Restaurant Manager

1. Objectives

• Main Objective

The main objective of this homework is the implementation of a restaurant management application. The application should be able to support 3 different users (Administrator, Waiter and Chef). The administrator can add, delete or update the menu of the restaurant. The waiter can place an order, generate a bill for a given order, and save/retrieve the restaurant's state at each step through serialization. The chef is signaled by the waiter every time he must cook a meal. The goal of the application is to simulate a restaurant, without the use of a database. Additionally, the application can provide a bill for each order with detailed billing information.

• Secondary Objectives

The main objective can be further divided into smaller objectives. Together they form the application as a whole.

I. Input checking

When inserting, updating, deleting menu items or when placing orders, the user has to provide certain input data to the application. To assure a good functioning, the application needs to protect itself against malicious or bad input. Therefore, a through input checking is enforced on the user defined data.

II. Serialization and deserialization

The application is strongly reliant on the serialization mechanism, since a database is not supported. At any point, the waiter can save or retrieve the state of the restaurant.

III. CRUD operations on the restaurant's menu

The administrator needs to be able to insert, update and delete products from the restaurant's menu by only interacting with the applications GUI. Moreover, other operations on the restaurant's state also need to be provided without the use of an underlying database.

IV. Order placing mechanism

The waiter needs to be able to select multiple items from the menu and place an order for a given table, directly from the application's GUI. These orders are stored in a specialized data structure for efficiency.

V. Bill generation

The waiter can also generate detailed billing information directly from the applications GUI. The bills are stored in a specialized folder. A bill is created for each order, containing all products that the customer ordered.

VI. Implementing a Graphical User Interface (GUI)

In order to facilitate the use and interaction with the application, a GUI was implemented. The work of the applications user is greatly simplified by means of an intuitive Graphical User Interface.

2. Problem Analysis

The application should be able to fulfill all the requirements in order to keep track of orders. The items on the menu along with the orders placed by the waiters represent the state of the restaurant and can be serialized/deserialized. Inserting, updating, deleting and other operations will be executed directly on the underlying data structures of the restaurant class.

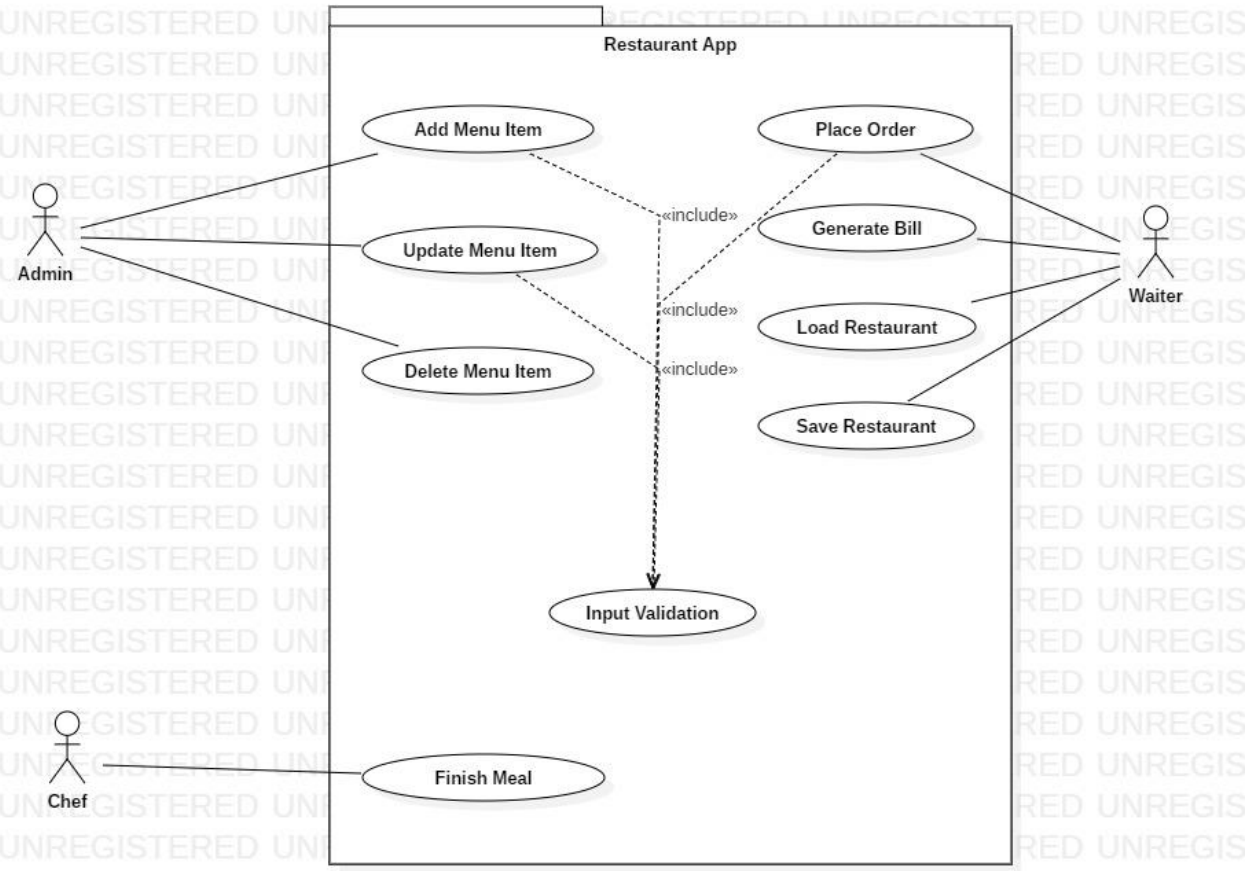
When running the application, all previous bills from previous uses are deleted. The user can either choose to load the former state of the restaurant through deserialization or provide different menu items/orders. However, changing the state of the restaurant requires the user to provide data to the application. The data needs to be validated before the application can continue. All the necessary conditions for valid data are presented below:

- The fields cannot be empty
- The price, grams and in-stock must be integers
- The price and stock of a product need to be positive

- The table number and ID need to be present in the restaurant

After providing all the necessary information, the waiter can proceed to placing orders. When placing an order, the waiter needs to select all the items from the menu and provide the application with a table number, which will be associated with the order. All the necessary conditions for a valid order are presented below:

- The fields cannot be empty
- The table number needs to be a positive integer



The main success scenario is presented below:

1) Administrator view:

Actors	The Administrator
Summary	Adds, removes and updates restaurant menu

Preconditions	The administrator provides correct input data
Main success scenario	1) The administrator provides all the needed input data 2) The application converts text into integers as needed 3) The application checks the data for correctness 4) All the provided products are inserted into the menu

2) Waiter view:

Actors	The Waiter
Summary	Place order/bill
Preconditions	The waiter provides correct input data
Main success scenario	1) The waiter provides all the needed input data 2) The application converts text into integers as needed 3) The application checks the data for correctness 4) All the provided products are associated an order and are stored in a specialized table on the GUI 5) Composite products are sent to the chef for cooking 6) The waiter generates a bill for the selected order

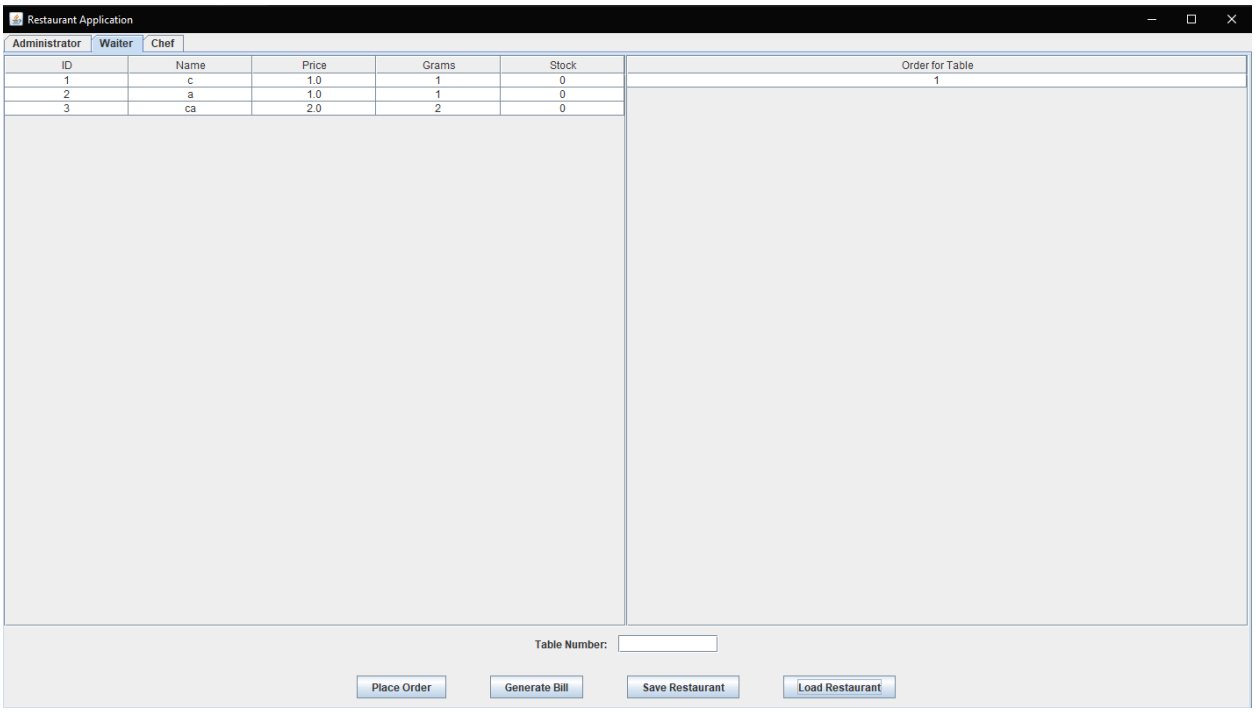
In case the input data in not correctly provided, an alternating sequence occurs:

Actors	The Admin/Waiter
Summary	Orders and menu managing
Preconditions	The actor provides incorrect input data
Alternating sequences	a. Non-integer input: 1) The actor provides all the needed input data 2) The application fails to convert text into integers. Return execution to step 1. b. Incorrect data (see above description): 1) The actor provides all the needed input data 2) The application converts text into integers if needed

	3) Input checking fails. Return execution to 1.

3. Design

From an interface design standpoint, the application combines 3 different panels (Administrator, Waiter and Chef) into one, and allows the user to choose at any time which panel he wants to interact with, by means of a tabbed pane.



The application is structured into 3 different layers. Each logical layer has its own package. The layers are: data access, presentation and business logic.

From a design standpoint, the application uses a layered architecture. The underlying logic of the application is completely isolated from the GUI itself. The data access layer provides helper classes for serialization, deserialization and writing data to a file. The business logic layer implements all the necessary logic for the good functioning of the application. Lastly, the presentation layer provides the Graphical User Interface. Utilizing this pattern helps maintaining

the application and allows for certain modules to be modified without affecting the other modules.

The application is started by running the main method, situated in the **Main** class. This class is straightforward, and its only purpose is to declare and initialize all the GUI components.

Other straightforward classes are found in the **Presentation** package. This package provides all the view related components, such as the administrator panel, waiter panel and chef panel. Each panel contains one or more JTables which allows for a neat display of the restaurant's menu/orders on the GUI. These components provide methods for resetting the view, adding action listeners, showing errors, etc.

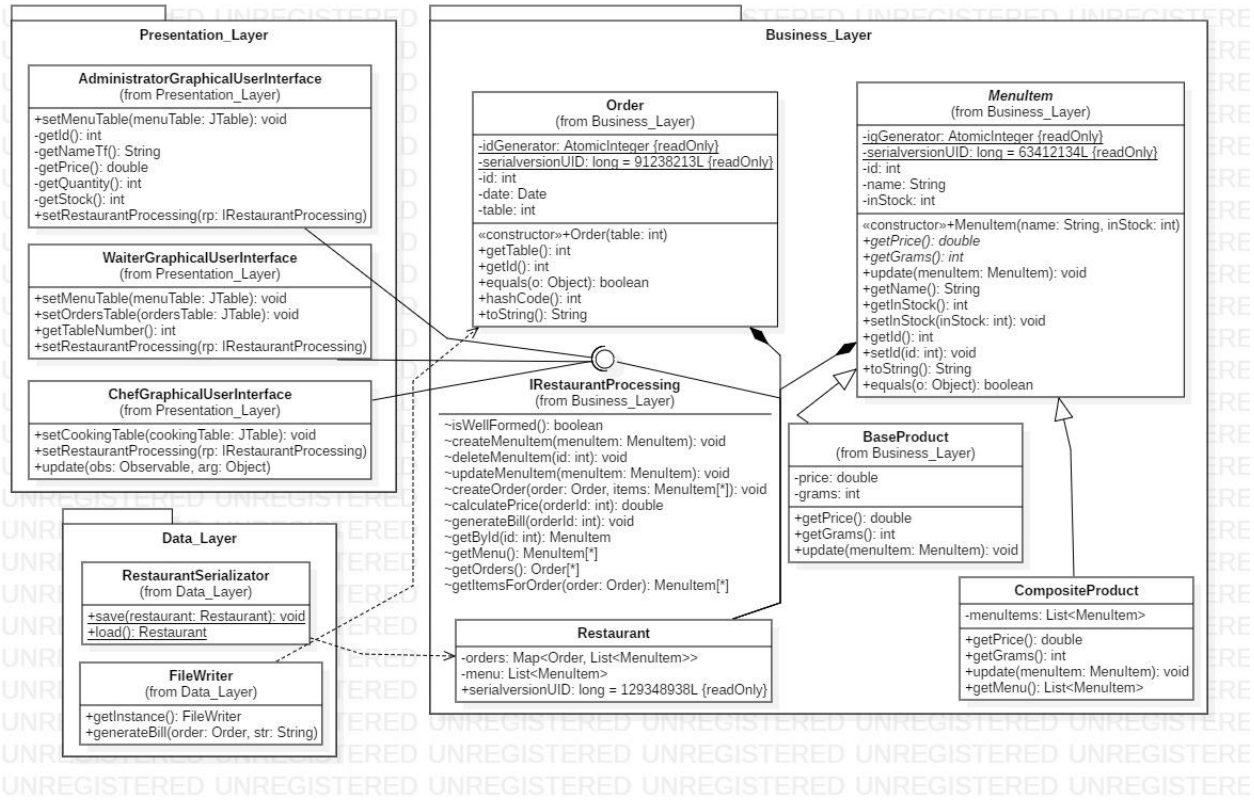
The link between the logic of the application and the user interface is realized by each panel. There are special inner classes for each button on the GUI:

- **Administrator Controller**: Provides the functionality for the 3 main operations performed on the restaurant's menu (insert, update, delete). Moreover, this controller is tasked with generating an error message in case a failure occurred in the application (bad input, etc.)
- **Waiter Controller**: Provides the functionality for placing new orders, generating bills for already existent orders and saving/loading the state of the restaurant.
- **Chef Controller**: Provides the functionality for finishing cooking products. The products are sent to the chef controller by means of the Observer pattern (sent by the waiter whenever a new order is placed).

The **Data Access** package contains the class that implements the serialization/deserialization process on restaurant instances and another class that implements the application's interaction with the file system.

The **Business Logic** package contains the entire logic of the application, including the class that defines the restaurant itself, the classes that make up the menu of the restaurant and the class that defines orders. The methods of the restaurant class are defined in an interface and then implemented by the **Restaurant** class. Also, the design of the restaurant class was done using the design by contract technique.

A detailed class diagram is presented below. Packages are also described in a pictorial manner.



4. Implementation

This section provides implementation details for some very important methods/classes.

The application is based on the serialization technique. Therefore, the **RestaurantSerializator** class is very important. This class provides two static methods for saving and loading a restaurant instance from/to a file named restaurant.ser. Interaction with the file system also needs to be provided. The class **FileWriter** provides a method for writing to a file. This class is implemented using the singleton design pattern, because each time a new instance of this class is created, some setup needs to be performed regarding the folder in which bills are to be saved.

Another very important aspect is the applications ability to change the state of the restaurant directly from a user interface context. The interface that defines all the operations

needed to be performed on the restaurant instance is **RestaurantProcessing**. Using polymorphism, restaurant instances can be passed to the controller classes which can then use the implemented operations to change the state of the restaurant instance directly from the GUI.

The **Business Logic** layer provides all the key classes. The restaurant has a list of **MenuItems**, which can be of two types: Composite and Base products. These use the composite design pattern. Moreover, the restaurant contains a **HashMap** that maps orders to a list of menu items. In this fashion, each order has associated a list of products that can be easily and efficiently retrieved in case a bill needs to be created or other operations need to be performed on the products.

Another important class is the class implementing the chef functionality and panel of the application. This class, named **ChefGUI** is responsible is implemented using the observer design pattern. Each time a new order is placed by the waiter, all the composite products associated to the order are sent to the chef class for processing. Here, the chef could do a thorough processing of the data, but instead a simple "finish order" button was implemented.

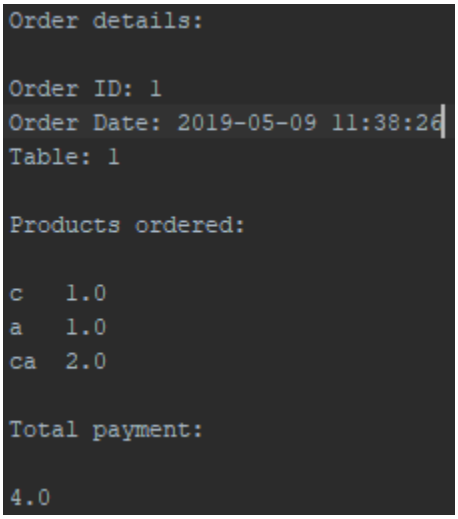
The design of the GUI was described in section 3. However, the **View** class defines two static methods which both, given a list of orders / menu items, returns a user-friendly representation of the given list as a JTable, ready to be displayed on the GUI. Moreover, this method provides some additional look and feel optimizations for the JTable, such as centered word positioning.

5. Results

This section shows a possible run of the application and its results. The application is very intuitive and easy to use. The only concern for the user is to correctly feed the input data to the application. What "correct" means was defined in section 2.

ID	Name	Price	Grams	Stock	Order for Table
1	c	1.0	1	0	1
2	a	1.0	1	0	
3	ca	2.0	2	0	

ID	Name	Price	Grams	Stock
3	ca	2.0	2	0



The first 2 pictures show the initial data that was provided by the user (3 menu items and 1 order). After pressing the "generate bills" button, the 3rd picture shows the format of the bill. It is composed of 3 main sections:

- The first part of the bill displays all order related information
- The second part of the bill displays each product that the customer ordered
- The last section displays the total amount that the customer must pay

6. Conclusions

Nowadays, the need for digitalization is increasing and each business tries to automate their work and make the work for its employees easier. This application provides an easy and comfortable way for multiple employees of a restaurant to efficiently communicate between each other, which in term contributes to the better functioning of the business overall. This application provides the building grounds for such a system and can be used as a baseline.

This assignment helped me improve my serialization and design pattern related knowledge.

The application can be further improved. For instance, the billing information can be displayed in a more user-friendly manner and the user interface can be improved. Moreover, the application could rely on a database instead of the serialization technique employed here.

7. Bibliography

- https://stackoverflow.com/