*Technical University of Cluj-Napoca*

*Faculty of Automation and Computer Science*

*2nd Semester 2019-2020*

**Programming Techniques**

**Homework 4**

Student: Osváth Tamás

Group: 30421

Supervising teacher: Pop Cristina

Contents

[1. Objectives 3](#_Toc39509168)

[2. Problem Analysis, Modelling, Scenarios, Use Cases 3](#_Toc39509169)

[2.1. Problem Analysis 3](#_Toc39509170)

[2.2. Modelling 3](#_Toc39509171)

[2.3. Scenarios, Use Cases 4](#_Toc39509172)

[3. Design 7](#_Toc39509173)

[3.1. Class Design, Packages and UML Diagrams 7](#_Toc39509174)

[3.2. Algorithms and Data Structures 9](#_Toc39509175)

[3.3. Graphical User Interface 9](#_Toc39509176)

[4. Implementation 13](#_Toc39509177)

[5. Usage and Testing 16](#_Toc39509178)

[6. Conclusions 17](#_Toc39509179)

[7. Bibliography 17](#_Toc39509180)

# 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 that is ordered through a waiter.

Secondary objectives:

* Design by Contract Programming Technique
* Observer design pattern
* Composite design pattern
* 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 can be useful for various businesses, such as restaurants, cafes, bistros which have the three basic users: Administrator, Waiter and Chef. The communication is done automatically between the users. The Waiter and the Chef communicate by using the Observer Design Pattern. Also, when the application is running, the Administrator can add, delete and update items from the menu, the Waiter can create new orders, compute order prices and generate bills based on the orders. The application has a simple, clear and well-organized graphical user interface, which makes it very user-friendly and easy to use.

## 2.2. Modelling

For such a system it is necessary to have some basic elements. The basic element for storing a product is the MenuItem, which is an interface. Two classes implement this interface, BaseProduct, a singular product, and CompositeProduct, which is basically a set of base products. Also, there need to be defined some operations for the application functioning, like adding a product in menu, deleting/ updating it, operations for the waiter: creating a new order, calculating price of the order and generating a bill, and so on.

The graphical user interface has to be specific for each user and give some limitations. For example, a waiter is not supposed to have access to menu manipulation operations (create/ delete/ edit product), also a chef should not be able to create order and so on.

The data of the application should be saved in some way. For the current application, serialization will be used. So, each time a new operation is done, the corresponding data is saved through serialization.

## 2.3. Scenarios, Use Cases

Scenarios include manipulating and maintaining the data of the restaurant.

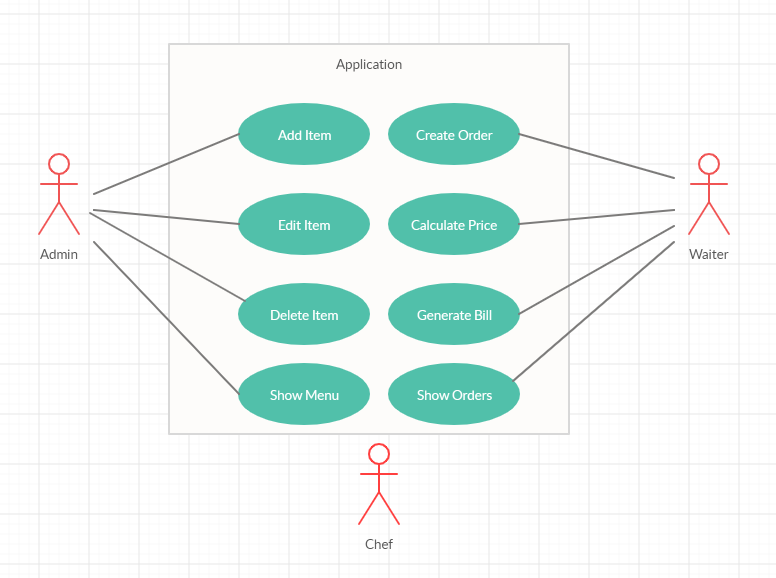
When the actor is the Admin, he can add, edit and delete items from the menu.

When the actor is the Waiter, he can create new orders, calculate prices for orders and generate bills.

When the actor is the Chef, there is no manipulation of the data, he can only visualize the orders, since he is an Observer.

Each user has its own interface, where he can see do its own job.

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 and displays some data.

**Use case title**: Add Item

Main success scenario:

1. Administrator introduces new data
2. Administrator presses the “Add” button
3. System takes information
4. New item is inserted into the menu and stored, success message displayed on the screen

Main failure scenario:

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: “Item creation failed”
5. The administrator can return to the first step

**Use case title**: Delete

Main success scenario:

1. Administrator introduces new data
2. Administrator presses the “Add” button
3. System takes information
4. The item is deleted from the menu, success message displayed on the screen

**Use case title**: Edit

Main success scenario:

1. Administrator introduces new data (the name of the base product to be modified and its price)
2. Administrator presses the “Edit” button
3. System takes information
4. The item is edited and stored in the menu, success message displayed on the screen

**Use case title**: Add Composite item

Main success scenario:

1. Administrator introduces the name of the composite product
2. Administrator selects the products from the menu which for creating the composite
3. Administrator presses the “Add Composite” button
4. System takes information
5. New item is inserted into the menu and stored, success message displayed on the screen

**Use case title**: Delete Composite item

Main success scenario:

1. Administrator introduces the Name of the composite product
2. Administrator presses the “Delete Composite” button
3. System takes information
4. The item is deleted, success message displayed on the screen

**Use case title**: Show Menu

Main success scenario:

1. Administrator presses the “Show Menu” Button
2. System takes information
3. The Menu is shown in a Table form (JTable)

**Use case title**: Place an order

Main success scenario:

1. Waiter introduces the table number (order ID and date are generated automatically by the application)
2. Waiter selects menu items from the table to be added to the order
3. Waiter presses “Add Order” button
4. The new order is saved and communicated to the chef, success message displayed on the screen

**Use case title**: Calculate price

Main success scenario:

1. Waiter selects the table for which wants to compute the price
2. Waiter presses “Calculate Price” button
3. The price of the order is displayed on the screen

**Use case title**: Place an order

Main success scenario:

1. Waiter selects the table for which wants to generate the bill
2. Waiter presses “Generate Bill” button
3. Bill is generated in a “.txt” file and success message is displayed on the screen

**Use case title**: Show Orders

Main success scenario:

1. Administrator presses the “Show Orders” Button
2. System takes information
3. The Menu is shown in a Table form (JTable)

# 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. The application is based on the Model View Controller pattern, and for designing the classes there were used some other design patterns: Composite and Observer.

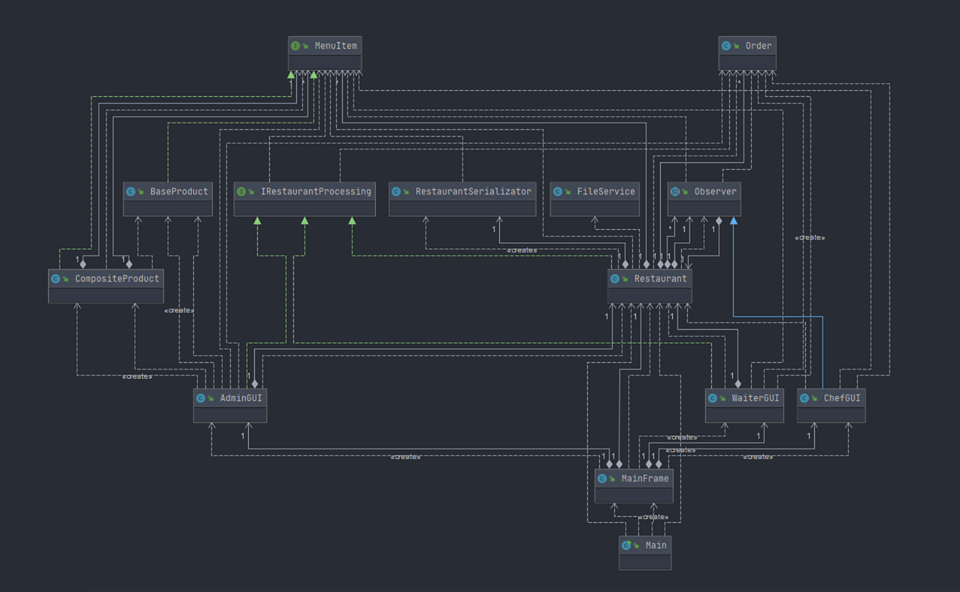
The project has 3 packages, as stated in the requirements: businessLayer, dataLayer and presentationLayer.

The business layer contains the model of the program, and is composed of the following: MenuItem and IRestaurantProcessing interfaces, and BaseProduct, CompositeProduct, Obsever, Order and Restaurant classes.

In the data layer two classes can be found: RestaurantSerializator for saving and loading the data of the restaurant from a file, and FileService, which is responsible for opening, closing and writing to a file (for the bills).

In the presentation layer the classes which implement the graphical user interface can be found, namely the AdminGUI, WaiterGUI, ChefGUI and MainFrame classes. Also, this package contains the Main class which is responsible for running the application.

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

The UML class diagram is the following:

The relationships are the same as in the laboratory requirements.

## 3.2. Algorithms and Data Structures

The algorithms needed for the implementation of the application are simple, they consist mainly in adding and removing from collections, and iterate through them.

As was stated in the assignment’s description, in the Restaurant class it was needed to use some predefined collection from Java Collections Framework.

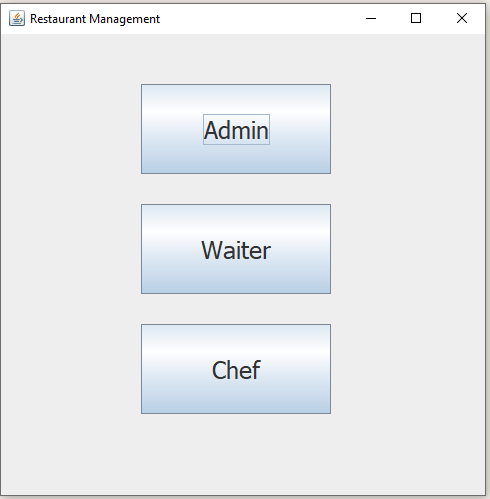
For storing the menu HashSet is used, since a Set can’t contains duplicates.

Fir storing the orders HashMap is used, having an Order type of key, and an ArrayList of MenuItems type of value. At this point, the HashSet of items is changed to ArrayList, since duplicates can appear.

## 3.3. Graphical User Interface

The Graphical User Interface provides all the necessary mechanisms for the user to use the application. It is implemented in a very minimalistic way, thus being easy to use for anyone. Every frame contains some text fields for introducing data, labels for information, representative buttons for doing operations and tables to display data. It is implemented with the swing library, no drag and drop tools or helper methods.

When the application starts, a “Login” frame is shown, where the user chooses the way he is using the application: As an Admin, Waiter or Chef. A future development can be a login based on some username and password. The application can be closed from this window, by clicking the ‘X’ on the upper right corner.

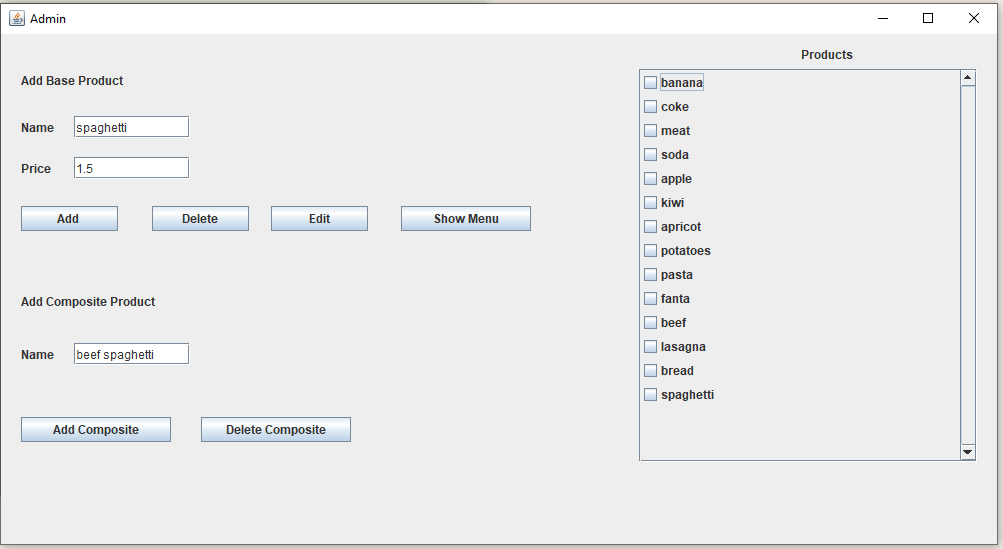


**Administrator Interface**

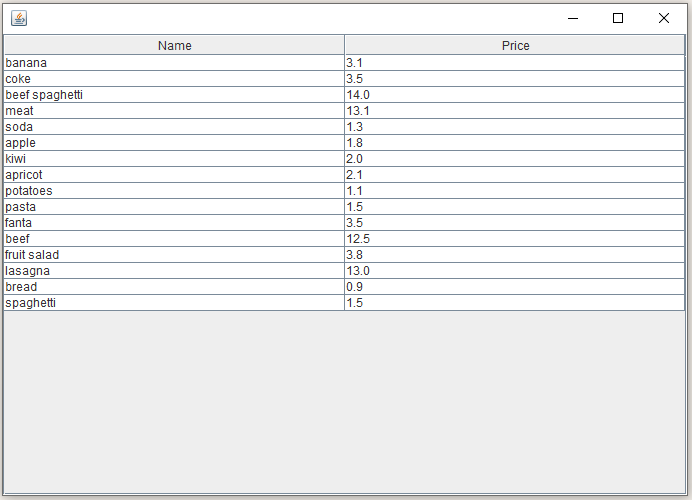
By clicking the ‘Admin’ button on the main frame, the Admin interface is opened.

Base products can be added/ edited/ deleted by typing the name and the price and pressing the corresponding button.

Composite product can be added or deleted by selecting the products from the box in the right, and then pressing the corresponding button.



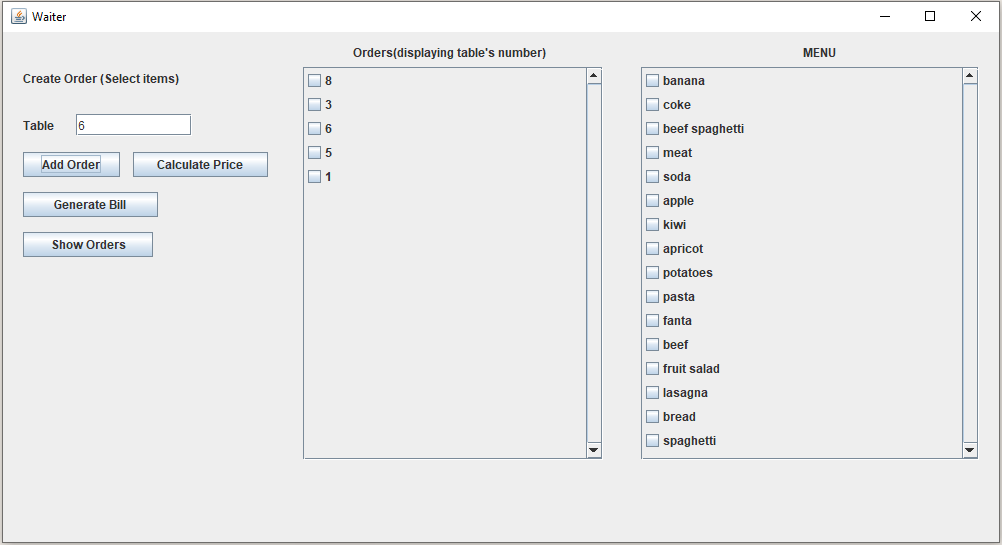
When the ‘Show Menu’ button is pressed, the entire menu is displayed.

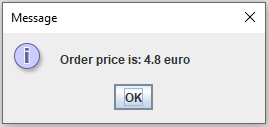


**Waiter Interface**

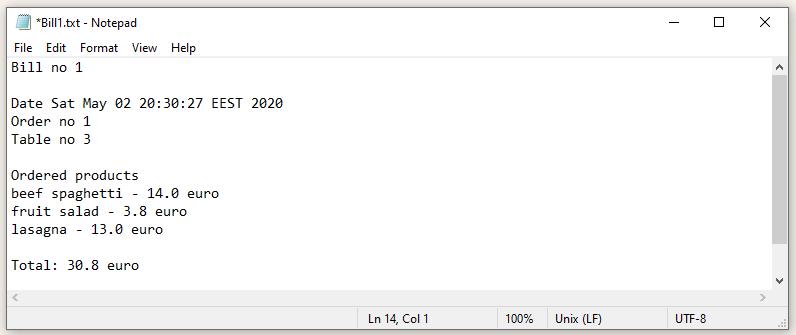
By clicking the ‘Waiter’ button on the main frame, the Waiter interface is opened.

For adding an order, the user types the number of the table, and selects the products from the menu to be added, finally presses the ‘Add Order’ Button.

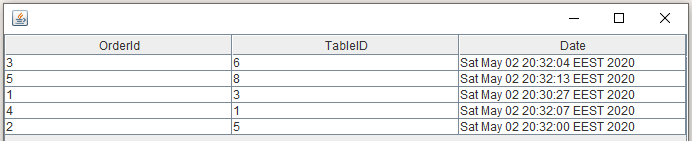


For calculating the price of an order, the order’s table from the list need to be selected and then the ‘Calculate Price’ be pressed. The price will be displayed as a message.

For Generating a bill, the table is selected and the ‘Generate Bill’ is pressed. The Bill will be generated as a “.txt” file in the project’s directory. The Bill contains all the relevant information from an order.



When the user presses the ‘Show Orders” button, a JTable is displayed with all the orders in the current session



**Chef Interface**

By clicking the ‘Chef’ button on the main frame, the Chef interface is opened.Since the Chef is a simple observer, only the orders are displayed.



# 4. Implementation

In this chapter specific details of the implementation will be presented. Since the UML Diagram as well as some class cards are given in the assignment’s description, there is no need to describe every single class.

The **MenuItem, BaseProduct and CompositeProduct** classes are designed according to the Composite Design Pattern.

The MenuItem is an interface having the following methods:

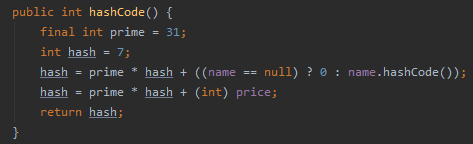
**Float** computePrice();

String getName();

**void** setName(String name);

**void** setPrice(**float** price);

HashCode is implemented in both BaseProduct and CompositeProduct classes, because the MenuItems are stored in the Restaurant class as a HashSet. The hashCode implementation is the basic Eclipse implementation:



The HashCode is also overwritten in the Order class, in the exactly same manner.

The **IRestaurantProcessing** interface contains the following methods:

**void** createItem(MenuItem item);

**void** deleteItem(MenuItem item);

**void** editItem(MenuItem oldItem, MenuItem newItem);

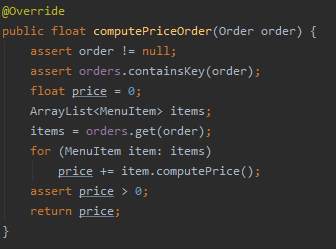
**void** createOrder(Order order, ArrayList<MenuItem> items);

**int** computePriceOrder(Order order);

**void** generateBill(Order order);

The **Restaurant** class implements the IRestaurantProcessing interface, and it is designed using the Design by Contract method, involving pre and post conditions, as well as invariants and assertions. The invariant of the class is the wellDefined method.

\*As a remark, the IRestaurantProcessing interface and the restaurant class contain the JavaDoc descriptions of the pre, post conditions and invariant.

For example, the computePriceOrder (Order order) method which computes the price for an order contains pre and post conditions, as well as assertions. It verifies if the order is not null, if the order is found in the orders list and also if the computed price is positive, assuring that everything works as expected.

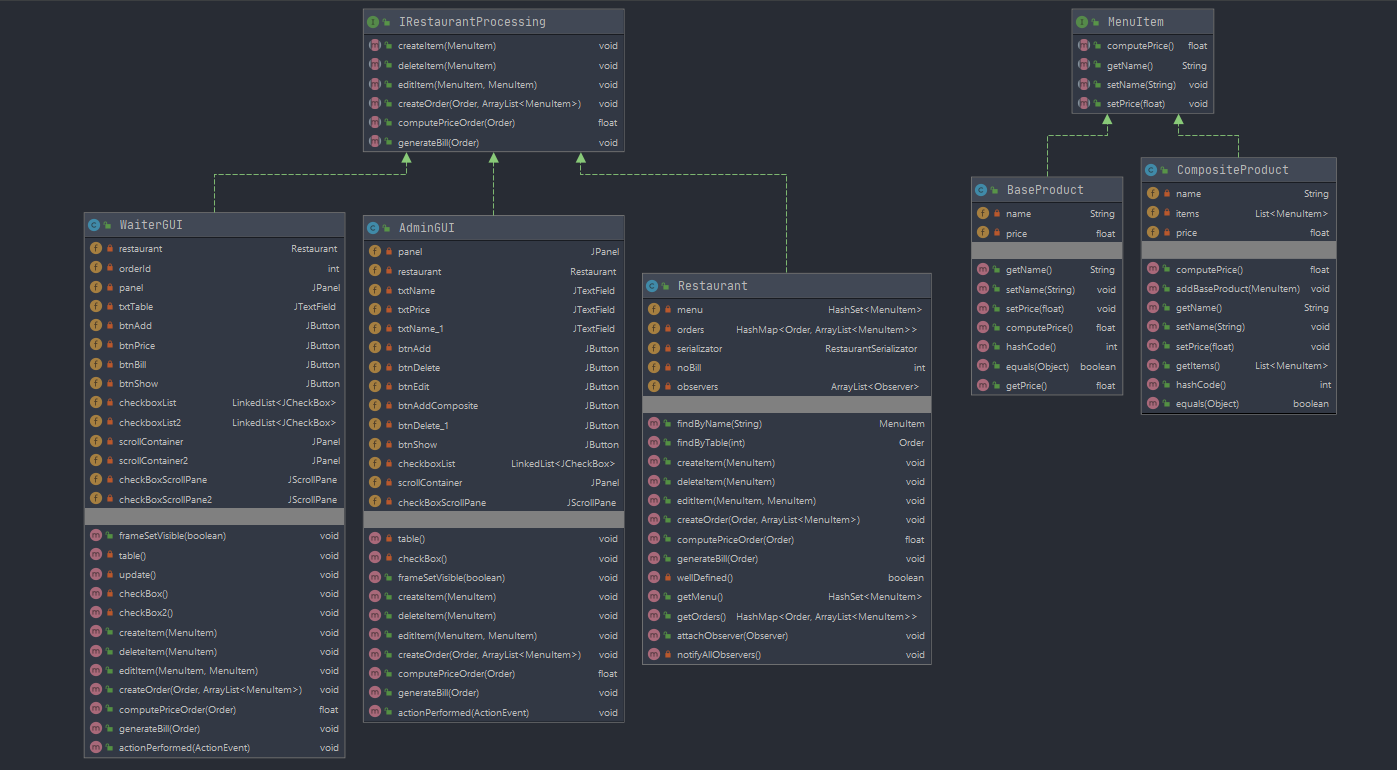
The Menu of the restaurant is kept in a HashSet (HashSet<MenuItem> menu), and the orders are kept in a HashMap (HashMap<Order, ArrayList<MenuItem>> orders), where the key is an order, and the value is an ArrayList consting in menu items. Most of the methods for manipulating these consist only of removing and adding elements.

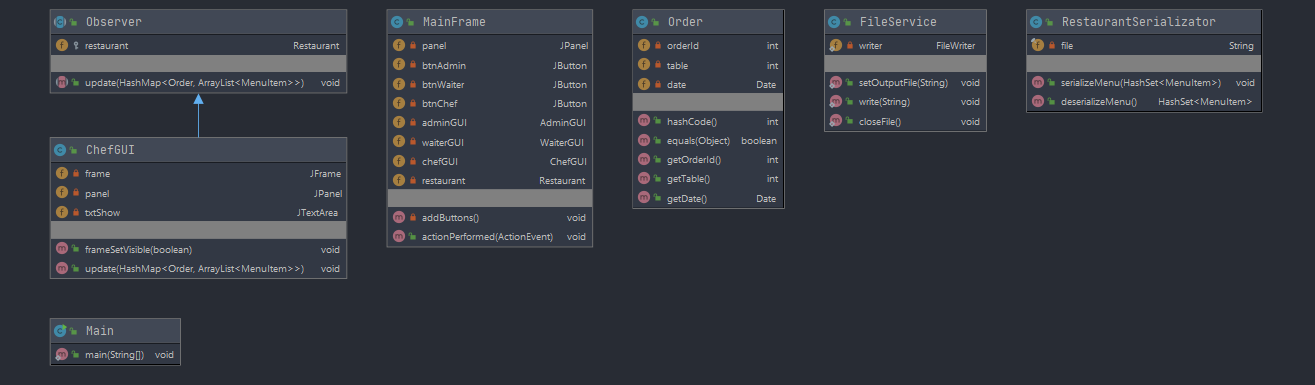
The **FileService** class is responsible for opening a file, writing in it and closing it.

The **RestaurantSerializator** class is responsible for serializing the Restaurant’s data in a “.ser” file.

The **Observer Design Pattern** is implemented with the help of the **Observer** abstract class, and the only observer in the application is the Chef. The Restaurant class is responsible for notifying the observers.

For running the application, the **Main** class is being run, which basically creates a new instance of the Restaurant Class and of the MainFrame class.

Below the class cards can be seen:



# 5. Usage and Testing

The usage of the application is very straightforward since the Graphical User Interface is user-friendly and intuitive.

The user starts the application, and can choose between three option, depending on his role: Admin, Waiter, Chef.

The Admin can add, edit and delete base products from the menu, he can add composite products consisting of products and delete them, and can also view the Menu in a table form (JTable). The data is introduced in the corresponding text fields and specific buttons are pressed. For selecting the products, the admin can use the checkbox from the right side where the products are located.

The Waiter can add orders, calculate prices, generate bills and view the list of orders in a table form. The data is again introduced in the corresponding text fields, and specific buttons are pressed. For selecting the ordered items, in the right side there is a checkbox list from where the products can be selected.

The Chef can only visualize the orders since he is an observer.

Testing was done by introducing all kind of data, including wrongly formatted, in order to be able to find all the exceptions. There are many products added in the menu, so the simulation can be the closest possible to a real life situation.

# 6. Conclusions

This application can be a starting point for developing a complex Restaurant Management System. Many other features can be added in order to fulfill the needs of an application of this type.

During the development of this assignment, some research was needed for understanding the design patterns and the way they can be used. I have learnt to use the Composite Design Pattern, the Observer Design Pattern and the Design By Contract method. Moreover, I exercised the serialization, the implementation of a more complex graphical user interface, and also gained experience on working on a predefined UML Diagram.

Future developments:

* More functionalities to be added
* Improvements on the graphical user interface can be added
* Creating a login system, based on credentials
* Chef can give feedback when an order is ready
* Storing data in a database instead of serializing it
* Adding statistics to the application (e.g. the most appreciated product, preferences, etc.)

# 7. Bibliography

* <http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_4/Assignment_4.pdf>
* <https://www.tutorialspoint.com/design_pattern/observer_pattern.htm>
* <http://www.tutorialspoint.com/java/java_serialization.htm>
* <https://en.wikipedia.org/wiki/Composite_pattern>
* <https://www.geeksforgeeks.org/composite-design-pattern/>
* <https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html>
* <https://www.w3schools.com/java/java_hashmap.asp>
* <https://www.baeldung.com/java-hashcode>