Programming Techniques

Semester II – 2019/2020

Assignment 4

Restaurant Management System

Name: Benedek Balázs
Group: 30424
Teacher: Viorica Chifu

Table of Contents

1. Objectives

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 ordered through a waiter**.**

Secondary objectives:

- IRestaurantProcessing interface used by the model and GUI classes
- Design by Contract
- Design Patterns: Observer and Composite
- Serialization (saving into ***"restaurant.ser"***)
- JCF HashMap and HashSet implementations

2. Problem Analysis, Modelling, Scenarios, Use Cases

2.1. Problem Analysis

Have you ever thought how restaurant management system works? With this application we can simulate the possible operation made by the stuff of the local. An application that implements the management system of a restaurant may be useful for every restaurant, bistro or café that has a chef and waiter next to the administrator. The communication becomes much easier by sending messages only (waiter to chef), administrating orders and managing them becomes an easier process, and the menu items can be modified at time by an administrator.

There can be registered some data or statistics in a more complex application, the logic behind is not so complex, and a well-organized, clear, easy to read graphical user interface makes the system easy to use. Not only can the friendly interface attract costumers, but also can get any the users closer to a better management method.

If the management of the restaurant is monitorized by a system like this, it gives you a confidence that your order history and your products are digitalized, and you can find them much easier, this to be said simply, the user gets the data of the restaurant organized.

2.2. Modelling

For such a system it is necessary to have some entities for representing the menu items, that can be basic items or composite items, but composite items are a set of basic elements. We need to define the operations that must be implemented, operations like adding new items, editing or deleting them, or taking a command, placing a command, computing price for an order, generating bills etc.

The graphical user interfaces must be different for the users, appropriate for each of them. For example, a waiter is not supposed to have access to the operations like editing a menu item, it should be done only by the administrator.

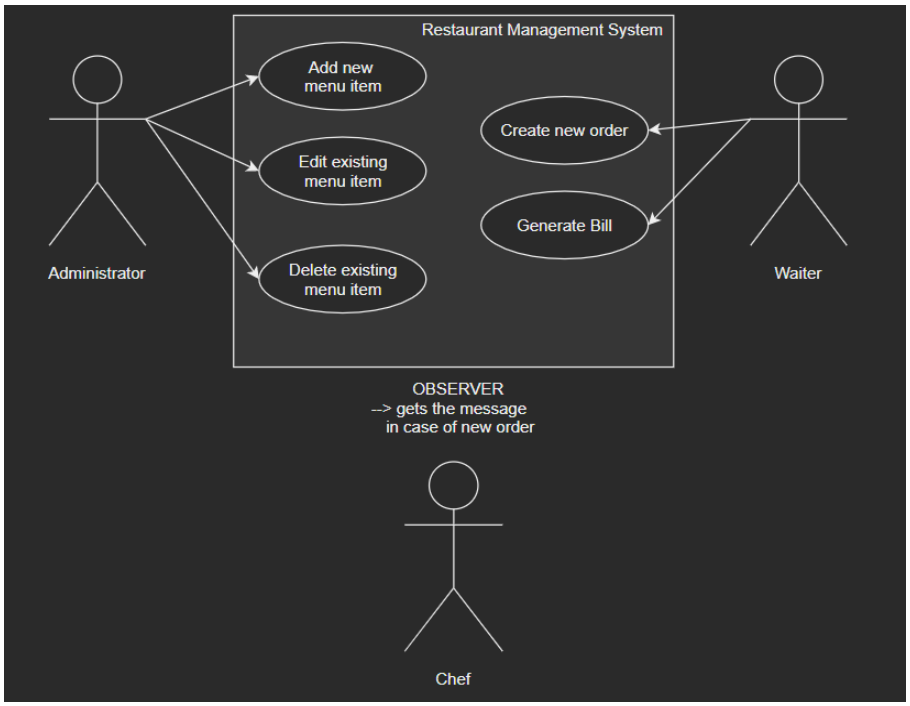
The data needs to be stored in some files (in this case into a ***".ser"*** file, using **serialization** method), from where it can be loaded every time the application is started. Let me show you a possible example: *Your restaurant is running properly, it is open, there are a lot of orders, saved in the system, but an electrical error appears, cutting the current, causing the system shut down. If you do not have a file which stands for loading the previous data, you get into a big trouble, unknowing the previous orders, resulting that your clients will be really upset and nervous.*

2.3. Scenarios, Use Cases

Scenarios includes maintaining data of the restaurant, which means adding, editing and deleting items from the menu, this happens when the actor is the administrator. In case of the waiter, he can create a new order by selecting elements from the menu (dropdown list), and he can also generate a bill for a certain order (the order id must be given in the text area). The chef is only an **observer**, when a new order is created by the waiter, the details regarding the chef appear in his window (together with a notification pane, containing the information as well).

Each user has its own interface. After the above-mentioned operations, the data appear in tables, containing name and price. The table is available for the administrator and the waiter, but only the administrator can edit it, the waiter can only read it and select products from there.

The use case diagram is on the right: →



Use cases

Use case title: Add, Create, Update and Refresh Menu

Actor: Administrator

Main success scenario:

1. Administrator introduces new data for Base Product
2. Can choose from the following operations:
    a. Administrator presses the ***"Create Item"*** button
    b. Administrator presses the ***"Update Item"*** button
    c. Administrator presses the ***"Delete Item"*** button
    d. Administrator presses the ***"Refresh Menu"*** button
3. System takes information
    a. There are 2 possibilities:
        i. The input data is correct, the new data is inserted
        ii. The input is incorrect, and an error message is displayed
4. The administrator can return to the first step

Use case title: Alternative Update Option

Actor: Administrator

Main success scenario:

1. Administrator introduces new data for Base Product
2. Administrator double clicks on the field in the table which wants to edit
3. Enters the new value for that field
4. Clicks outside the box, or presses enter
5. System takes information
    a. There are 2 possibilities:
        i. The input data is correct, the new data is inserted
        ii. The input is incorrect, and an error message is displayed

Use case title: Create Composite Menu Item

Actor: Administrator

Main success scenario:

1. Administrator introduces new data for Base Product
2. Administrator clicks on the ***"Create an entire menu"*** button
3. The Composite Menu GUI opens
4. The user chooses from the dropdown list the product to add to the list
5. Presses the ***"Add Item"*** button to add the product to the list
6. Repeating the 4-5. operations how many times it is needed
7. Administrator presses the "Create Menu" button
8. If there are no errors the new menu item is inserted to the table in the Admin view
9. The Admin can return to the 4th step whenever he wants
10. To exit this frame, the Admin presses the "***Back***" button

Use case title: Create Order

Actor: Waiter

Main success scenario:

1. Waiter introduces the table number choosing from the dropdown list (order ID and date are generated automatically)
2. Waiter selects menu items one by one after each one pressing the ***"Add Item"*** button
3. Selected items appear in a list (in the text area)
4. Waiter presses ***"Create Order"*** button
5. The order is saved and communicated to the chef
6. The administrator can return to the first step

Use case title: Generate bill

Actor: Waiter

Main success scenario:

1. Waiter introduces the ordinal number of order for which he wants to generate the bill (he or she can see in the table the orders made before)
2. Waiter presses ***"Generate Bill"*** button
3. Bill is generated in a *bill.txt* file

Use case title: See orders

Actor: Waiter

Main success scenario:

1. The Chef can see the current orders in text area placed on his interface

3. Design

3.1. Class and Package Diagram

The project is built on a layered architecture represented by three important layers, the business layer, presentation layer and data layer (named respectively***: "bll", "presentation", "dao"***). In designing the classes, design patterns like composite pattern and observer pattern were used, and the application is based on the Model, View, Controller pattern.

The number of packages is five: *bll, dao, presentation, control and main*.

The business layer contains the entities like ***Order, Restaurant*** and ***MenuItem***. For designing the ***MenuItem*** entity, composite pattern was used, there are ***Base Products*** and ***Composite Products*** composed by base products. Both

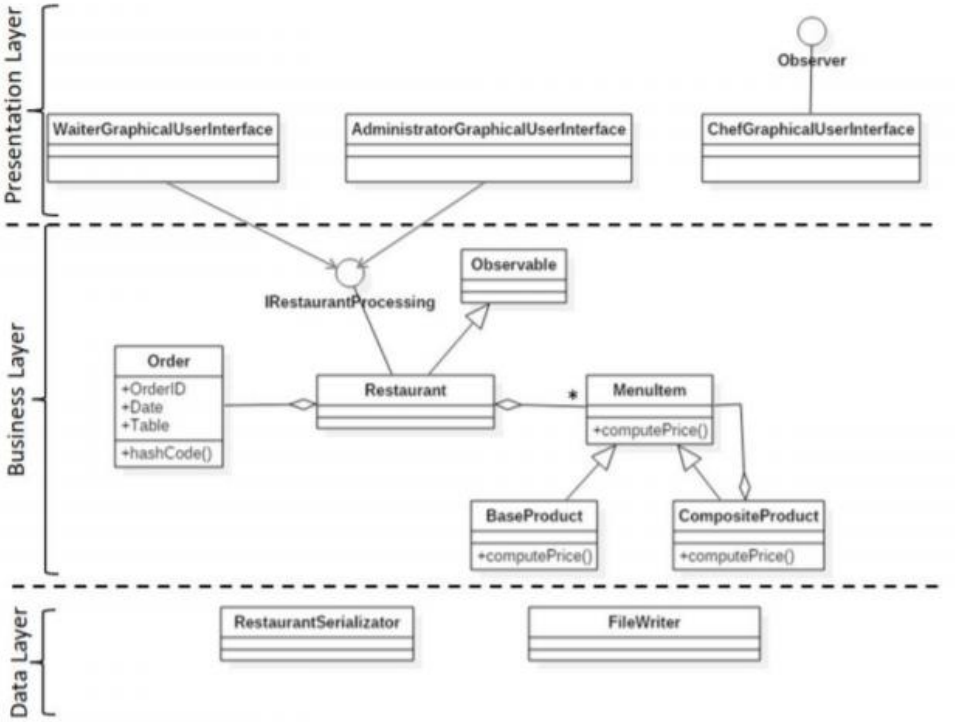
categories can represent a Menu Item which is an abstract class extended by BaseProduct and CompositeProduct. The class Restaurant implements the operations described in the ***IRestaurantProcessing*** interface (maintaining data, generating bills, creating orders etc.).

In the data layer the class RestaurantSerializator, and FileWriter are placed which saves and loads the data from a file. The serializator class makes the serialization, and deserialization, while the file writer class generates the text file for the bill.

In the presentation package and layer, the classes that are part of the GUI are placed. There is a main window (named ***SelectorGUI***) from where the users can access their own windows (the administrator gets the ***AdminGUI*** (in cse of create composite item he gets the ***CompositeGUI***, the waiter the ***WaiterGUI*** and the Chef, the ***ChefGUI***).

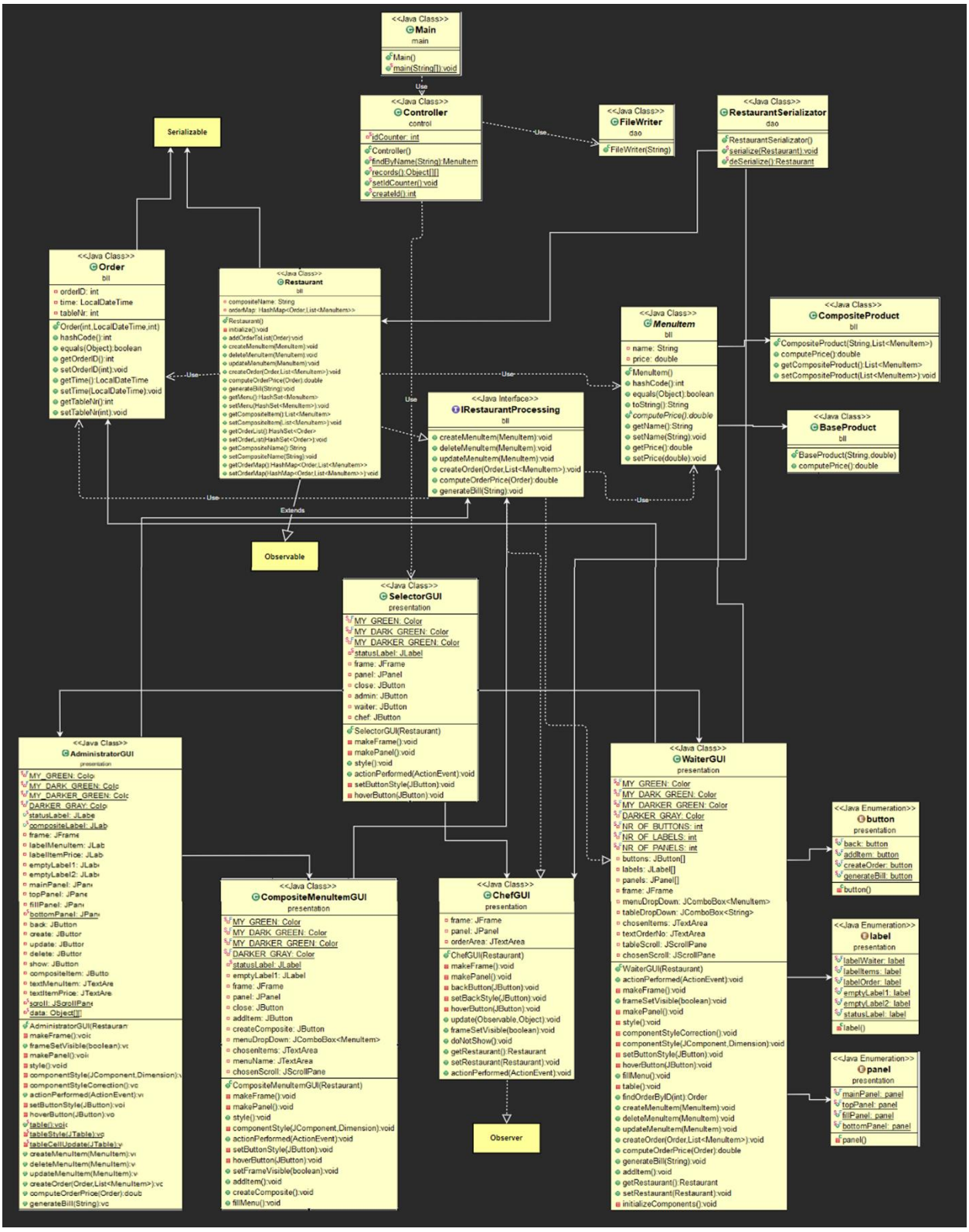
The control package contains the ***Controller*** class which is the connection between the presentation and business logic(model). It works like an interface between the mentioned classes.

The main package contains the ***Main class*** that has the public static main method to start the application.

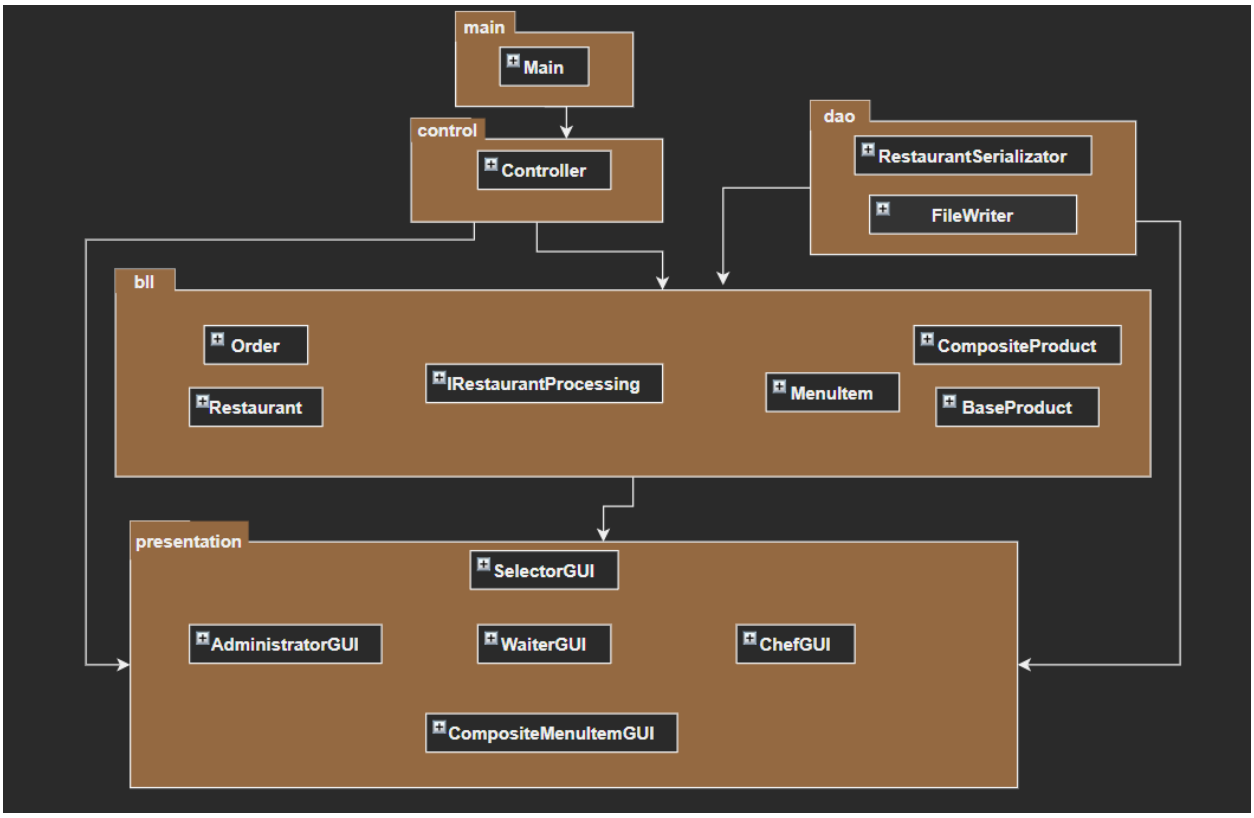


- Required Class Diagram for the restaurant management system -

Programming Techniques – Assignment 4



- Class Diagram for the restaurant management system -



- Package Diagram for the restaurant management system -

3.3. Algorithms

The algorithms needed are simple, they consist mainly of adding, removing from collections or setting attributes. The algorithm for computing the price of an order consists of computing the price of each element and summing them together.

There are some important data structures used from the Java Collections Framework. For storing the menu of the restaurant, the HashSet is used, since a Menu Item is unique (by its name AND price). There are overridden the equals and hash code functions to set our sorting methods, and, being a hash set or hash map, identical item cannot enter these collections.

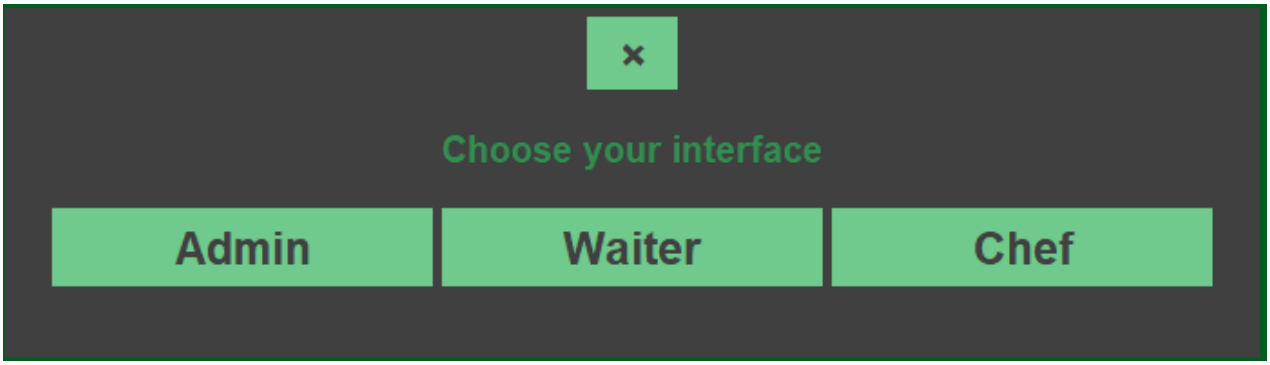
The orders are stored in a HashMap having as a key and Order object and as value a list (***ArrayList*** was used) containing the products selected (***HashMap<Order, List<MenuItem>>***). Here a list is used instead of a set, because at this part there can appear duplicates (such as 2 bottles of coke, or 4 hamburgers and so on).

The Design by Contract is appearing in the restaurant class, using some assertion, as pre- and postconditions to make sure that the required operations can be applied. For example, for creating a new menu item the precondition is that the menu item is correct, and is not null, then comes the invariant, adding to the hashset the new product, then the postcondition is to check if the difference between the resulting list size and the size before is exactly one, to be sure that the insertion went correct.

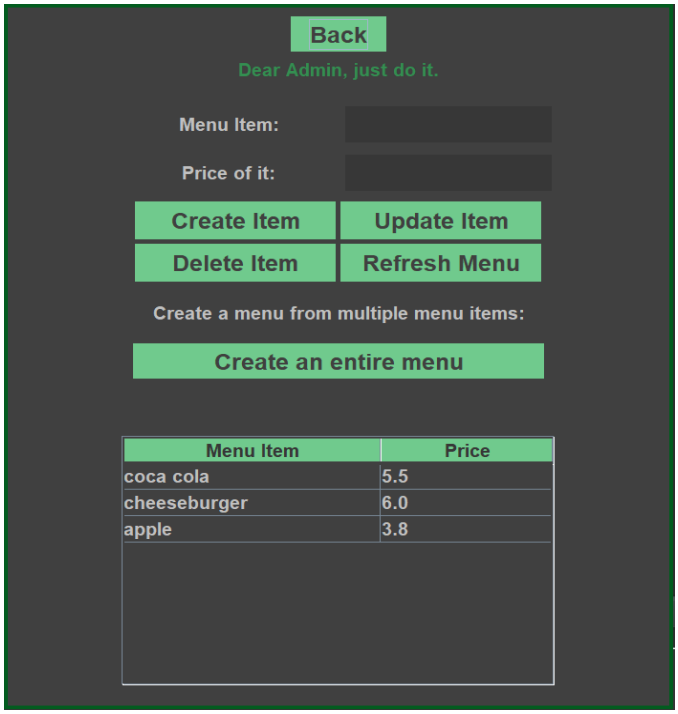
3.4. Graphical User Interface

The graphical user interface provides an easy handling of the system for the user. It has some labels (for information), text fields (for introducing data and output of the result) and buttons. The buttons are representative since they have a label describing shortly the operation it performs.

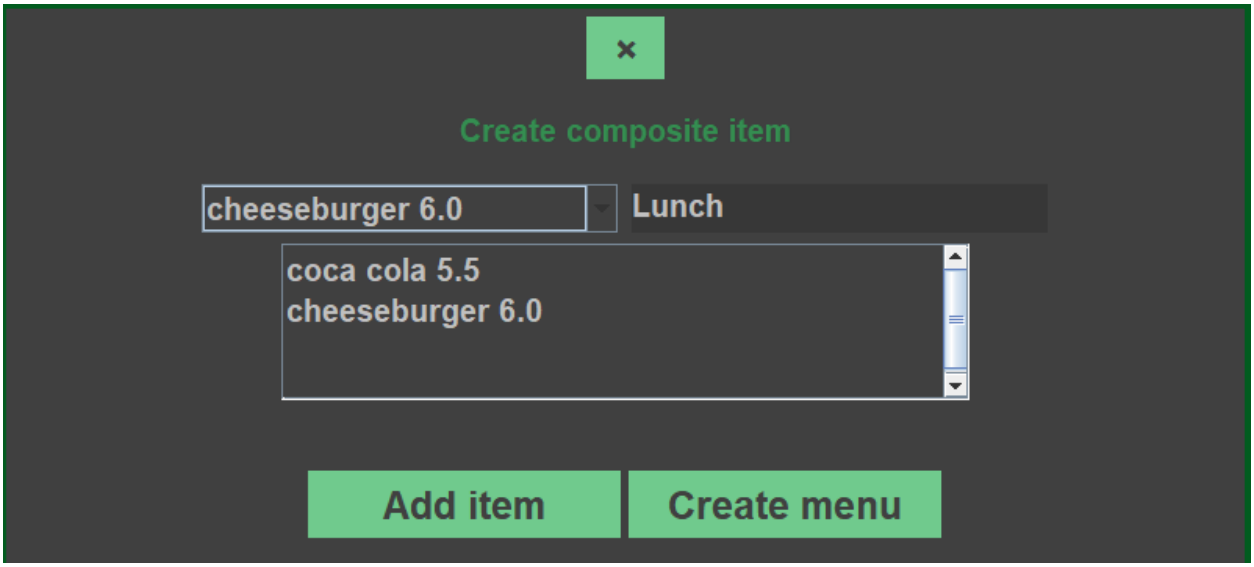
It is constructed using the swing library. There are five frames **(JFrame)** in total. When the application starts, the first frame appears, where the user is told that he can choose an option: Admin, Waiter or Chef. In this project it is not required to create login, thus in this case the user can choose any option without confirming that he, or she is allowed to enter the respective interface.



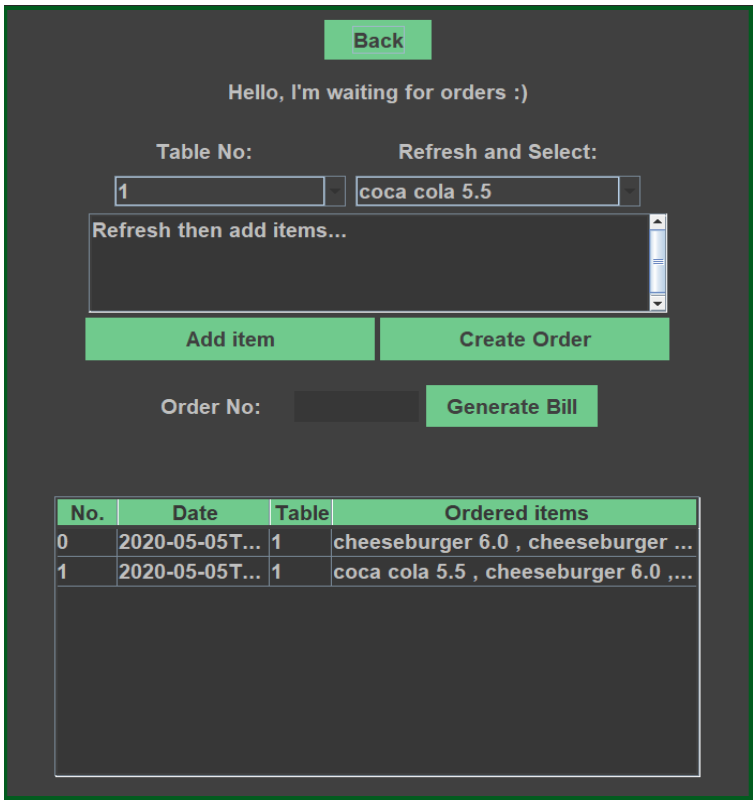
Choosing the **Admin** interface there appears a friendly frame for creating, editing and adding products to the menu list. The first button which appears is the back button which drives the user back to the main frame. There are two text area which should be mentioned. The first one stands for the name of the menu item, while the second one is for how much does the product cost. In general case, at every modification, the *menu item table refreshes automatically*, but to be sure that we the table will be refreshed there is a refresh button to update the content of the table. The other buttons, how their names show, they are for create, update and delete item. **To create** an item the user is required to introduce the name of the menu item, and the price of it, then click on the create button. **To delete**, the user must introduce the name of the item then click on the delete item button. **For updating** items there are two possibilities: the first one is to write the name of the product and the new price of it into the text areas, then click on the update item button; the second possibility is double click on the field in table that you want to change, then click outside of the box. At the bottom part of the frame there is the **JTable** containing the menu items, even if they are composite items (*an entire menu*) or base products (*only a single item with its own price*). It has 2 columns for the mentioned fields, and a scrollbar appears in case that the menu list is bigger than the table box.



The ***"Create entire menu"*** button is for creating a composite menu item. In this case a new frame opens for an easier collaboration with the system. The admin can add as many items as he wants, then pressing the create menu button. How to use this window? This method is presenting in the ***Use Cases*** section.



In case that the user chooses the waiter interface to handle the orders and to generate bills for existing orders, then the following window shows up. The waiter can choose from the dropdown list the ordinal number of the table, then adding new items to the order list, then creating the order itself. In a JTable the waiter is able to see the made orders, which are loaded from the serialization file made before, or created at the moment.



The design to this part of view can be optimized, and can be done better, but in my opinion, it is pretty enough for a restaurant to get to know to a system like this.

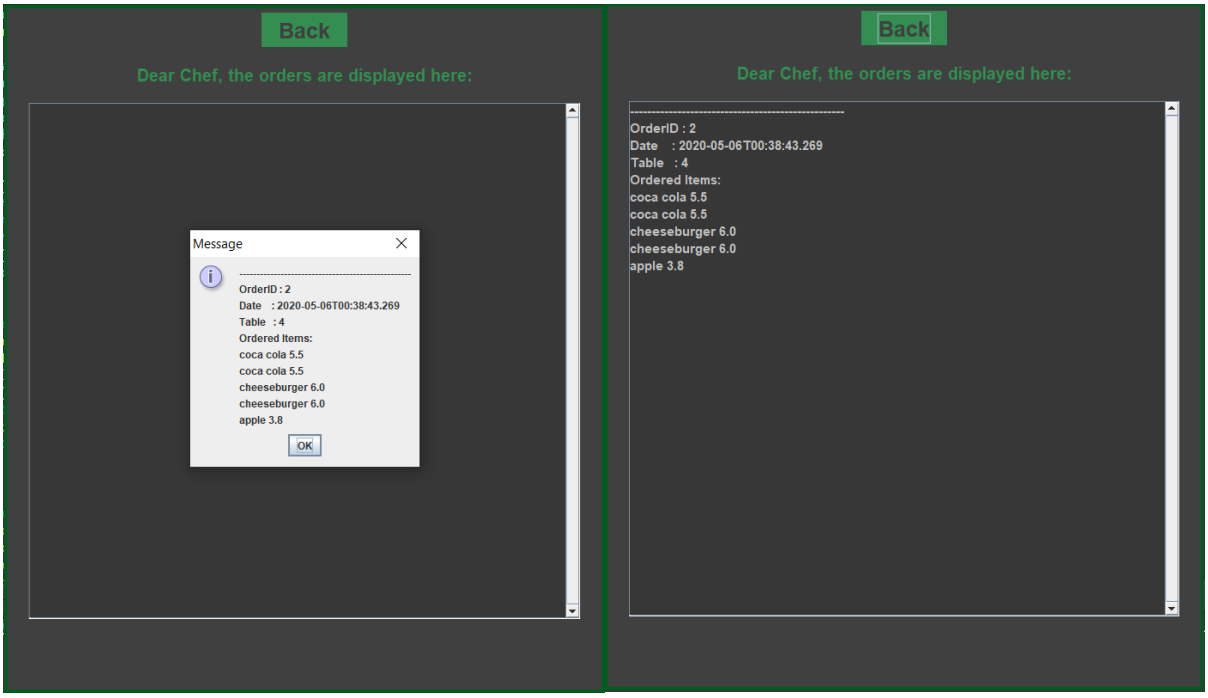
The colors are chosen wisely to be more comfortable for the eyes.

To see how the operations can be done, these are explained in the use case section, feel free to apply them.

If a new order is placed the chef is notified by a new notification pane on his interface, then appearing in the text area placed on the chef's text area.

And here is represented how the chef can see the system:

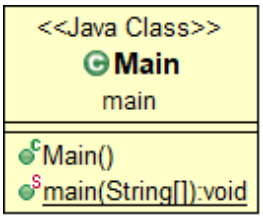
First of all, the notification pane appears, then clicking on ok, the order appears int the text area. In case there are made more orders, they are appended to the list, and not replaced.



4. Implementation

4.1. Main (“main”) package

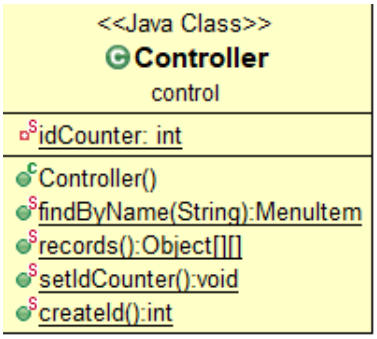
4.1.1. Main Class



The main class is for testing the application, where the Controller, and View objects are created. There is one public static void method, named main() to the exact start of this processing system.

4.2. Controller (“control”) package

4.2.1. Controller Class



The Controller is like an interface between Model (bll, dao) and View (presentation). It receives the user requests from the view layer and processes them. Then the requests are sent to model for data processing. Once they are processed, the data is again sent back to the controller and then displayed on the view.

In the controller part near the constructor there are 4 methods, namely the find by name method which returns the menu item with the name, or null in case it is not found. The records method returns the data for the JTable, while the id methods are setting the orderID.

4.3. Presentation Layer ("presentation") package

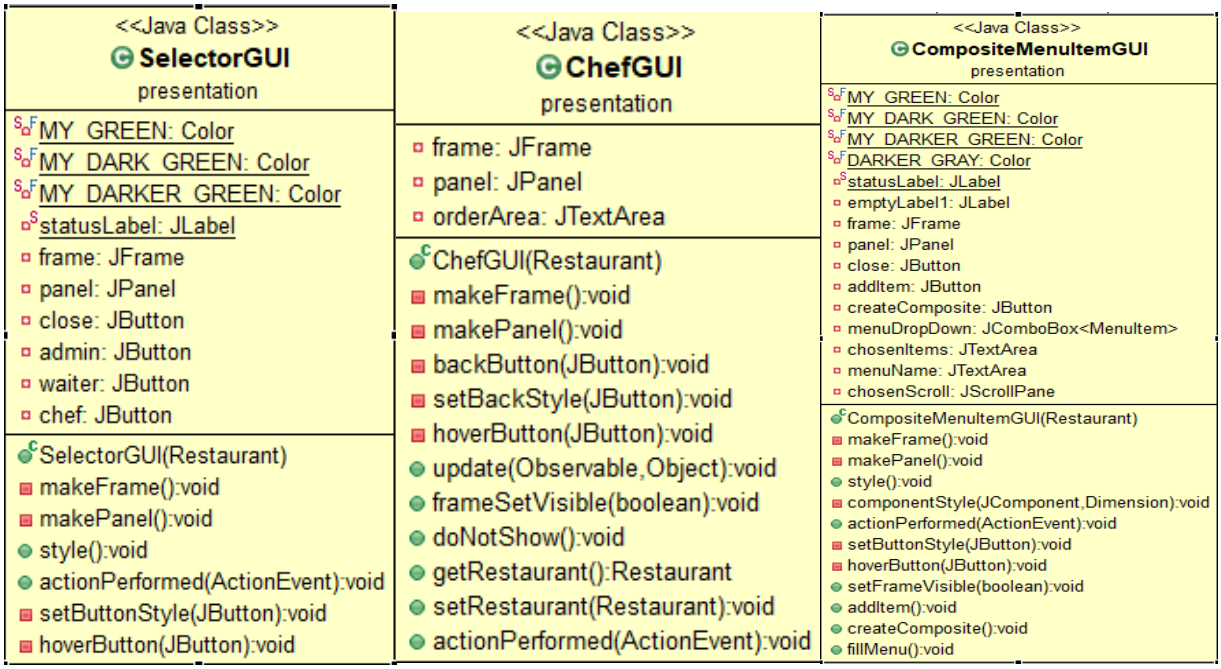
This class implements the view of the application. The Swing library stays at the base of this class and view, it has as attributes swing elements like JButton, JTextArea, JLabel, JPanel, or JFrame.

The frontend part of the gui classes are explained in the graphical user interface section. The classes are implementing not only the IRestaurantProcessing interface, but the ActionLIstener as well. There is not used any layout helper, every element is placed by its own (this method of using gui is more difficult near the fact that I do not use any FrameWork options to construct the view by element placing method).

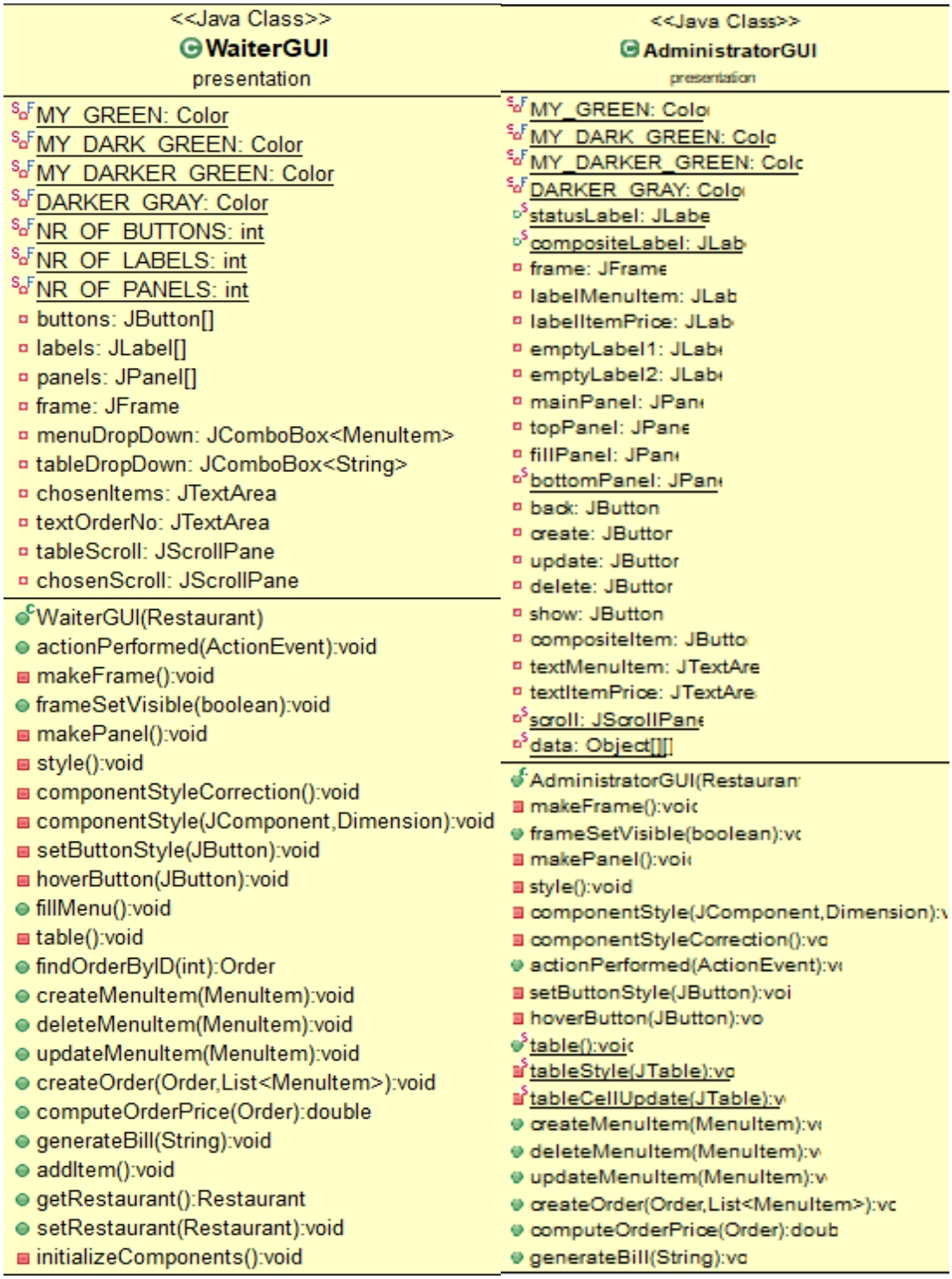
Let's see the model of classes in this package:

(They are not placed in order, because the size of their classes, but as explained before, the selector opens the waiter, the chef and the administrator (which opens the composite) interfaces.)

The gui classes are implementing the IRestaurantProcessing interface, inheriting the methods from there. It must be highlighted that the ChefGUI is an observer, thus every time a new order is placed, the waiter gui uses the notifyAllObservers() method, to notify the chef about the new order, and the observer uses the update() function from the chefGUI saying exactly what happens at the time that it is notified.

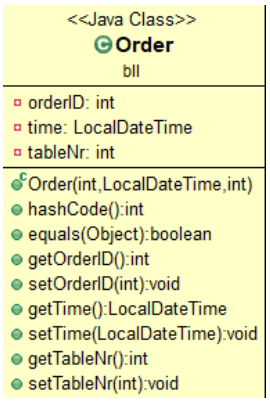


These gui classes are bigger than the others, having more operations possible, each having an automatically refreshing table, having multiple buttons for the different functions, that they are allowed to make.



4.4. Business Layer ("bll") package

4.4.1. Order class



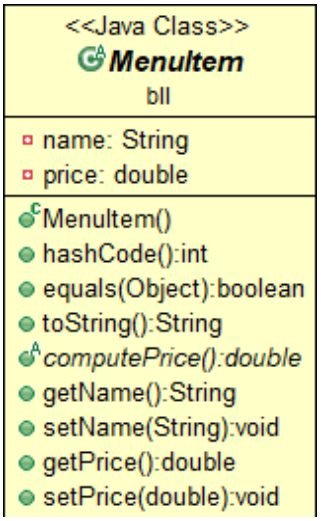
The Order class has the following attributes:

```
private int orderID;
private LocalDateTime time;
private int tableNr;
```

Order is used as a key later in a HashMap structure, so hashCode() and equals(Object obj) are implemented, and the default Eclipse methods were used.

It also contains the getters and setters for the attributes presented in the class

4.4.2. MenuItem Class



The MenuItem is an interface having the following methods (shown in the right):

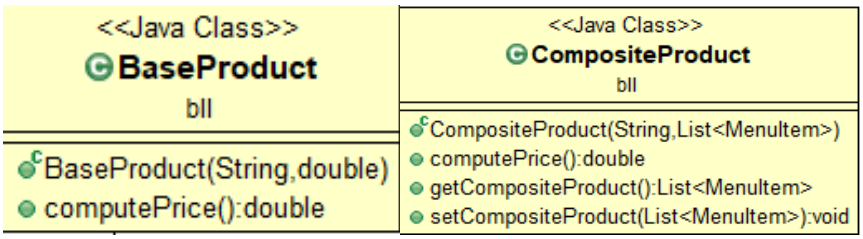
Abstract class that resembles a product from the restaurant's menu. It can be a BaseProduct or a CompositeProduct, for those options, the price is computed accordingly.

from the composite pattern point of view the compute price method is the most important, for which the user does not have an exact access, but it is used by the generate bill method to find the total price of the order, while it is used to calculate how much does a menu cost, built from different base products.

The toString() overridden function is for diplaying the menu items nicely, to be able to represent them in tables, in the bill and so on.

4.4.3. BaseProduct and CompositeProduct Classes

These classes extend the MenuItem class, being known that a menu item can be a base product and a composite item as well, depending on that it is a single product (like an apple) or it is a menu itself (like a lunch, containing soup, second plate and dessert for instance). The BaseProduct class represents a base product which has a name and a price as an attribute (private String name, and private double price). In this class, the compute price method simply returns the price of the object, because in its constructor it was specified. The CompositeProduct class represents a composite product having multiple base products.

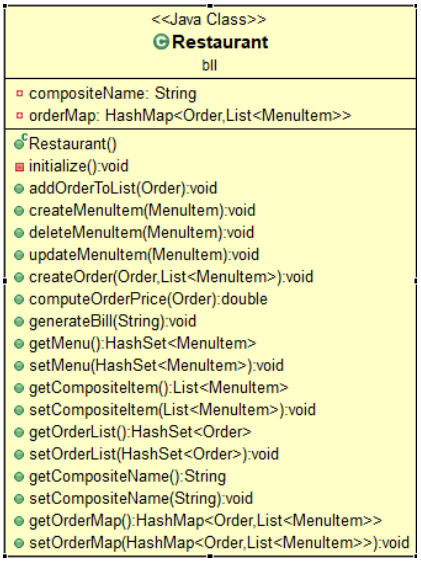


4.4.4. IRestaurantProcessing Interface

The restaurant interface which is implemented by the restaurant and the GUI classes as well. It contains the main operations that can be executed by the above-mentioned classes.

```
public interface IRestaurantProcessing {
    /**
     * creates a menu item, choosing the right type (composite or base)
     * @pre the given item is not null
     * @inv the menu item itself, putting in the set
     * @post the difference between the old size and the new size is exactly 1
     * @param item, the new item which is about to be added to the menu
     */
    public void   createMenuItem(MenuItem item);

    /**
     * deletes a menu item, choosing the right type (composite or base)
     * @pre the given item is not null
     * @inv the menu item itself, putting in the set
     * @post the difference between the old size and the new size is exactly minus1
     * @param item, the selected item which is about to be deleted from the menu
     */
    public void   deleteMenuItem(MenuItem item);

    /**
     * updates a menu item, choosing the right type (composite or base)
     * @pre the menu item is not null
     * @inv the update method, itself
     * @post the new price and the given item price are equal
     * @param item, the selected item which is about to be updated in the menu
     */
    public void   updateMenuItem(MenuItem item);

    /**
     * creates a new order, required by the waiter, notifying the chef
     * @pre the given order is not null
     * @inv the order itself, putting in the map
     * @post the given order and the created order are equals
     * @param order, the information about the order
     * @param menuItem, the list of menu items, ordered by the clients
     */
    public void   createOrder(Order order, List<MenuItem> menuItem);

    /**
     * computes the total price of an ordered, which has to be paid by the client
     * @pre the given item and order are not null
     * @inv the menu item itself, putting in the set
     * @post the difference between the old size and the new size is exactly 1
     * @param order, the information about the given order
     * @return the total price of the order
     */
    public double computeOrderPrice(Order order);

    /**
     * generates the bill in .txt file
     * @pre the given bill content (String) is not null
     * @inv creating the bill.txt file
     * @param content, the content printed in the text file
     */
    public void   generateBill(String content);
}
```

4.4.5. Restaurant Class



First and foremost, the class view looks like the one on the right →

where the attributes are the following:

```
private HashSet<MenuItem> menu;
private List<MenuItem> compositeItem;
private HashSet<Order> orderList;
private String compositeName;
private HashMap<Order, List<MenuItem>> orderMap;
```

This class that is the center of this application, every action must at least pass through the restaurant if not executed by it. It implements the IRestaurantProcessing interface and all the methods present there. It is also built using Design by Contract, testing the preconditions and postconditions with assert instructions in each implemented method.

4.5. Data Layer ("dao") package

4.5.1. RestaurantSerializator Class

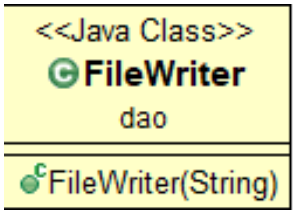
This The RestaurantSerializator class contains methods for storing data. There are methods for serialization the whole Restaurant object or only the HashSet used to store the menu. In my application a used the methods for storing the content of the HashSet. This class uses the following built in libraries:

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
```

And in this class there are two methods, a method for serialization, for saving the data about menu items and order to a .ser file, and a method for reading the data from there.

```
public static void serialize(Restaurant restaurant) {
            try {
                  FileOutputStream fileOut = new
FileOutputStream("restaurant.ser");
                  ObjectOutputStream out    = new ObjectOutputStream(fileOut);
                  out.writeObject(restaurant);
                  out.close();
                  fileOut.close();
                  System.out.println("Serialized data is saved in restaurant.ser");
            } catch (IOException i) {
                  i.printStackTrace();
            }
      }
```

4.5.2. FileWriter Class



In this class the text file is generated, with de bill, containing the relative information about the selected order, such as the ordinal number of the table, the orderID, the date when the order was taken, the products that were ordered, and the prices of it. In this project I tried to personalize a bit the order, to look like a real bill. The generated bill.txt looks like the following:

```
 _____________________________________
|                                     |
|               BILL                  |
|                                     |
|        Restaurant Ciumani           |
|_____________________________________|
|                                     |
| OrderID: 2                Table: 4  |
|       2020-05-06  00:38:43          |
|_____________________________________|
|                                     |
| coca cola ---------------- 5.5 RON  |
| coca cola ---------------- 5.5 RON  |
| cheeseburger ------------- 6.0 RON  |
| cheeseburger ------------- 6.0 RON  |
| apple -------------------- 3.8 RON  |
|_____________________________________|
|                                     |
| Total price: ------------ 26.8 RON  |
|_____________________________________|
|        Reaturant Ciumani            |
|  Romania, HR, Ciumani 14, 537050    |
|        Tel: +40753624586            |
|     www.restaurantciumani.ro        |
|=====================================|
|                                     |
|                                     |
|        ___________________          |
|        (name, signature)            |
|                                     |
|                                     |
|    Thank you for your visit!        |
|        Have a nice day!             |
|_____________________________________|
```

5. Usage and Testing

The user starts the application, he can choose from the three options depending on who it is. There is a window for the admin, a window for the waiter and one for the chef.

The admin can add, edit and delete items from the menu and build some composite products selecting base products. He can introduce data to the labeled text fields and press the corresponding button.

The waiter can select items from the menu preparing a new order and when all the items are selected, by pressing the Create Order button, the order is saved and a message is sent to the chef, who will see the details of the order.

The exact steps are shown in the use case section.

6. Conclusion

After I finished this project, I reckon that my mind has widened while I was able to dive into the Java implementation, improving my java skills, learning new techniques, and working more beautiful than until now. There is no doubt this project was a good continue for our set of experiences.

During this assignment I have got more confidence to create a view manually (this was my second real GUI), because before the last two gui-s I have done it only with using some drag and drop based applications, like NetBeans. Other aspects that I learnt was Design by Contract implementation, the Composite Design Pattern, the Observer Design Pattern, which helped me to understand how to use such a pattern, which are the advantages of it, and why should I work with it.

During the development of the restaurant management system, I met a lot of new information, built-in classes, GUI nucleus, and so on. I can say that I am more experienced now in working on predefined UML diagrams. The other thing I observed was how can I optimize this application, and how can I develop in the future.

Future developments:

- More functionalities
- Chef being able to give feedback when an order is ready
- Easier handling of base and composite product
- Clearer graphical user interfaces
- Creating a login system
- Bug fixes and improvements

7. Bibliography

Programming Techniques Courses from Ioan Salomie

http://www.tutorialspoint.com/java/java_serialization.htm

https://en.wikipedia.org/wiki/Composite_pattern

https://www.geeksforgeeks.org/composite-design-pattern/

https://www.tutorialspoint.com/design_pattern/observer_pattern.htm

http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW4_Tema4/

https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html

https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

https://www.w3schools.com/java/java_hashmap.asp