**Lab: FoodItem** **20 Points**

**Goal:** Create a class which will create food item objects. This class will be used in a supermarket's inventory and checkout systems software.

**Part 1:**

You should make up appropriate, descriptive ***instance variable*** (also called ***field***) names. Each food item object may contain the following data:

* description of the item
* unit sale price to supermarket's customers
* unit cost price from supermarket's suppliers
* number of units in stock
* weight of one unit in ounces

**Part 2:**

This class should have constructors.

* First write the constructor that takes enough arguments (parameters) to initialize all its instance variables (fields).
* Write two more constructors: one ***default constructor***, and one that allows you to initialize only the description, the sale price to customers, and the cost price from suppliers.

**Part 3:**

Users of this class should be able to *access* the values of a FoodItem object’s instance variables through public methods – write an ***accessor method*** for each instance variable.

Users of this class may need to *change* the data of FoodItem objects – write an appropriate ***mutator method*** for each instance variable. In doing so, be sure you encapsulate your data so that instance variables are never assigned improper values – that is, provide appropriate error-checking in each mutator method. Also, follow these guidelines:

* Errors should result in an appropriate System.out.println() message describing the error.
* The names of your mutator methods should be:
  + setDescription
  + setUnitPrice
  + setUnitCost
  + addToStock
  + setWeight
  + sell
* If the sale price to customers is changed by more than 50%, the change is made but a warning message is sent to the output screen. ("Price changed by over 50%! …") The warning message should include specifics about the old price and the new price.
* Anytime the cost price from suppliers is increased, a warning message is sent to the output screen. ("Unit cost increase!...") The warning message should also include specifics about the old price and the new price.
* The number of items in stock can only be increased by the addToStock method.

* The number of items in stock can only be decreased by the sell method, which also does a few other things. The sell method returns the total price (before tax) of all items sold. It also allows for the sale of any number of items not more than the number of items that are actually in stock. If someone tries to sell more items than are in stock, this method will sell the entire stock and send a warning message that there were not enough items to fulfill the request.

**Part 4:**

* Add an appropriate toString method.
* Add an equals method that returns true if and only if both items' descriptions are the same.

**Part 5:**

The supermarket wants to keep track of its cumulative *revenue* and cumulative *profit* for each food item so it can see which items are the most popular and profitable over time. In addition, it wants to keep track of its total *revenue* and total *profit* for all food items in the store.

* Add variables to this class called cmItemRev, cmItemProfit, totalRev, and totalProfit. Two of these should be ***instance variables*** and two of them should be a ***class variables***. Determine which is which and how to implement these ideas.
* Next, you will need an ***accessor method*** for each variable. Also, update your constructors appropriately. (Hint: this should *not* be much work at all.)
* Next, enhance your sell method so that it appropriately updates each of these new values when an item is sold.

The supermarket has also decided to give each food item a unique ID number for scanning and inventory purposes. Assigning unique ID numbers could be tedious and error-prone for a human so the computer should do it automatically.

* Add variables to this class called idNum and nextAvailableID. One of these should be an ***instance variable*** and one of them should be a ***class variable***. Determine which is which and how to implement these ideas.
* ID numbers should be 5 digits, assigned automatically by the computer, and should *not* be changeable by users. Update all constructors to assign appropriate values automatically.
* Provide an ***accessor method*** where appropriate and update the toString method to include the item's ID.

It is now time to test your FoodItem class. When your class compiles successfully, close it and open a new class called FoodItemTester. Create a static method called runTest. In this method, create several FoodItem objects and call their various methods. Try to do dangerous things: sell more units than are in stock, raise the price by a hundred times the original price, check the ID numbers of each item to see if they're unique. Basically, stress-test your FoodItem class.

When you think your FoodItem class behaves the way it should, copy the DriverForFood class into your BlueJ project and call the foodItemTester() method. Check your output against the desired output as shown in the *FoodItem Tester Output* file. Call me over when your output matches.

**Part 6:**

Write a CashRegister class that will create CashRegister objects and keep track of cumulative cash register data. The CashRegister class should have a method called scanItem, which takes two parameters: a FoodItem variable and an int variable (representing the quantity of each particular FoodItem so duplicates do not also need to be scanned.) The scanItem method should print a message to the output screen like this:

Customer bought 3 cupcakes for a total of $4.41.

This class should have variables for customerTotal and registerTotal, which are updated each time scanItem is called. This class should also have a closeSale method, which returns the price of the customer's scanned objects and resets the customerTotal variable for the next customer. The closeSale method should also print a message to the output screen like this:

Customer Total: $23.54

Add another variable called totalAllRegisters and write any code required to keep it constantly updated. Provide an accessor method for this variable.

Now add a verifySales method, which returns true if the total sales (revenue) stored with the FoodItem class matches the total sales stored with the CashRegister class. Otherwise, it returns false.

When you are finished, call the cashRegisterTester() method in the DriverForFood class. Compare your output with the desired output as shown in the *CashRegister Tester Output* class. Call me over when your output matches.