



TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FUNDAMENTAL PROGRAMMING TECHNIQUES
ASSIGNMENT 4

FOOD DELIVERY MANAGEMENT SYSTEM

Student: Iordan Diana

Contents

1. Assignment Objective.....	3
Sub-objectives.....	3
2. Problem analysis, modeling, scenarios, use cases	3
Problem analysis	3
Use Cases.....	4
Level 1 : Overall system design.....	8
Level 2: Division into packages.....	8
Level 3: Division into classes	9
Class diagramaam.....	10
GUI	11
4. Implementation.....	13
5. Result.....	15
6. Conclusions	17
7. Bibliography	17

1. Assignment Objective

The main objective of this assignment is to design and implement a food delivery management system with a dedicated graphical user interface through which the users can perform the needed operations.

Sub-objectives

- Analyze the problem and identify requirements.
- Design and implement the operations for administrator.
- Design and implement the operations for clients.
- Design and implement the operations for regular employee.

2. Problem analysis, modeling, scenarios, use cases

Problem analysis

The purpose of the project is to solve the following problem: managing the orders and make the communication between client, admin and employee faster in a catering company. Old ways of ordering food may take a lot of time and can result in different types of misunderstanding such as getting the wrong order or taking too much time to take a call and process the information.

Solution: As a solution, the project provides a faster way managing the whole process of food delivery, using a virtual system, making it easier for the administrator to manage the products and offering a better communication between the employee and client.

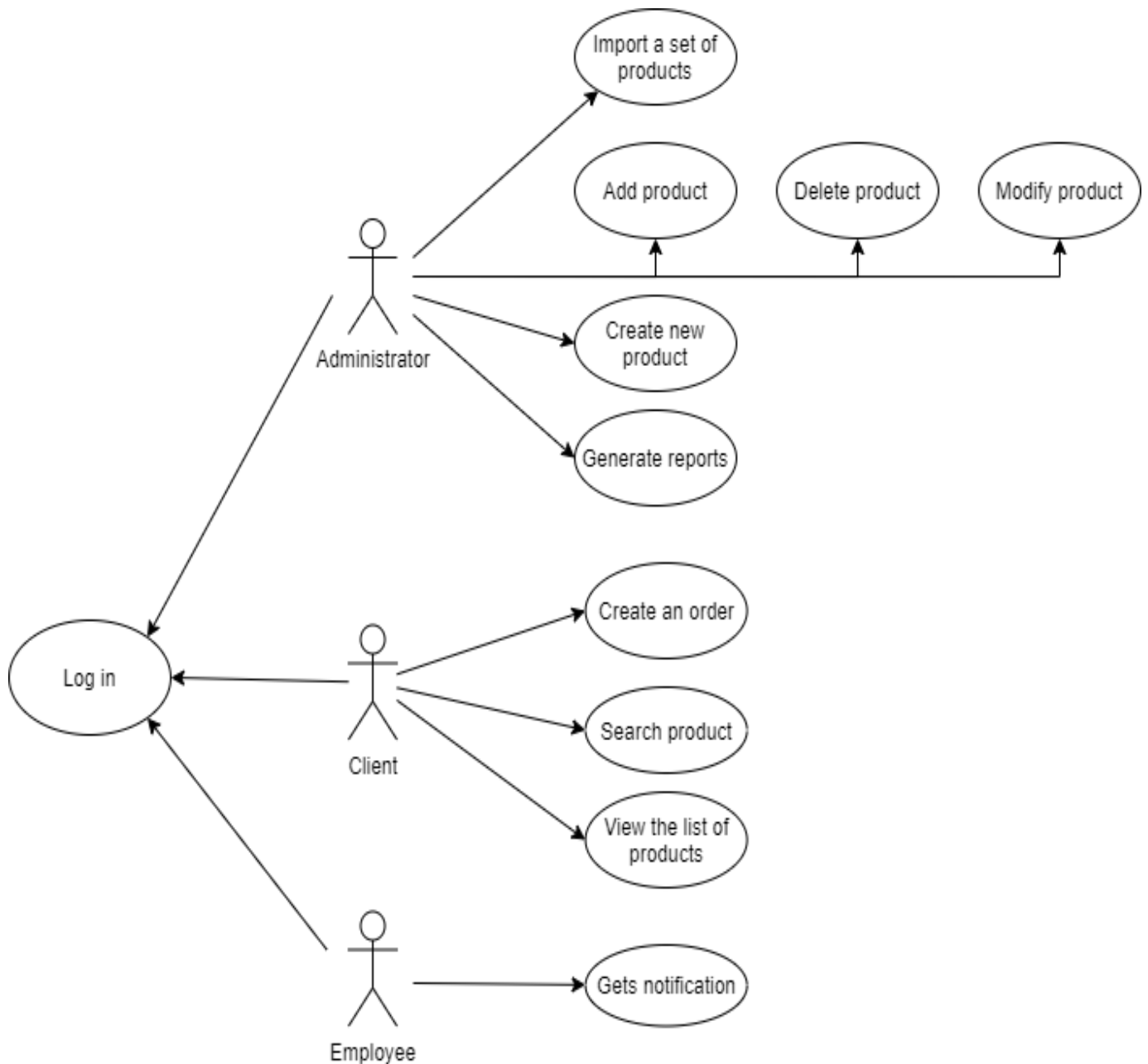
Functional requirements:

- The food delivery management system should allow importing a .csv file.
- The food delivery management system should allow logging in for a specific type of user.
- The food delivery management system should allow the administrator to add/delete/modify products.
- The food delivery management system should allow the administrator to create new products.
- The food delivery management system should allow the client to view a list of products.
- The food delivery management system should allow the client to search for products based on one or multiple criteria.
- The food delivery management system should allow the client to create an order consisting of several products.
- The food delivery management system should be able to generate a bill with the list of products ordered and the total price of the order.

- The food delivery management system should notify the employee when a new order is performed by a client so it can prepare the delivery of the ordered dishes.

Use Cases

Use Case Diagram:



Use case: Log in

Main actors: Administrator, Client, Employee

Main successful scenario:

1. The user introduces the username.
2. The user introduces the password.
3. The application will open the window corresponding to the type of user that logged in.

Alternative sequence:

1. The user introduces an inexistent username.
2. The password and the username do not match.

Use case: Import products

Main actor: Administrator

Main successful scenario:

1. The admin introduces his credentials.
2. The admin selects "Import products" button.
3. The admin introduces the path of the .csv file.
4. The system imports the products.

Alternative sequence:

1. The path is not correct.

Use case : add product;

Primary actor: admin;

Main success scenario:

1. The admin selects "Manage product" from the operation field.
2. The admin completes the necessary fields for inserting a product.
3. The admin finishes the action and presses the button for insertion.
4. The application displays a message for successfully inserting a product.

Alternative sequence:

- The user inserts invalid input (e.g. Invalid price)
- The scenario returns to the first step.

Use case : delete product;

Primary actor: admin;

Main success scenario:

1. The admin selects the "Manage product" field from the main frame.
2. The admin inserts the id for the product to be deleted.
3. The admin presses the delete button and deletes the entry.

Alternative sequence:

- The scenario returns to the first step.
- The user inserts an invalid id

Use case : edit product;

Primary actor: admin;

Main success scenario:

1. The admin selects the “Manage product” field from the main frame.
2. The admin inserts the id for the product to be edited.
3. The admin inserts the fields to be modified.
4. The admin selects the “Edit product” button.

Alternative sequence:

- The user inserts an invalid input.
- The scenario returns to the first step.

Use case : view products

Primary actor: user;

Main success scenario:

1. The admin selects the “Manage product” field from the main frame.
2. The admin chooses the view products operation.
3. The application shows the table with all the entries from the field.

Alternative sequence:

1. The scenario returns to the first step.

Use case : add new product;

Primary actor: admin;

Main success scenario:

1. The admin selects the “Manage product” field from the main frame.
2. The admin completes the id field with the ids from the products that he want to add for the composite product.
3. The admin finishes the action and presses the button for insertion.

Alternative sequence:

- The user inserts invalid input (e.g. Invalid id)
- The scenario returns to the first step.

Use case: Add order

Primary actor: user;

Main success scenario:

1. The user selects the order operations field from the main frame.

2. The user chooses one client from the dropdown menu.
3. The user chooses one product from the other dropdown menu.
4. The user places the order.
5. The app generates a bill for the specific order.

Alternative sequence:

2. The scenario returns to the first step.
3. The user tries to place an order with an invalid quantity.
4. The user tries to place an order with a quantity bigger than what is left in stock.

Use case: generate reports

Primary actor: admin

Main success scenario:

1. The admin selects the “Manage product” field from the main frame.
2. The admin selects the “Generate reports” section.
3. The admin introduces the input in order to generate some reports.
4. The reports are generated.

Alternative sequence:

- The admin introduces an invalid input.

Use case: order

Primary actor: client

Main success scenario:

1. Client selects the item he wants to order.
2. The orders are placed in the shopping cart.
3. The client selects the button for finishing the order.

Alternative sequence:

- Client does not want a product he selected and wants to remove it from the cart.
- Wrong button.

Use case: gets notified

Primary actor: Employee

Main success scenario:

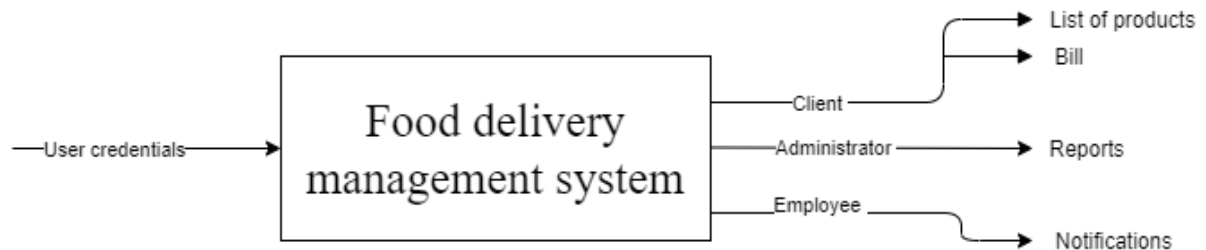
1. Employee logs in.
2. Employee gets notified when an order is placed.

Alternative sequence:

- Wrong credentials.

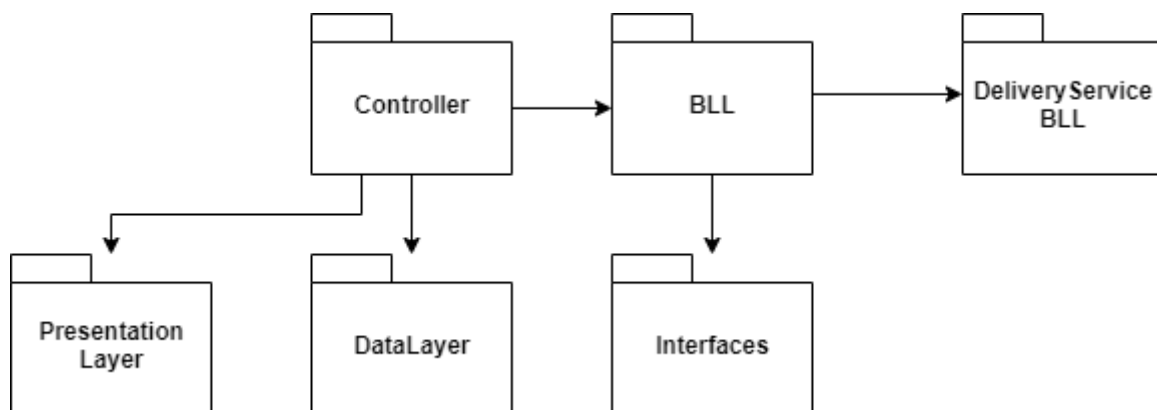
3. Design

Level 1 : Overall system design



The application will have as input the credentials from different type of users such as employee, client, administrator and each of them will be able to perform some specific operations which will generate different outputs regarding the user. For client, as an output a list of products and a bill will be generated, for the administrator as an output some reports will be generated, as for the employee the output is represented by the notifications which will appear in the app.

Level 2: Division into packages



Level 3: Division into classes

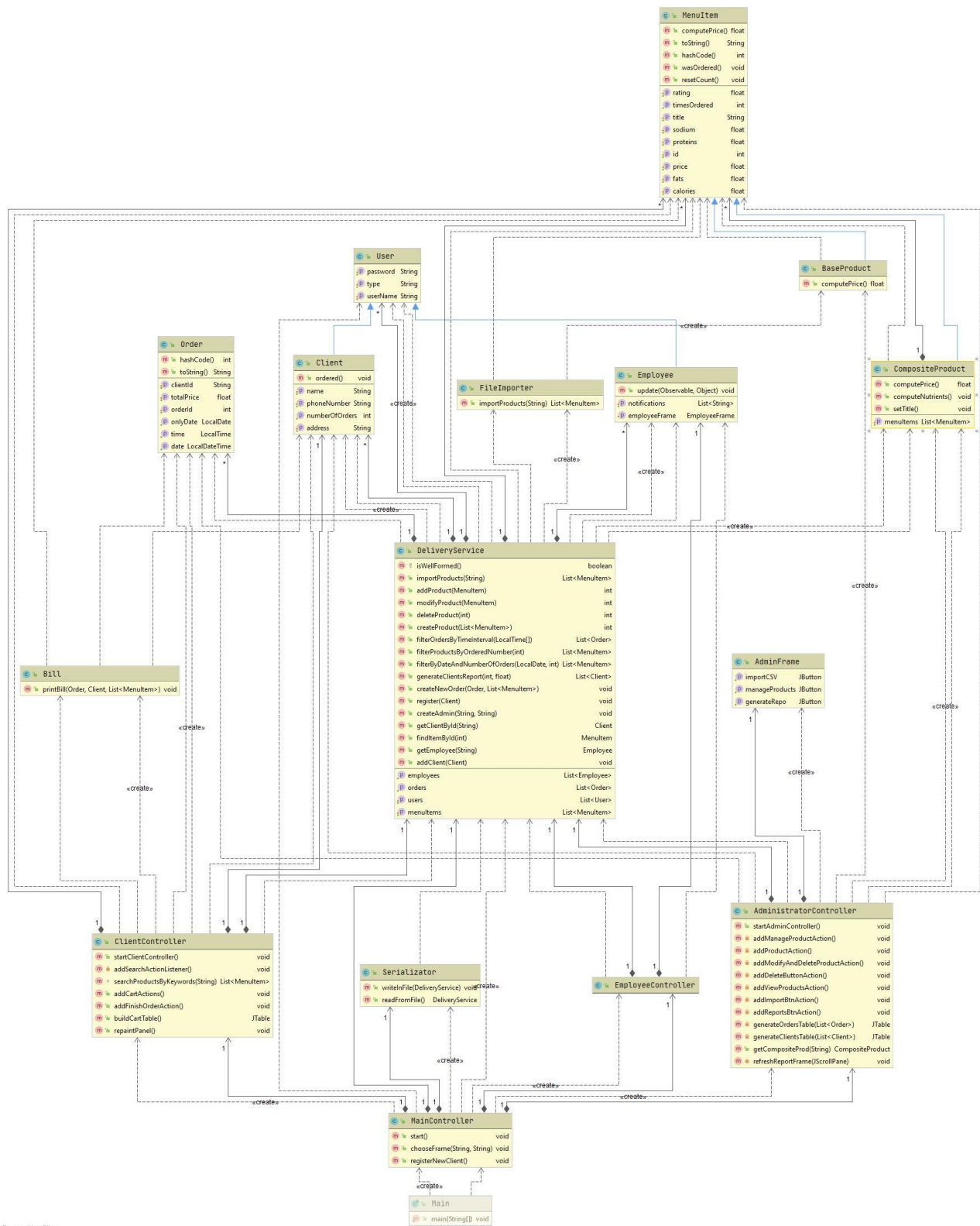
The DeliveryService package – contains the main classes: DeliveryService, Client, BaseProduct, MenuItem, Order, Employee, User and Order. The DeliveryService class plays the main role in managing the objects. It stores the main information that will be worked with. The User class will store the credentials for each type of users. The class is extended by Employee and Client which also contain different fields outside the one contained by the User class. The classes BaseProduct, MenuItem and CompositeProduct respect the CompositeDesignPattern such that MenuItem is an abstract class which is extended by the other two types of products. The Order contains the clientId which is similar to the username, the date and the orderId. The items ordered will be connected using an HashMap.

The presentation package – contains the classes which implement the graphical user interface. Each class is represented by a JFrame and plays a specific role for the application. The main frame, named LogInFrame is formed from a multiple buttons which will eventually lead to the other frames: ClientFrame, AdministratorFrame, EmployeeFrame. .

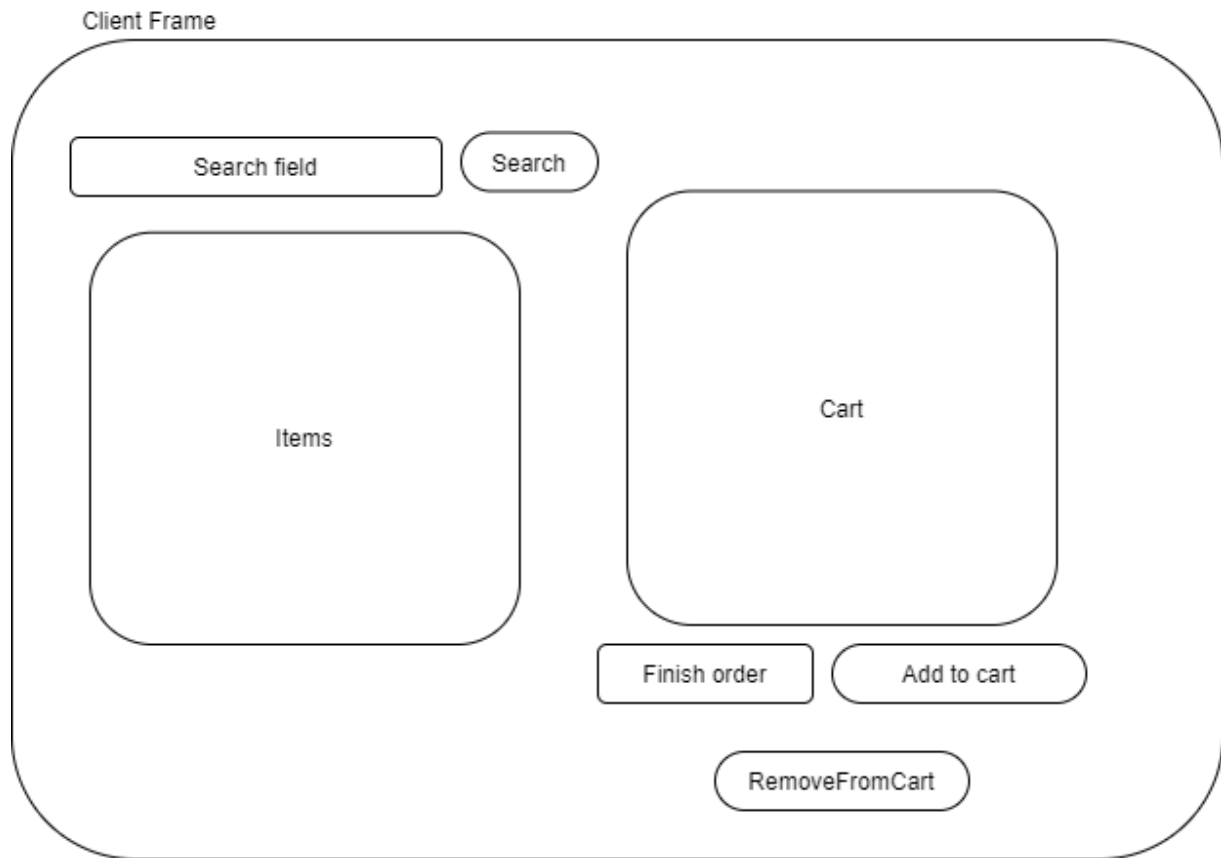
The Data Layer package is used mainly for serialization and working with the data. It contains the following classes: Bill, FileImporter, MenuItemTableModel and Serializer. The Bill is the object generated after an order was placed, the Serializer is an object which helps the delivery service to save the state of the application and the FileImporter is an object which helps reading from the .csv file.

The Controller package implements controller for each frame. The administratorController adds action listener for each button from the mainframe and also for the other frames that are created when the buttons are pressed. The EmployeeController connects the EmployeeFrame and the Objects and updates the frame each time there is an update. The mainController starts each controller when it is needed.

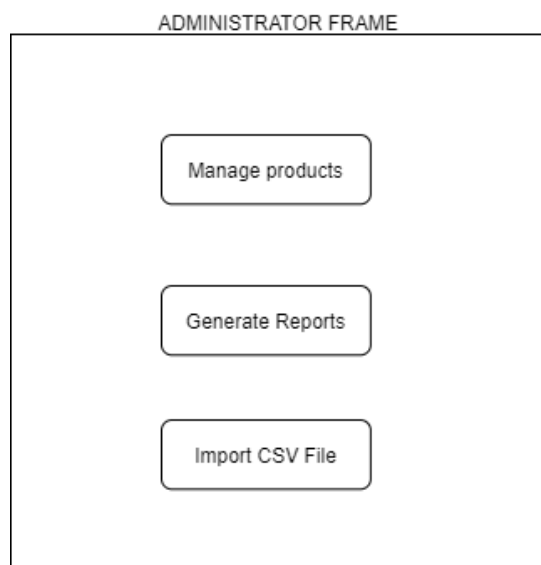
Class diagram



GUI



The interface for the client is formed from two panels, one of them representing the items to be viewed and the other one being the items from the cart. There is also a searching field which will help finding items using a specific keyword. The buttons are performing operations on the items from the cart.



The administrator frame is pretty simple and has three main buttons that will open new frames for performing different operations.

Administrator Frame Mock Up

A mockup of an administrator frame. It features a large rounded rectangle labeled 'TEXT FIELDS' in the top left. To its right is a large rectangle labeled 'TABLE'. Below the text fields, there are five buttons arranged in two columns: 'Add Product', 'Add Composite Product', 'Modify Product', 'Import CSV File', 'Delete Product', and 'Edit Product'.

The frame for managing products is simple. It has a panel for viewing the specific items needed and multiple buttons for performing the needed operations.

Employee Frame

A mockup of an employee frame. It consists of a large rounded rectangle. Inside, at the top, is a horizontal rectangle labeled 'Name:: Employee Name'. Below this is a large rounded rectangle labeled 'Notifications'.

The employee frame contains only a panel that will show the notifications received when an order is placed.

4. Implementation

For implementation the following patterns were considered: Composite Design Pattern, Observer Pattern and Design by Contract.

For the Design By Contract pattern, there was implemented a “well formed” method for checking the invariants.

```
protected boolean isWellFormed() {  
  
    ArrayList<Integer> idList = new ArrayList<>();  
    ArrayList<Integer> menuIds = new ArrayList<>();  
    ArrayList<String> userIds = new ArrayList<>();  
    for(Order order : orderListMap.keySet()){  
        if(!idList.contains(order.getOrderID())){  
            idList.add(order.getOrderID());  
        }else{  
            return false;  
        }  
    }  
    for(MenuItem menuItem : menuItems){  
        if(!menuIds.contains(menuItem.getId())){  
            menuIds.add(menuItem.getId());  
        }else{  
            return false;  
        }  
    }  
    for(User user : users){  
        if(!userIds.contains(user.getUserName())){  
            userIds.add(user.getUserName());  
        }else {  
            return false;  
        }  
    }  
    return true;  
}
```

The method verifies if every user, order or menuItem has an unique id.

The Observer Pattern with the help of the classes Observable and Observer. The Employee implements Observer while the DeliveryService extends Observable. Those two communicate with the help of the methods notifyObserver() and update() which were implemented to be called when an order is finished. The update method will add a new notification to the list of notifications from the Employee's attributes.

```
@Override  
public void update(Observable o, Object arg) {  
    notifications.add((String) arg);  
    if(employeeFrame != null){  
        employeeFrame.updatePanel(notifications);  
    }  
}
```

The DeliveryService object is keeping its state using serialization. The Serializator object implements two methods which read and write into a filestream,

```

public DeliveryService readFromFile() throws IOException, ClassNotFoundException
{
    inputStream = new ObjectInputStream(new FileInputStream(fileName));
    return ((DeliveryService) inputStream.readObject());
}

public void writeInFile(DeliveryService object) throws IOException {
    outputStream = new ObjectOutputStream(new FileOutputStream(fileName));
    outputStream.writeObject(object);
    outputStream.close();
}

```

The reading is performed each time the application is loaded. The main controller loads the object maintaining its state even after closing. Before closing the app the new state of the deliveryService object is loaded in the file. This is used by adding a window listener.

```

if (JOptionPane.showConfirmDialog(logInFrame, "Are you sure you want to close
this window?", "Close Window?",
    JOptionPane.YES_NO_OPTION,
    JOptionPane.QUESTION_MESSAGE) == JOptionPane.YES_OPTION) {
    try {
        serializator.writeInFile(deliveryService);
        logInFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    } catch (IOException e) {
        e.printStackTrace();
    }
} else {
    logInFrame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
}

```

For generating the reports, the items were filtered using streams. The reports were filtered following different criteria such as :

- 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.

```

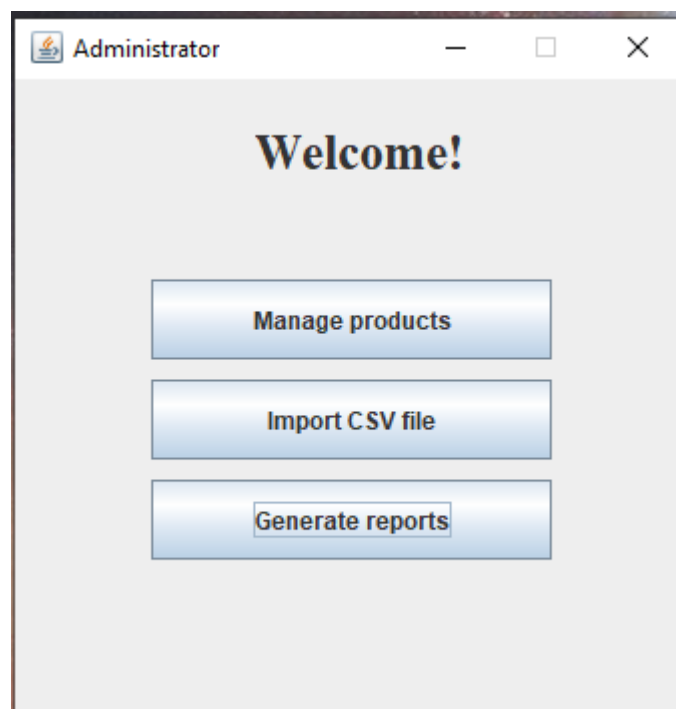
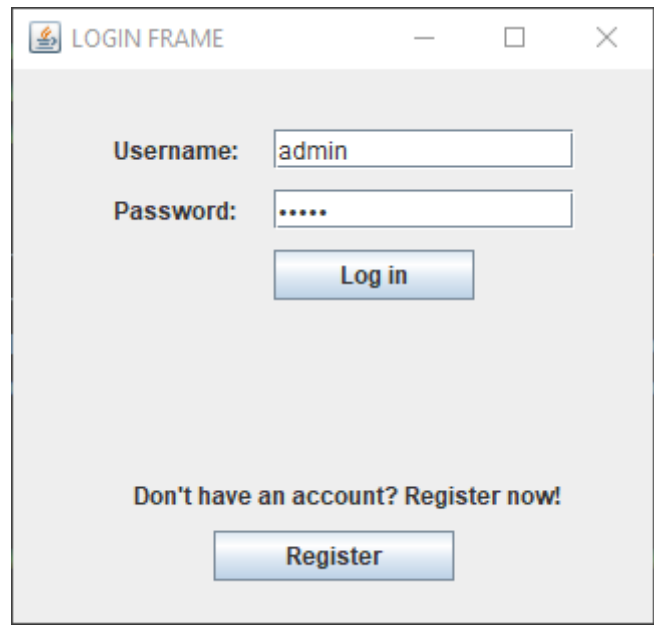
public List<MenuItem> filterByDateAndNumberOfOrders(LocalDate date, int
numberOfTimes) {
    List<MenuItem> filteredList = new ArrayList<>();
    List<MenuItem> result = new ArrayList<>();
    orderListMap.entrySet().stream().filter(entry->
entry.getKey().getOnlyDate().isAfter(date)).forEach(e->
e.getValue().stream().filter(k->k.getTimesOrdered() >=
numberOfTimes).forEach(k->filteredList.add(k)));
    for(MenuItem m : filteredList){
        if(!result.contains(m)){
            result.add(m);
        }
    }
    return result;
}


```

```
public List<MenuItem> filterProductsByOrderedNumber(int numberOfOrders){  
    List<MenuItem> filteredList = new ArrayList<>();  
    filteredList = menuItems.stream().filter(m->m.getTimesOrdered() >=  
numberOfOrders).collect(toList());  
    return filteredList;  
}
```

5. Result

The username and password have to be inserted and then the corresponding window will be displayed.




Proceeding order

Search for products...

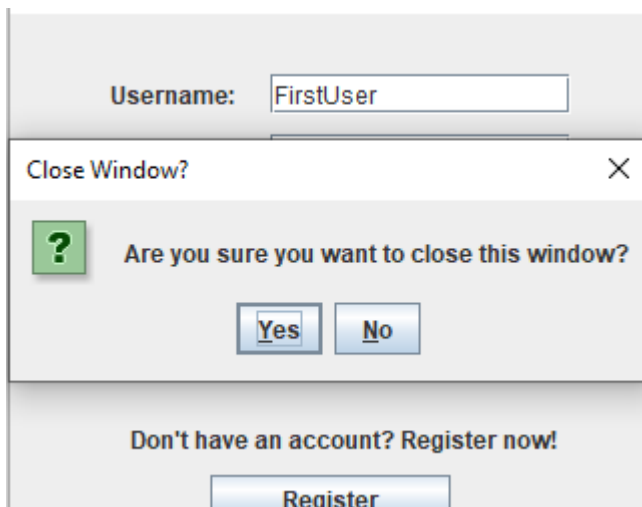
soup

Search

Added to cart

id	title	rating	calories	protein	fat	sodium	price	numbe...
3479	"Butter...	0	200	9	11	1,195	10	0
5603	"Mixed ...	5	273	17	6	821	45	0
12951	"Vietna...	3.75	1,084	101	54	338	75	0
8685	"Wont...	3.125	435	16	18	2,059	62	0
3757	3-Ingr...	3.75	209	4	16	684	86	0
6377	Albond...	1.875	302	27	16	555	34	0
10713	Anasa...	4.375	602	49	22	1,871	71	0
6703	Apple ...	3.75	314	7	12	186	15	0
13696	Aroma...	0	1,951	28	33	1,779	73	0
1106	Asian ...	3.75	98	8	2	1,091	79	0
4912	Asian ...	3.75	248	23	5	867	95	0
2266	Aspar...	4.375	154	8	8	323	17	0
2024	Aspar...	4.375	144	4	10	541	39	0
3549	Aspar...	4.375	203	13	9	294	21	0
6770	Aspar...	4.375	317	11	21	252	38	0
5253	Autum...	3.75	260	7	12	268	30	0
4147	Avoca...	3.75	223	3	20	28	49	0
3562	Avoca...	3.125	203	7	16	264	10	0
10056	Bacon ...	4.375	537	22	26	674	60	0
7858	Barbe...	3.75	381	18	9	556	66	0
6657	Barley ...	3.75	312	16	7	2,715	32	0
10720	Beef B...	4.375	603	17	41	630	36	0
7781	Beef a...	4.375	376	22	19	682	80	0
2531	Beet S...	3.75	165	2	14	370	81	0
13056	Beet a...	3.75	1,139	18	106	86	40	0

[illegible]



The question window is generated in order to offer the application to serialize the object.

6. Conclusions

- The application proves the utility of the Design Patterns and also the use of the invariants. It is a good example for proving how a real-life type of situation can be implemented using different patterns.
- The Observer Pattern can be used in order to update the GUI in real time.
- Streams in java can help in organizing and filtering different structures in a clean way.

7. Bibliography

- https://en.wikipedia.org/wiki/Composite_pattern
- <https://www.geeksforgeeks.org/stream-in-java/>
- <http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html>
- <http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html>