CS211 Programming Assignment (revised 2019)

You are to write a set of supporting classes for a simple shopping program.

This assignment uses ArrayList two ways, you are given a program that HASA (database language for "has a") ArrayList, then you create a ShoppingCart Class where ISA (yes, that's "is a") ArrayList. The GUI program (Graphical User Interface) provides the "front end" to the program, you are writing the back end which is often referred to as the "domain specific code," or the data structures that work behind the scenes. Here's the front end after some user selections:

Above is a screen shot of what the program might look like when the user has selected various items to order. The terms "item" and "SKU" can be considered interchangeable, like when you go to the store and select and item, you really place a SKU into your cart. And if you buy more, we need to know the number selected.

Prices are expressed using doubles and quantities are expressed as simple integers (e.g., you can't buy 1.23 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. Un-check removes discount. This is computed using simple double arithmetic, computing a price that is 90% of what it would be otherwise.

You need to add 3 classes that are used to make this code work:

SKU (Stock Keeping Unit) will store information about the individual items. SKU is the term for what a barcode means in the store. This Class must have the following public methods and fields.

Method	Description
SKU(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 is price is negative.
SKU(name, price, bulk quantity, bulk price)	Constructor that takes a name and a single-SKU 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 SKU: name followed by a comma and space followed by price. With a bulk price, you must append an extra space and a parenthesized description of the bulk pricing that has the bulk quantity, the word "for" and the bulk price (see example at top of GUI)
equals(other)	Returns a Boolean that helps us know if two SKU's are really the same. Each SKU object must get a unique id number, called a primary key. We need to know when an identical SKU is selected, so we can remove the old and replace with the new.
getSKU()	Returns the int unique SKU number for this item

Above requires at least 5 data fields, all of which must be private.

In addition, the SKU Class must have a private static int pkey_next = 123018; which will define the starting "primary key" identification number for items we have in our store. It is "static" so there is only one for all objects from this Class. This quarter we start at #123018. When you create a new object from your Class code, you simply ++ the pkey. Now we know what static is really for!!!

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

Method	Description
NumSelected(SKU, quantity)	Constructor that creates an order for the given SKU and given quantity. The quantity will be an integer.
getPrice()	Returns the cost for this order.
getSKU()	Returns a reference (i.e. an SKU reference) to this order.
toString()	Not required for GUI, but nice for console debugging.

You should implement a class called ShoppingCart that extends ArrayList<NumSelected> so it stores information about the overall order. It must have the following public methods.

Method	Description
ShoppingCart()	Constructor that creates an empty list of orders.
add(NumSelected yes)	Adds an order to the list, replacing any previous order for this SKU(s) with the new order. Return true when added successfully.
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()	Not required for GUI, but nice for console debugging.

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 a NumSelected 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 is NOT 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 SKU 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 SKU.java, NumSelected.java and ShoppingCart.java. You will need to place ShoppingFrame.java from the Canvas web site in the same folder as your ShoppingCart, NumSelected, and SKU Classes to run the GUI. You should open and compile ShoppingFrame to run the program.