COMS W1007 Homework 2

Submission instructions

All programs must compile and run on CUNIX to receive credit. Submit your electronic files via `http://courseworks.columbia.edu`. Post your archived submission directory to the CLASS FILES section in the appropriate homework directory. We prefer electronic submission of written portions, though you will not be penalized for paper submissions. (Do **not** print out your programs.) Include a separate README text file that explains exactly what each file in your submission is. Place all the files you want to submit into a submission directory with the following naming scheme.

`<your_uni>_hw<number>`

So if my UNI is `uni1234` am submitting homework 5, my directory would be `uni1234_hw5`. Archive your submission directory using zip, tar, or gzip format. In CUNIX, you can tar and gzip with the following command:

```
tar -czvf uni1234_hw5.tar.gz uni1234_hw5
```

and upload `uni1234_hw5.tar.gz` to courseworks. (You will probably need to first download the file to a local directory using an FTP program. See CUNIX tutorial for more info.) Your submission archive should contain your code, written problems, and your readme file. Do not include your compiled .class files or your compiled javadoc, since we will generate those from your code.

Multiple Submissions: You can submit multiple times, but we will only consider the latest submission based on the timestamp in courseworks. Please give at least 1-2 minutes between two submissions so we can tell which is the newest submission.

Keep a pristine copy of your submission folder in case there is any submission error.

Written Problems

1. **6 points** (Horstmann 3.2) Find a total ordering for `Rectangle` objects. *Hint*: Use lexicographic order on (*x*, *y*, *width*, *height*).

2. **6 points** (Horstmann 3.16) This chapter [Chapter 3] discusses the drawbacks of mutator methods for setting the year, month, and date of a `Day` object. However, the `Calendar` class of the standard library has a `set` method for just that purpose. Does that method solve the issues that were raised in our discussion?

3. **6 points** Interfaces in Java are related to abstract classes. Describe in your own words the differences and similarities between interfaces and abstract classes. Give at least one hypothetical example for when you would use each construction. Cite any sources you used to learn about these types (I recommend looking around the java.sun.com site, or skim ahead to Ch. 6 in Horstmann).

4. **6 points** The COMPOSITE pattern and the DECORATOR pattern are similar. Describe the subtle difference between the two and give original examples (not the examples in the book or from class) of when you would use each.

5. **6 points** Antipatterns are the opposite of design patterns; they are strategies you should avoid. A commonly cited antipattern is "The Blob", which Horstmann describes as "a class that has gobbled up many disparate responsibilities". Describe such a class and list its methods. You may be able to find an example in old code you have written[1], which are welcome responses for this problem (make sure to cite yourself). Totally new, hypothetical examples are also welcome.[2]

Programming Assignment

(70 points) For this homework assignment, you will build a **point of sale** GUI for a restaurant wait staff.

Description

Point of sale interfaces are designed to simplify the process of making transactions, often in a retail environment. We often see them used in restaurants where managers can input the menu and waiters can quickly enter customer orders, which are transmitted to the kitchen and recorded. Modern systems usually include a touchscreen interface, which we will simulate with a mouse-based GUI.

The program you should design and build will read a menu from a text file formatted with a menu item on every other line with its price listed on the next line. To simplify your text-parsing code, we will omit the dollar sign from the price. For example:

```
Royale with cheese
3.99
Le Big Mac
4.99
Milkshake
5.00
```

The program should load the file at launch and create a panel full of large buttons (ideal for a touchscreen) for each menu item. A waiter should be able to click on each menu item to add it to the current order. This should add the item to a receipt panel which displays the full order and the total cost. The total cost should be accurate at all times, updated as each item is added (not only when the order is finalized).

The waiter should be able to click a "Place Order" button that simulates transmitting the order to the kitchen by appending the order to a log file. This log file will contain each order from the session. You should also include a "Clear" button that will clear the current order without saving it to the log (used when a waiter makes a mistake and needs to start over).

[1] I know I wrote many "blob" classes when I was learning to program.

[2] The purpose of making you give an example is to help solidify the concept of what to avoid. Remember, this is to be an example of a specific kind of poor design. This example should not show up in your programming submission!

Your program should be able to handle a large menu (which may not fit on the screen all at once), and should provide a way to order more than one of each item. The program should have no keyboard entry, since we are designing a touchscreen interface, for a bustling, fast-paced NYC restaurant.

You should design your own layout for the interface, which should be straightforward as it only needs a text panel for the receipt and a panel of menu item buttons. It is up to you where your special "Place Order" and "Clear" buttons go.

Process and Deliverables

For this programming assignment, you are only required to submit code that compiles via javac and javadoc. At least all public methods must have javadoc comments for any parameters and return values. You should include your code and a "readme" file that briefly describes each included file, and can be used to indicate to us any notes that would be helpful for grading (such as any extra functionality you have added or were unable to finish in time).

Nevertheless, you are encouraged to use the design tools we practiced on the previous homework, but you do not need to submit them.

Your design must follow the Model/View/Controller pattern, and the related Observer pattern, as described in class.

As always, to get full credit, your code must not crash for any reasonable user behavior. You may exit with an informative error message, but uncaught exceptions or crashes will lose some points.

Notes

- Since there may be more menu items than fit on the screen, you will want to use a JScrollPane
- Explore the Java 6 API for how to make your buttons larger than the default.
- A combination of FlowLayout and GridLayout should be sufficient for laying out the buttons. But you are welcome to explore other layout managers.
- Depending on your design, your objects representing the model, view and controller may be relatively small since the program itself is not very complex; this is fine. It is still worth following the programming pattern to provide good organization for extendibility and such.
- Scanner objects sometimes don't behave the same between different operating systems due to text file formats being different. You should consider using a `BufferedReader` to read from the menu file.