CS6311
Final Exam

The Great Purchase Kiosk

Objectives:

- Apply the programming techniques taught this semester to implement a small software application

Part 2 (26 points)

Be sure to read through all items before you begin. The following descriptions are not written in any particular order. You should apply the iterative development techniques we've stressed this semester to develop your code in an appropriate fashion.

Model

Develop a class named `Customer` such that it defines:

Instance Variables (only):

- Customer name (String)
- Amount to spend (double)
- Shopping cart (ArrayList<Computer>)

Methods:

- A 2-parameter constructor that accepts the `Customer`'s name and amount they have available to spend on the `Computer`. Also be sure that the shopping cart is initialized so that it can hold new computers. Be sure to error check each parameter so that if invalid values are passed, the default value for that data type is stored (the Empty String for String variables and zero for numeric variables). Use these values to initialize the instance variables.
- Accessor methods for the Customer's name and amount to spend
- `getCart` – This method will return a `String` listing the complete information about each computer to be purchased on its own line. If the shopper's cart is empty, then this method will return an appropriate String describing that situation.
- `enoughMoneyToBuy` – This method accepts a `Computer` object and a quantity and return a Boolean value. This method will check to be sure the Customer has enough money left to purchase the given quantity of computers. If they do, then it will return true, otherwise it will return false.
- `addComputer` – This method accepts a `Computer` object and a quantity and not return any value. If this Customer has enough money and there are enough in stock to be purchased, then this method will add the appropriate computer to the cart. Note

that you'll need to be sure to remove those computers from the inventory and reduce the amount the Customer has to spend appropriately.

- `completePurchase` – This method will accept no parameters and have no return value. This method will simply clear the shopping cart. (Yes, in the real world you would need to certify the transaction and update the inventory, but we're not giving our users the option of backing out of a transaction after they've put something in the cart. This is just more than I want you to do this semester – but maybe next semester we can consider something more realistic like this.

Package - View

Develop a class named `KioskTUI` such that it defines:

Instance Variables (only):
- An `InventoryManager` object
- A `Scanner` object

Methods:
- A 1-parameter constructor that accepts an `InventoryManager` object and initializes the instance variables.
- `runKiosk` – This method serves as the 'director' of the main user. Basically this method is just going to give the user the ability to choose to enter the manager's application (where new items can be added, etc) or the shopper's application (where shoppers can add items to a cart and purchase them).
- A variety of other `private` helper methods that will provide the functionality for allowing the user to interact with the program and display the necessary output. Every method should perform exactly one cohesive task and should have an appropriate name to describe this task. As a general rule, each menu option will have at least one method associated with it.

Modify the existing `GreatPurchaseDriver` class such that it makes a `KioskTUI` object, rather than a `ManagerTUI` object and then uses this object to call `runKiosk()` instead of calling `runManager()`.

Develop a class named `CustomerTUI` such that it defines:

Instance Variables (only):
- An `InventoryManager` object
- A `Customer` object
- A `Scanner` object

Methods:
- A 1-parameter constructor that accepts an `InventoryManager` object and initializes the `InventoryManager` and `Scanner` instance variables

- `getCustomerInformation` – This private helper method will accept no parameters and not return any value. Code inside this method will prompt the user for a name and an amount to spend. With that information, it will then initialize the `Customer` instance variable.
- `runCustomer` – This method will serve as the 'director' of the user interface by calling on private helper methods to do their work, as appropriate.
- A variety of other `private` helper methods that will provide the functionality for allowing the user to interact with the program and display the necessary output. Every method should perform exactly one cohesive task and should have an appropriate name to describe this task. As a general rule, each menu option will have at least one method associated with it.

Application Functionality

The user will interact with the application using a console-based, textual user interface (TUI). This interface should be menu driven. To keep things simple, menu options will be numbered so that the user can type a single integer value as a menu choice (instead of typing in a character or word). See the sample output for an example on how this might be done.

Menu options include:

1. View computers
   This option will display a listing of all computers (name, price, and quantity) in the inventory

2. Add computer to the cart
   This option will allow the user to type in computer SKU. If a computer with that SKU exists, then the user will be able to enter a quantity. If there are enough on hand and the user has enough money left to make this purchase, then these computers will be added to their cart. A message will also be displayed on the console to let them know how much money they have remaining after the cart is purchased.

3. View cart
   This will display a listing of the name, price, and quantity of computers in the shopping cart

4. View money remaining
   This option will display the amount of money the shopper has left to spend

5. Checkout
   This option will complete the purchase.

6. Quit shopper application
   This option will end the shopper's menu and return to the Kiosk main menu

When all methods are written, close Eclipse, and Zip the folder holding your Eclipse project using 7-Zip. Give your file the name CS6311*YourLastName*FinalPart2.zip. Upload the Zip file to the appropriate link in Moodle.

Moodle is set up to accept exactly one file for this exercise. No big deal, the Eclipse Project you end up with will contain *all* of the code from Parts 1 and 2 mixed together inside the same project folder. And if you followed the directions from Part 1 and uploaded your Part 1 zip file to Moodle already, just know that the file that was present will be replaced by your latest version.

(Note: this means that the Moodle assignment link is a useful backup system – be sure to upload early and often!)