Technical University of Cluj-Napoca

Programming Techniques

Laboratory-Assignment 4

Teacher: prof. Ioan Salomie

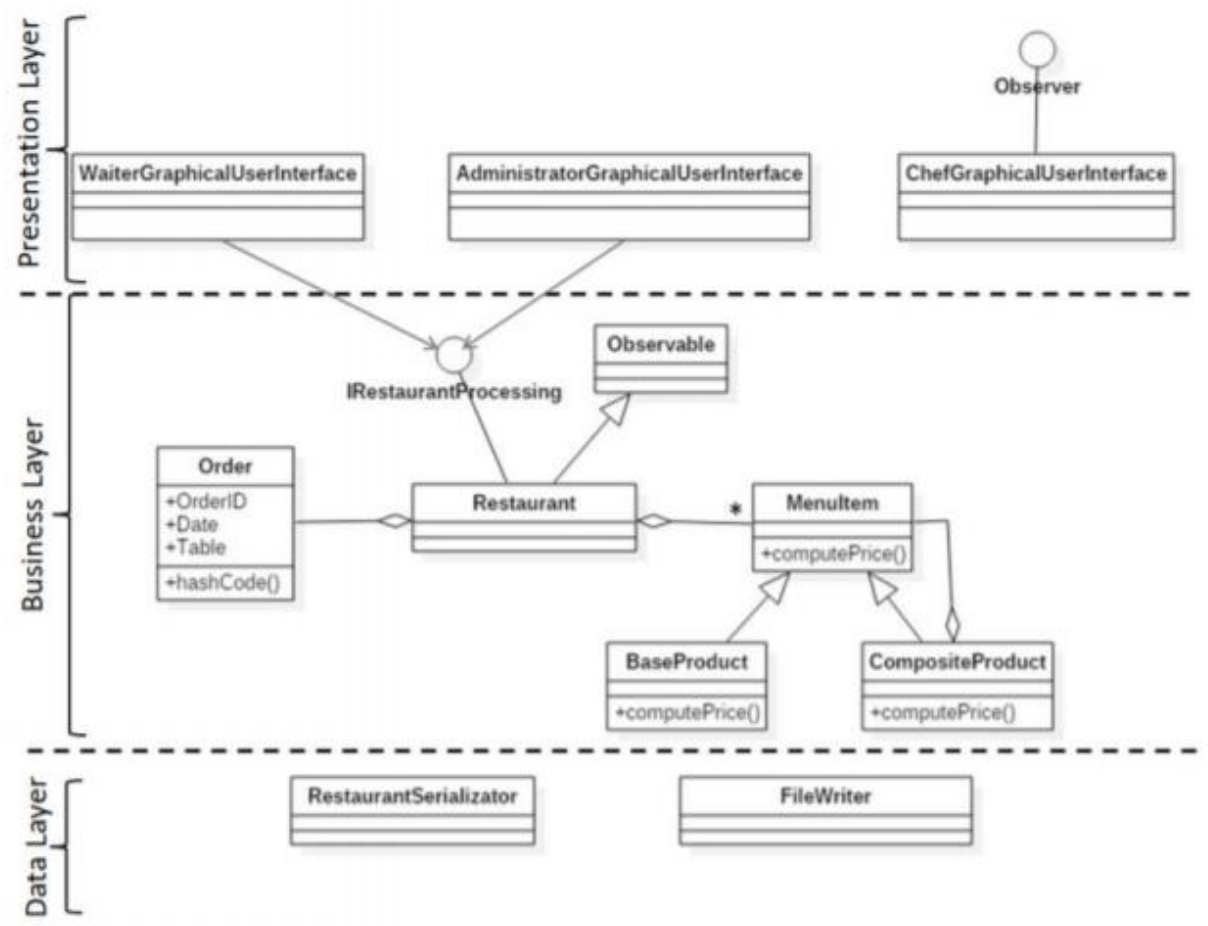
Teacher Assistant: Viorica Chifu

Student:Stioiu Denis-Adrian

Group:30424

1.Objective

Consider implementing a restaurant management system. The system should have three types of users: administrator, waiter and chef. The administrator can add, delete and modify existing products from the menu. The waiter can create a new order for a table, add elements from the menu, and compute the bill for an order. The chef is notified each time it must cook food that is ordered through a waiter. Consider the system of classes in the diagram below.



To simplify the application, you may assume that the system is used by only one administrator, one waiter and one chef, and there is no need of a login process.

Solve the following:

1) **Define the interface** IRestaurantProcessing containing the main operations that can be executed by the waiter/administrator, as follows:

• Administrator: create new menu item, delete menu item, edit menu item

• Waiter: create new order; compute price for an order; generate bill in .txt format.

2) **Define and implement the classes** from the class diagram shown above:

• Use the Composite Design Pattern for defining the classes MenuItem, BaseProduct and CompositeProduct .

• Use the Observer Design Pattern to notify the chef each time a new order

containing a composite product is added.

3) **Implement the class Restaurant** using a predefined JCF collection that is based on a hashtable data structure. The hashtable key will be generated based on the class Order, which can have associated several MenuItems. Use JTable to display Restaurant related information.

• Define a structure of type Map<Order, Collection<MenuItem>> for storing the order related information in the Restaurant class. The key of the Map will be formed of objects of type Order, for which the hashCode() method will be overwritten to compute the hash value within the Map from the attributes of the Order (OrderID, date, etc.).

• Define an appropriate collection consisting of MenuItem objects to store the menu of the restaurant.

• Define a method of type “well formed” for the class Restaurant.

• Implement the class Restaurant using Design by Contract method (involving pre, post conditions, invariants, and assertions).

4) **The menu items** for populating the Restaurant object will be loaded/saved from/to a file using Serialization.

2. Problem analysis, scenarios, use cases

A. General overview

The application simulates a management for a restaurant. There are three main users: the administrator, who can create new menu items, edit the price of the existing items, or delete a menu item. The waiter is able to create an order taken from the clients at a speicified table, he can create the order and deliver it to the chef, and generate a bill after a client finished. The chef is notified by the waiter to cook the food.

B. Input and output

The input is given through a button command from the graphical user interface. Each type of user has access to different facilities. The administrator has the operations to create a

new menu item, delete a menu item, or update an existing one. If the waiter is using the appllication he may create a new order, generate a bill or calculate the cost of an order. The chef is notified by a pop up when an order has to be prepared.
The output is a bill in the form of a .txt file, with the details of the order, and the total price.

C. Use cases

Administrator:

a. Create a new menu item: Enter the admin panel, add the information about the menu item, and press the button to add it in the menu. In case of an error the user will be notified.
b. Edit a menu item: Enter the admin panel, add the updated information about the menu item, and press the button to finish the operation. In the case of an error the user will be notified.
c. Delete a menu item: Enter the index of the menu to be deleted, in the admin panel, and press the corresponding button. In case of an error, the user will be notified.

Waiter:

a. Create a new order: In the waiter panel, select the create option, add the required information about the order, and place the order by pushing the button corresponding with the operation. In case of an error the waiter will be notified.
b. Calculate the price: This operation is automatically performed, whenever the waiter creates a new order. In case of an error the waiter will be notified.
c. Generate the bill: When the order is the delivered, the waiter will press the button to generate the bill to the order. A txt file will be created with the details of the order and the total price.

The chef:

a. Notification: When the waiter gets a new order the chef will be notified through a pop up to prepare the food. In case of an error there will be a notification.

D. Data structures

Array list, where the menu items are stored. The orders placed are stored inside a hashmap for an efficient access.

E. Packages

Java packages are useful to organize multiple modules and group together related classes and interfaces.

As the application uses a graphical user interface I chose the MVC pattern. The MVC is a software pattern that has as the main advantages: enabling logical grouping of related actions on a controller together. It is easy to modify due to the separation of responsibilities, the future development or modification becoming easier. The view and controller of the program are stored in the presentation layer.

The interface IRestaurantProcessing is implemented by the Restaurant class. It declares the methods necessary for the implementation of the operation of the administrator and waiter users.

The abstract class MenuItem is inherited by BaseProduct and CompositProduct. These is done using the composite design pattern.

The project has a layered architecture.

F. Class design

Main- is the main class of the project, it initializes the gui and al the components. It also deserialises the .ser file in order to get the data for the restaurant object.

presentation.View: It is the graphical user interface of the project. There the user can enter a panel and use the functions

presentation.Controller: It creates the listeners for the operations implemented by the restaurant object.

dao.FileWriter: Used to generate the bill. The class creates a bill with information about the order and the price to be paid.

dao.Serializator: It creates and extracts data from the .ser file.

bll.IRestaurantProcessing: The interface declares the methods for the desired functionality of the application. It will be implemented by the restaurant class

bll.Restaurant: Implements the interface, and has getters and some additional methods along with some invariants.

bll.MenuItem: abstract class made using the composite design pattern. It has two abstract methods to calculate the price of the product and show the ingredients of a food. It also has getters and setters.

bll.BaseProduct: Extends the MenuItem class, implementing its abstract methods, and is the leaf of the design pattern.

bll.CompositeProduct: Extends MenuItem, and implements its abstract methods, It is the composite of the design pattern, being more complex.

bll.Order: Class having the model and the attributes of a real life order

3. Conclusion

The restaurant management system project was a good opportunity to remember and perfectionate the Model – View – Controller pattern, and to learn about the composite design pattern.I was excited to learn about serialization. Also, I learned more about hash maps and was a good practice for the future. An improvement could be adding more functions to the program.

4. Bibliography

Materials offered during the courses and the laboratories