Assignment\_4

**FOOD DELIVERY MANAGEMENT SYSTEM**

**Teacher**: Viorica Rozina Chifu

**Student**: Furcovici Teodora-Melania

**Group**:30421

1. **Assignment objective**

* **Main objective**

The purpose of this assignment was to simulate the process of  customer orders for a warehouse.

* **Sub-objective** 
  1. Analyze the problem and identify the requirements
  2. Design the relation between database and application
  3. Implement relation between database and application
  4. Test the relation between database and application

1. **Problem analysis, modelling, scenarios, use cases**

* **Overview**

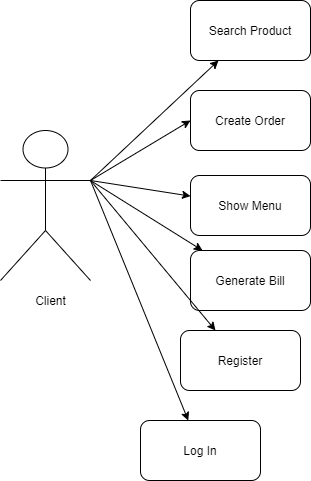
This application should be able to fulfil all the requirements in order to add, modify and delete items. These are stored in a Excel file , along with the information about the items, like calories, proteins, price and so on. This way, all the data is easier to retrieve and access from different computers.

* **Modelling**

The user will be able to use the INSERT button in order to insert a new client, order or product in the database. Also, if the user presses the DELETE button it deletes certain row, and if the UPDATE button is pressed then it will modify a certain row from the database.

After the SHOW MODIFICATIONS button is pressed, the result will appear in the interface.

* **Scenarios and use cases**

****

1. **First use case (Show)**

**Normal scenario**

The user presses the SHOW MENU button and the table is displayed in the interface.

**Alternative scenario**

The user doesn`t fill in all the compulsory fields or in some fields the data introduced doesn`t respect the imposed format. In this case, a message will issue informing the user that the values are not correct.

1. **Second use case (Create Order)**

**Normal scenario**

The user inputs in the mandatory fields the order details and then presses the CREATE ORDER button, then a table with its order details will be displayed in the interface.

**Alternative scenario**

The user doesn`t fill in all the compulsory fields or in some fields the data introduced doesn`t respect the imposed format. In this case, a message will issue informing the user that the values are not correct.

1. **Third use case (Generate Bill)**

**Normal scenario**

The user presses the GENERATE BILL button and a notification is displayed in the interface, with information about the current order.

**Alternative scenario**

The user doesn`t fill in all the compulsory fields or in some fields the data introduced doesn`t respect the imposed format. In this case, a message will issue informing the user that the values are not correct.

1. **Fourth use case (Search)**

**Normal scenario**

The user presses the SEARCH PRODUCT button and the table is displayed in the interface containing the elements which fulfil the entered criteria.

**Alternative scenario**

The user doesn`t fill in all the compulsory fields or in some fields the data introduced doesn`t respect the imposed format. In this case, a message will issue informing the user that the values are not correct.

1. **Fifth use case (REGISTER)**

**Normal scenario**

The user presses the REGISTER button and client is registered.

**Alternative scenario**

The user doesn`t fill in all the compulsory fields or in some fields the data introduced doesn`t respect the imposed format. In this case, a message will issue informing the user that the values are not correct.

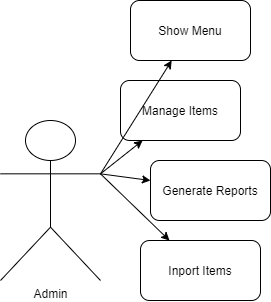
1. **Sixth use case (Log In)**

**Normal scenario**

The user presses the LOG IN button and the interface for the client is going to be displayed in the interface.

**Alternative scenario**

The user doesn`t fill in all the compulsory fields or in some fields the data introduced doesn`t respect the imposed format. In this case, a message will issue informing the user that the values are not correct.



1. **First use case (Add Item)**

**Normal scenario**

The user presses the ADD PRODUCT button and the table is displayed in the interface.

**Alternative scenario**

The user doesn`t fill in all the compulsory fields or in some fields the data introduced doesn`t respect the imposed format. In this case, a message will issue informing the user that the values are not correct.

1. **Second use case (Delete Item)**

**Normal scenario**

The user presses the DELETE ITEM button and the table is displayed in the interface.

**Alternative scenario**

The user doesn`t fill in all the compulsory fields or in some fields the data introduced doesn`t respect the imposed format. In this case, a message will issue informing the user that the values are not correct.

1. **Third use case (Modify Item)**

**Normal scenario**

First, the user pressed the DELETE BUTTON and then after the desired modifications, he/she will presses the MODIFY BUTTON and the table is displayed in the interface.

**Alternative scenario**

The user doesn`t fill in all the compulsory fields or in some fields the data introduced doesn`t respect the imposed format. In this case, a message will issue informing the user that the values are not correct.

1. **Fourth use case (Show Menu)**

**Normal scenario**

The user presses the SHOW MENU button and the table is displayed in the interface.

**Alternative scenario**

The user doesn`t fill in all the compulsory fields or in some fields the data introduced doesn`t respect the imposed format. In this case, a message will issue informing the user that the values are not correct.

1. **Fifth use case (Import Products)**

**Normal scenario**

The user presses the IMPORT PRODUCTS button and the table with data retrieved from the products.csv file is displayed in the interface.

**Alternative scenario**

The user doesn`t fill in all the compulsory fields or in some fields the data introduced doesn`t respect the imposed format. In this case, a message will issue informing the user that the values are not correct.

1. **Sixth use case (Generate Reports)**

**Normal scenario**

The user presses the CREATE REPORT button and the a new interface will be displayed, where the criteria will be introduced. Based on them, certain reports will be generated.

**Alternative scenario**

The user doesn`t fill in all the compulsory fields or in some fields the data introduced doesn`t respect the imposed format. In this case, a message will issue informing the user that the values are not correct.

O imagine care conține text

Descriere generată automat

1. **First use case (Get Notified)**

**Normal scenario**

The user presses the GENERATE BILL from the client interface button and a notification with information about the current order is going to be displayed. Besides this, the employee interface will appear.

**Alternative scenario**

The user doesn`t fill in all the compulsory fields or in some fields the data introduced doesn`t respect the imposed format. In this case, a message will issue informing the user that the values are not correct.

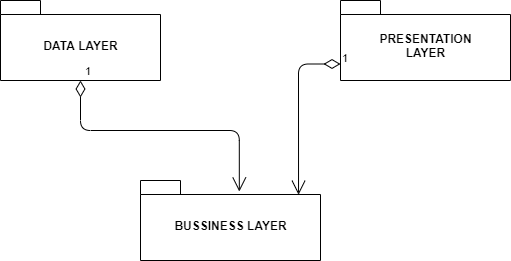
1. **Design (design decisions, UML diagrams, data structures, class design, interfaces, relationships, packages, algorithms, user interfaces)**

* **Black Box**

O imagine care conține text

Descriere generată automat

* **Packages**

****

In develop my application after the Layered Architecture Design Pattern. I believe that the data becomes more re-useable and easy to follow.

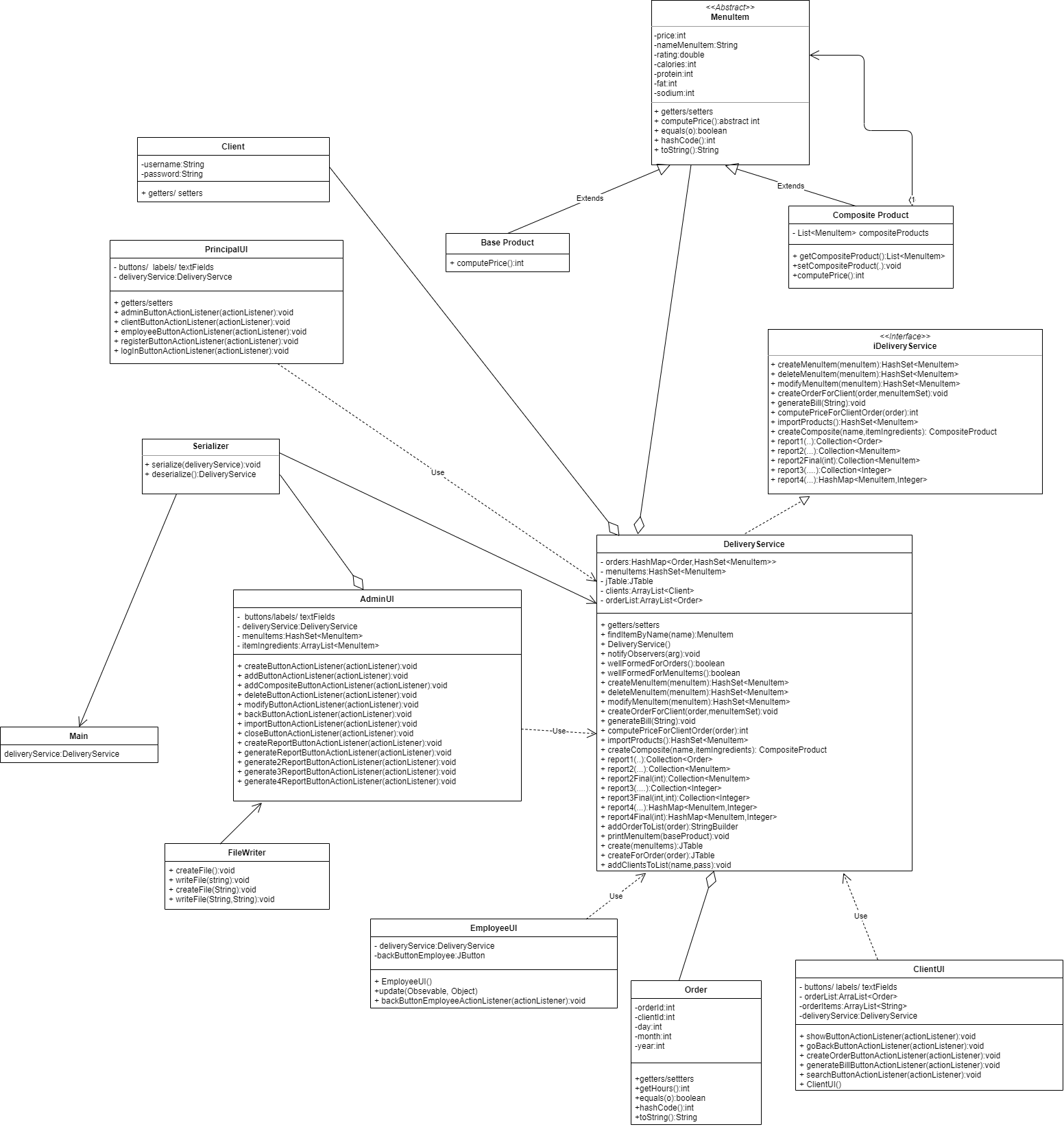
My application is divided into 3 parts because it respects the Layered Architecture Design Pattern.

**Business Layer** package represents the Business Logic of the application, and it contains all the classes necessary to implement the application and also to control it.

**Presentation Layer** package represent the UI package, where the User Interface are implemented.

**Data Layer** package contains the classes for writing/creating a file and also for the serialization/deserialization.

* **UML Diagrams**

****

* **Data structures**

Data Structures I have used in this project are basic, such as primitive data types (integer and double, mostly) and reference data types. Some complex data structures would be the HashMap, for storing the orders and the HashSet for keeping information about the Menu Items.

The reference type I used developing this application was Client, Order, BaseProduct, Composite Product and MenuItem.

HashSet are preferred here due to the fact that they are very easy to use and have a lot of methods, which makes the life of the programmer easier. In this case we don`t have to worry about always keeping the length(as it would have happened for simple arrays) and also it easier to add to an Vector rather than to an array.

* **Class Design**

I choose not to write all the code in one class, because it is more readable for persons who see the application for the first time and because if it happens to appear an error, I have to search it in the entire program and it is loss of time.

Fora better understanding of the application and to be more accessible, I choose to divide the program in 3 major parts: **BUSINESS,** **DATA** and **PRESENATION**. Each of this parts in the application is represented as a separated package, which contains one or more classes.

For the PRESENTATION package, I designed a class where I take care of all problems that may arise when designing the GUI.

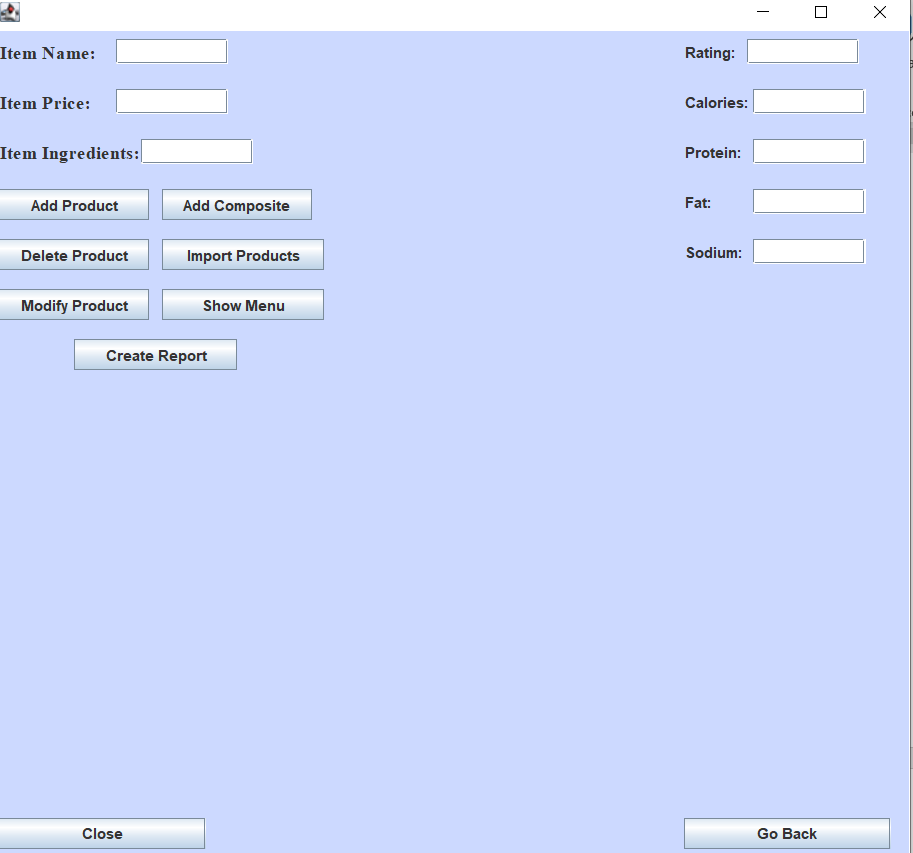
For the DATA package, I designed one class for writing or creating a file and another class for the Serialization part.

For the BUSINESS package, I designed one class for each reference type (e.g Order, Base Product and so on) and also a Delivery Service class, which is kind of a controller of the whole application.

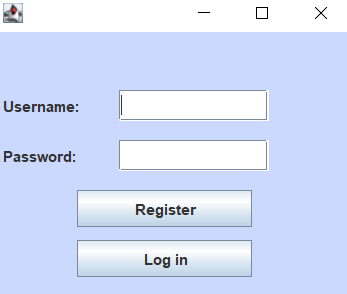
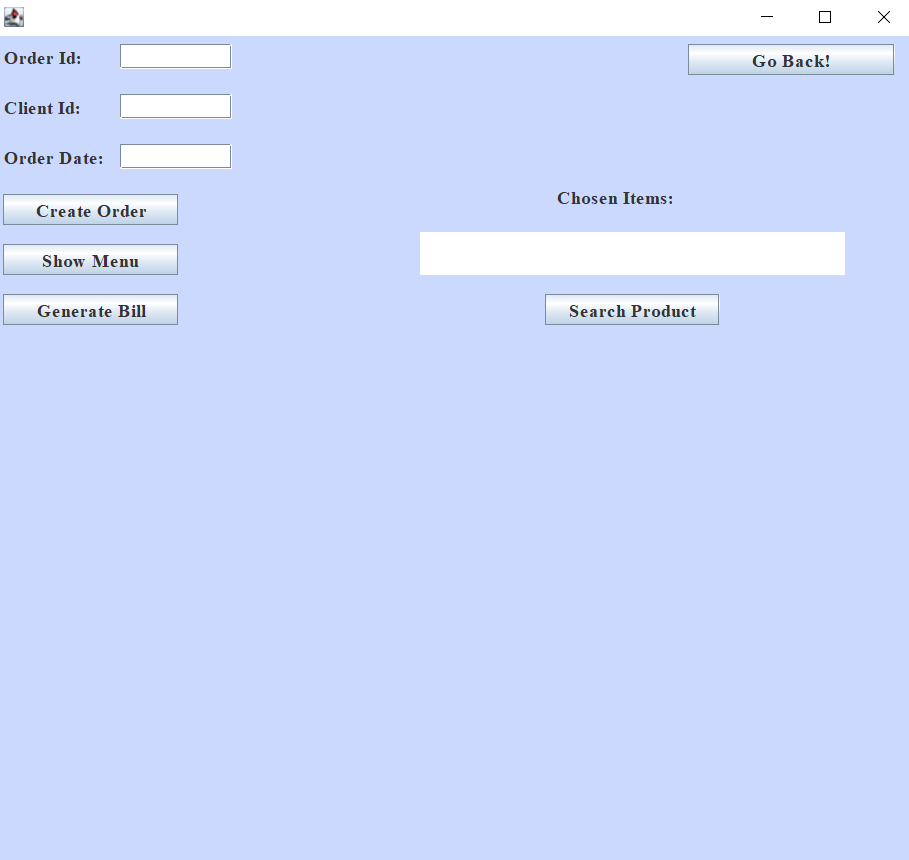
* **User Interfaces**

The user interface is the connection between the user and the programmer, because with it the application can executes the commands.

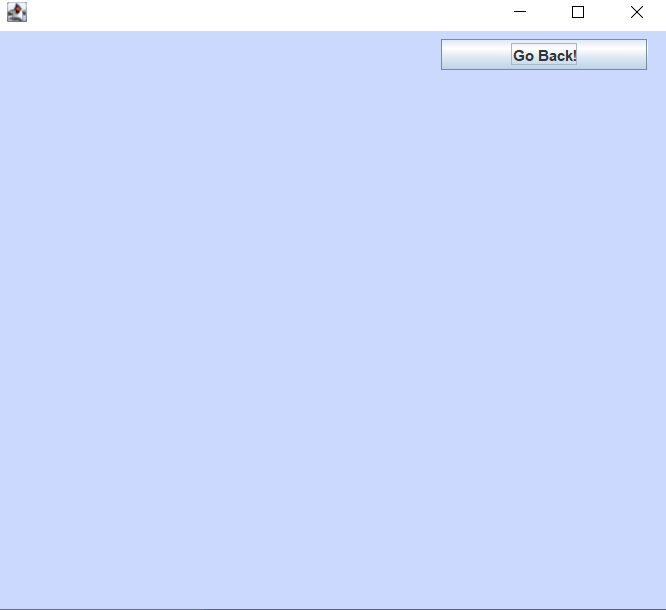
****The first user interface, from where the user chooses based on its type in the food delivery application.



The interface for the users that are admins.



Both interfaces are for the users that are clients to the app.



The user interface for the employee

The user inputs the necessary data for performing one certain operations and see the results. The result will be visible in the Table area, just after he/she pressed the button with the operation she/he wants to perform.

Beside this, the user has to respect some conditions in order to see the result:

* + The values introduced in the destinated fields must contain only digits and be a positive integer value, in case an ID is required, or a String otherwise.
  + Then another compulsory condition that must be fulfilled is that all the fields must contain a certain number/name, otherwise the application won`t work.

1. **Implementation**

As I said above, my application follows the “rules” of the Layered Architecture pattern, and so I decided to split my program into more classes, which, practically, is the scope of Java Programming. The application is divided as follows:

* 1. **DATA LAYER**
     1. **FileWriter**

The main purpose of this class is to make the relation between the files and the UI.

* + 1. **Serializer**

In this class I deal with the problems that may arise when performing serialization/deserialization on a file.

* 1. **PRESENTATION LAYER**
     1. **Employee UI**

In the EMPLOYEE UI class are defined all the elements needed to form the Graphical User Interface for the employee and the methods for handling the button actions.

* + 1. **AdminUI**

In the ADMIN UI class are defined all the elements needed to form the Graphical User Interface for the admin user and the methods for handling the button actions.

* + 1. **ClientUI**

In the CLIENT UI class are defined all the elements needed to form the Graphical User Interface for the client and the methods for handling the button actions.

* + 1. **Principal UI**

In the PRINCIPAL UI class are defined all the elements needed to form the Graphical User Interface for that link all the others 3 interfaces and the methods for handling the button actions.

* 1. **BUSINESS LAYER**
     1. **Base Product**

This class only extends the MenuItem class, and it has an implementation for one method from MenuItem.

* + 1. **Composite Product**

This class only extends the MenuItem class, and it has an implementation for one method from MenuItem. Besides this, in this class I declared an ArrayList of Base Products, such that is easier to store the items from a composite product (e.g a daily menu).

* + 1. **Client**

This class contains 2 instances variables, one for username and one for password.

* + 1. **MenuItem**

In this class, I declared all the details about a base product (such as: fat ,protein and so on). Because this class implements Serializable, I implemented also the hashCode and equals method.

* + 1. **Order**

This class contains the order id, client id and the date of the order.

* + 1. **Delivery Service**

This class is the heart of all the application, because from here I perform all the operations that are available for the client and admin. I have implemented one method for each operations that can be done, such as: add/delete/modify, create new products, generate reports and so on.

O imagine care conține text

Descriere generată automat

**This method is used when we add a new base product.**

O imagine care conține text

Descriere generată automat

**This method performs the desired modifications on a base product, and here after the item is deleted, we insert it back with the details modified.**

**O imagine care conține text

Descriere generată automat**

**The most complex and probably the longest one is the method where I import products from .csv file. Here I create a new Base Product to insert it into the HashSet<MenuItem> and besides this, I verify not to have duplicates.**

**The logic behind choosing from multiple identical base products was to add into the HashSet the first that it will be encountered into the .csv file.**

**O imagine care conține text

Descriere generată automat**

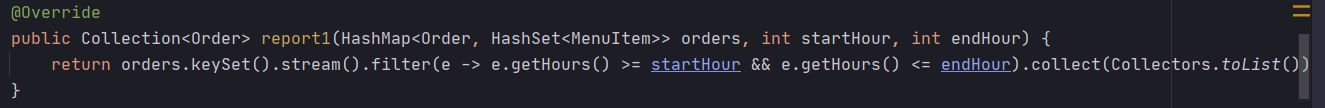
**This method uses reflection for creating a table based on the fields that are in the Menu Item class.**

**O imagine care conține text

Descriere generată automat**

**Here, I used streams to generate reports having the following criteria:**

**The products ordered within a specified day with the number of times they have been ordered. A HashMap with key equal to the MenuItem and the value the value of the day parameter. Using some getters from the Order class, I verify if the day is the same as the one transmitted by the user.**

****

**The method for filtering the reports based on their start and end hour. Using the getter getHours() from Order class, which take the current hour when the order has been created, and its result is compared to see if it meets the desired standards.**

* + 1. **IDeliveryService interface**

In this interface I declared only the headers of each method implemented in the DeliveryService class. Using, pre and post conditions we verify the correctness of the data inputed by the user.

1. Results

O imagine care conține masă

Descriere generată automat

The result after adding a base product.

O imagine care conține masă

Descriere generată automat

The result after deleting a base product.

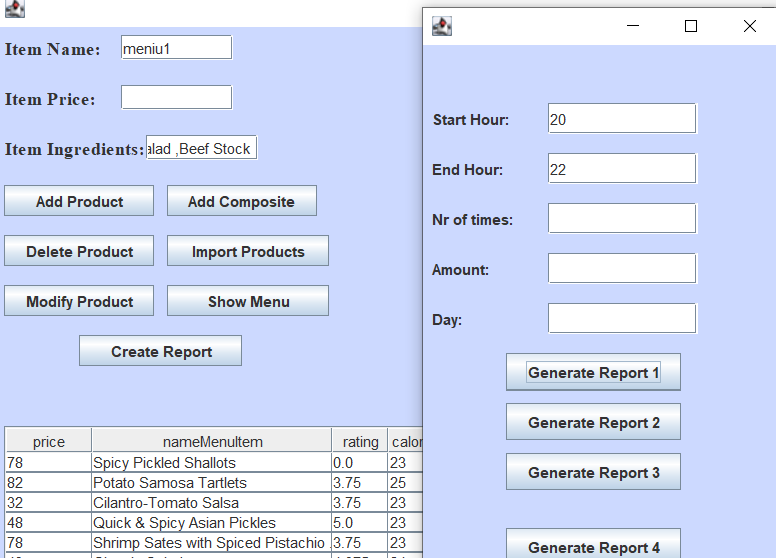
O imagine care conține masă

Descriere generată automatThe result after modifying a base product.

O imagine care conține masă

Descriere generată automat

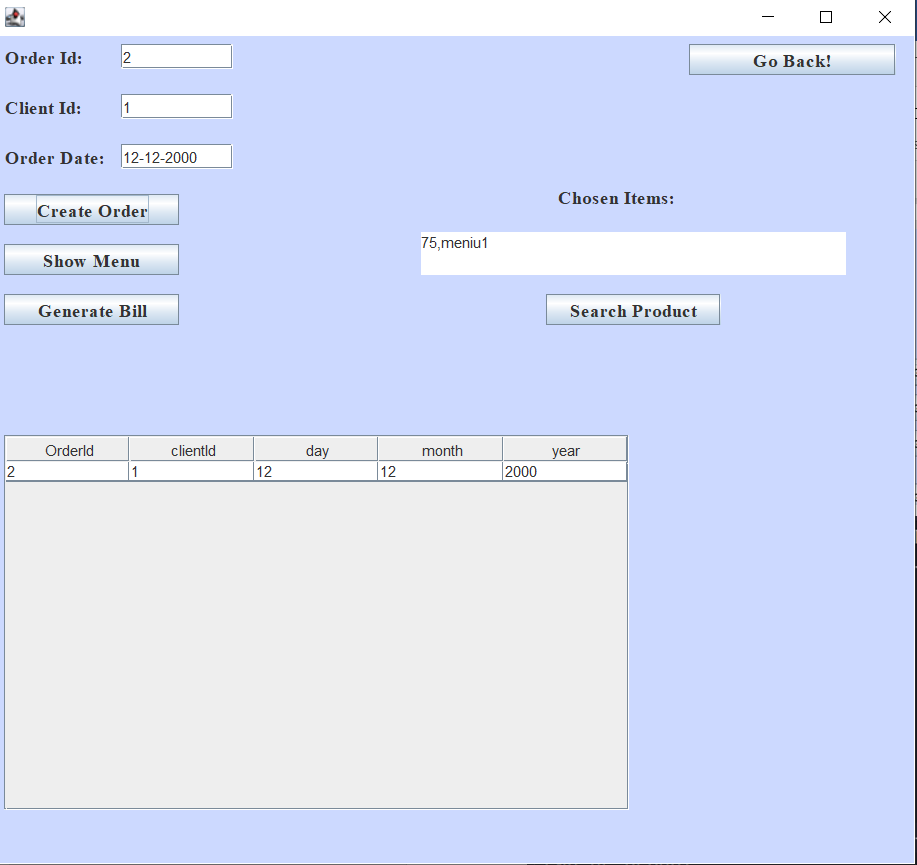
The result after inserting a composite product.



O imagine care conține text

Descriere generată automat The result after generating report based on the first criteria.



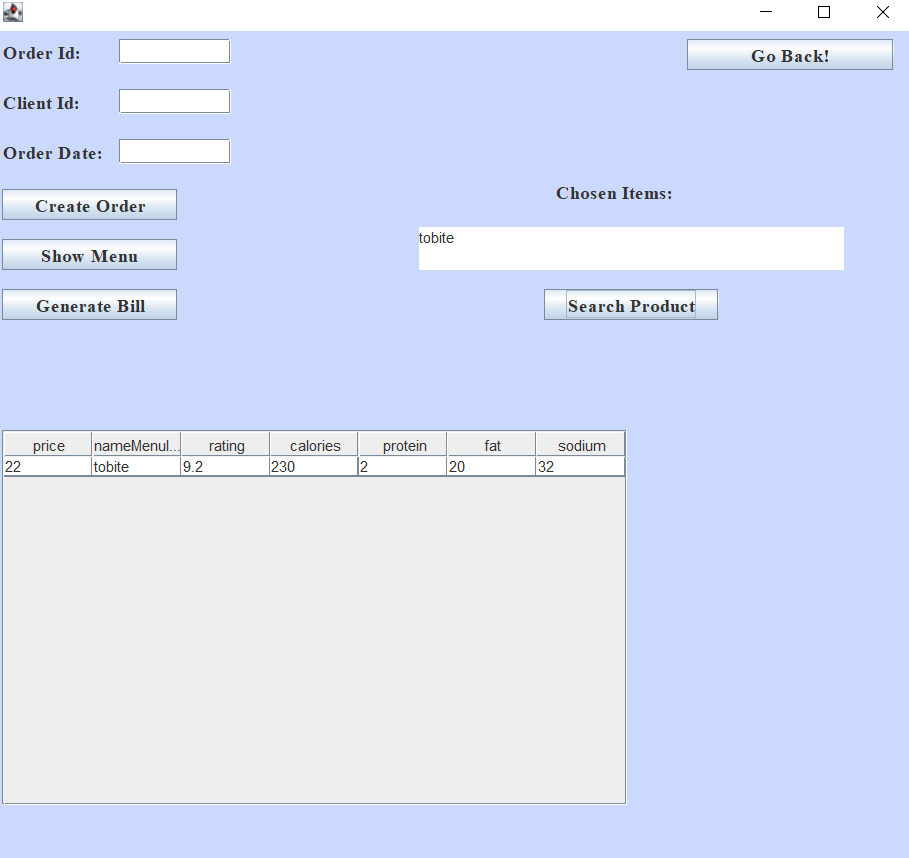
The result after entering one order.

O imagine care conține text

Descriere generată automatThe result after generating a bill.

O imagine care conține text

Descriere generată automat

The result after searching an item.



1. **Conclusions**

For this assignment it was harder to build a strategy and implement it, because the notion of thread was totally new to me. Anyway, I found this very challenging at first, but then I got used to it. I consider that it was useful, due to the fact that we learned how the connectivity with a database working and how we are supposed to modify or list the contents of a table.

I arrived at the conclusion that facing problems with your code and trying to make it work by yourself, through the mean of research, has the benefit of learning new concepts and a better use of the known ones.

About further developments, I think that one advantage would be to have a more user friendly interface.

1. **Bibliography**

[1] <http://tutorials.jenkov.com/java-reflection/private-fields-and-methods.html>

[2] <http://tutorials.jenkov.com/java-reflection/fields.html>

[3] <https://www.javatpoint.com/java-int-to-string>

[4] <https://www.geeksforgeeks.org/vector-setsize-method-in-java-with-example/>

[5] <https://www.javatpoint.com/java-date-getdate-method>

[6] <https://stackoverflow.com/questions/15105739/how-to-make-entire-jtable-invisible-in-swing>

[7] <https://app.diagrams.net/>