Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

2nd Semester 2018-2019

Programming Techniques
Homework 4

Student: Elekes Lukacs-Roland

Group: 30424

Supervising teacher: Viorica Chifu

Contents

1. Objectives 3
2. Problem Analysis, Modelling, Scenarios, Use Cases 3
    2.1. Problem Analysis 3
    2.2. Modelling 3
    2.3. Scenarios, Use Cases 4
3. Design 6
    3.1. Class Design, Packages and UML Diagrams 6
    3.2. Algorithms and Data Structures 7
    3.3. Graphical User Interface 8
4. Implementation 10
5. Usage and Testing 13
6. Conclusions 13
7. Bibliography 14

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:

- Design by Contract Programming Techniques
- Design Patterns: Observer, Composite
- Serialization
- JCF HashMap and HashSet implementations

2. Problem Analysis, Modelling, Scenarios, Use Cases

2.1. Problem Analysis

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 more 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 any 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.

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 have to 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 have to 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.

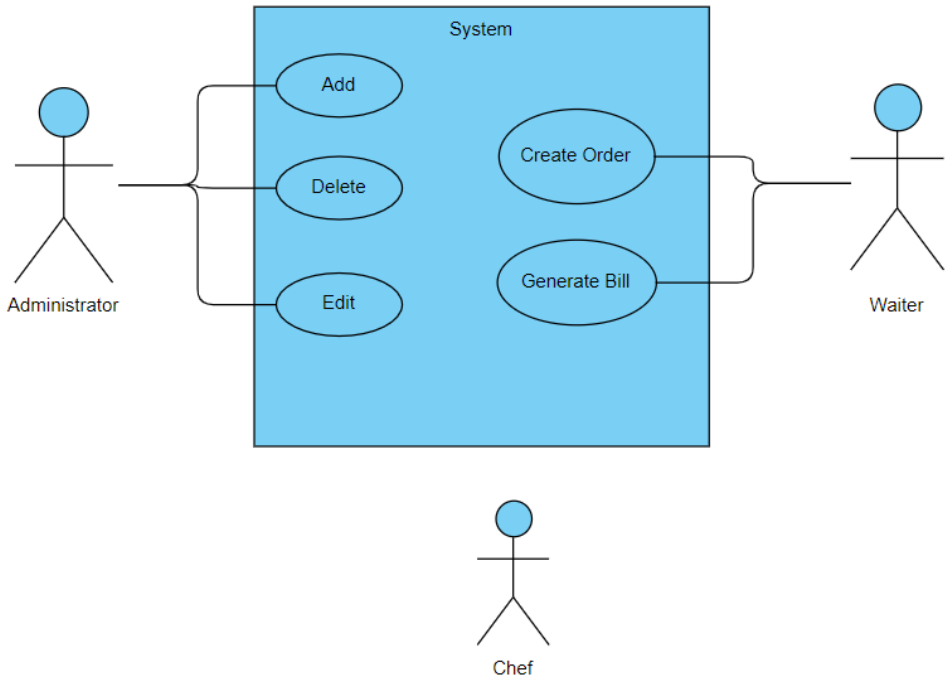
The data needs to be stored in some files, from where it can be loaded every time the application is started.

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, and he can compute the price (also generating a bill) for a certain order. The chef is only an observer, when a new order is created by the waiter, the details regarding the chef appear in his window.

Each user has its own interface. After the abovementioned 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 the following:



The actors are the administrator and the waiter, the chef is only an observer, who receives information from the waiter.

Use case title: Add

Main success scenario:

1. Administrator introduces new data
2. Administrator presses the “Add” button
3. System takes information
4. New item is inserted and stored

Alternative sequence:

Data introduced by the administrator is not correct (e. g. administrator introduces text instead of number)

1. Administrator introduces new data
2. Administrator presses the “Add” button
3. System takes information
4. System displays a message with the error
5. The administrator can return to the first step

Use case title: Edit

Main success scenario:

1. Administrator introduces new data
2. Administrator selects the item to be edited
3. Administrator presses the “Edit” button
4. System takes information
5. The item is edited and stored

Use case title: Delete

Main success scenario:

1. Administrator introduces new data
2. Administrator presses the “Add” button
3. System takes information
4. New item is inserted and stored

Use case title: Place an order

Main success scenario:

1. Waiter introduces the table number (order ID and date are generated automatically)
2. Waiter selects menu items one by one after each one pressing the “Add Selected” button
3. Selected items appear in a list
4. Waiter presses “Place Order” button

5. The order is saved and communicated to the chef

Use case title: Compute price/Generate bill

Main success scenario:

1. Waiter introduces the number of table for which he wants to generate the bill
2. Waiter presses “Compute Price” button
3. Bill is generated in a .txt file, also it is displayed on the screen

3. Design

3.1. Class Design, Packages and UML Diagrams

The project is built on a layered architecture represented by three important layers, the business layer, presentation layer and data layer. 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 four: businessLayer, dataLayer, presentationLayer and start.

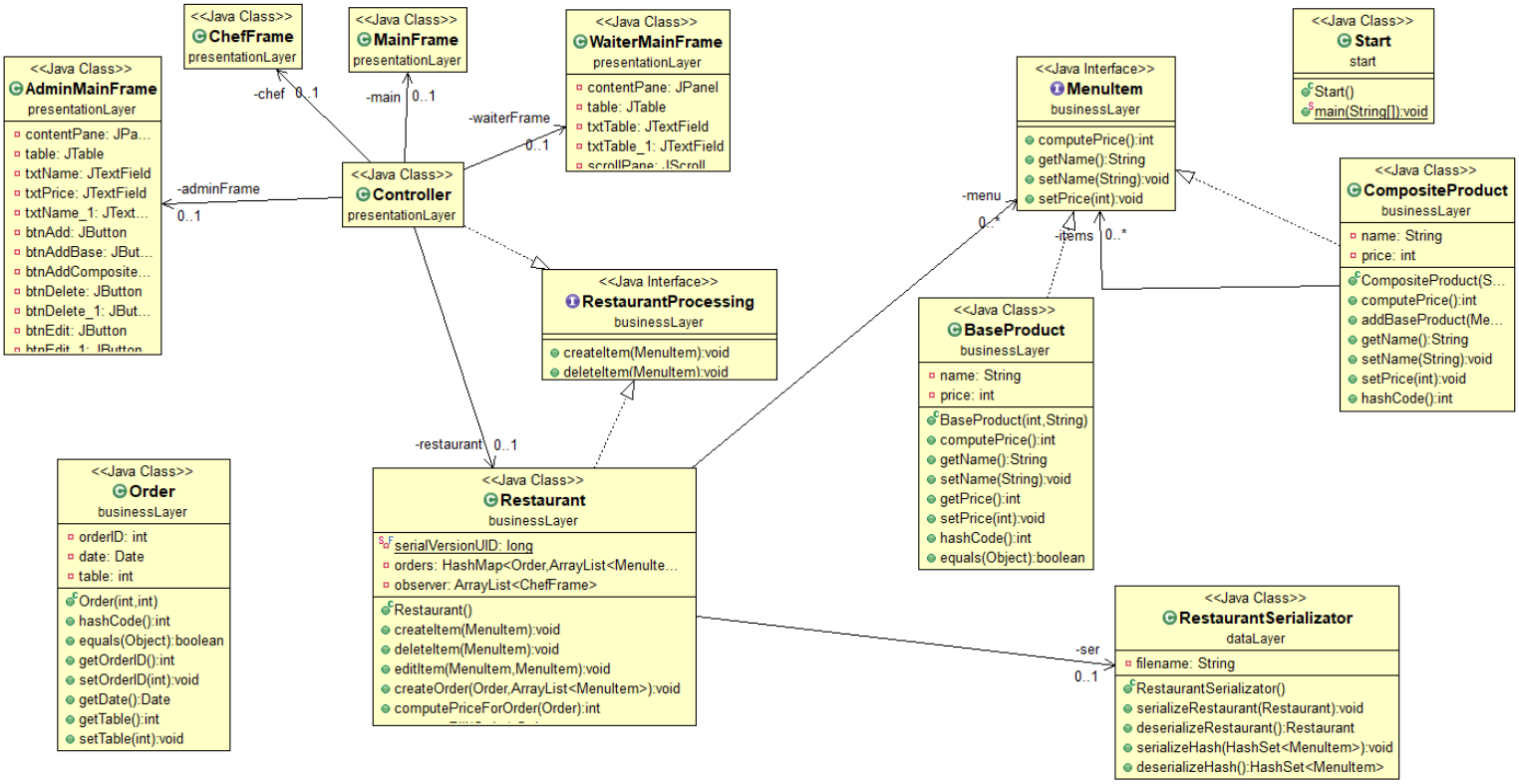
The packages businessLayer and dataLayer are part of the model. The business layer contains the entities like Order, Restaurant and MenuItem. For designing the Menuitem entity, composite pattern was used, there are Base Products and Composite Products composed by base products. Both categories can represent a Menu Item which is an interface. The class Restaurant implements the operations described in the RestaurantProcessing interface (maintaining data, generating bills, creating orders etc.) .

In the data layer the class RestaurantSerializator is placed which saves and loads the data from a file.

In the presentation package and layer, the classes that are part of the GUI are placed and the Controller class, which is the connection between the presentation and business logic(model). There is a main window from where the users can access their own windows.

The start package contains the Start class that has the main method to start the application.

The UML class diagram is the following:



3.2. Algorithms and Data Structures

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.

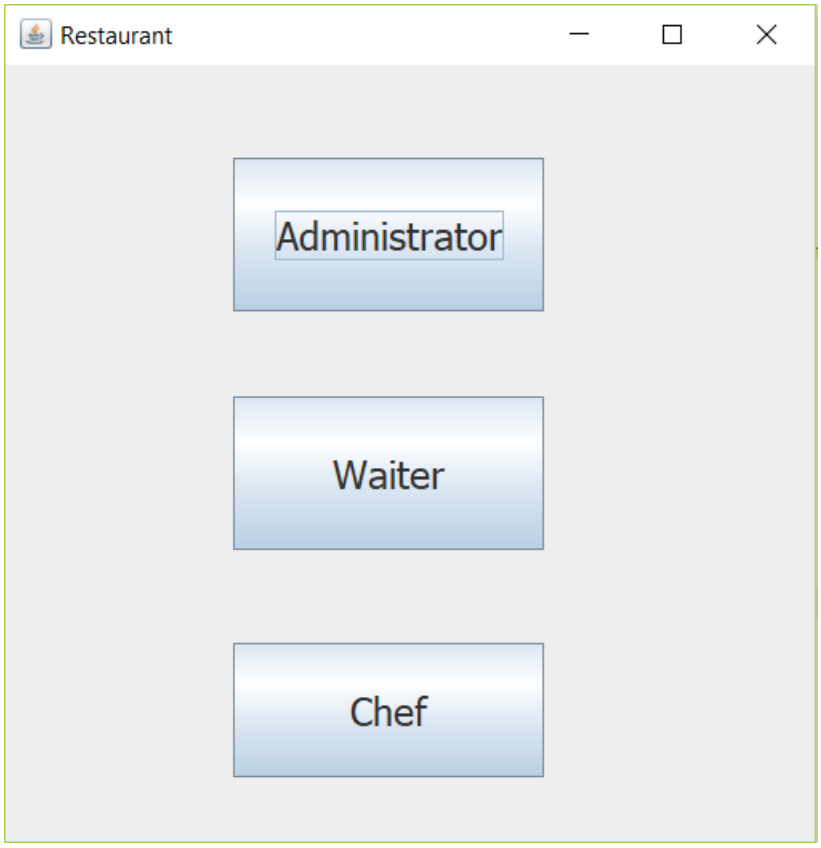
There are some important data structures used from the Java Collections Framework. For storing the menu of the restaurant the HashSet is used, since a Menu Item is unique.

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. Here a list is used instead of a set, because at this part there can appear duplicates.

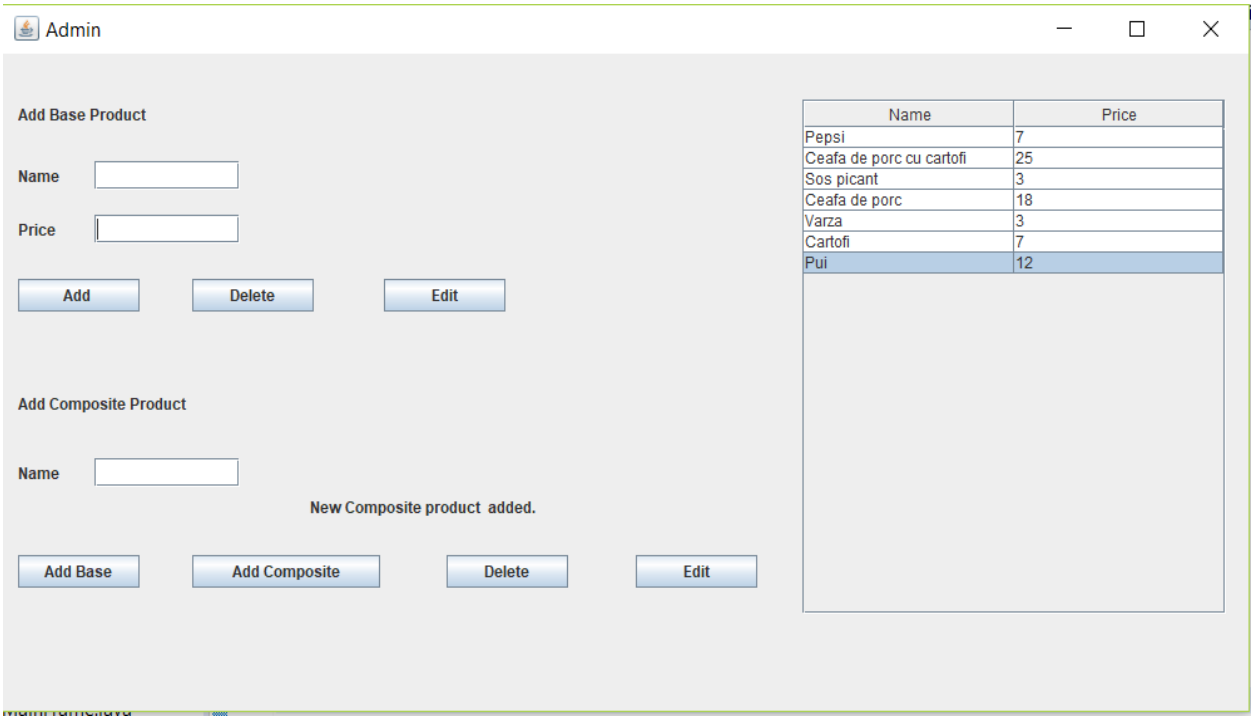
3.3. Graphical User Interface

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.

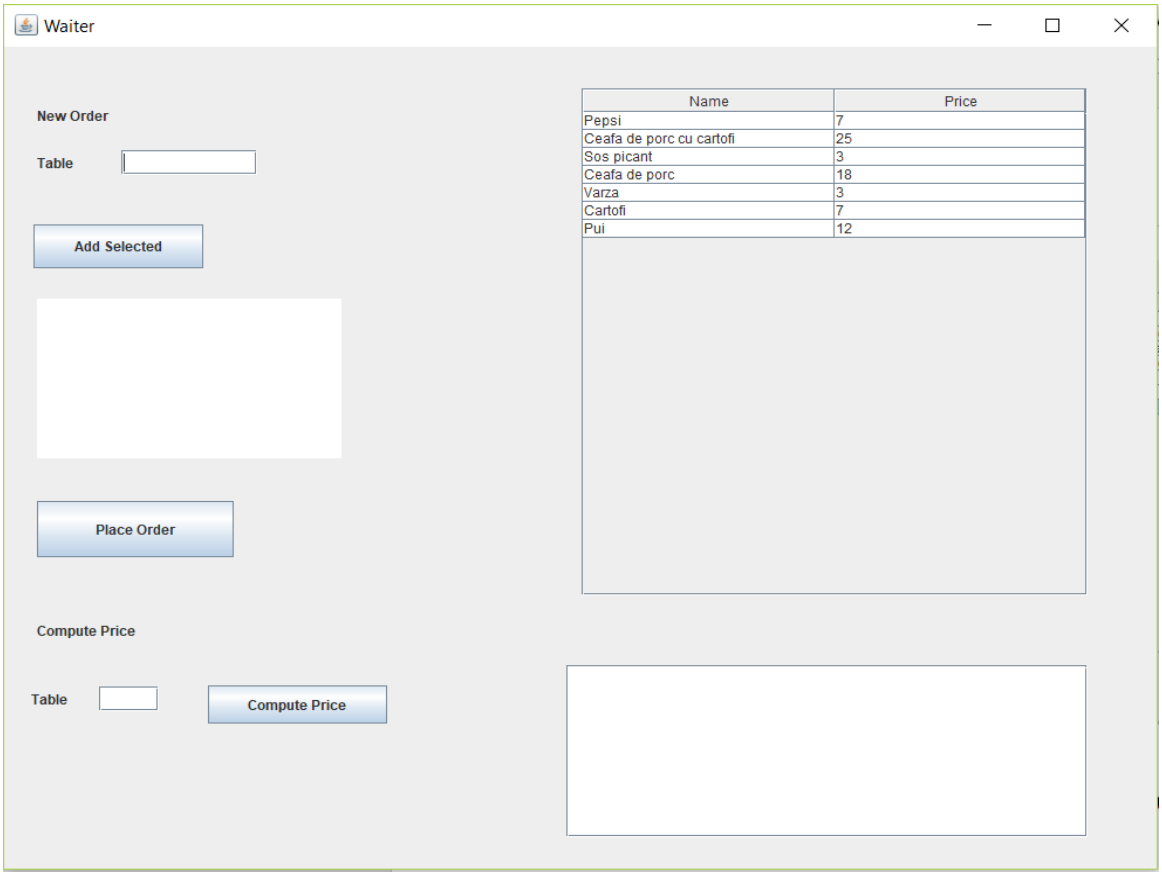
It is constructed using the swing library and the windowbuilder of Eclipse to have a nicer GUI and to save time. There are four frames (JFrame) in total. When the application starts, the first frame appears, where the user is told that he can choose an option : Admin, Waiter or Chef.



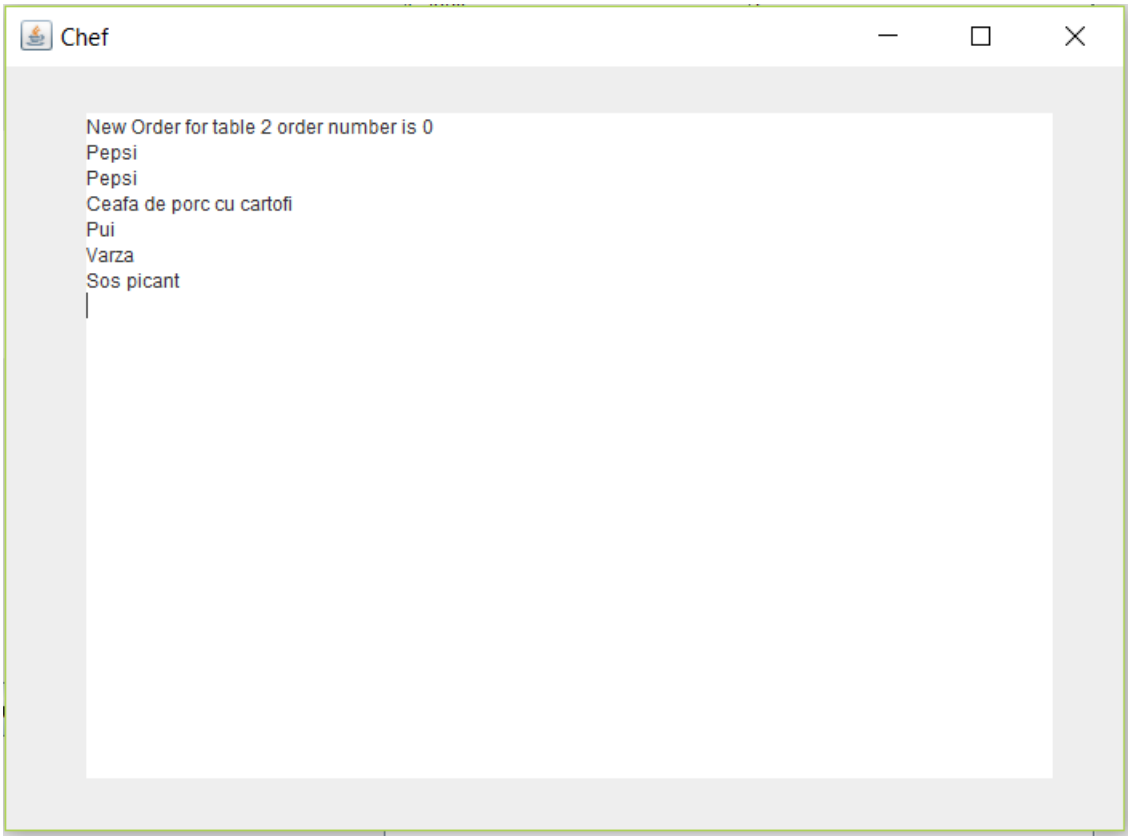
The application can be closed from this window. On this window by clicking the Administrator button, the interface for the administrator is opened.



When the user chooses Waiter the window dedicated to the waiter appears.



When the user chooses Chef, the window for the chef appears, where the information of last order is displayed.



If a wrong value of data is introduced (text instead of number for example) a message dialog box appears with the information regarding the error.

4. Implementation

The Order class has the following attributes:

```
private int orderID;
private Date date;
private int table;
```

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

The MenuItem is an interface having the following methods:

```
public abstract int computePrice();
public abstract String getName();
public abstract void setName(String name);
public abstract void setPrice(int price);
```

from the composite pattern point of view the compute price method is the most important.

The BaseProduct class represents a base product which has a name and a price as an attribute:

```
private String name;
private int price;
```

HashCode is implemented in this class also, because be store the items in a hash set:

```
public int hashCode() {
	final int prime = 31;
	int result = 1;
	result = prime * result + ((name == null) ? 0 : name.hashCode());
	result = prime * result + price;
	return result;
}
```

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.

The compute price method is the following:

```
public int computePrice() {
	int computedPrice = 0;
	for(MenuItem item : items) {
		computedPrice += ((BaseProduct)item).getPrice();
	}
	this.price = computedPrice;
	return computedPrice;
}
```

The RestaurantProcessing interface contains the following methods:

```
public void createItem(MenuItem item);
public void deleteItem(MenuItem item);
public void editItem(MenuItem itemOld, MenuItem itemNew);
public void createOrder(Order order, ArrayList<MenuItem> items);
public int computePriceForOrder(Order order);
public String generateBill(Order order);
```

and the Restaurant class implements them. Firstly, its attributes are :

```
private HashSet<MenuItem> menu;
private HashMap<Order, ArrayList<MenuItem>> orders;
private RestaurantSerializator ser ;
private ArrayList<ChefFrame> observer;
```

Menu is the menu of the restaurant, orders is a HashMap where orders are saved, key is an Order and value is a list of menu items. Ser is responsible for serialization and observer is a list containing the observers, restaurant being the subject in the observer design pattern.

Most of the methods consist of removing from or adding to the data structures defined.

```
public void createOrder(Order order, ArrayList<MenuItem> items) {
		this.orders.put(order,  items);
		notifyAllObservers(order);
	}

public void deleteItem(MenuItem item) {
		this.menu.remove(item);
		ser.serializeHash(this.menu);
	}
```

The generateBill method constructs a string containing the information of the orders.

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.

Classes from the presentation layer represent the GUI, they were built by the Eclipse windowbuilder and later modified by hand to process easier the buttons and events.

The Controller class creates the connection between the presentation layer and business layer and implements the RestaurantProcessing interface to assure all the functionalities that this application gives.

```
	public void createItem(MenuItem item) {
		this.restaurant.createItem(item);
		//this.serializator.serializeRestaurant(this.restaurant);
		adminFrame.setTable(createTable(restaurant.getMenu()));
		waiterFrame.setTable(createTable(restaurant.getMenu()));
	}

	case "addadmin" :
		price = adminFrame.getPrice();
		name = adminFrame.getNameBase();
		createItem(new BaseProduct(price, name));
		break;
```

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 Place Order button, the order is saved and a message is sent to the chef, who will see the details of the order.

6. Conclusions

This application is a basic restaurant management system, only an example but it can be a good base for developing such an application. From this application, many other features and a more complex application can be developed.

During this assignment, some research was needed for being able to use the design patterns correctly. I have learnt how to use the composite and observer design patterns, I used an alternative for database for storing data (serialization). I can say that I am more experienced now in working on predefined UML diagrams.

Future developments:

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

7. Bibliography

- http://users.utcluj.ro/~cviorica/PT2019/
- 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