**Assignment nr. 4**

1. **Assignment objective.**

The scope of this assignment is to simulate an application for a restaurant. This application includes features for the administrator, for an employee and for a regular costumer. In order to use the application, the first thing you need to do is to create an account or use an existing one. The administrator should be able to use the application in order to add new products to the menu of the restaurant, create menu items composed of more other existing items, modify menu items, delete menu items, generate a report regarding clients, orders and products. The features for a client are: search for menu items with some specific attributes, make an order, receive a bill. The employee should be able to see the most recent orders and to mark them for delivery.

In order to make accomplish all this, some secondary requirements had to be done. Any user should not be able to create an account with a user name that is already registered. A costumer should be able to search for a menu item just after some attributes (like price), it shouldn’t be necessary to know all the characteristics of that product. Data for the application should be saved and recover every time the app is closed or opened, new items and orders should be available for the following users.

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

* Functional assignments: user can: create a new account, use an existing account to log in, see menu items, modify, add, delete, menu items, get a bill, get a report based on multiple parameters, create a menu, see what a menu contains, see undelivered orders, mark an order as delivered.
* Non-functional requirements: the application should be easy to use, provide helpful messages in cases of errors, display the result in a friendly form.

Use cases:

Success scenario:

1. User enters values for a username and password, clicks “Sign up” and get a message saying that he/she was added as a user.

Success scenario:

1. User fills the text field for username and password with the same values he/she used when creating an account, the application switches to a window where the user is able to search for products and make an order.

Alternative scenario:

1. User enters a username that already exists and is notified by the application.

Alternative scenario:

1. User tries to sign in but the password entered doesn’t match the username. A message saying that the password or username is wrong is displayed on the screen.

Success scenario:

1. In the user window, a specific menu item is searched. User fills up the text field describing the menu item. One or multiple menu items with the required characteristics are shown on the screen.

Alternative scenario:

1. In the user window, a specific menu item is searched. User fills up the text field describing the menu item. No such item is found so the text area remains empty.

Success scenario:

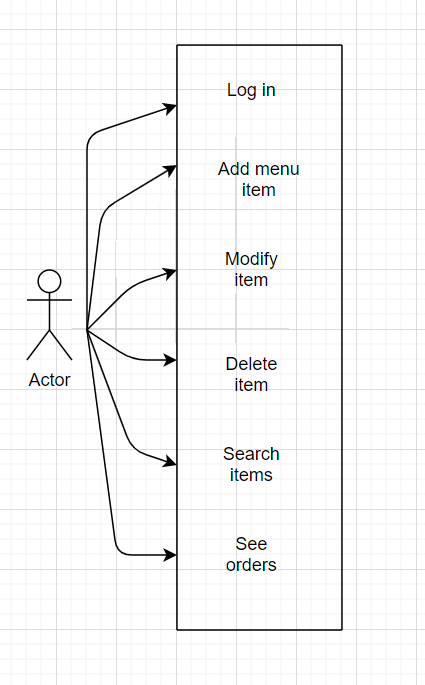
1. Admin tries to modify a menu item. Enters the name of that menu item, clicks the “Modify” button and is informed that that item exists. Fills up the text fields corresponding to the values of the product to modify and clicks the “Modify” button. The item is successfully modified.

Alternative scenario:

1. Admin tries to modify a menu item. Enters the name of that menu item, clicks the “Modify” button. No such item is registered in the application so a warning message is displayed on the screen.

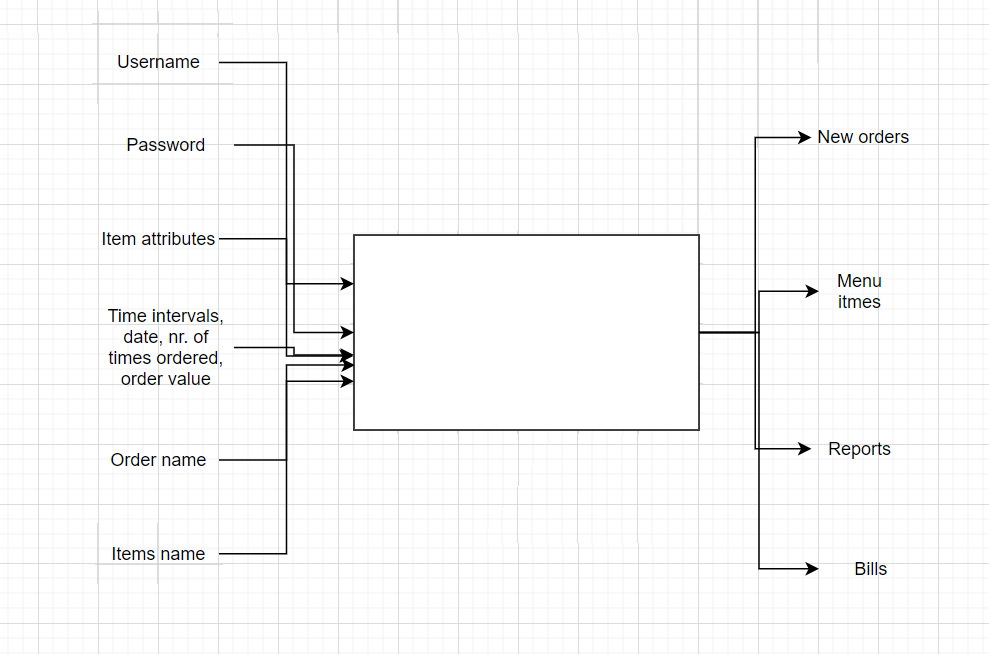
Success scenario:

1. Am employee logs in. Clicks the button to see the new orders, undelivered by other employees. Enters the id of the orders he/she wants to take care of. That order will no longer be seen by the rest of the employees.

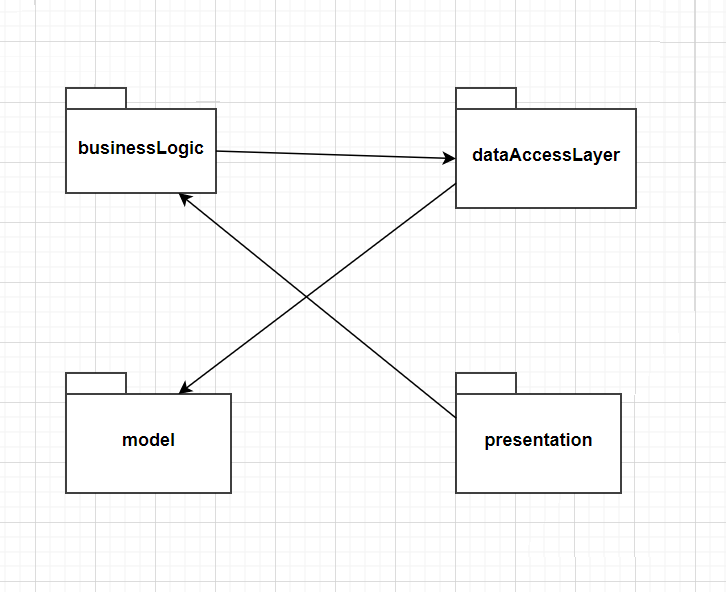


1. **Design.**

* Application

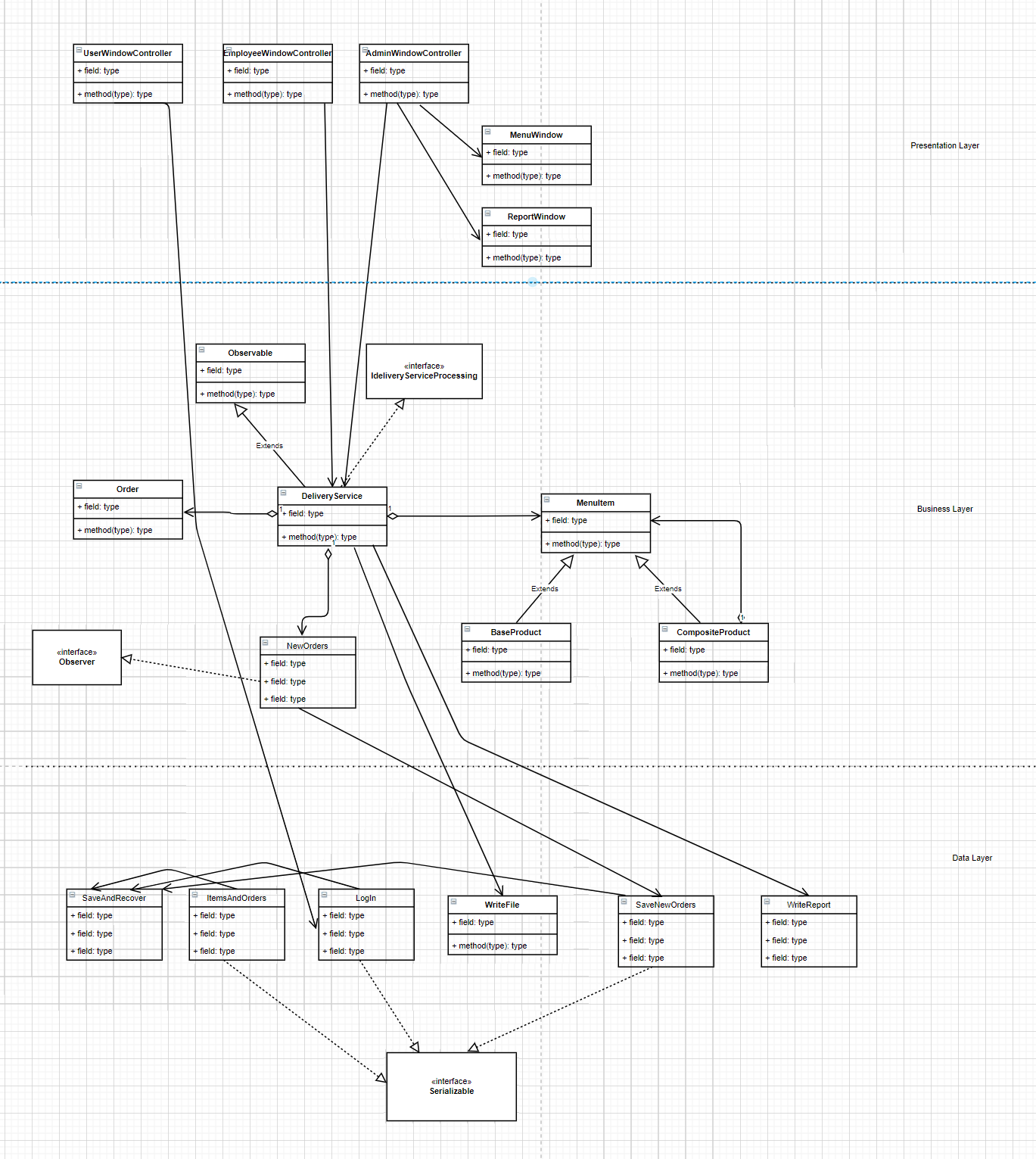


* Packages



* Classes

Data that must be retrieved and then saved before starting and ending an application is: the usernames and passwords, the orders and the menu items and the orders that have to be delivered by the employees. The classes containing this information are in the data layer:”SaveNewOrders”, “ItemsAndOrders”, “LogIn”.



1. **Implementation.**

The first thing an application for a restaurant must have is a list on menu items, with the menu names, prices and some more additional information about what the food contains. The second thing is the orders performed by the clients. Both types of data are used by the “DeliveryService” class. Every time a new instance of this class is created, in the constructor, the status of the class (when it was last time used) is restored via serialization. This way, menu items or orders added do not remain in the application only when the user keeps it open. They are saved in text file and changes made by one user will be seen by the following users. The class that is saved, restored and contains the menu items and orders is not in fact “DeliveryService”. “DeliverySerive” uses some getters and setters of the class “ItmesAndOrders” in order to keep this important data from use to use of the application. Because in the “DeliveryService” we have the orders and the menu items, this will also be the place where new items are being created, items modified, searched or deleted, orders added and used to generate reports or bills for the costumers. One menu item contains many characteristics and when a costumers wants to search for a specific item (like a menu item that has the rating at least 6 and price at least 40), it shouldn’t be mandatory to know all the other characteristics (like sodium, fat, protein,…). So the costumer fills only some specific text fields, the unused ones will be initialized by the application and if in the “DeliveryService” those values initialized values are found, that attribute is not considered when searched for the items the costumers requires. The correct functionality of this important class is assured by some pre conditions, post conditions and invariants. The invariant “isWellFormed()” checks if the data structures with the orders and menu items are not null, when this information is required by other classes.

“DeliveryService” is the subsect of the observer class “NewOrders”. Every time an order is performed, the “NewOrders” class is notified. This class uses getters and setters of the “SaveNewOrder” class in order to add the most recent orders, so that they can be displayed for the employee.

In the window for the employee, the new performed orders can be seen on the graphical interface and when the employee decides to deliver one of them, he has to enter the id of that order in the text so to delete it from “NewOrders” and other employees will not see them anymore.

In the case of the classes that implement the “Serializable” interface, when data needs to be recovered, this is done in the constructor of the class. So to recover data is enough to just instantiate the object. Saving the new modified object requires calling the method for saving. This method firstly calls a set method, to update the data in the class responsible for storing the data and then calling the “save” method of “SaveAndRecover”, where the serialization is done.

The “LogIn” class is responsible for storing the users of the application. It provides a method to search for a user and the search is done in the list corresponding to the admin, employees and regular costumers. This method returns a string describing the role of the user with the inserted username and password. This way, when the class managing the log in window uses this class, from the returned string, it can know if the user should be redirected to the window for the admin, employee or costumer. The “LogIn” class can also register a new user. All the usernames and passwords are saved through serialization.

The “SaveAndRecover” class contains static method responsible for saving and recovering the state of the classes that must be saved. Here can be found the names of the text files where the status of the objects are saved as bytes. The “save()” method takes 3 arguments corresponding to the 3 objects that have to saved, so the same “save()” method is called whenever one of those objects must be saved. Although, we sometimes need to save only one of those object, so if the “save()” method receives a null reference, the object corresponding to that null reference will not be saved.

The “MenuItem”, “BaseProduct” and “CompositeProduct” are defined using the Composite Design Patter. This design patterns makes creating menus very easy to implement.

The “Order” class is where information about the orders is stored. The fields refer to an order id, a client name for each order, a total price and the date and hour when the order was performed. The “Time” class is defined inside this “Order” class and stores the information regarding the date and hour. The collection with the orders must be saved when the application is closed, so the “Order” class itself must implement the “Serializable” interface. That collection is a HashMap so the “hashCode()” and “equals()” methods had to be overwrite. The orders are placed in the HashMap after it’s id field and two orders are considered to be equal when they have the same id, since orders with same costumer, price and date are possible to exist. The “Order” class requires a “toString()” method so that to display the orders when an employee has to perform a delivery.

The “GenerateReport” class is responsible to generate the reports the admin requires. Apart from the values specific to the report (like the day in which the admin wants to see what orders where performed), this class requires the map with the orders and the list with the menu items. All those static methods of the "GenerateReport” class rely on traversing the map with the orders and what items where ordered. In the case of displaying the menu items that were ordered more than a specified number, all the products ever ordered are added in a single list. Then that list is being traversed with two indexes to find duplicates. When the case of two products with the same name is met, one of them is deleted from the list. Also, I used an integer array where the value i-th value of the array corresponds to the i-th product in the list, so if array[5] = 10 it means that the 5th item in the list was found 10 times. Values in this integer array are incremented every time a duplicate is found. Since in a case of the duplicate we have to decide between deleting the first one that appears or the second one and increment in the integer array the value corresponding to the first or second of the duplicates, the first appearance is chosen to remain in the list and used in the integer array. After we have the list with the products ordered, each one appearing only once, and the array to used to see how many times a menu item has been ordered, we have to traverse this array one more time to see which products to write in the report, which products where ordered more than the number inserted by the administrator .The same strategy is used also to count the clients that have ordered more than a specified number of times. In the case showing orders performed between two specified hours, the values are converted from the format with hours and minutes (example 15:30) in only minutes ( minutes = 60\*15 + 30) since is easier to perform arithmetic operations.

1. **Conclusions.**

This application made me more comfortable to some design patters. I also learned about a new way to save and recover data, so that changes made by a user can be seen by other users. Apart from the last homework, when data was saved in databases, serialization and deserialization is easier to implement and use. Also, this homework introduced me to lambda expressions and streams, concepts that seem to be essential knowledge for a programmer.

1. **Links used.**