+

Food Delivery Management System

Assignment 4

- Documentation -

Student: Alexandra Ilovan

Group: 30422Laboratory Professor: Andreea-Valeria Vesa

TABLE OF CONTENTS

1. Assignment Objectives

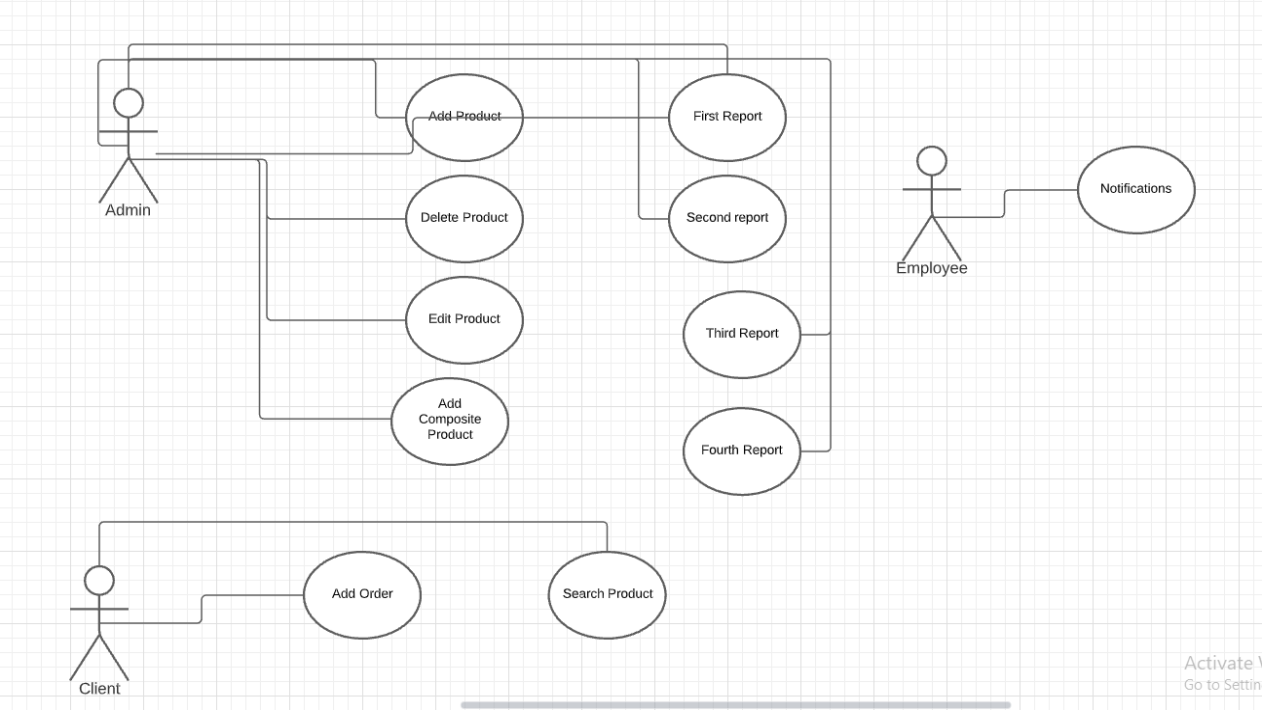
The goal of this assignment, called "Food Delivery Management System" is to create and implement an application with an intuitive graphical interface that mimics the management met among the classic food delivery applications. Such an application contains a login window, in order to establish the type of user who is currently using the application, and depending on the information extracted at login, other windows will open corresponding to each user role. The actions a window offers differ depending on the logged in user type. There are three types of users: Administrator, Employee and Client. The administrator can import the available menu items, add, edit, or remove a menu item (or create an entirely new menu item, composed of the existing products), as well as generate several reports that have to do with the restaurant management. A client can view the entire menu or search for specific items depending on one or more criteria, and then choose the menu items he/ she desires. Once all the wanted products are added to the cart, the client can place an order. The employee receives notifications each time an order is send in by a client.

The secondary objectives of this theme are represented by the correct and efficient implementation of all classes that make possible the actions mentioned above. More exactly:

- Adding a product
- Deleting a product
- Adding a composite product
- Generating specific reports
- Creating an order
- Searching for a product
- Modifying a product
- Notifying the employee
- Viewing products

2. Problem analysis, modeling, scenarios, use cases

Use case diagram:



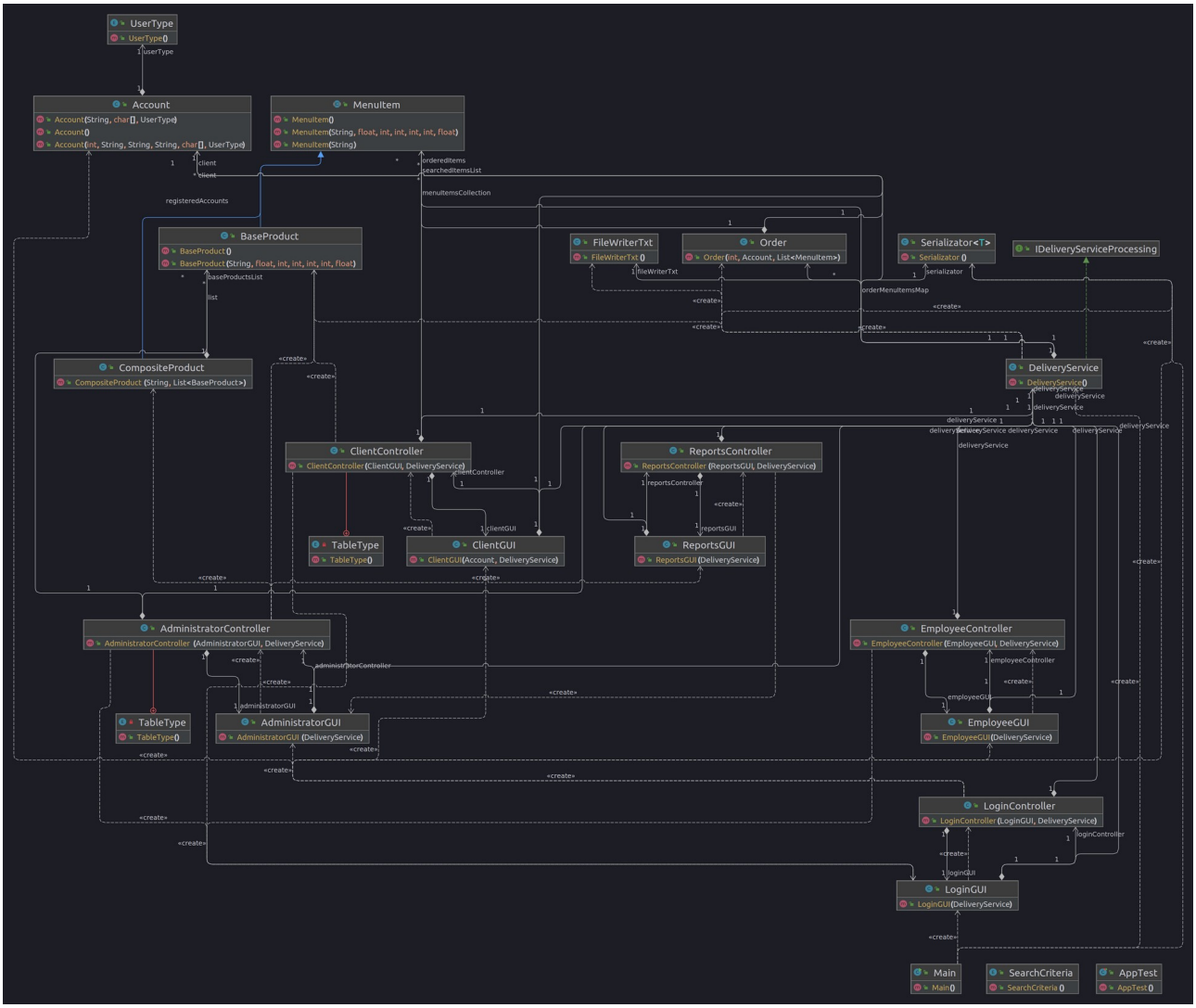
In day to day life we have a multitude of examples of such food delivery applications, some of the more popular ones being Glovo, Food Panda, Bolt Food, as well as newer ones that are starting to appear more and more often, such as Tazz by Emag. Thus, the implementation of a system for organizing such an application is a very good practice for our future. After running the application, the user will be greeted by a Log In/ Sign Up window. If the user does not already own an account, he or she will be encouraged to create a new one. Otherwise, the user will be free to log in directly into the application. Depending on the type of account selected upon the account creation (admin, employee or client), three new interfaces will appear.

The admininistrator operations window will display all the products available in the menu, it will provide the options to modify a product, add a new one, delete a product or even add a composite product consisting of several base products, such as a "daily menu" for example. At the same time, the admin can visualize report for different criteria.

The customer will also be able to see all the products, but will have a more limited number of functionalities. He will only be able to search for a desired item based on certain criteria and place an order.

The employee will be greeted by a graphical interface that will contain notifications regarding the orders placed by customers. When a customer places an order, a new invoice presenting the ordered products, their price and the date of the order placement will be created instantly.

3. Design (design decisions, UML diagrams, data structures, class design, interfaces, relationships, packages, algorithms, user interface)



In this Food Delivery Management System application several packages and design patterns were combined, but at the very root, the idea was again to use a Layered Architecture, in which each different package represents a layer that has a certain purpose, each having an individual purpose.

The "bll" package contains the classes that encapsulate the application logic for its correct and efficient operation. In addition to the implementation of the main classes dealing with the real-world transposition (Account, BaseProduct, CompositeProduct, MenuItem, Order), there are also two enumerations, one used to determine the type of user that accesses the application (UserType), and the other one (SearchCriteria) is used to differentiate between the types of criteria that a user can use to filter the product search operation. The “brain” of the applications, stands within the DeliveryService class, which implements the IdeliveryServiceProcessing interface, and implements the entire functionality of the application, managing all the operations performed by the admin, employee or client.

The "dao" package contains two classes: FileWriterTxt, which is used to write and create the reports that the admin can generate, as well as the individual bills (generated in .txt format) for the orders placed by the customers, and the Serializator class, which implements the operations of serialization and deserialization, respectively. By serializing the login, menu items and order data, the application can be used with more ease, as there will be no need for doing things multiple times.

The third package is the "presentation" package, which contains the GUI and controller classes for the Login/ Signup Window, Employee Window, Admin Window, Client Window and Reports Window.

We also have the Main class implemented in the default package which is used for starting the application.

This project has as role the implementation of a management application for food delivery, which extracts its menu items from a products.csv file. This is done by using OOP paradigms, lambda expressions and stream processing to make our work easier and to get familiarized with current technology.

4.Implementation

For the implementation of this project, a total of 22 classes was used, among which there was also an interface and two enumerations, which will be discussed more in depth below. The “bll” package contains all the default attributes and methods, as well as logicistics we might need for the products in this application, the BaseProduct class represents the most simple product class in the application, and the CompositeProduct class represents a list of base products, both of them extending the MenuItem super class. The IdeliveryServiceProcessing interface is used to add signatures of several methods such as addProduct, editProduct, deleteProduct and many more, more precisely the methods that will be used by the DeliveryService class within the same package, the default implementation of these methods taking place in this class. The Account class represents the class used to create a user, having a UserType attribute, this being an enumeration that contains the 3 instance a user can take, more precisely admin, client or employee. The Order class is the class that stores in memory, how an order should look. Within the given package, we have the FileWriterTxt class in which 5 different methods are implemented, but all these methods have a similar role because it represents the creation of the 4 reports seen by the admin itself, and also the automatically generated bill when a client place an order. As mentioned above, the classes in the Presentation package are the ones used to implement a graphical interface at a fairly simple level, so that the user is not confused at the time of use and is as user-friendly as possible because we know that that is a characteristic that is currently being sought after,

The Account class from the "bll" package is the class that describes and stores an account in memory. This class has attributes such as String username, used to store the username, String password used to store a user's password, a UserType attribute representing the role of the user, an order counter, and additional information about the user (his first and last names). The methods defined in this class mostly consist of getters and setters.

The UserType enumeration in the "bll" package is an enumeration that contains the three types that the user can identify as, these three types being Administrator, Employee and Client. This listing is very useful when creating an account.

The Order class within the same package mentioned above is the class that describes and creates in memory an order that a customer will place. This class has attributes such as Integer orderID that represent the ID of an order, Integer ClientID, a list of ordered items of type MenuItem, be they composite or base products, depending on what the customer wants, an orderDate of type LocalDateTime which represents the date on which the order was placed and an object of type Account, so that the customer-order connection is established. The methods in this class are represented by getters and setters, and by overriding the equals() and hashcode() methods.

The MenuItem class is the abstract class that will contain all the attributes that the products represent from the received file for extracting them. Thus we have a String title, a float-type rating, and the number of calories, protein, fat, sodium and price, all represented by Integers except for the price, which will be a floating point number. There are getters and setters provided for each, along with the abstract method computePrice() which will be implemented inside the two classes that extend this abstract class, BaseProduct and CompositeProduct.

The BaseProduct class in the "bll" package extends the MenuItem class and presents the storage of a simple product, extracted from the list of products just mentioned. It presents only two constructors and a computePrice() method, method that implicitly returns the price that one of the products may present at that time.

The CompositeProduct class in the "bll" package extends the MenuItem class and presents the memory retention of a product consisting of several other base products – it can basically be seen as a Menu in itself. It contains an list of BaseProducts and within the constructor it receives a title and a list of the corresponding base products. Thus, its attributes are calculated in a forum, depending on the products added. The abstract method computePrice() is implemented in this class by initializing the price with 0, after which we go through each product in the list of base

products made of this composite product, and add the price of each item in the list to the total price of the composite item.

The IdeliveryServiceProcessing interface is an interface that the class DeliveryService will implement, both classes being part of the "bll" package. Here the methods that will be implemented in DeliveryService are presented: addProduct, deleteProduct, editProduct, a method for generating the first report, second report, third report, fourth report, createOrder, searchItem, getMenu, importItemsFromCSV.

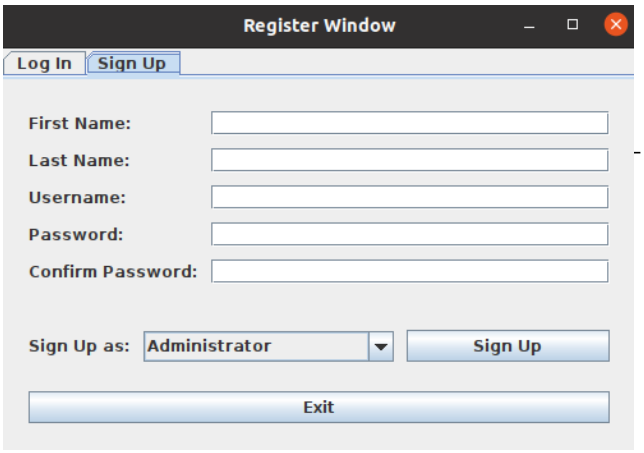
The DeliveryService class presents a list of MenuItems, a map which has Orders as keys, and their corresponding lists of MenuItems as values, a list of registered accounts, a serializator and a filewriter attribute for reporting and billing. The report generating methods use stream processing and lambda expressions. The createOrder() method receives as parameters an account which represents the account representing the client and a list of menu items, representing the items that were ordered by that client. A new order will be placed with the information sent, the number representing how many times a product has been ordered will be increased and an employee will be notified by the fact that a new order has been created through the use of the Observer Design Pattern, and a bill for the newly-created order will be saved. The FileWriterTxt class in this package contains the necessary methods for creating files for the necessary reports and bills.

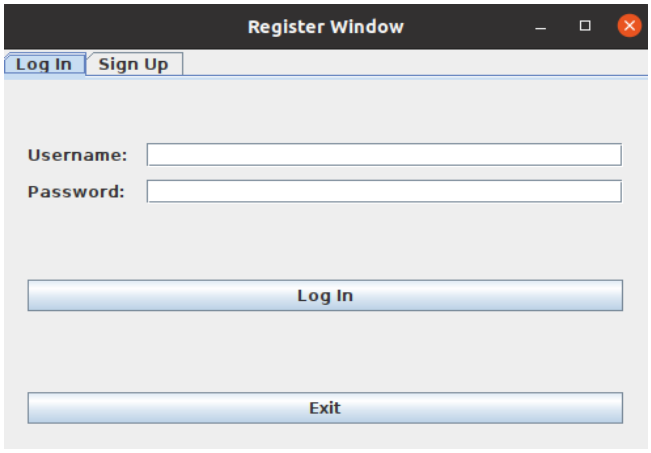
5. Results

The results of this application take the form of the graphical interface, the notifications received by the employees, or of the generated reports and order bills.

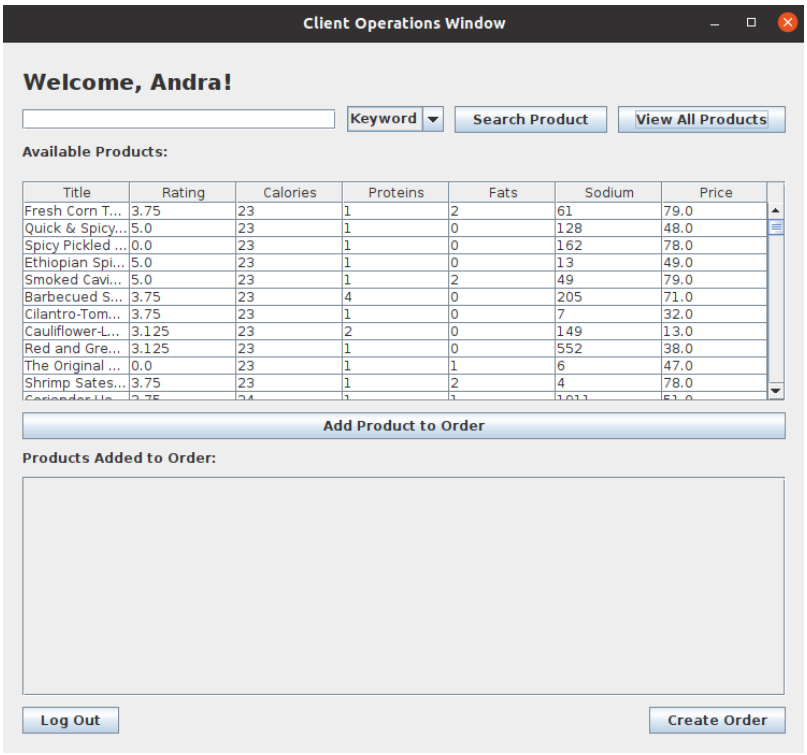
Following are attached a few pictures displaying the functionality of each window of the application:

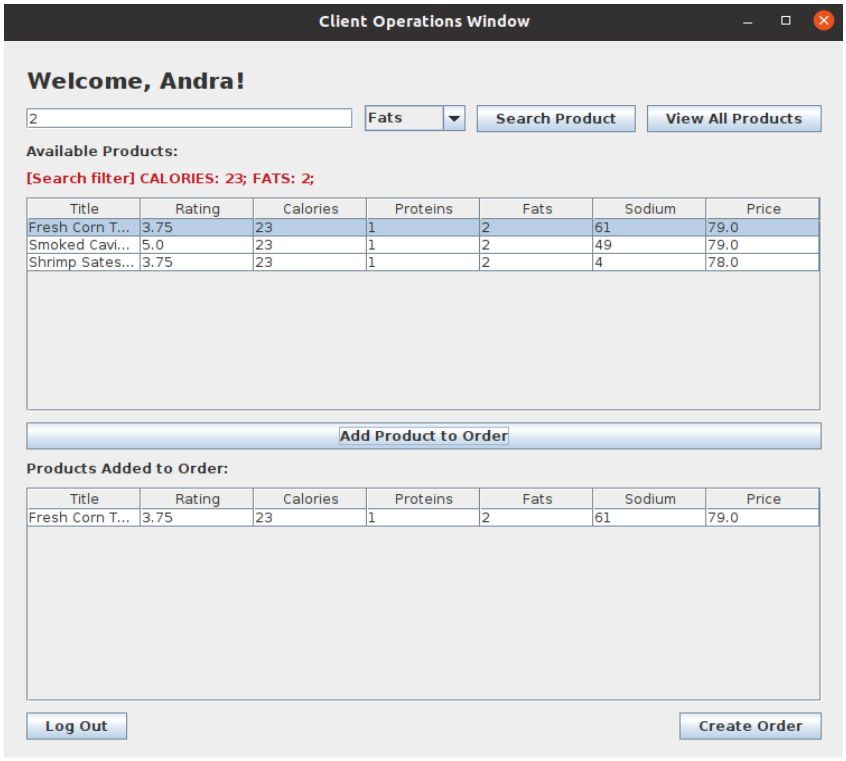
1) Log in / Sign up Window



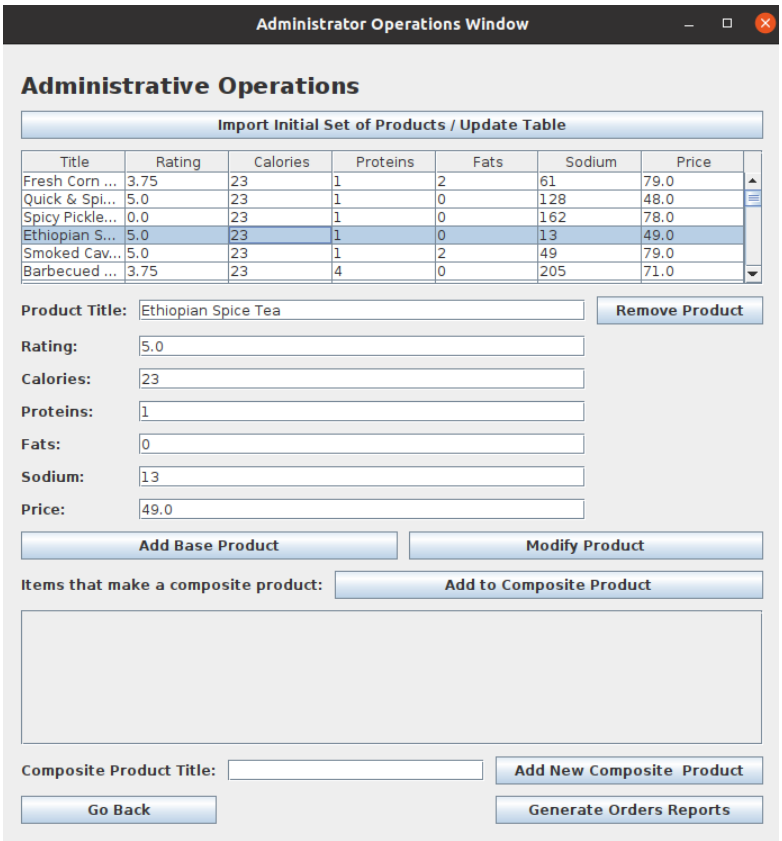


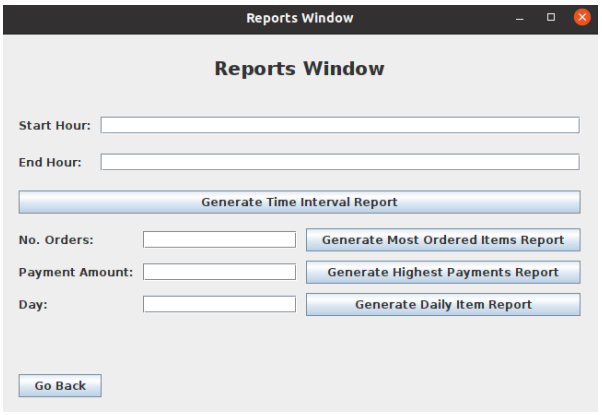
2) Client Window



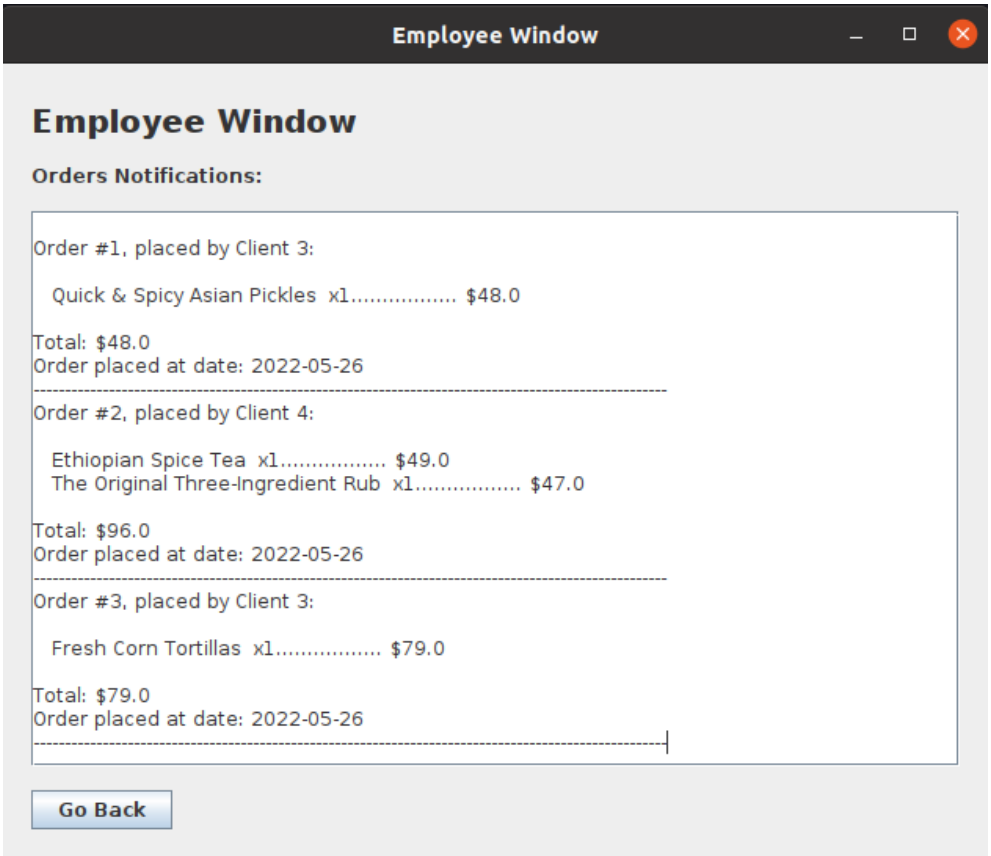


3) Administrator Window and Reports Window





4) Employee Window



6. Conclusions

By working on this assignment, I learned how to use lambda expressions and stream processing, as well as how to work with interfaces for real-time adaptations of windows, using observables and observers, and how to maintain data by the means of serialization. This was overall a very complex project, that used me enrich my Java programming knowledge significantly.

7.Bibliography

https://www.baeldung.com/java-observer-pattern

https://www.baeldung.com/java-composite-pattern

http://www.java2s.com/Tutorials/Java/Swing/JTable/Get_selected_value_from_JTable_in_Java.htm

https://www.section.io/engineering-education/assertion-and-design-by-contract-in-java/

https://stackoverflow.com/questions/34639928/parsing-a-csv-file-for-a-unique-row-using-the-new-java-8-streams-api

https://www.baeldung.com/java-stream-filter-lambda

https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html

https://www.baeldung.com/java-stream-filter-lambda

https://www.geeksforgeeks.org/traverse-through-a-hashmap-in-java/