Automated Restaurant System (Command Design Pattern)

120 pts.

PROBLEM

You are to implement an automated restaurant system utilizing the Command design pattern, minimally consisting of the following three commands. You will then extend this basic system in any way that you choose to demonstrate the use of additional design patterns.

- Display menu
- Submit Order
- Display tab

SCENARIO (of the basic system)

In the basic system, we assume that all orders are from the same table (i.e., there is only one table in the restaurant). Therefore, a tab is generated by simply totaling all of the ordered items, and tabs are not stored.

The menu of the basic system will consist of just main entrees. It will not include appetizers, desserts, drinks, etc. The information for each entrée will just be the name of the dish ("Roast Beef", "Chicken Korma", "Jiaozi", etc.) and the price.

APPROACH

You should implement the Command design pattern. This includes the following:

- Text-based user interface
- SystemInterface class
- **Invoker class**
- **Command interface**
- **Command classes (one for each command)**
- **Aggregator class**
- Menu class
- MenuItem Class
- Orders class
- OrderItem Class
- Tab class

classes of the Command design pattern

The user interface should just be a text-based numbered list of options, implemented in the main method. (It can be a GUI if you desire and are familiar with the development of GUIs, but no extra points will be given for this).

The SystemInterface can be a class of all static methods – one for each of the three commands of the user interface, if it does not have any state in your extension of the program.

The Aggregator class maintains references to the Menu object and the Orders object. It should provide a getter method for retrieving the Menu and Orders objects (no setters are needed). The Menu and Orders classes store a collection of MenuItem and OrderItem objects, respectively. A MenuItem object will store the menu item #, the description, and its cost. An OrderItem object will store an order by its item number only (not its description).

The Invoker class has methods cooresponding to the methods of the system interface. Each method simply creates a Command object of the appropriate Command class (constructed with a reference to the Aggregator object), calls its execute method, and returns the results (as a single object) back to the system interface. (Execute methods should not be passed any parameter values - any values needed are passed to the constructor.)

A tab will be constructed from the Tab class containing all of the ordered items, returned as an array of strings for the user interface to display. Note that a tab needs information from both the Menu and the Orders objects. (The Orders object indicates what menu items were ordered, and the Menu class has the description of each item to include in the Tab.)

EXTENSION OF THE BASIC SYSTEM

You are to extend the design and capabilitities of the program in any ways that you wish, by using the design patterns we learned in the class. The intent is to demonstrate the significant and appropriate use of design patterns. In additional to the command design pattern, at least **TWO** other design patterns need to be used/identified from your system.

Following are some ideas on how the system may be extended:

- Capability of maintaining orders (and tabs) for multiple tables.
- Capability of maintaining a queue of waiting groups of different size to be seated, and automatically seated as people leave tables (triggered when tab paid).
- Capability of maintaining an inventory of items (tomatoes, etc.) and their depletion as orders are prepared, which may have various effects (e.g., what is offered on the menu, the price of certain items, the options offered for certain items)
- Capability of "dynamic pricing" in which menu items change price based on factors such as how well certain items is selling.
- Capability of generating various reports (sales, inventory, etc.)
- Capability of manager adjusting a given table's tab based on some factors.
- Anything else you can think of!

WHAT TO SUBMIT

- A document includes all the screenshots that indicates how your system works with steps.
- A class diagram for your system. Make sure to include all of the classes and interfaces of your implementation. The class diagram should include the relationships of association, composition, aggregation, generalization (inheritance), and dependency where they exist; notation of multiplicity, navigation, and role names where appropriate; and the inclusion of the methods within each class, including access modifiers (public, protected, private), parameters, and return type (instance variables do not need to be included).
- Discussion of the use of design patterns within your design, and
    - *why* you made the decision to incorporate the use of each pattern,
    - the *benefits* of each included pattern,
    - a "walk through" of your design.

GRADES

- Basic system implementation (50 pts.)
- Additional features with design patterns (20 pts.)
- Class diagram (20 pts.)
- Document with screenshots (10 pts.)
- Discussion (20 pts.)

Please submit everything required above along with your code (all source code should be submitted in a zip file).