Technical University of Cluj-Napoca

Fundamental Programming Techniques

Laboratory – Assignment 3

Food Delivery Management System



Teacher: prof. Ioan Salomie

Teacher Assistant: Ciprian Adrian Stan

Student: Anca Itu

Group: 30422

1. Assignment objective

Design and implement a food delivery management system 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.

In the application the client should be able to register and use the registered username and password to log in within the system, view the list of products from the menu, search for products based on some criteria such as a keyword for the name, rating, number of calories, proteins, fats, sodium, price and order products from the company’s menu.

The administrator can import the initial set of products, manage the products from the menu by adding, deleting or modifying products, creating new products composed of several existing ones or he can generate reports about the performed orders considering some criteria: time interval of the orders, the products ordered more than a specified number of times so far, the clients that have ordered more than a specified number of times and the value of the order was higher than a specified amount, the products ordered within a specified day with the number of times they have been ordered.

The employee must be notified each time a new order is created by a client.

1. Problem analysis, modeling, scenarios, use cases

Problem analysis

Hash Table is a data structure used to implement an associative array with constant access time to its elements.

Hashing is a technique or process of mapping keys, values into the hash table by using a hash function. It is done for faster access to elements. The efficiency of mapping depends on the efficiency of the hash function used.

Hash Map class implements the Map interface which allows us to store key and value pair, where keys should be unique. It is easy to perform operations using the key index like updating or deletion.

Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.

Modelling

In the main stage there are 2 buttons: one for sign up and another one for log in. In the sign up stage the user must introduce a username and a password and select the type from combo box (administrator, employee, client). Then he is redirected to the window corresponding to the options characteristic for his type. In the log in stage the user has to introduce the same data as in the previous case and he is redirected from here to the window corresponding to his options.

In the administrator window there are several buttons the user can click on in order to import the initial set of products from the .csv file or to perform one of the following operations: add/delete/modify product, create new products, generate reports or view products. There is a window corresponding to each one of these operations in which the administrator can enter the data corresponding to the new item to be added to the menu, select item to be deleted from a combo box, change the content of the text fields corresponding to the item desired to be modified, select 2, 3 or 4 products in order to create a new daily menu, select to see the list of base products or the one containing the composite products (the daily menus) by clicking on the corresponding button. In case he chooses to generate reports, he is redirected to a window where he can choose the kind of report he wants to generate and then, according to his choice, a window containing some text fields or combo boxes will appear, each window including a text which explains the type of report and specifies the data that the user should enter in order to generate it accordingly to his desires, a “Generate report” button which, by clicking it, sets off the appearance of a listView containing the data corresponding to the selected criteria.

In the client window there are 3 buttons in the beginning: “View products”, “View menus” and “Create order”. By clicking the last one, other controls appear and the client can choose to search for a product or for a menu by clicking on the corresponding button. According to the made selection, he can introduce some desired data in the text fields that showed up and then by pressing “Search” a list containing the matching data will appear. If the user wants to select product from the list, he just has to click on the desired option and then it will also be displayed on the listView in the lower part of the window. He can delete options he doesn’t want anymore to be included in his order and then he can create the order by pressing on the “Create order” button, after visualizing the total price which updates each time the content of the order is changed and which is displayed in a textField. After the order is created, a confirmation message will appear and the client is informed that the employees were notified regarding his order, also through a confirmation alert box.

The employee window contains only a listView which displays the performed orders.

Use cases

A use case is a definition of a specific objective that the system needs to accomplish. The two types of use-cases can be described at an abstract level (business use-case) or at an implementation-specific level (system use-case).

The **business use-cases** in my project are “sign up “log in”, “choose one operation to perform”, “view the result of chosen operation”, “view the modified menu”.

The **system use-cases** are written at a lower level of detail than the business use case and refer to the specific processes that will be carried out by different parts of the system. In my project, such use-cases are “add product to the menu”, “delete product from the menu”, “modify product from the menu”, “view menu”, “create order”.

Scenarios

The main success scenario would be the following one:

-the user creates a new account or logs in the already existing one

in case of administrator or client:

-the user selects the operation he wants to perform

-introduces some valid data

-the corresponding result will be generated and he can see the effects of his operation

If the operations performed succeed, an appropriate message is displayed in a combo box. If not, an alert box with an error message is displayed and the user must change the filled in data in order to perform the desired operation.

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

Design decisions

The user has the possibility to log in his/her own account or to create an account if he/she doesn’t have one. After that, he is redirected to the window corresponding to his type. If he is an:

- administrator: the user can import the products which will populate the menu from the file .csv, add a new product, delete a product, modify a product, create a new menu consisting of 2, 3 or 4 base products, generate some reports and view the products or the menus. The reports are about the performed orders considering them (the orders), the products or the clients according to some specified criteria.

- client: the user can view the list of products or the list of menus and he can make an order; in order to create one, he can search one base product by name, rating, calories, protein, fat, sodium or price, or he can search one menu by name of some product from it or by the price of the entire menu. He has to select his options from the list and then, if he agrees with the total price which is displayed, he confirms his order and a bill is generated. He also can delete an option if he cahenges his mind and doesn’t want his order to contain it anymore.

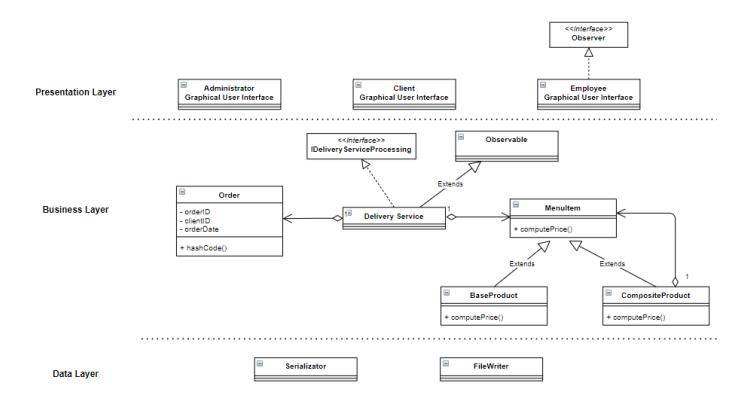
- employee: the employee is notified when a new order is done and he can see a list including all the orders made until then.

Data structures

As data structures, I used arrays of type List and ArrayList. Normal arrays have the disadvantage that they can store only a fixed set of elements and they don’t grow their size at the runtime. To solve this problem, I preferred to use ArrayList, which is a resizable-array implementation of the List interface. In some cases I had to use Lists because the data that I want to serialize or deserialize has to be put in List.

Class design

In order to design the classes, my logic was based on the following diagram:



I used Composite Design Pattern for modelling the classes MenuItem, BaseProduct and CompositeProduct because it is meant to allow treating individual objects and composite objects in the same way. I also used the Observer Design Pattern to notify the employees each time an order was performed because it is a behavioral design pattern and specifies communication between objects: observable (an object which notifies) and observer (an object which is notified about the changes made in the observable’s state).

Packages

My application includes classes organized in the following main packages: application, businessLayer, dataLayer and presentationLayer.

I used in my application a **layered architecture**. It is consistent across different projects and guarantees separation of concerns which leads to the possibility to organize very easily the code. The used layers are the following ones:

• Presentation layer – UI takes the action made by the user and sends it to the controller and at the end the result is taken from the controller and send to the UI;

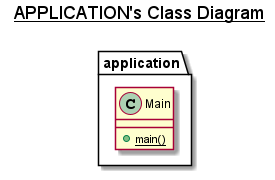
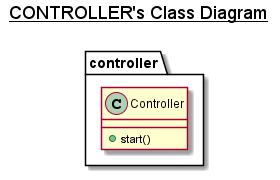
• Business logic – handles the application logic; it takes information from the user and sends it to the Data layer and vice versa.

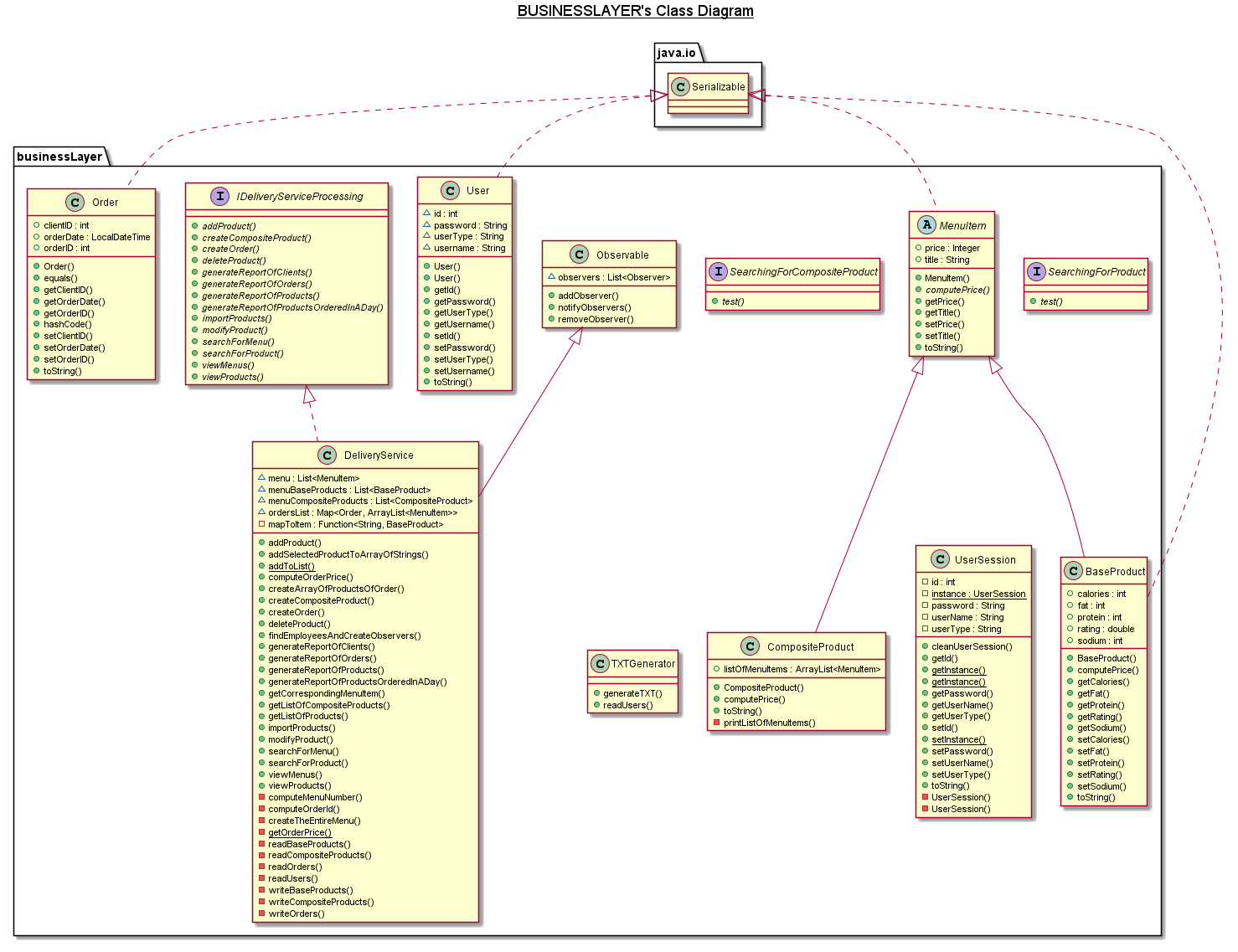
• Data layer – it is the layer which handles the storing of data, for example the products or the users’ data in this case; it contains the FileManager and the Serializator class

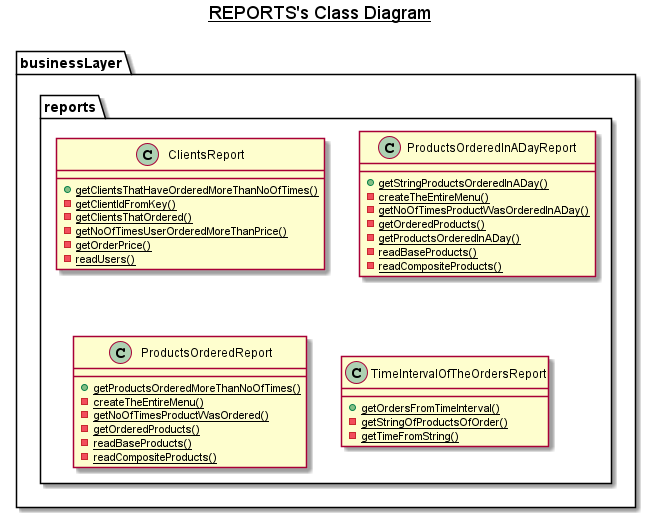
My project contains classes organized in packages the following way:

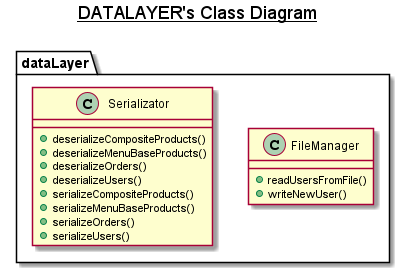
* application
* Main – here I create the object controller of the class Controller and I call the start() method from the Controller
* businessLayer
  + reports
    - ClientsReport
    - ProductsOrderedInADayReport
    - ProductsOrderedReport
    - TimeIntervalOfTheOrdersReport
* BaseProduct
* CompositeProduct
* IDeliveryService – this interface includes the main operations that can be executed by the user and it is implemented by the DeliveryService class
* DeliveryService
* MenuItem
* Observable
* Order
* SearchingForCompositeProduct
* TXTGenerator
* User
* UserSession
* dataLayer
  + FileManager
  + Serializator
* presentationLayer
  + fxmlFiles – package including the fxml files
  + AdministratorController
  + AddProductController
  + DeleteProductController
  + ModifyProductController
  + CreateNewProductController
  + GenerateReportsController
  + ViewProductsController
  + ClientController
  + EmployeeController
  + ListViewHandler
  + SignUp
  + LogIn
  + Observer
  + Window
  + InterfaceLoader
  + Controller
* txt

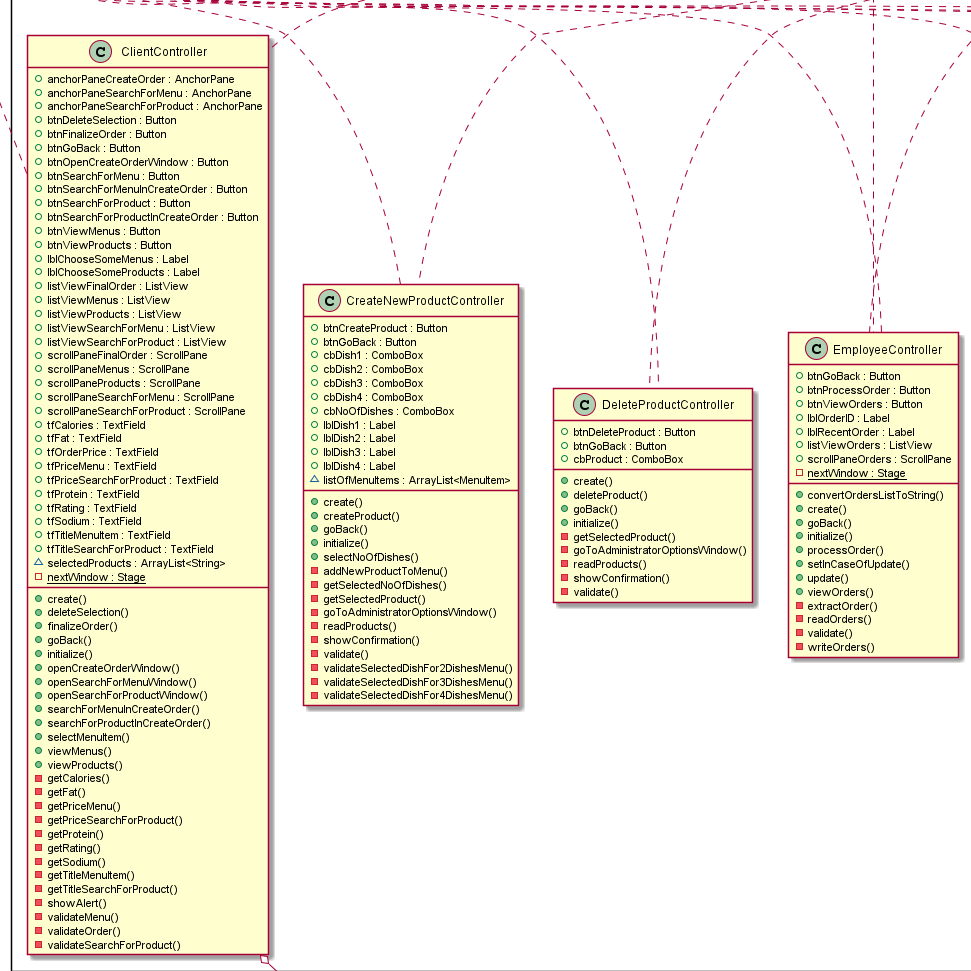
UML digrams

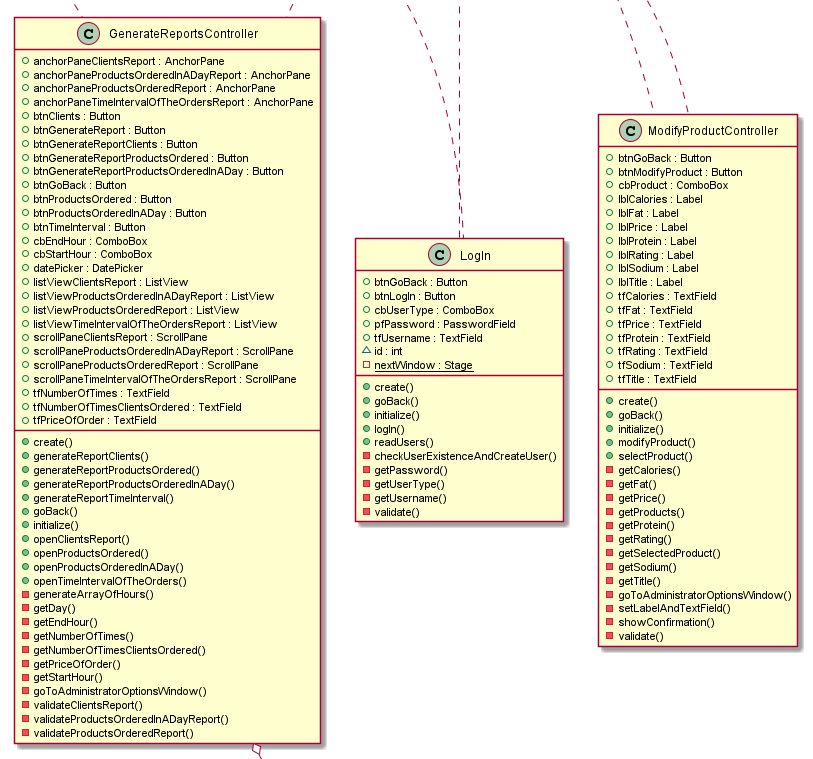
 

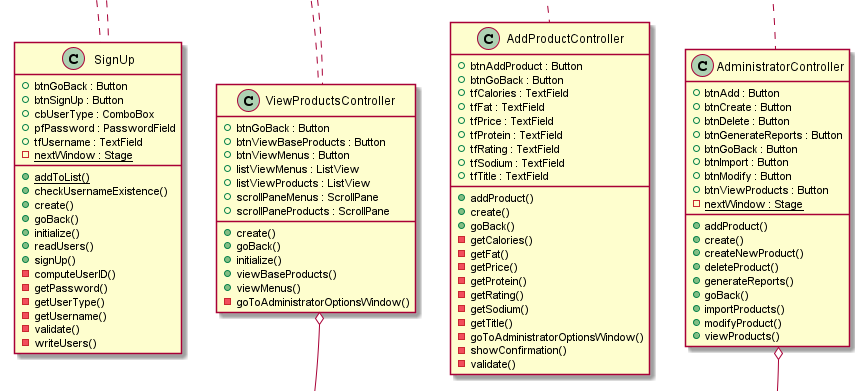


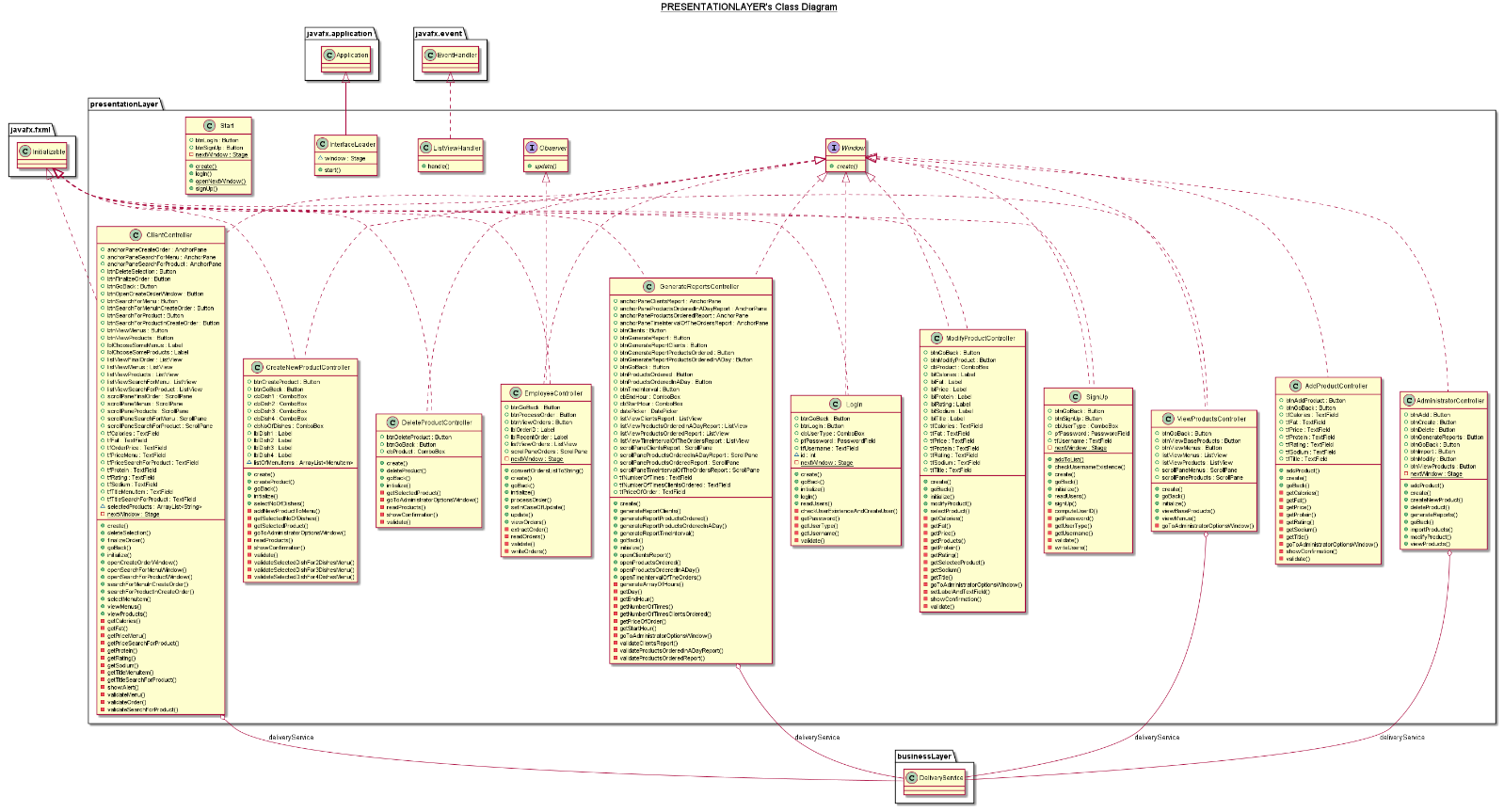












1. Implementation

• application

* Main – here I create the object controller of the class Controller and I call the start() method from the Controller
* businessLayer
  + reports
* ClientsReport

*public static List<String> getClientsThatHaveOrderedMoreThanNoOfTimes(int noOfTimes, int priceOfOrder, Map<Order, ArrayList<MenuItem>> ordersList)*

*private static List<User> getClientsThatOrdered(Map<Order, ArrayList<MenuItem>> ordersList)*

*private static int getNoOfTimesUserOrderedMoreThanPrice(int priceOfOrder, Map<Order, ArrayList<MenuItem>> ordersList, User user)*

*private static int getOrderPrice(ArrayList<MenuItem> listOfProducts)*

*private static int getClientIdFromKey(Order order)*

* ProductsOrderedInADayReport

*public static List<String> getStringProductsOrderedInADay(LocalDate day, Map<Order, ArrayList<MenuItem>> ordersList)*

*private static List<MenuItem> getProductsOrderedInADay(LocalDate day, Map<Order, ArrayList<MenuItem>> ordersList)*

*private static int getNoOfTimesProductWasOrderedInADay( MenuItem product, LocalDate day, Map<Order, ArrayList<MenuItem>> ordersList)*

*private static List<MenuItem> getOrderedProducts(Map<Order, ArrayList<MenuItem>> ordersList)*

* ProductsOrderedReport

*public static List<String> getProductsOrderedMoreThanNoOfTimes(int noOfTimes, Map<Order, ArrayList<MenuItem>> ordersList)*

*private static int getNoOfTimesProductWasOrdered(Map<Order, ArrayList<MenuItem>> ordersList, MenuItem product)*

*private static List<MenuItem> getOrderedProducts(Map<Order, ArrayList<MenuItem>> ordersList)*

* TimeIntervalOfTheOrdersReport

*public static List<String> getOrdersFromTimeInterval(String startHour, String endHour, Map<Order, ArrayList<MenuItem>> ordersList)*

*private static String getStringOfProductsOfOrder(ArrayList<MenuItem> productsOfOrder)*

*private static LocalDateTime getTimeFromString(String stringHour)*

* BaseProduct
* *Constructor, setters and getters, toString method*
* CompositeProduct
* *Constructor, setters and getters, toString method*
* IDeliveryService

*void importProducts();*

*void addProduct(String title, double rating, int calories, int protein, int fat, int sodium, int price);*

*void deleteProduct(BaseProduct productToDelete);*

*void modifyProduct(BaseProduct selectedProduct, String title, double rating, int calories, int protein, int fat, int sodium, int price);*

*void createCompositeProduct(ArrayList<MenuItem> listOfMenuItems);*

*List<String> generateReportOfOrders(String startHour, String endHour);*

*List<String> generateReportOfProducts(int noOfTimes);*

*List<String> generateReportOfClients(int noOfClients, int amountOfOrder);*

*List<String> generateReportOfProductsOrderedInADay(LocalDate day);*

*ArrayList<String> viewProducts();*

*ArrayList<String> viewMenus();*

*List<String> searchForProduct(String title, double rating, int calories, int protein, int fat, int sodium, int price);*

*List<String> searchForMenu(String itemTitle, int price);*

*void createOrder(ArrayList<MenuItem> selectedProducts);*

* DeliveryService
* Mainly, the methods defined in the implemented interface and also the following ones, which are used to call the serialization or deserialization methods (they are also used in other classes, according to the needed data):

*private List<BaseProduct> readBaseProducts(){*

*return new Serializator().deserializeMenuBaseProducts(); }*

*private void writeBaseProducts(){*

*new Serializator().serializeMenuBaseProducts(menuBaseProducts); }*

*private List<CompositeProduct> readCompositeProducts(){*

*return new Serializator().deserializeCompositeProducts(); }*

*private void writeCompositeProducts(){*

*new Serializator().serializeCompositeProducts(menuCompositeProducts); }*

*private HashMap<Order, ArrayList<MenuItem>> readOrders(){ return new Serializator().deserializeOrders(); }*

*private void writeOrders(){ new Serializator().serializeOrders(ordersList); }*

*private ArrayList<User> readUsers(){ return new Serializator().deserializeUsers(); }*

* MenuItem
* *Constructor, setters and getters, toString method*
* Observable

*public void addObserver(Observer observer);*

*public void removeObserver(Observer observer);*

*public void notifyObservers(Object ordersList)*

* Order

*public Order(int orderID, int clientID, LocalDateTime orderDate)*

*public int hashCode()*

*public boolean equals(Object o)*

* *Constructor, setters and getters, toString method*
* SearchingForCompositeProduct

*boolean test(CompositeProduct product);*

* TXTGenerator

*public void generateTXT(Order order, int orderPrice, ArrayList<MenuItem> products)*

* User
* *Constructor, setters and getters, toString method*
* UserSession
* *Constructor, setters and getters, toString method*
* dataLayer
* FileManager

*public void writeNewUser(User user)*

*public ArrayList<User> readUsersFromFile()*

* Serializator

*public void serializeUsers(ArrayList<User> users)*

*public ArrayList<User> deserializeUsers()*

*public void serializeMenuBaseProducts(List<BaseProduct> menuBaseProducts)*

*public List<BaseProduct> deserializeMenuBaseProducts()*

*public void serializeCompositeProducts(List<CompositeProduct> menuCompositeProducts)*

*public List<CompositeProduct> deserializeCompositeProducts()*

*public void serializeOrders(Map<Order, ArrayList<MenuItem>> ordersList)*

*public HashMap<Order, ArrayList<MenuItem>> deserializeOrders()*

* presentationLayer
* fxmlFiles – package including the fxml files
* AdministratorController
* AddProductController
* DeleteProductController
* ModifyProductController
* CreateNewProductController
* GenerateReportsController
* ViewProductsController
* ClientController
* EmployeeController
* ListViewHandler
* SignUp
* LogIn
* Observer – interface implemented by EmployeeController class
* Window – interface implemented by all the controller classes
* It declares the *create* method used to open a new window
* InterfaceLoader
* Controller
* Most of the Controller classes contain methods for user input validation, initialization in case of combo boxes or list views, create method used to open the corresponding window and actions on the buttons;
* Txt
* This package includes the generated bills

1. Results

The modifications made on the data are not lost but applied to the content of each file representing a byte stream. These files are managed through serialization and deserialization. Each time the content of such a file is needed, it is deserialized and when the modifications are made, the updated data is serialized, so it is written back into the file by replacing the old content. If a product is added, deleted or modified, the updated data is write into the file. If an order is made, and is serialized as well, a bill is generated and the employees are notified.

1. Conclusions

In conclusion, this application provides the user with an intelligible way to manage the products from the menu or to generate some reports if he is an admin, to make an order if he is a client and to see the made orders in case the user is an employee.

1. Bibliography

[https://www.baeldung.com](https://www.baeldung.com/java-composite-pattern#:~:text=The%20composite%20pattern%20is%20meant,a%20whole%20hierarchy%20of%20objects).

[https://dzone.com/articles](https://dzone.com/articles/layered-architecture-is-good#:~:text=What%20Is%20Layered%20Architecture%3F,the%20particular%20concern%20it%20represents).

<https://searchsqlserver.techtarget.com>

<https://www.geeksforgeeks.org/hashing-data-structure/>

<https://www.geeksforgeeks.org/serialization-in-java/>

<file:///D:/Facultate/An%202/Sem%202/PT/Laboratory/Assignment4/ASSIGNMENT_4_SUPPORT_PRESENTATION.pdf>