Programming Assignment

This assignment will allow you to review defining Java classes and the use of an ArrayList. You are to write a set of supporting classes for a simple shopping program. I am providing GUI code (Graphical User Interface) that will provide the "front end" to the program. You are to write the back end (what is often referred to as the "domain specific code").

Below is a screen shot of what the program could look like when the user has selected various items to order.

Prices are expressed using doubles and quantities are expressed as simple integers (e.g., you can't buy 2.345 of something). Notice that some of the items have a discount when you buy more. For example, silly putty normally costs $3.95 each, but you can buy 10 for $19.99. These items have, in effect, two prices: a single item price and a bulk item price for a bulk quantity. When computing the price for such an item, apply as many of the bulk quantity as you can and then use the single item price for any leftovers. For example, the user is ordering 12 buttons that cost $0.99 each but can be bought in bulk 10 for $5.00. The first 10 are sold at that bulk price ($5.00) and the two extras are charged at the single item price ($0.99 each) for a total of $6.98.

At the bottom of the frame you will find a checkbox for an overall discount. If this box is checked, the user is given a 10% discount off the total price. This is computed using simple double arithmetic, computing a price that is 90% of what it would be otherwise. For example, if we turn on that checkbox, the frame looks like this:

You are to implement four classes that are used to make this code work. You should implement a class called Item that will store information about the individual items. It should have the following public methods.

Method	Description
Item(name, price)	Constructor that takes a name and a price as arguments. The name will be a String and the price will be a double. Should throw an IllegalArgumentException if price is negative.
Item(name, price, bulk quantity, bulk price)	Constructor that takes a name and a single-item price and a bulk quantity and a bulk price as arguments. The name will be a String and the quantity will be an integer and the prices will be doubles. Should throw an IllegalArgumentException if any number is negative.
priceFor(quantity)	Returns the price for a given quantity of the item (taking into account bulk price, if applicable). Quantity will be an integer. Should throw an IllegalArgumentException if quantity is negative.
toString()	Returns a String representation of this item: name followed by a comma and space followed by price. If this has a bulk price, then you should append an extra space and a parenthesized description of the bulk pricing that has the bulk quantity, the word “for” and the bulk price.

You should implement a class called Catalog that stores information about a collection of these items. It should have the following public methods.

Method	Description
Catalog(name)	Constructor that takes the name of this catalog as a parameter. The name will be a String.
add(item)	Adds an Item at the end of this list.
size()	Returns the number of items in this list.
get(index)	Returns the Item with the given index (0-based).
getName()	Returns the name of this catalog.
toString()	Returns a String representation of this catalog: name followed by a new line followed by each item in the catalog on a separate line.

You should implement a class called ItemOrder that stores information about a particular item and the quantity ordered for that item. It should have the following public methods.

Method	Description
ItemOrder(item, quantity)	Constructor that creates an item order for the given item and given quantity. The quantity will be an integer.
getPrice()	Returns the cost for this item order.
getItem()	Returns a reference to the item in this order.
toString()	Returns a String representation of this item order: item followed by a space followed by the quantity.

You should implement a class called ShoppingCart that stores information about the overall order. It should have the following public methods.

Method	Description
ShoppingCart()	Constructor that creates an empty list of item orders.
add(item order)	Adds an item order to the list, replacing any previous order for this item with the new order. The parameter will be of type ItemOrder.
setDiscount(value)	Sets whether or not this order gets a discount (true means there is a discount, false means no discount).
getTotal()	Returns the total cost of the shopping cart.
toString()	Returns a String representation of this shopping cart: each item order on a separate line followed on a separate line with “Qualify for discount” or “Does not quality for discount” depending on whether this order qualifies for a discount.

You are not to introduce any other public methods to these classes, although you can add your own private methods. You are allowed to redefine toString in any of these classes (you might find that helpful in testing and debugging your code). You should use an ArrayList to implement the ShoppingCart and Catalog classes, but they should not extend ArrayList. In other words, we are following Joshua Bloch’s tip #14 to favor composition (“has a”) over inheritance (“is a”).

You will probably want to write your own testing code so that you can develop these classes in stages rather than all at once. When you have confidence that your classes are working, you should combine them with the GUI classes to make sure that they are working properly.

Most of these methods are fairly simple to write, but notice that when you add an ItemOrder to a ShoppingCart, you have to deal with replacing any old order for the item. A user at one time might request 3 of some item and later change the request to 5 of that item. The order for 5 replaces the order for 3. The user isn't requesting 8 of the item in making such a change. The add method might be passed an item order with a quantity of 0. This should behave just like the others, replacing any current order for this item or being added to the order list.

In the Item class you need to construct a String representation of the price. This isn't easy to do for a number of reasons, but Java provides a convenient built-in object that will do it for you. It's called a NumberFormat object and it appears in the java.text package (so you need to import java.text.*). You obtain a formatter by calling the static method called getCurrencyInstance(), as in:

```
NumberFormat nf = NumberFormat.getCurrencyInstance();
```

You can then call the "format" method of this object passing it the price as a double and it will return a String with a dollar sign and the price in dollars and cents. For example, you might say:

```
double price = 38.5;
String text = nf.format(price);
```

This would set the variable text to "$38.50".

There are several potential errors that you are required to handle, as outlined above. If the client requests an illegal index for the catalog, the ArrayList you are using will throw an IndexOutOfBoundsException. This is the right thing to have happen, so you don't have to introduce your own check for this case.

You will be graded on program style including the use of good variable names, comments on each class and each method, using local variables when possible, correct use of generics and the other standard style guidelines.

Your classes should be stored in files called Item.java, Catalog.java, ItemOrder.java and ShoppingCart.java. You will need to include the files ShoppingFrame.java and ShoppingMain. from Edmodo in the same folder as your files to run the GUI. You should open and compile ShoppingMain to run the project. Create a project named LastnameShoppingProject that includes the six java files noted above. The project file also needs to be in the same folder. Name your folder LastnameShoppingProject and turn in a zip file of this folder with these files into Edmodo.