METROPOLITAN STATE UNIVERSITY

ICS 141 - 02: Problem solving with programming
Spring 2023

Assignment 3 – Part2: Objects that contain objects

Out: Wednesday, February, 22nd, 2023
Due: Wednesday, March, 1st, 2023

Total points: 25

Problem Description

Building on ***Part1***, follow the instructions below to add the **CashRegister** to the **RetailStoreApplication** class. At this point you should already have a **RetailStoreDriver** from ***Part1*** and have test the methods in the **RetailItem** class.. The **CashRegister** and the **RetailStoreDriver** should both be in the ***retail*** package. To receive full credits, **create a video recording less than 10 minutes with your face on camera** describing the additional changes to the code. Also upload the java source code.

`class CashRegister`

The CashRegister simulates the sale of a retail item where a cash register is identified by the following instance variables:

1. clerk: a String that represents the name of the employees who processes the purchase.
2. **item**: a ***RetailItem*** that represents the item being sold.
3. quantity: an int variable that represents the number of units being sold of that item.

Implement the following methods for the `CashRegister` class:

1. A constructor that takes as input (1) **the clerk's name**, (2) a **RetailItem** Data type, and (3) a **number to represents the number of units to be purchased**. You can assume that the input quantity is always less than the number of units on hand for the given **item**. The constructor should use the **item**'s `getUnits` method to modify the **item**'s units on hand by subtracting the quantity to be purchased.
2. Getter methods for all the instance variables.
3. **`getSubTotal`**: a method that calculates and returns the sub total of the sale which the quantity being sold multiplied by the **item**'s price.
4. **`getItemAvailabilty:`** this method returns the number of available unitsOnHand of the **items** being purchased.

5. **`modifyQuantity:`** a method that takes a number as input and the method adds this number to the current quantity. Note that the input number may be positive or negative to either increase or decrease the purchased quantity. This method should modify the **item**'s units on hand accordingly. For example, if 2 more units are added on the quantity, then 2 units should be subtracted from the item's units on hand. Similarly, if 2 units to be subtracted from the quantity, then the 2 units should be added back to the **item**'s units on hand.
6. **`getTax`**: a method that takes a double input that represents the tax rate (e.g., 0.03 for 3% tax) and the method then calculates and returns the amount of sales tax on the current purchase.
7. **`getTotal`**: a method that returns the total of the sale which is the sum of the subtotal and tax. The method should take a double input that represents the tax.
8. **`toString`**: a method that returns a string representation of the cash register item that includes clerk's name, details of the item, quantity to be sold, and the sub total sale price. Note that the output string must be nicely formatted with all the data listed in one line.

Class RetailStoreDriver

If you haven't already, crate a driver class to test all the methods that you implemented in both the `RetailItem` and `CashRegister` classes. Basically, the driver should include the following actions:

- Creates two items with your choice of description and price.
- Add 1000 units to the first item and 2000 units to the second item.
- Test all the methods of the `RetailItem` class on the two items.
- Create two cash registers, for each one of the items that are created in the previous step.
- Test all of the methods of the CashRegister class on the two objects that are created in the previous step.

Important Requirements

- The output of our program must be nicely formatted.
- Method names and variable names must exactly match the assignment requirements.
- The program should display your name at the end.
- At the top of the Java files, include a comment with your your name, a brief description of the program and what it does and the due date.
- Add appropriate comments to your code.
- All code blocks must be indented consistently and correctly. Blocks are delimited by opening and closing curly braces. Opening and closing curly braces must be aligned consistently.
- Variable names should convey meaning.
- The program must be written in Java and submitted via D2L.

Submission Instructions

- Draw a UML diagram of your application that includes the **`RetailItem`** and **`CashRegister`** classes but not the **`RetailStoreDriver`** class. Draw the UML in a word document and upload that document to D2L.
- Follow the following steps to upload your code to D2L:
    - Create a java project and call it `<yourlastname>RetailStoreApplication` (e.g., mine will be called `DillonRetailStoreApplication`). o Create three.java files as described above. o Archive **only** your .java files into **one zip** file using Eclipse using the following steps:
        - In Eclipse Project Explorer, right click on the **src** folder of the project and click on Export. Note that you just include only the **src** folder and do not include the bin or any other folder.
        - Choose General->Archive File and click Next. Use the Browse key to choose a folder to store the archive file on your hard drive and give the file the same name as your project (e.g., `DillonAssginment4.zip`), then click Save, then click Finish.
        - Upload the **.zip** file you created to the D2L folder called Assignment 4. It is important that you upload only **one** zip file. Your assignment will not be graded if you upload individual .java files to the drop box.