MILESTONE 2

Overview

In this milestone, you will be defining and coding the beginnings of the *StoreView* and *ShoppingCart* classes. You will likely have to refactor some of your code from Milestone 1 and make additions to the *StoreManager* class.

*Keep in mind that most of the tasks you will be required to do are relatively **open-ended**. You must justify any decisions you made that were not obvious in your Change Log. Refer to the examples in Milestone 1.*

The ShoppingCart Class

The *ShoppingCart* class is very similar to the *Inventory* class in purpose. A *ShoppingCart* will keep track of the state of the user's shopping cart. It should maintain the contents that the user adds to it. Remember, the user can also remove items. *ShoppingCarts* are maintained for every user by the *StoreManager* class as previously mentioned.

The StoreView Class

The *StoreView* class will manage the GUI for your system. For this milestone, it will be textually displayed in the console. Note that the *StoreView* class must contain the **main()** method (i.e. the entry point of the program).

Each instance of the *StoreView* class contains a *StoreManager* and a unique *cartID* used to identify the user of the system. In our case, each *StoreView* instance will have a unique *cartID*. You can choose how the *StoreView* class will obtain the *StoreManager* and *cartID* (you can pass them via a constructor if you want). However, the *cartID* should be generated by the *StoreManager* class. An example of how the *StoreView* textual User Interface (UI) can look is shown in the figure below.

```
CHOOSE YOUR STOREVIEW >>> 0
CART >>> 0
Enter a command...
browse
|--------------------THE COURSE STORE--------------------|
\-------------------------BROWSE-------------------------/
Type 'help' for a list of commands.

Stock | Product Name | Unit Price
76 | SYSC2004 | $100.0
0 | SYSC4906 | $55.0
32 | SYSC2006 | $45.0
3 | MUSI1001 | $35.0
12 | CRCJ1000 | $0.01
132 | ELEC4705 | $25.0
322 | SYSC4907 | $145.0

GO TO ANOTHER STOREVIEW? (y) >>> n
CART >>> 0
Enter a command...
addtocart
|--------------------THE COURSE STORE--------------------|
\--------------------------ADD-------------------------/
Stock | Product Name | Unit Price | Option
76 | SYSC2004 | $100.0 | (0)
0 | SYSC4906 | $55.0 | (1)
32 | SYSC2006 | $45.0 | (2)
3 | MUSI1001 | $35.0 | (3)
12 | CRCJ1000 | $0.01 | (4)
132 | ELEC4705 | $25.0 | (5)
322 | SYSC4907 | $145.0 | (6)
Option:
```

In the example, the user types a command to enter a certain subroutine. Selections can then be made by using the number displayed adjacent the option on the page.

This is just an idea for how you can implement your UI. In the end, you will be creating a GUI anyway. Remember to document how a user should navigate your system. This should be a combination of a good explanation in your Change Log, and helpful command line prompts.

You should think of each instance of the *StoreView* class as a separate user browsing the store. Like multiple users on the internet browsing an online store. However, your store is obviously much simpler.

Some code is supplied in the Appendix of this document that might be helpful for simulating multiple users connected to the store.

Updates to the StoreManager Class

The *StoreManager* needs some new functionality. Now, it will not only be managing the *Inventory*, but it will also be managing user *ShoppingCarts*.

Each user that connects to the store (new *StoreView* instance) should have their own unique *ShoppingCart*. If a user adds something to their cart, the Product's stock in the store Inventory should be decreased accordingly. A user can also remove items from their cart. Note that if the user removes a product from the shopping cart, the inventory must also be updated accordingly.

Upon request, the *StoreManager* should return a new, unique *cartID*. This means *StoreManager* should be keeping track of *cartIDs* in some way. It could be as simple as having a counter that increments every time a new *ShoppingCart* is made. This implementation is ultimately up to you. Just be sure to document what you do!

A user needs to be able to checkout once they are ready (your method for processing a transaction from Milestone 1 will likely need to be changed, or completely removed…). This method should return the total and summary of the items in the cart (print it for the user to see). You can choose to disconnect the user at this point or reset the cart – up to you! If the user quits before checking out, any items in the cart should be returned to the Inventory stock. **Note**: quitting means the user entered 'quit', not your program suddenly closes; you do not need to worry about that.

Now that you will be implementing the UI for the store, you need some way to get the information needed to drive this UI. *StoreManager* should have some methods that return needed information about *ShoppingCarts*, or available *Products*. The *StoreView* class will be using this information to populate the UI for the user. Remember, all communication by the *StoreView* class must be done with the *StoreManager* only!

Questions

1. What kind of relationship is the StoreView and StoreManager relationship? Explain.
2. Due to their behavioral similarity, would it make sense to have ShoppingCart extend Inventory, or the other way around? Why or why not?
3. What are some reasons you can think of for why composition might be preferable over inheritance? These do not have to be Java-specific.
4. What are some reasons you can think of for why inheritance might be preferable over composition? These do not have to be Java-specific.

Milestone 2 Deliverables

1. The following classes completed according to Milestone 2 specifications: *StoreView.java*, *ShoppingCart.java*, *Inventory.java*, *StoreManager.java*, and . Do not jump ahead!
2. Everything applicable to Milestone 2 from General Submission Requirements. Do not forget the updated UML diagram, to document all your classes, and the report with the change log and the answers to the questions.

APPENDIX

```
public static void main(String[] args) {
    StoreManager sm = new StoreManager();
    StoreView sv1 = new StoreView(sm, sm.assignNewCartID());
    StoreView sv2 = new StoreView(sm, sm.assignNewCartID());
    StoreView sv3 = new StoreView(sm, sm.assignNewCartID());
    StoreView[] users = {sv1, sv2, sv3};
    int activeSV = users.length;

    Scanner sc = new Scanner(System.in);

    while (activeSV > 0) {
        System.out.print("CHOOSE YOUR STOREVIEW >>> ");
        int choice = sc.nextInt();
        if (choice < users.length && choice >= 0) {
            if (users[choice] != null) {
                String chooseAnother = "";
                while (!chooseAnother.equals("y") && !chooseAnother.equals("Y")) {
                    // this implementation of displayGUI waits for input and displays the page
                    // corresponding to the user's input. it does this once, and then returns
                    // true if the user entered 'checkout' or 'quit'.
                    if (users[choice].displayGUI()) {
                        users[choice] = null;
                        activeSV--;
                        break;
                    }
                    System.out.print("GO TO ANOTHER STOREVIEW? (y) >>> ");
                    chooseAnother = sc.next();
                }
            } else {
                System.out.println("MAIN > ERROR > BAD CHOICE\nTHAT STOREVIEW WAS DEACTIVATED");
            }
        } else {
            System.out.println(
                    String.format("MAIN > ERROR > BAD CHOICE\nPLEASE CHOOSE IN RANGE [%d, %d]",
0, users.length - 1)
            );
        }
    }
    System.out.println("ALL STOREVIEWS DEACTIVATED");
}
```