|  |  |
| --- | --- |
| **CMSC 131** |  |
| **Sections 020X/040X** |

Assignment:     Project #6

Due Date:     Thursday 11/29 at 11:00PM

Open/Closed policy:     CLOSED

**Cafe 131**

****

**Objective**

This project will give you practice with interfaces, arrays, working in a complex environment where several classes depend on one another, dealing with mutable vs. immutable and careful array copying.

**Overview**

This project will simulate an operational restaurant.  It isn't very realistic, but it may amuse you.  The image below represents the GUI (Graphical User Interface) that we have provided.



The entire project can be completed without ever running the GUI -- really the GUI is just for fun.  You should write your classes according to the [JavaDoc](http://docs.google.com/doc/index.html), and test them using your own JUnit tests.  (We are not grading your tests this time, but you are always expected to write them as any good Java programmer would.)   Once you believe that your classes are working properly, please have fun playing around with the GUI.

**Notes on the GUI:**

* At the top you see the amount of cash that the restaurant currently has available.
* The left side of the screen represents the Menu for the restaurant (you can easily add entrees to the menu by clicking the buttons at the top of the Menu.)
* The right side represents your inventory (you can easily add items to the inventory by clicking the delivery buttons -- but your available cash will *decrease* by the wholesale cost of the items that are delivered.)
* Beside each entree on the menu there is an "order" button.  Clicking one of these buttons is supposed to simulate a customer ordering one of the entrees from your menu:  the appropriate items will be removed from your inventory, and your "available cash" will be *increased* by the retail value of the entree.

**Things that are Provided for You**

As usual, we are supplying [JavaDoc](http://docs.google.com/doc/index.html) to provide most of the project specification.

* **First, read the documentation for the *Listable* interface, which has been written for you.**  In this project there will be two classes that implement this interface.  They are:  Food and Entree.
* **Next, read the documentation for the Food class, which has been written for you.** Note that the Food class is immutable and implements the *Listable* interface.  You will need to use this class while writing *your* classes.
* We have also written a class called ***GUIDriver***, which contains the main method that you can run to invoke the GUI.

**What You Must Implement**

You will be implementing the following classes.  It is strongly recommended that you implement them in the order below:

* ***SortedListOfImmutables --*** This is a list that can be used to store many different kinds of objects.  The list always starts off empty, and as objects are added to the list, they are inserted in such a way that the list is always in alphabetical order.  The list can hold any types of objects that satisfy two requirements:

1. The objects must be immutable.
2. The objects must be instances of a class that implements the *Listable* interface.

This project will use this class to store lists of Food objects, and lists of Entree objects.

* ***Entree --***  an entry on a menu.  The *Entree* class implements the *Listable* interface.  Each Entree consists of a name (which is a String) and a list of food (which is a SortedListOfImmutables containing Food objects).  Once an entree is created, it will never change.  (It is an immutable class.)
* ***Restaurant --*** the top-level class representing the restaurant.  A Restaurant consists of a name (which is a String), a menu (which is a SortedListOfImmutables containing Entree objects),  an inventory (which is a SortedListOfImmutables containing Food objects), and a certain amount of available cash (an int, representing the amount in pennies).  If you think about it, the menu is very interesting:  It is essentially a list of lists.  (The menu is a list of entrees; entrees are basically lists of food items.)

**Hints:**

* The methods of the Entree class should be very short.  The entree class has an internal variable called foodList, and some of the methods of the entree class should just be calling instance methods on this variable instead of re-writing essentially the same code!
* The Restaurant class has instance variables called menu and inventory.  The methods of the Restaurant class should rely heavily on making calls to instance methods of these two variables instead of re-writing essentially the same code!
* When writing the "placeOrder" method of the Restaurant class, do not pass the Entree item directly to the "checkAvailability" method of the inventory, and do not try to remove the Entree object directly from the inventory.  Instead, use the Entree method "getFoodList" to retrieve the list of food items that make up the Entree, and then pass that list of food items in to these two methods.

**Restrictions:**

* **You may not use any loops at all anywhere in the Entree or Restaurant classes.**  This will hopefully steer you away from writing redundant code!  (There is a loop in the toString method of the Entree class, but there should be no loops anywhere else in these two classes.  There will be plenty of loops in the SortedListOfImmutables class, though.)
* When elements are added to or removed from the SortedListOfImmutables, the size of the internal array must end up the same as the number of items that are currently in the list.  You may NOT have any extra "unused" elements in your array after calls to either of these methods.
* The instance variables you will need for your classes are already declared for you in the code -- you may not add any others.
* You may not use any static variables anywhere in this project.
* You may not use the operator *instanceof* , except in your equals method.
* You may not use any of the Java collections classes, such as ArrayList, HashSet, LinkedList, etc.(In case you know what these are.)
* You may not call any static methods of the Collections class (in case you know what that is.)
* You may not use any sorting algorithms at any time.  The elements of the lists must be inserted in the correct position to maintain the alphabetical order of the list.  You never sort the whole list.
* You are expected to use good style (as always), including commenting, proper indentation, proper choice of variables names, etc.

**Grading**

Your grade will be computed as follows:

* Public Tests:  12%
* Release Tests:  78%
* Style:  10%  (You will be penalized heavily for redundancy on this project -- you could lose all 10 points.)

[Web Accessibility](https://www.umd.edu/web-accessibility)