different customer names, different Item names, and different store names, you will be all good.
- Remember that when you enter a customer into a store remember to take user input from a Scanner or a **BufferedReader** to determine which store, they are in.
- If you are using Parallel ArrayLists make sure you are handling edge cases
  - Generally speaking, the ArrayLists should always have the same size, so if you erase an element from one ArrayList, you need to erase an elements from the other.
- If you would prefer to use a HashMap to make the System more efficient, then here are some resources that will help you out
  - https://www.geeksforgeeks.org/java-util-hashmap-in-java-with-examples/
  - https://www.youtube.com/watch?v=ceh8s-r53m0