DOCUMENTATION

FOOD DELIVERY ORDER MANAGEMENT

4th ASSIGNMENT

STUDENT NAME: IAZ ANIA MEDEEA

GROUP:30424

Content

1.Theme objective

Main objective is to design and implement an application for managing the food orders for a catering company. Some sub-objectives are: analyzing the problem and identifying the requirements, designing the orders management application, implementing the orders management application and testing the orders management application.

The system should have three types of users that log in using a username and a password: administrator, regular employee, and client. The client can order products from the company's menu.

The administrator can:

- Import the initial set of products which will populate the menu from a .csv file.
- Manage the products from the menu: add/delete/modify products and create new products composed of several products from the menu
- Generate reports about the performed

The client can:

- Register and use the registered username and password to log in within the system.
- View the list of products from the menu.
- Search for products based on one or multiple criteria
- Create an order consisting of several products

The employee is notified each time a new order is performed by a client so that it can prepare the delivery of the ordered dishes.

2.Problem analysis, modeling, scenarios, use cases

2.1 Problem analysis

In this application the user can be client, administrator and employee each having their own operations to perform described briefly in the first chapter.

The client can search depending on the fields he or she has chosen the products that satisfy their needs. The products need to be added to the cart and when the client wants to place the order, he or she, presses the button named order and this will automatically be placed in list of orders. After that the employee is notified about the newly made order.

The admin has more operations to perform. First there is a load button that lets him or her load the table with products from a .csv file. He can add, modify and delete the base products or create a menu with more items and name it as he wishes. Besides these operations there is one more, generating the reports based on the fields that are not left empty.

All these information is placed in different serialized files in which I write at the end of the application and read from at the beginning of it.

Another thing to consider besides the operations of each user and how the information is stored is how the user will interact with the application. The user can introduce anything, but the application should stop him from introducing strings where integers are required, for example. That is why with the help of regex I have validations for most of the fields that need to be completed.

2.2 Modeling

Since the application is very complex I need to store the information somewhere and I do that with the help of serialization for three different files. One if for the client information, one for the products information and one for the orders where I store the user that placed it, the date and items bought. The information for the last component is stored back in a HashMap where the key is the order.

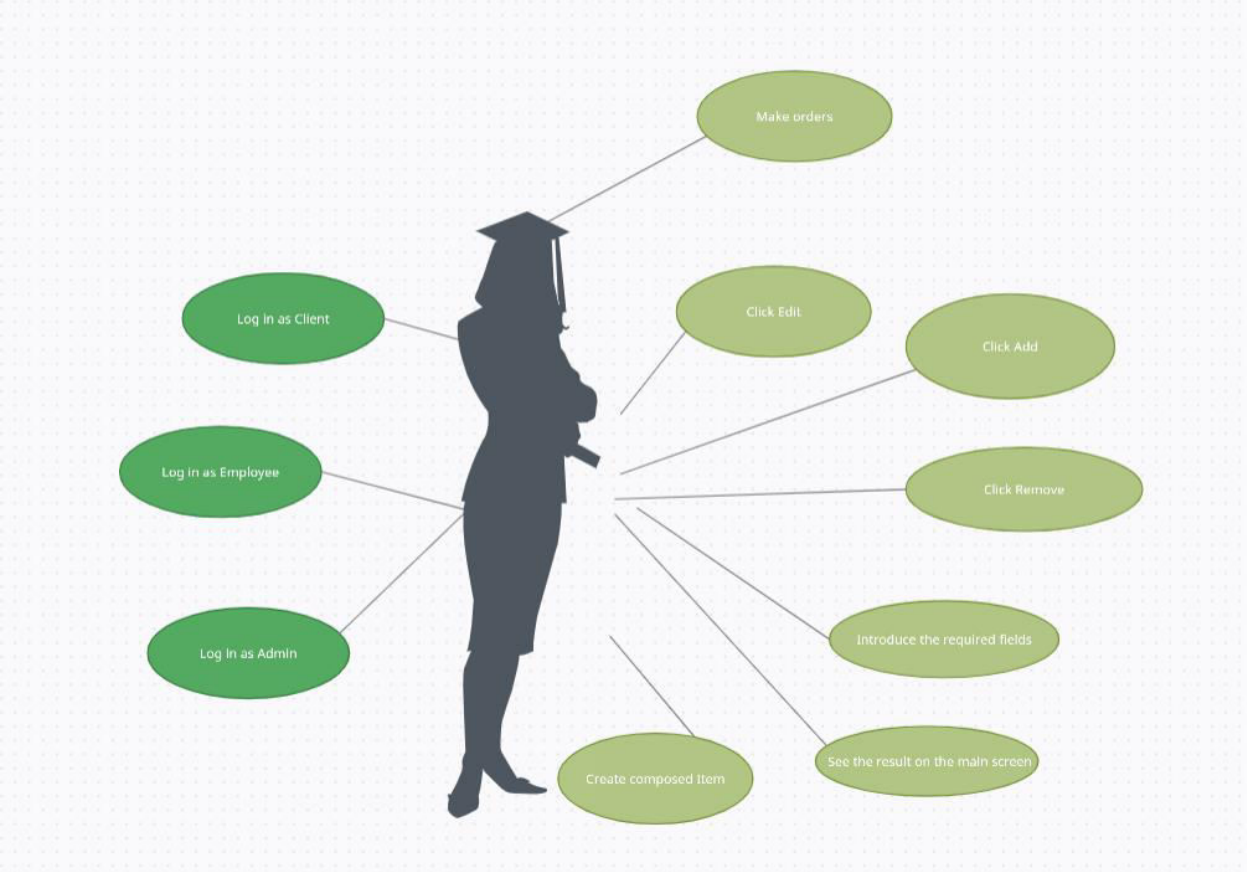
2.3 Scenarios

Because it is not sure how the user will interact with the graphical interface, I needed to make sure that he or she cannot introduce incorrect data or leave fields uncompleted. As long as the input it is not correct, the application will not do anything with the data, but some messages will be displayed. The message will indicate the error that was made so that the user can easily identify it. This is mostly for the clients, but I have validators for administrator as well. The most serious validations are made for the log in and register where the user must have an account and he or she must introduce the fields correctly.

After that the only thing that I need to do is to illustrate the result on the graphical user interface after every operation performed.

2.4 Use Cases

The use case diagram can be a pleasant way to visualize the interaction between the user and the system that he or she may use. It also shows what the user needs to do in order to get the right results, giving a general description of the system.



As it can be seen in the picture above the actions will take place in the following order:

- The user can choose based on log in account which role he wants to get
- After that we have three options, each with its operation:
  - For the Client: we can add products to the cart, search a product, make an order, load products
  - For the Administrator: we can add, edit or remove base products, create a composed product, generate reports about the orders, load base products
  - For the Employee: we can visualize the information about the orders
- Complete the required fields if necessary
- See the results of the operation performed after the data is introduced correctly

3. Design

In this application I have used many architectural patterns such as: Observer Design Pattern. One of the most important patterns is the Singleton one. This allows the application to have global access to the class DeliveryServiceProcessing and the information stored there in hashMap or ArrayList. This class will also be notified when to perform the opration desired.

The approach architectural pattern is Layered architecture that organizes the application on layers and it can be described as an architectural pattern composed of several separate horizontal layers that function together as a single unit of software. The layers are : presentation layer, business layer and data layer. This is available regarding my application. Presentation data contains the controllers and implicit the views, the business layer contains the operations performed and data layer contains the operation to store the information.

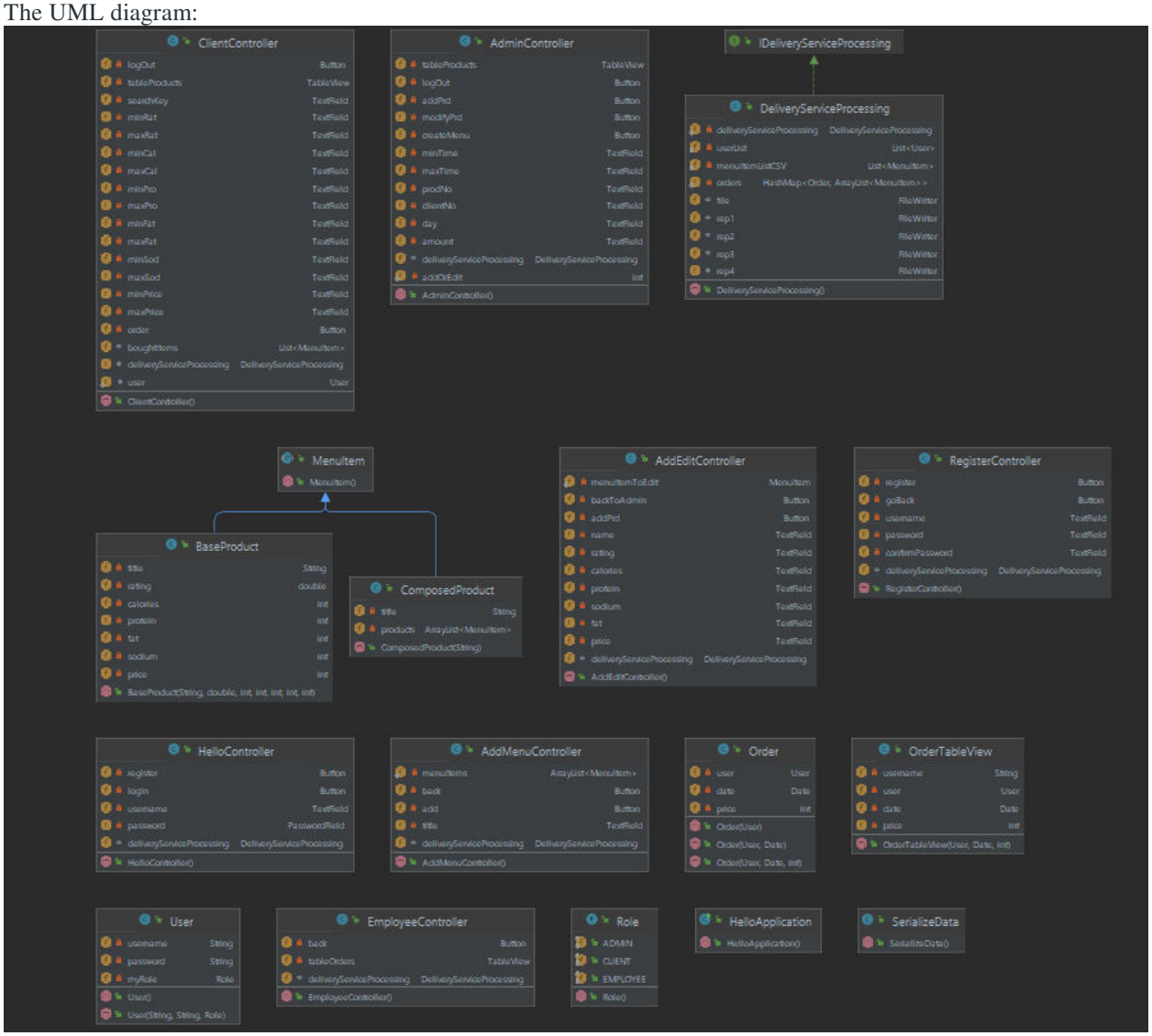
The classes MenuItems, BaseProduct and ComposedProduct use the Composite Design Pattern. Both BaseProduct and ComposedProduct extend the class MenuItem. Class ComposedProduct contains a title and an ArrayList of either BaseProduct or ComposedProduct .

Class EmployeeController implements the interface Observable in order to be notified each time an order is added. This pattern is called Observer Design Pattern.

The class DeliveryServiceProcessing uses hashMaps as one of the data structures. The key is based on the date of the order that is unique since it takes into consideration even the second when the ordered is performed and only one user can use it at a time. This class implements the Observable Design Pattern and every time an order is performed it notifies the employee. It also implements the Observable Design Pattern that consists of pre and post conditions for the operations performed in order to check if the input is correct and that the operation was performed error-free.

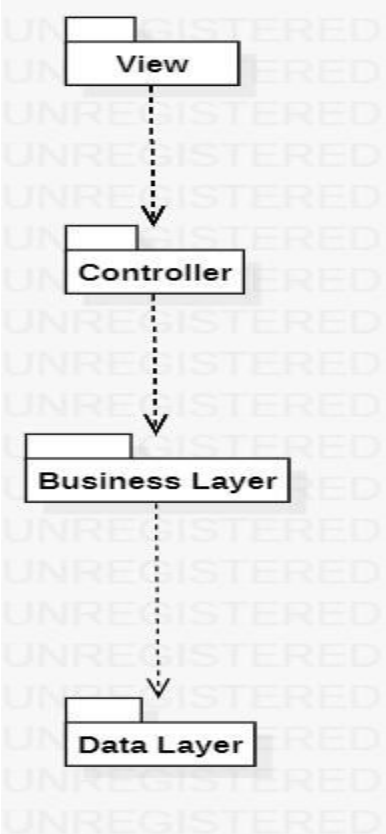
In the package PresentaionLayer each view is assigned a controller which will manipulate the view and how the user can see if the data introduced is incorrect. Some controller, like the EmployeeController communicates with the DeliveryServiceProcessing class in order to display the orders in the table.

The UML diagram:



It cand be easily seen that the view is missing due tot the fact that it is a .fxml file and not a java class. Also the class that contains the main is missing, its only purpose being to launch the application and set the title, width and length of the screen that will be displayed and others.

The package diagram will be presented below:



- **Data Structures**

For this application I used as data structure Lists, more specific ArrayLists. Some advantages of this data structure are: they are resizable, elements can be inserted and deleted at or from a specified indexes, they have many methods to manipulate the objects stored and they can store any type of object that we define. Another reason for which I have chosen them is that they are very compatible with the database and tableView from the view that helps to display every product, client or order that is present in the tables.

Besides ArrayList I have used HashMap formed of Order and an ArrayList of MenuItems to store information about the order a client placed.

4. Implementation

- **Model Classes**

In the Model package I have seven classes: BaseProduct, ComposedProducte, MenuItem, Order, User, OrderTableView and Role. I use the role class for the user to know if he or she is a client, admin or employee. The class OrderTableView is very similar with the Order class but I use it to display information about the orders in the table view from one of my controllers. The most interesting class is the MenuItem one which is an abstract class that only has abstract methods. Classes BaseProduct and ComposedProduct extend this class. I do this in order to use both BaseProduct and ComposedProduct in the same list without having to differentiate the two. For each class I have getters and setters plus one or two constructors.

For class BaseProduct I have declared as attributes the following ones: title, rating, calories, protein, fat, sodium and price. All of them are private.

For class ComposedProduct I only have a title and a list of MenuItem, both of them being private.

In class Order I have three attributes: user, price and date. For date I use java.util.date that takes the actual date and time when I instantiate it.

In class User I declared also three attributes: username, password and role. The password is not stored as a simple String, I have a method that converts it into something else with the help of SHA-256.

```
public Boolean checkPassword(String passwordToCheck){
    try {
        MessageDigest messageDigest =  MessageDigest.getInstance("SHA-256");
        passwordToCheck = Arrays.toString(messageDigest.digest(passwordToCheck.getBytes(StandardCharsets.U
        if(passwordToCheck.equals(password))
            return true;
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    return false;
}
```

- ## PresentationLayer Classes

In this package I have in total seven classes, each corresponding to an .fxml file where the view for eachscreen is stored. Each button from every view has associated a method that either performs an operation, either loads a new screen. In this class I have introduced some validators and pop-up messages to inform the user where he or she might be wrong when completing the data.

For every view that contains a tableView I have a method called initialize that sets the field of the table and displays the content of a static list.

These seven classes are called: ClientController, AdminController, EmployeeController, HelloController (main controller), RegisterController, AddMenuController and AddOrEditController.

- ## DataLayer Classes

In this package I have only one class called SerializeData that saves information at the end and the beginning of the application. This class performs two operations called serialization (write the information) and deserialization (read the information). These two operations are implemented for three types of data: ArrayList of User, ArrayList of MenuItem and HashMap of Order and ArrayList of MenuItem.

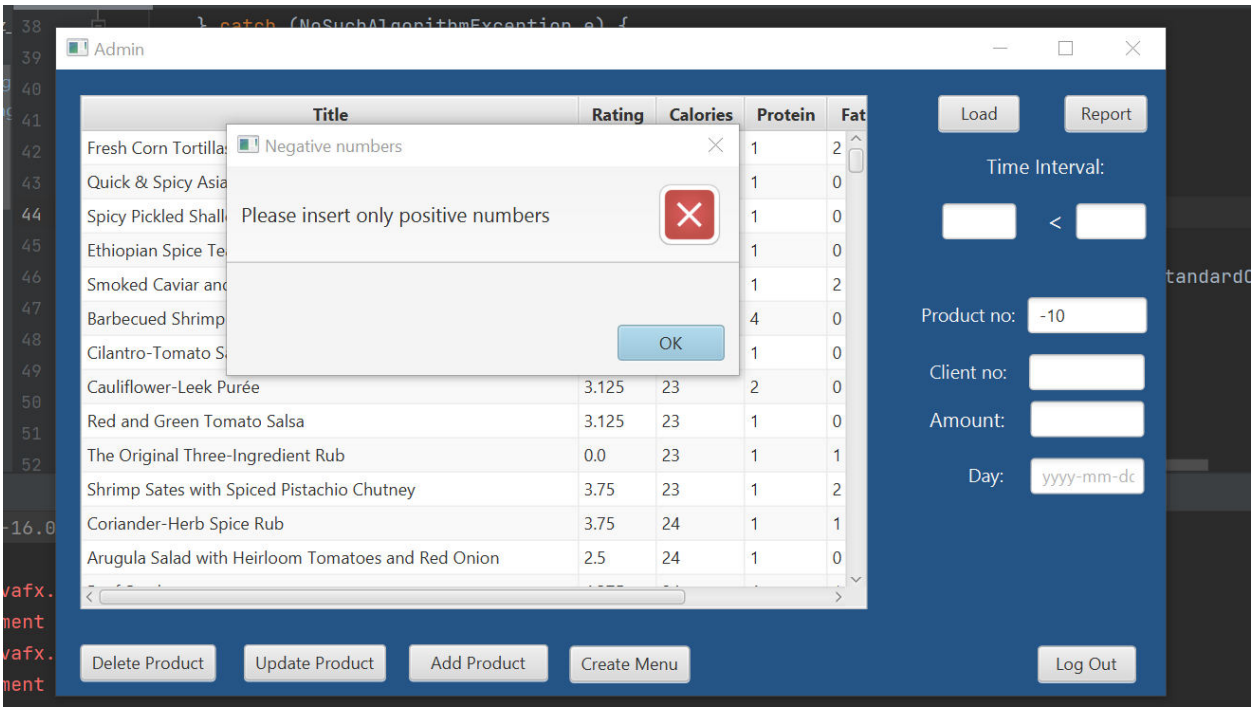
```
public static void writeUser(List<User> writeFile){
    try {
        FileOutputStream fileOut =
                new FileOutputStream( name: "C:\\Users\\Medeea\\Desktop\\ANUL2(SEM2)\\Projects PT\\PT2022_
        ObjectOutputStream out = new ObjectOutputStream(fileOut);
        out.writeObject(writeFile);
        out.close();
        fileOut.close();
    } catch (IOException i) {
        i.printStackTrace();
    }

}
```

- ## BusinessLayer Classes

These package, the most important one, implements all the operations a user, does not matter if he is a client, employee or admin. I have one interface that only indicated the methods and a class called DeliveryServiceProcessing that implements all the methods declared in the interface. This class is also considered a singleton since I only work with this in order to store the information. All the fields that use serialization and deserialization are static fields since I want every user to be aware of the changes in one field.

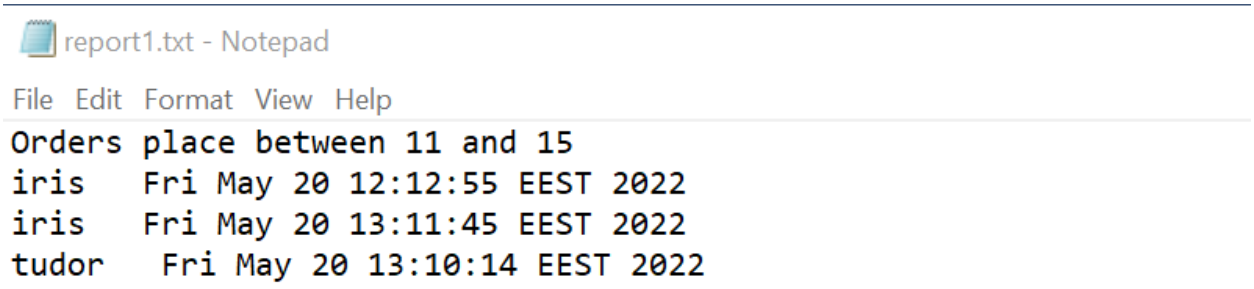
- ## HelloApplication Class

This class only takes care of launching the application, loading the fxml screen and storing and reading the information, but since is the first time that something very important happens in this class it needed to be mentioned in the documentation.



5. Results

The results of the application can be seen, besides the graphical interface, in the .txt files that contain the bill and four different reports. First report contains the orders performed between a start hour and end hour. The second report contains the products ordered more than a specified number of times. The third contains the clients that have ordered more than a specified number of times so far and the value of the order was higher than a specified amount. The last one contains the products ordered within a specified day with the number of times they have been ordered.



6. Conclusions

This application was very complex and it helped me a lot to get familiar with several different patterns used in real life applications. I learnt how to read a .csv file and how to use serialization and deserialization in order to store information. Another good thing learnt from this project is how to work with streams, filters and lambda expressions.

One further development could be display in the employee table view the list of items bought by the user. This is a little bit more difficult since javafx work with specific getters and setters.

7. Bibliography

- https://regexr.com/
- https://stackoverflow.com/

- https://app.creately.com/d/pfulq18TWJw/edit
- https://online.visual-paradigm.com/diagrams/features/package-diagram-software/
- https://gitlab.com/utcn_dsrl/pt-reflection-example