DOCUMENTATION

ASSIGNMENT NUMBER 4

FOOD DELIVERY MANAGEMENT SYSTEM

Jimborean Oana Elena

Group 30422

CONTENTS

1. Objectives. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ..3
2. Analysis. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3
3. Design. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .5
4. Implementation. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .7
5. Results. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
6. Conclusions. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ..10
7. Bibliography. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11
8. Objectives

• Main objective:

Design and implement an application for managing the food orders for a catering company.

The system should have three types of users that log in using a username and a password: administrator, regular employee, and client.

- The administrator can import the initial set of products and generate the menu, can add, delete, modify products or create new products composed of several products and generate different reports related to orders made according to certain criteria.

- The customer can register and use the registered name to log in, see the list of products on the menu, search for the products on the menu according to different criteria, and create orders consisting of several products, and for each order will be generated an invoice with the date and time of creation, the list of products ordered and the total price of the order.

- 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.

• Sub-objectives:

-Analyze the problem and identify requirements- the modeling of the problem will be performed.

-Design the orders management application- the OOP design decisions will be presented (i.e. UML diagrams, data structures, class design, relationships, packages, interfaces, user interfaces).

-Implement the orders management application- the implementation of the classes (i.e. fields and important methods) and of the graphical user interface will be presented

-Test the orders management application- results will be written in the .txt files.

1. Problem analysis, modeling, scenarios, use cases

This application must meet all the necessary requirements so that users first log into the application and then, depending on the identity of the user (administrator, customer or employee), be able to perform the required operations.

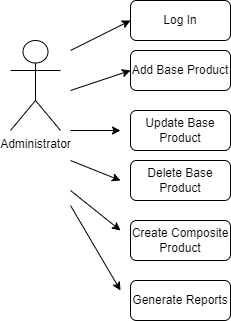
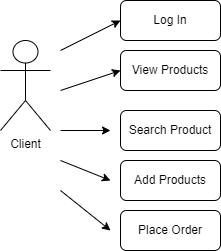
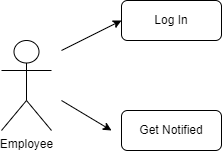
To meet the requirements, observer design patterns must be used to notify employees each time a new order has been placed) and Composite Design Pattern.

The Observer Design Pattern defines a 1..\* (one to many) dependency relationship between objects so that when an object changes its state, all its dependents are notified and updated automatically. The use of this pattern involves the existence of an object with the role of a subject, which re associated with a list of dependent objects, with the role of observers, which it automatically calls each time an action happens.

This pattern is behavorial (behavioral) type, because it facilitates a better organization of communication between classes according to their roles.

Composite Pattern design is used when we need to treat a group of objects similarly as if it were a single object. Within this application, it is used in the production of products composed of several basic products, these composite products being treated as a single MenuItem product.

* Use cases

**Use Case 1**: Add, create, update the menu

**Lead actor**: administrator

**Main success scenarios**: 1. The administrator enters in the login window the username: admin and password: admin, then press the button Administrator and opens the corresponding window.

2. The administrator enters new data for a BaseProduct product.

3. The administrator may choose to modify, add or delete the product with the data entered, by pressing the button corresponding to the operation.

4. The system takes over the information, and if the data is correct, the operation will be performed.

**Alternative sequences**: - the user enters wrong data, in this case an error message will be displayed.

- the user enters the wrong name and password when logging in.

**Use Case 2**: Create menu

**Lead actor**: client

**Main success scenarios**: 1. The client enters in the login window his username and password, then press the Client button and the corresponding window is opened.

2. The customer enters in textField the name of the product he wants to add to the order.

3. If you want to add more products, press the Add Product button and enter the name of the next product. This operation is repeated until the customer no longer wants to add other products to the order.

4. When he has finished adding all the products to the order, the customer presses the Finish Order button.

5. The order is saved, it is sent to the employee and an invoice in .txt format is generated.

**Alternative sequences**: - the user enters a product name that is not found in the menu, in this case an error message will be displayed

- the user enters the wrong name and password when logging in.

**Use Case 3**: View commands

**Lead actor**: employee

**Main success scenarios**:

1. The employee visualizes in the appropriate window the orders, as they are created by the customers.
2. Design

A Layered Architecture is the organization of the project structure into four main categories: presentation, application, domain, and infrastructure. Each of the layers contains objects related to the particular concern it represents.

The presentation layer contains all of the classes responsible for presenting the UI to the end-user or sending the response back to the application.

The application layer contains all the logic that is required by the application to meet its functional requirements and, at the same time, is not a part of the domain rules.

The domain layer ( business layer ) represents the underlying domain, mostly consisting of domain entities and, in some cases, services. It contains the classes that encapsulate the application logic.

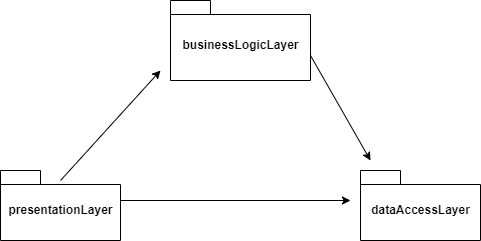
The Data Layer contains all the classes responsible for saving and processing the sets of objects.through serialization and deserialization and the classes which handle the file writing.

The whole idea of splitting your program into classes is based on a general rule named divide and conquer. This paradigm can be used almost everywhere: you divide a problem into smaller problems and then you solve these little, simple and well-known problems.

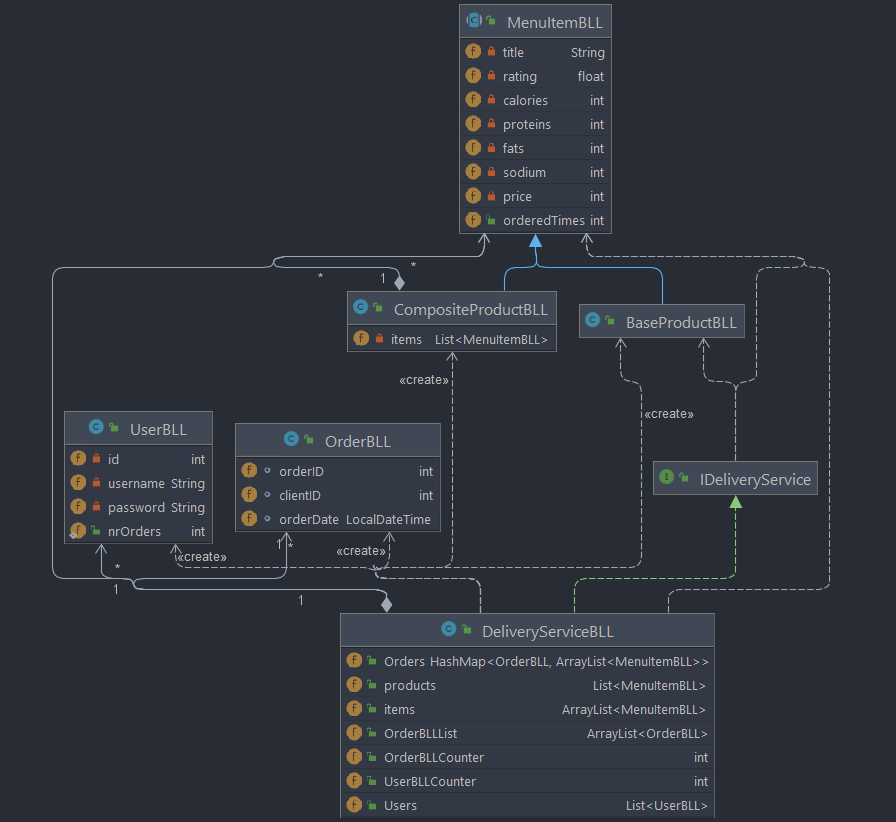
Dividing your program into classes is one of the types of division which started to become common in last decade. In this programming paradigm we model our problem by some objects and try to solve the problem by sending messages between these objects.

I also used the Observer and the Composite design patterns with respect to the restaurant + employee + admin and the MenuItem + BaseProduct + CompositeProduct respectively.

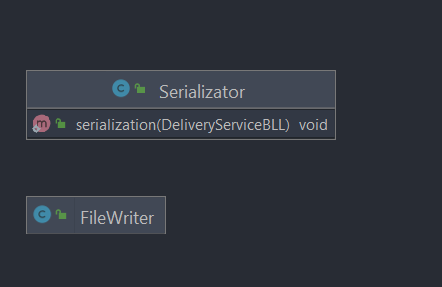
* Package Diagram:



* UML diagram for classes from the businessLogicLayer package:



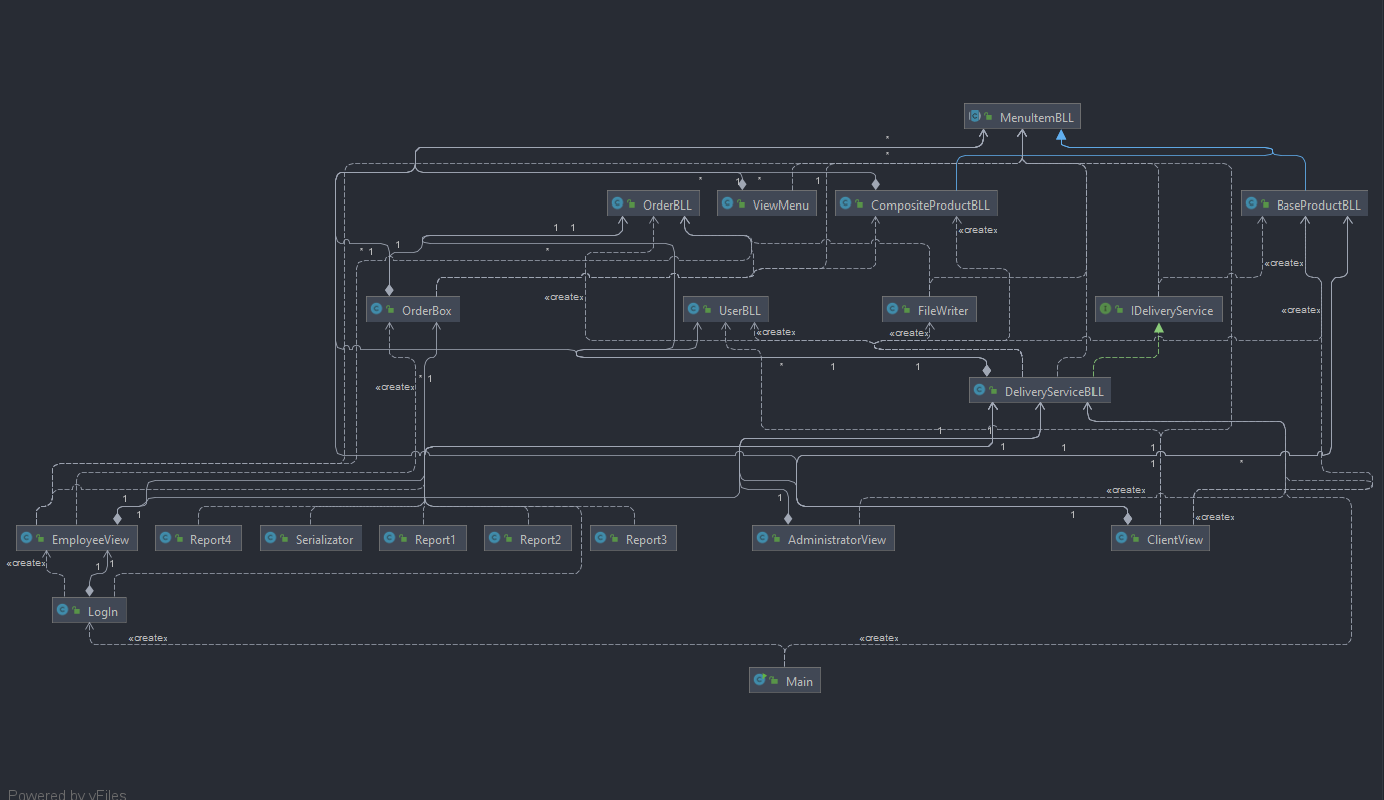
* UML diagram for classes from the dataAccessLayer package:



* UML diagram for classes from the presentationLayer package:



* Class diagram:



1. Implementation

I used 3 packages:

- businessLogicLayer: contains the classes that encapsulate the logic of the application. It contains the classes that together represent the logic of the application. The main class in this package is DeliveryService which implements the IdeliveryServiceProcessing interface, having defined the methods that achieve all the necessary requirements to meet the needs of customers, administrator and employees.

- dataAccessLayer: contains the serialization and writing classes in the file. It contains the Serializer class to retain deliveryservice class information using the serialization method, in this way, the changes made to the data will be stored, the FileWriter class for generating the invoice for the client .

- presentationLayer: contains the classes that define the user interface, in which the interface windows corresponding to each type of user are implemented

- the main window (MainGUI): offers the possibility of authentication or registration in the application of users

**Main Class:**

-the main class of application testing.

-creates an instance of the Login class that starts the application's start window.

**PresentationLayer Package:**

-contains the necessary classes to achieve the graphical user interface

-*Admin, Client, Employee, Login* classes are the main windows

-*ViewMenu* class is the window of displaying a table with the products in the menu

-classes *Report1, Report2, Report3, Report4* are the classes where the reports are generated by the administrator

-*OrderBox* class, represents the boxes that are added to the Employee when an order is created

**DataAccessLayer Package:**

*FileWriter Class*:

-generates an invoice in txt format when a customer creates a new order

*Serializer class*: -contains 2 methods: a method for serialization, saving menu data and commands in a .serum file, and a method for reading data from there.

**BusinessLogicLayer Package:**

*BaseProductBLL Class:*

-for defining this class I used The Composite Design Pattern

-expand MenuItemBLL class

-represents a basic product that has the attributes of menuitemBLL class

-overwrite the computePrice() method which in this class simply returns the price of the product

*CompositeProductBLL Class:*

-for defining this class I used The Composite Design Pattern

-expand MenuItemBLL class

-represents a product composed of several basic products (BaseProduct)

-overwrite the computePrice() method, which here returns the sum of the prices of all the products of which the CompositeProduct product is composed.

*DeliveryServiceBLL class:*

-implements Ideliveryservice interface

-DeliveryService class is the center of the application, all actions, functionalities are found in this class.

-it is built using Design by Contract, testing preconditions and postconditions using assert instructions in all methods implemented in this class.

*IDeliveryService interface:*

-interface that is implemented by DeliveryServiceBLL

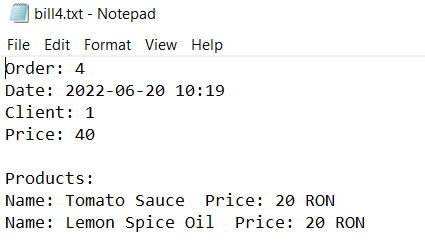
-contains the methods for implementing the functionalities for the client and the administrator.

1. Results

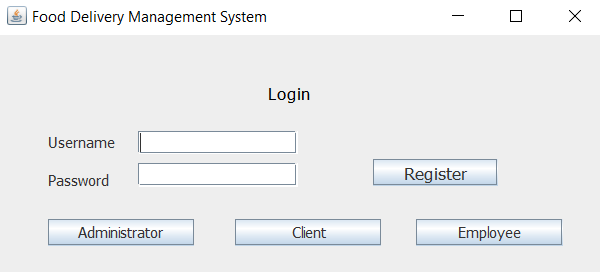
When starting the application, choose which user will use the application, then redirect the user to the corresponding window. They can enter data in the available fields and execute the functionalities of each one.

For each order, a new file is created and it is inserted in the employee's window.

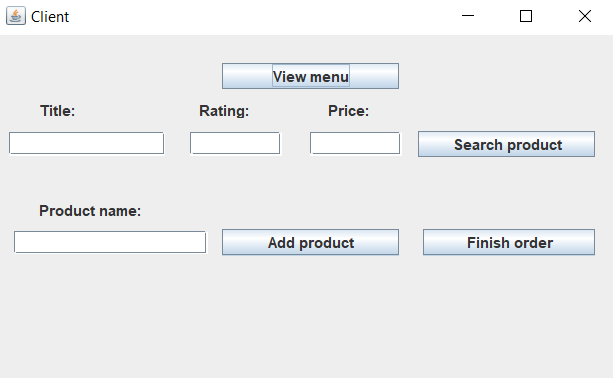
The contents of a file containing a command:



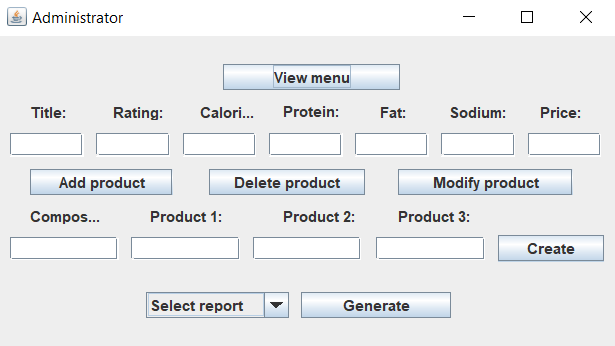
The interface for the LogIn window:



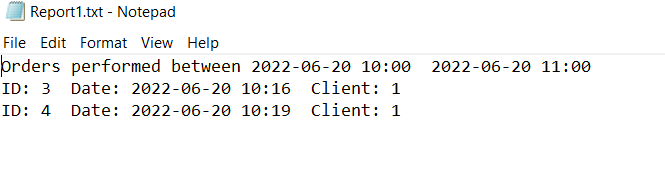
The interface for the Client window:



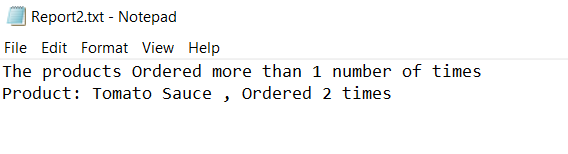
The interface for the Admin window:



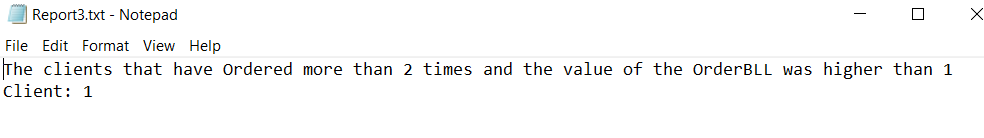
The contents of the generated report1:



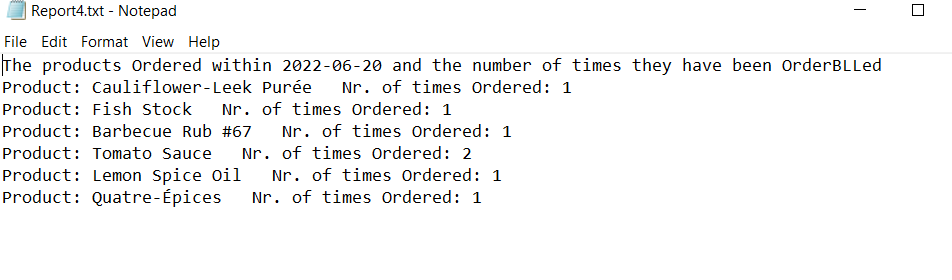
The contents of the generated report2:



The contents of the generated report3:



The contents of the generated report4:



1. Conclusions

The conclusions include that a well-thought-out planning of classes makes things easier and saves time. With the increasing difficulty in projects, the problem must be approached with an intelligent perspective, and not just immediately jump to writing the code. Thus, starting with the UML diagrams, I gave myself time to reflect for the proper approach to the implementation of the desired program.

After implementing this project I learned to manage and process the information in the tables in order to be able to display it and to be able to interact with it, to use streams and lambda expressions, which made my work very easier, reaching me to do searches by writing less code.

Other aspects I learned were the implementation of Design by Contract, Composite Design Pattern, Observer Design Pattern, which helped me understand how to use such a model, what are its advantages and why I should work with it.

I believe that a further development could be to add a real-time simulation of orders, the measurement of the delivery time of practically a simulation application close to reality, the implementation of more functionalities, a more friendly user interface.

1. Bibliography

[https://app.diagrams.net/#](https://app.diagrams.net/)

<http://www.tutorialspoint.com/java/java_serialization.htm>

<https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>

<http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html>

<http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html>