**ASSIGNMENT 4 DOCUMENTATION**

**FOOD DELIVERY MANAGEMENT SYSTEM**

**Name: Zarnescu Radu**

**Group: 30423**

**Laboratory Assistant: Antal Marcel**

Contents

[1. Assignment Objective 3](#_Toc476131445)

[2. Problem Analysis 3](#_Toc476131446)

[3. Design 7](#_Toc476131449)

[3.1. Data Structures 7](#_Toc476131451)

[3.2. UML Diagram 7](#_Toc476131452)

[4. Implementation 8](#_Toc476131454)

[4.1. Class Description 8](#_Toc476131453)

[4.2. Algorithms 11](#_Toc476131453)

[5. Testing 11](#_Toc476131454)

[6. Conclusions and Further Improvements 12](#_Toc476131455)

[7. Bibliography 13](#_Toc476131456)

1. **Assignment Objective**

Design and implement a food delivery management system for a catering company. The client can order products from the company’s menu. The system should have three types of users that log in using a username and a password: administrator, regular employee, and client.

1. **Problem Analysis**

**Serialization in Java** is a mechanism of writing the state of an object into a byte-stream. It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies. The reverse operation of serialization is called deserialization where byte-stream is converted into an object. The serialization and deserialization process are platform-independent, it means you can serialize an object in a platform and deserialize in different platform.

One issue with anonymous classes is that if the implementation of your anonymous class is very simple, such as an interface that contains only one method, then the syntax of anonymous classes may seem unwieldy and unclear. In these cases, you're usually trying to pass functionality as an argument to another method, such as what action should be taken when someone clicks a button. Lambda expressions enable you to do this, to treat functionality as method argument, or code as data.

A use case is a written description of how users will perform tasks on your website.  It outlines, from a user’s point of view, a system’s behavior as it responds to a request. Each use case is represented as a sequence of simple steps, beginning with a user's goal and ending when that goal is fulfilled.

Basic Flow: Placing an order as a client:

* Actor: User
* Description: The main success scenario is the situation where the client selects the desired products and orders them successfully.

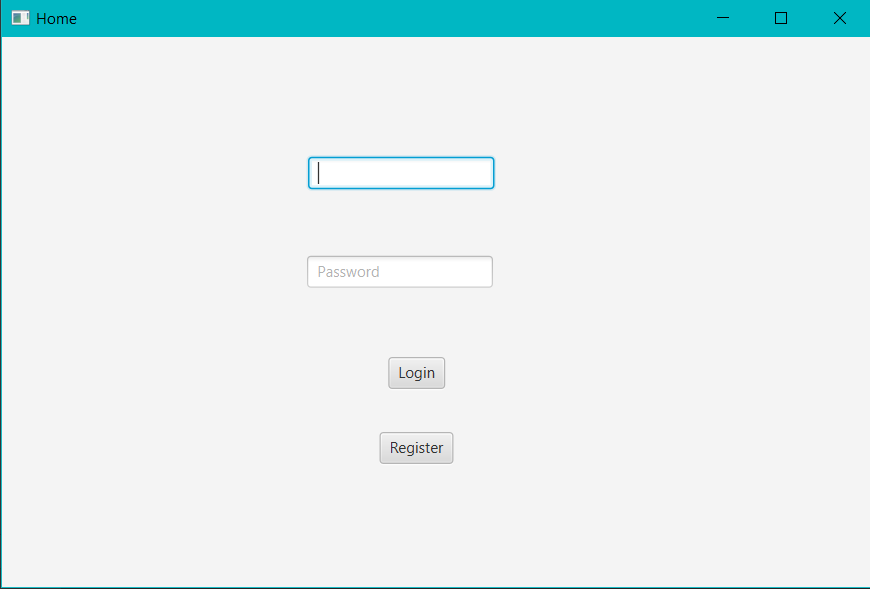
1. Client logs in using their username and password
2. Client selects each product and then presses the “add to order” button (client can use filters to search for specific products)
3. Client sends the order

* Termination outcome: The employee user interface displays a message to confirm an order has been made.

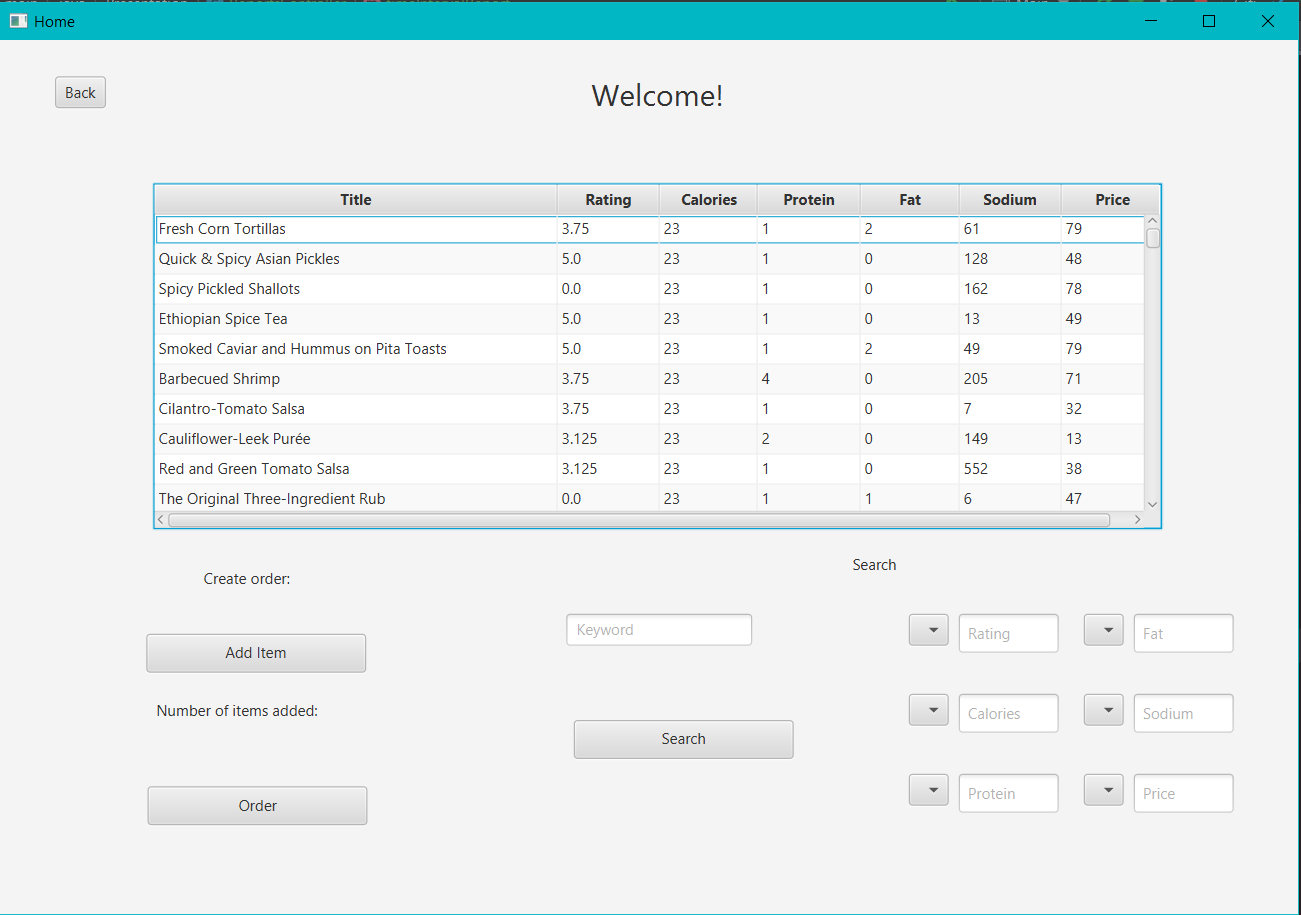
Illustration of the steps a user has to make in order to correctly use the application for the use case presented above:

The user interacts with the program through a dedicated UI found in the screenshot below. It is very simple to use and it indicates the user what every field represents.

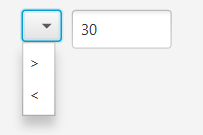
The first step is to log in using a valid account. If the user doesn’t have one, they can register.



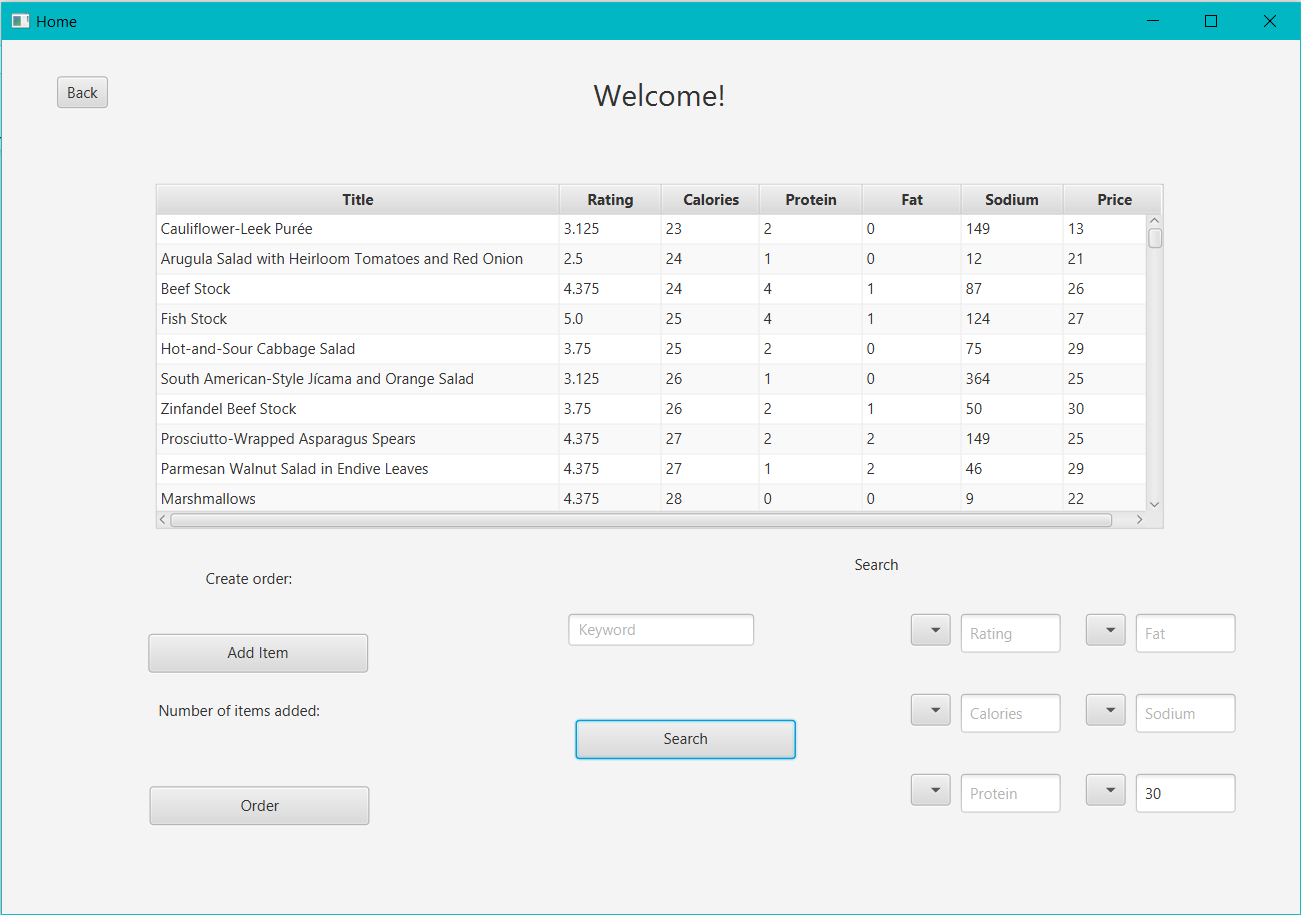
After the user logs in, the following window will pop up:



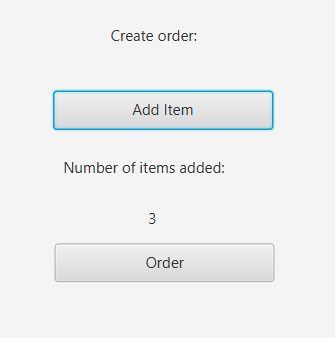
Now, the client can search for what products he/she wants. For example, let’s say the client wants to eat something with beef but does not want to spend a lot of money. He/She types the word “beef” in the text field marked with “Keyword”. Then, they can search for a price lower than 30 using the menu and text field labeled with “Price”.



Now, they have to selected greater or smaller than the inserted value. Let’s selected smaller than 30 and press the “Search” button.



Now, if the client wants to add a product to the order, they need to click the desired product and then click on the button labeled with “Add Item”. The user can browse through all the menu or search for specific products and add how many they want to the order the same way as before.



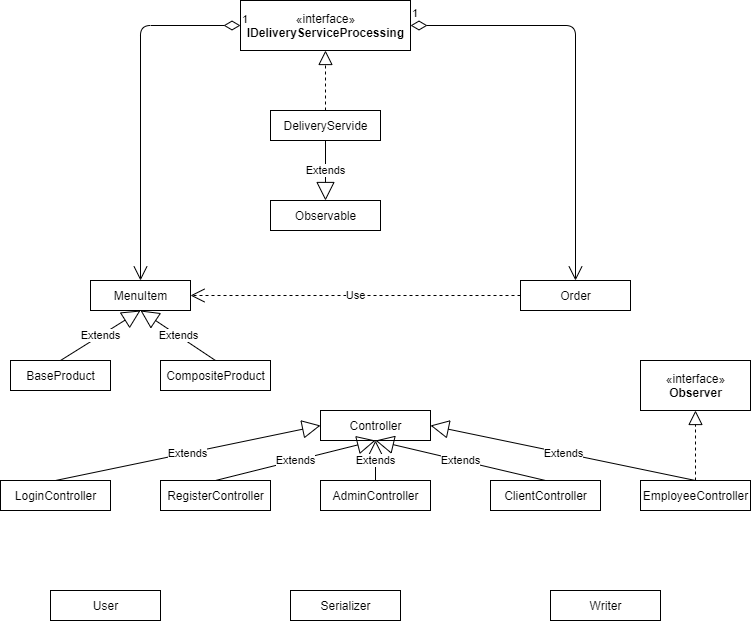
The number of added items will be displayed like shown above. When the user added all the products they want all they need to do is to press “Order” and wait for their food to arrive.

1. **Design**
   1. **Data Structures**

The main data structures I used were ArrayList and HashMap to implement the back end part of the application. I also used ObservableList to display the products on the user interface.

I have chosen to use ArrayList instead of a simple Array because it is more efficient for memory management and also, it does not have a fixed number of elements. An alternative data structure would be HashMap, but I found it easier and more convenient to implement the operations using ArrayLists.

* 1. **UML Diagram (Class Diagram)**

****

1. **Implementation**
   1. **Class Description**

I based my class structure on 3 layers: Business, Data and Presentation.

1. Business includes the classes used to implement the algorithms and the most important operations made by the app.
2. Presentation represents the classes that form the user interface, the part with which the user interacts.
3. Finally, data is the collection of classes which deal with reading and writing data from/to files.

* MenuItem Class

The MenuItem class represents all the products in the menu, including base products, as well as composite products. It contains 7 fields: title, rating, calories, protein, fat, sodium and price.

* BaseProduct Class

BaseProduct is a subclass of MenuItem. It represents the products imported from the csv file.

* CompositeProduct Class

This class is a derived class of MenuItem. It represents a new product formed from other instances of MenuItem, be it base or composite products.

* User Class

A class used to store the users. It contains a username and a password with which each user logs in and an ID used to place orders. It also has a type field, used to differentiate the types of users (administrator, client and employee) and an array list of orders that client has made.

* Order Class

Order is a class used to create new product orders by the clients. It is represented by an ID (generated automatically), a client ID, the date is has been made and a total price. It is used together with an array list of menu items to form a hash map. The order is the key while the array list is the value.

* DeliveryService Class

This class is basically the brain of the application. It contains all the information about the application (all the users, all the products, all the orders). DeliveryService extends Observable and implements IDeliveryServiceProcessing. This class is kept as the same instance throughout the running cycle of the application and at the end is serialized in a file. At each start of the app it is deserialized.

Methods:

* importProducts(): this method is used to import all the base products from the csv file. It is only performed once. Afterwards all the data is being serialized so there is no need to perform it anymore.
* createCompositeProduct(ArrayList<MenuItem>composition, String name): method used to create a new composite product, from the composition list with the given name.
* createOrder(ArrayList<MenuItem>menuItems): this method creates a new order from the given list of items and stores it in the hash map.
* FileReader Class

Class used to read the data from the given csv file.

* Writer Class

This class is used to write information to .txt files. It generates bills for each order and also generates the required reports.

* Serializer Class

The name describes what this class represents. It is used to serialize any object and to deserialize the DeliveryService class and an array list of users.

* Controller Class

This class is used to ensure all the controllers use the same instance of DeliveryService. All the other controllers extend this class.

* LoginController

Subclass of Controller; it handles the operations from the log in window.

* RegisterController

Subclass of Controller; it handles the operations from the register window.

* AdminController

Subclass of Controller; it handles all the operations an administrator performs using the dedicated admin user interface.

* ClientController

Subclass of Controller; it handles all the operations a client performs using the dedicated client user interface.

* ReportsController

This controller is used to generate the requested reports by the administrator.

* EmployeeController

This is the class used to implement the observer. All it does is to print a message when a new order is made.

* Main Class

The main class extend the Application class and is used to start and shape the User Interface.

* 1. **Algorithms**

In this section I will briefly describe how the app works.

When the application starts, the first operation performed is the deserialization of the delivery service. This will recover all the user and products stored previously. Then, after the user inserts their username and password the list of users is checked to see if the inserted combination exists. If it does not, a new client can be registered using the “Register” button. If the user exists, a new window will appear, based on the type of the current user. Each type of user (administrator, client, employee) has a dedicated user interface with corresponding operations. If the current user is an administrator they can perform the following operations: create composite product, add a new base product, delete a product, modify a product and generate reports. In the case the user is a client, they can search for products using a keyword or filters and then place a new order. A bill will be generated. The employee can only see if a new order has been placed.

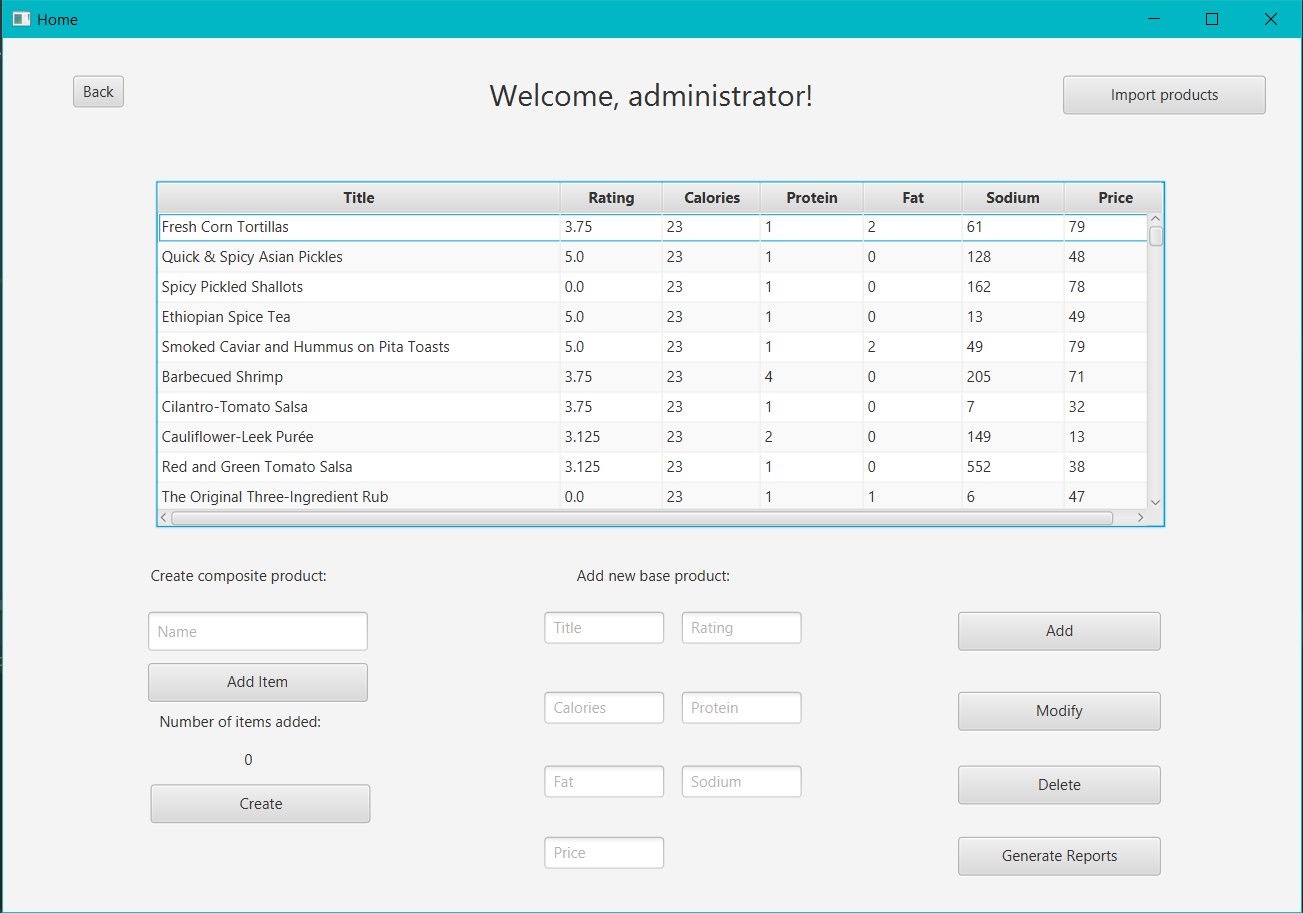
In the Serializator class I implemented a single serialization method, which works for any type of object, and a deserialize method for each type of object I wanted to serialize.

The reports were one of the hardest parts of the project for me. For each one I made a list of map sets in order to filter them with stream processing and lambda expressions. At this point, to make my job easier, I added one more variable to the MenuItem class, which increments every time that product is ordered and another one used to count how many occurrences a menu item has in a list as mentioned above.

To implement filters for the clients when searching for a product, I wanted the clients to be able to filter with multiple options, so any number of them will work. They are implemented using lambda expressions and stream processing. What a client has to do is insert a value in each deserved field and select from the 2 options in the button menu.

1. **Testing**

The most important part of the project is the testing. If the results are not correct, then the program is not doing it’s job properly. In order to test whether the app is working well, we can just run the program, then follow the steps presented in section 2. However, there are more types of use cases which all work properly. I am not going to visualize them all, but here is a screenshot of the admin interface.



If the administrator wants to add a new composite product, they should click on every item they want to add and immediately click on add item. After all the desired products are added, the admin must insert a name for the product and all that’s left is to click “Create”, and the new product will be displayed as the last element in the table.

To add a new base product the user has to complete each field in the middle of the page and then click Add. The new product will be displayed at the bottom of the table. To delete a product, all that needs to be done is to click the product desired to delete and click the delete button. Lastly, to modify a required product, it needs to be clicked and the desired modifications should be written in the fields in the middle and then click modify.

1. **Conclusions and Further Improvements**

To conclude, this assignment was a very good exercise in writing java and object oriented programming code. Now, I would consider myself more experienced in the techniques of programming applications.

The most important concepts I learned were serialization, lambda expressions and stream processing and how to use and implement them.

Overall, I would say that this project was the most difficult one and it was pretty challenging for me, especially in working with serialization and generating the reports.

As for further improvements, there is still plenty to do. One of the main improvements is to make the Observer pattern work properly, as I didn’t manage to get it working at the moment. Also, some alerts and errors in the GUI would do good for cases when the user makes a mistake with the input.

1. **Bibliography**

* wikipedia.org
* [www.stackoverflow.com](http://www.stackoverflow.com)
* geeksforgeeks.org
* usability.gov
* <https://mkyong.com/java/how-to-read-and-parse-csv-file-in-java/>
* <https://www.baeldung.com/java-streams-distinct-by>
* <https://www.tutorialspoint.com/java/java_serialization.html>
* <https://stackoverflow.com/questions/30191264/javafx-tableview-how-to-get-the-row-i-clicked>