Project 5: Grocery List

Assigned: September 28, 2010
Due: Sunday, October 2, 2010 by 11:59 PM

Deliverables:

The following project files must be uploaded to Web-CAT by the due date and time specified above (see the Lab Guidelines for information on submitting project files). **If you are not able to submit your project via Web-CAT, then you must e-mail your project to gofflau@auburn.edu before the deadline. Therefore, it is important to give yourself time to submit via e-mail if you have trouble submitting to Web-CAT.**

- GroceryItem.java
- GroceryList.java

Specifications:

GroceryItem.java

- **Requirements**: You have been hired by a local software company to build a class that stores grocery items. Their client (a supermarket) needs each grocery item to have a name, a category, and a base price (maximum $500). Additionally, a total price should be calculated based on the the base price, the category, and a local tax rate as follows:

Category	Total Price Calculation
General	base price with local tax
Produce	base price with a set 5% tax (no local tax)
Refrigerated	base price plus a $1.5 refrigeration fee* with local tax
Frozen	base price plus a $3 refrigeration fee* with local tax

*Taxes are always applied after applicable fees are added to the base price.

- **Design**: Your GroceryItem class must have public constants of type String with names GENERAL (value: "General"), PRODUCE (value: "Produce"), REFRIGERATED (value: "Refrigerated"), and FROZEN (value: "Frozen") to represent each category. The constructor must accept as input a String representing name and a String representing the category. If the name is an empty string or consists only of whitespace, then the name should default to (No Name). If the category is not one of the valid categories GENERAL ("General"), PRODUCE ("Produce"), REFRIGERATED ("Refrigerated"), and FROZEN ("Frozen"), then the category should default to GENERAL. The base price should default to 0 until it is set.

  The GroceryItem class must also contain the following methods (you can choose the return type if it is not specified):

  - `setName`: accepts a String parameter respresenting the name of the item. If the name is an empty string or consists only of whitespace, then the name should default to (No Name).
  - `getName`: returns the name of the grocery item as a String.

  - `setBasePrice`: Takes a double parameter representing base price. Sets the base price of the item and returns 1 (an int) if the price is valid (above 0 and less than or equal to the maximum price of 500). If the price input exceeds the maximum, then the method should return a 0. If the price input is lower than the valid range, then the method should return a -1.
  - `getBasePrice`: Returns the base price (a double).
  - `calculateTotalPrice`: accepts the local tax rate (a double, for example 0.08 would be an 8% tax rate) as a parameter and returns the total price (a double) based on the calculations listed in the requirement. If the tax rate is invalid (less than 0 or greater than 1), then the method should return a value of -1.
  - `setCategory`: takes a String parameter representing the category. Changes the category and returns true only if the parameter is one of the valid categories ("General", "Produce", "Refrigerated", or "Frozen").
  - `getCategory`: returns the category of the grocery item as a String.
  - `toString`: Your toString should return a String representation of the grocery item including its name, its category ("General", "Produce", "Refrigerated", or "Frozen") and its total price. You can decide the format of the toString return.

- **Code**: Keep in mind the following tips when writing your GroceryItem class:
  - Declare variables locally if possible. Reducing the number of instance data can help with the readability of your class.
  - Do not repeat code in your class. For example, if the constructor needs to set the category of the item, then use the setCategory method and its return instead of rewriting all of the code to set the category.
  - The trim method of the String class (see the API documentation) can help you determine whether a String is composed of whitespace only.
  - Do not use "magic numbers" (literal values) in your code. Instead, use a constant field. Constant fields should only be public if they need to be used by the client program.
  - Your get methods should not change the state of the object (i.e. get methods should not change the values of instance variables).

- **Test**: Test your class by instantiating objects in the interactions pane and checking method returns. Remember to test all methods separately. Do not assume that there are any dependencies on methods; for example, do not assume that the client program will invoke calculateTotalPrice before invoking the toString.

GroceryList.java

- **Requirements**: Though the software company for which you are working will be using the GroceryItem class in a driver program that they build, you have been requested to create a simple driver program that will read items from a file and display them in an easily readable manner.

- **Design**: Your program should take as input the name of a file. Each file contains a tax rate followed by a list of grocery items (one per line, listing name, category, and base price). Download groceries.txt and shoppinglist.txt for two examples. Your program should then create GroceryItem objects based on the file input and add the items to an ArrayList with generic type GroceryItem. Once that is complete, your program should display items based on their category (General, Produce, Refrigerated, and then Frozen) and calculate the total cost of all items rounded to 2 decimal places. Your program should follow the format below exactly.

```
Prompt user for file name: groceries.txt

Grocery List
------------
General:

Produce:
- Cilantro
- Parsley

Refrigerated:
- Eggs
- Hamburgers
- Pizza

Frozen:
- Meatballs

Total Cost of Items: $23.20
```

- **Code & Test**: Though it is not good to do in practice, you can assume that every file will be properly formatted. You can also assume that every item's name is only 1 word and that no item will have an invalid price. Your code will be tested using different files, so you may want to create your own additional files for testing (do not place a blank line at the end of your file unless you plan to skip blank lines).

  The following suggestions are optional. In order to process your file, you can read in the tax line and then have a while loop to parse through each line of text. For an example of how to read from a file and insert into an ArrayList see `ReverseWordsFromFile.java` (download from Lab page). For the project, you'll need to read the file, create GroceryItem objects, and add them to your ArrayList. You can then have 4 separate while loops to print general items only, produce items only, refrigerated items only, frozen items only. You can calculate total cost in the file I/O while loop or in a separate while loop.