DOCUMENTATION

PROJECT 4

NAME OF STUDENT: Jakab-Gyik Sarolta
GROUP: 30423

TABLE OF CONTENTS

1. Objectives

Main objective:

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

Sub-objectives:

- Analyze the problem and identify requirements
- Design the orders management application
- Implement the orders management application
- Test the orders management application

2. Problem analysis, modelling, scenarios, cases of utility

Functional requirements:

The application should allow the administrator to add a new product to the menu
The application should allow the administrator to delete a product from the menu
The application should allow the administrator to modify a product from the menu
The application should allow the client to order products
The application should allow the client to search for products
The application should allow the client view all the menu items
The application should allow the employee to view all orders
The application should allow the employee to delete the orders that have been prepared

Use Case: <u>add product to the menu</u>
Primary Actor: administrator
Main Success Scenario:

1. The administrator selects the option to add a new product
2. The application will display a form in which the product details should be inserted
3. The administrator inserts the name of the product, the number of calories, proteins, fats, sodium and its price
4. The administrator clicks on the "Add" button
5. The application saves the product's data and displays an acknowledge message

Alternative Sequence: Invalid values for the product's data
- The administrator inserts a negative value for the number of calories/proteins/fats/sodium/price of the product
- The application displays an error message and requests the administrator to insert a valid value for the number of calories/proteins/fats/sodium/price
- The scenario returns to step 3

Use Case: <u>delete product from the menu</u>
Primary Actor: administrator
Main Success Scenario:

1. The administrator selects the option to delete a Product
2. The application will display a form in which the product's name should be inserted
3. The administrator inserts the name of the product
4. The administrator clicks on the "Delete" button
5. The application deletes the product's data and displays an acknowledge message

Alternative Sequence: Invalid values for the product's name
- The administrator inserts a nonvalid product name
- The application displays an error message and requests the administrator to insert a valid name
- The scenario returns to step 3

Use Case: <u>modify product to the menu</u>
Primary Actor: administrator
Main Success Scenario:

1. The administrator selects the option to modify a product

2. The application will display a form in which the product details should be inserted
3. The administrator inserts the name of the product, the number of calories, proteins, fats, sodium and its price
4. The administrator clicks on the "Modify" button
5. The application saves the product's data and displays an acknowledge message

Alternative Sequence: Invalid values for the product's title
- The administrator inserts a negative value for the number of calories/proteins/fats/sodium/price of the product
- The application displays an error message and requests the administrator to insert a valid value for the number of calories/proteins/fats/sodium/price
- The scenario returns to step 3

Use Case: create composite product
Primary Actor: administrator
Main Success Scenario:
1. The administrator selects the option to create a composite product
2. The application will display a form in which the product details should be inserted
3. The administrator inserts the name of the product, the number of calories, proteins, fats, sodium and its price
4. The administrator clicks on the "Create" button
5. The application saves the product's data and displays an acknowledge message

Alternative Sequence: Invalid values for the product's fields
- The administrator inserts a negative value for the number of calories/proteins/fats/sodium/price of the product
- The application displays an error message and requests the administrator to insert a valid value for the number of calories/proteins/fats/sodium/price
- The scenario returns to step 3

Use Case: generate report
Primary Actor: administrator
Main Success Scenario:
1. The administrator selects the type of the report, then the option to generate the report
2. The application will display a form in which the details should be inserted
3. The administrator inserts the given parameters for the report
4. The administrator clicks on the "Generate" button
5. The application searches through the products list and writes the report to a .txt file

Alternative Sequence: Invalid values for the report parameters
- The administrator inserts a negative value for the parameters
- The application displays an error message and requests the administrator to insert a valid value for the parameters
- The scenario returns to step 3

Use Case: search in the menu
Primary Actor: client
Main Success Scenario:
1. The client inserts the name of the product, the number of calories, proteins, fats, sodium and its price, as many parameters as he would like to.
2. The administrator clicks on the "Search" button
3. The application displays in the table the products that correspond to the criteia

Alternative Sequence: Invalid values for the search options
- The client inserts a negative value for the number of calories/proteins/fats/sodium/price of the product
- The application displays an error message and requests the employee to insert a valid value for the number of calories/proteins/fats/sodium/price
- The scenario returns to step 1

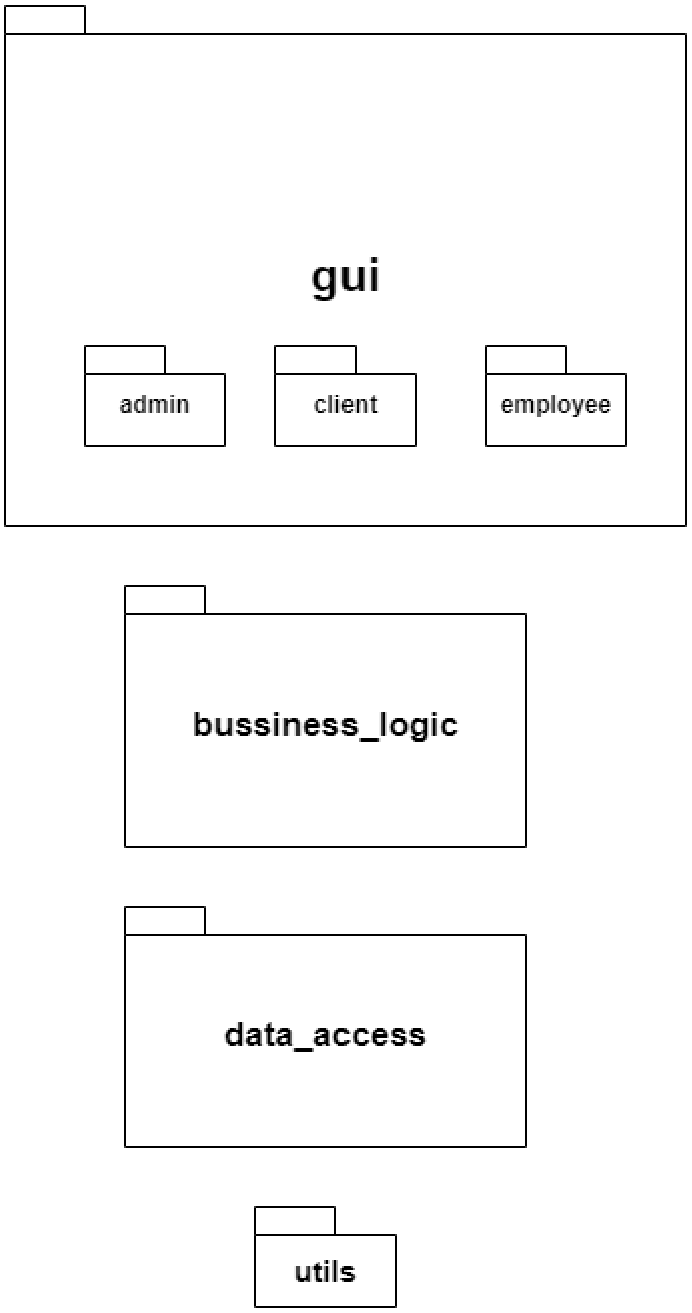
Use Case: place an order
Primary Actor: client
Main Success Scenario:

1. The client selects on of the menu items and the quantity of the item
2. Clicks on the “Add” button and the items is added to the current order
3. The first 2 steps can be repeated until all the products were added to the order
4. The client clicks on the “Create” button
5. The “Bill.txt” file gets generated with the details of the created order
6. The application displays a message to the user in case of successful creation of the order

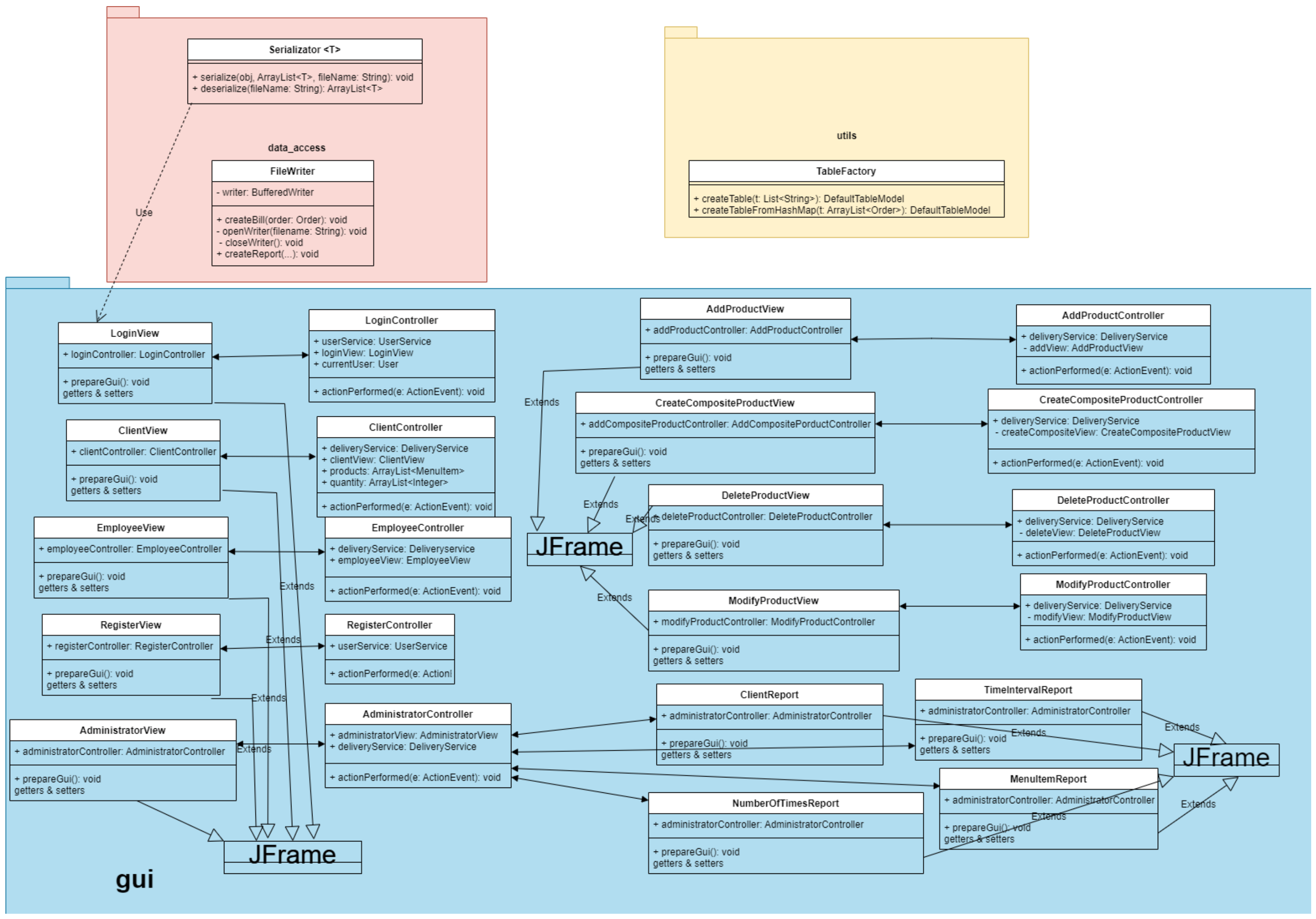
Use Case: delete an order
Primary Actor: employee
Main Success Scenario:

1. The employee is able to see all the orders placed by the users
2. If an order has been prepared, the employee can click on the delete button
3. The application updates the current orders and displays the new table on the screen

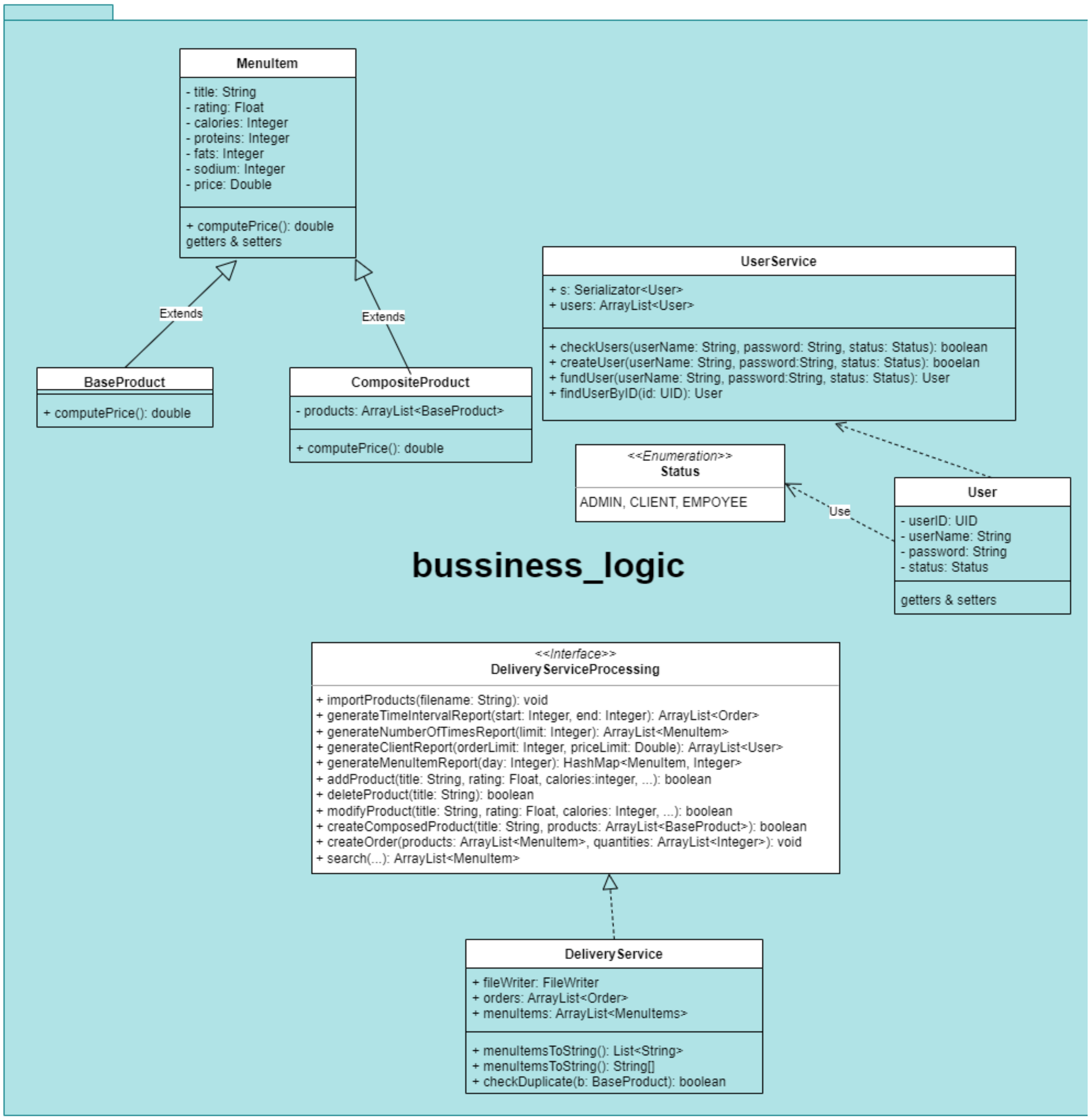
Alternative Sequence: Delete when there is no order in the waiting list
- The employee gets notified that there are no more orders, wait until an order gets placed
- The scenario returns to step 1



~ Package diagram~



Serializator <T>
+ serialize(obj, ArrayList<T>, fileName: String): void
+ deserialize(fileName: String): ArrayList<T>
data_access
FileWriter
- writer: BufferedWriter
+ createBill(order: Order): void
- openWriter(filename: String): void
- closeWriter(): void
+ createReport(...): void
Use
utils
TableFactory
+ createTable(t: List<String>): DefaultTableModel
+ createTableFromHashMap(t: ArrayList<Order>): DefaultTableModel
LoginView
+ loginController: LoginController
+ prepareGui(): void
getters & setters
LoginController
+ userService: UserService
+ loginView: LoginView
+ currentUser: User
+ actionPerformed(e: ActionEvent): void
ClientView
+ clientController: ClientController
+ prepareGui(): void
getters & setters
ClientController
+ deliveryService: DeliveryService
+ clientView: ClientView
+ products: ArrayList<MenuItem>
+ quantity: ArrayList<Integer>
+ actionPerformed(e: ActionEvent): void
EmployeeView
+ employeeController: EmployeeController
+ prepareGui(): void
getters & setters
EmployeeController
+ deliveryService: Deliveryservice
+ employeeView: EmployeeView
+ actionPerformed(e: ActionEvent): void
RegisterView
+ registerController: RegisterController
+ prepareGui(): void
getters & setters
RegisterController
+ userService: UserService
+ actionPerformed(e: ActionI
AdministratorView
+ administratorController: AdministratorController
+ prepareGui(): void
getters & setters
AdministratorController
+ administratorView: AdministratorView
+ deliveryService: DeliveryService
+ actionPerformed(e: ActionEvent): void
Extends
Extends
Extends
Extends
JFrame
gui
AddProductView
+ addProductController: AddProductController
+ prepareGui(): void
getters & setters
AddProductController
+ deliveryService: DeliveryService
- addView: AddProductView
+ actionPerformed(e: ActionEvent): void
CreateCompositeProductView
+ addCompositeProductController: AddCompositePorductController
+ prepareGui(): void
getters & setters
CreateCompositeProductController
+ deliveryService: DeliveryService
- createCompositeView: CreateCompositeProductView
+ actionPerformed(e: ActionEvent): void
Extends
Extends
Extends
JFrame
DeleteProductView
+ deleteProductController: DeleteProductController
+ prepareGui(): void
getters & setters
DeleteProductController
+ deliveryService: DeliveryService
- deleteView: DeleteProductView
+ actionPerformed(e: ActionEvent): void
Extends
ModifyProductView
+ modifyProductController: ModifyProductController
+ prepareGui(): void
getters & setters
ModifyProductController
+ deliveryService: DeliveryService
- modifyView: ModifyProductView
+ actionPerformed(e: ActionEvent): void
ClientReport
+ administratorController: AdministratorController
+ prepareGui(): void
getters & setters
TimeIntervalReport
+ administratorController: AdministratorController
+ prepareGui(): void
getters & setters
Extends
Extends
JFrame
MenuItemReport
+ administratorController: AdministratorController
+ prepareGui(): void
getters & setters
Extends
Extends
NumberOfTimesReport
+ administratorController: AdministratorController
+ prepareGui(): void
getters & setters


bussiness_logic
MenuItem
- title: String
- rating: Float
- calories: Integer
- proteins: Integer
- fats: Integer
- sodium: Integer
- price: Double
+ computePrice(): double
getters & setters
Extends
Extends
BaseProduct
+ computePrice(): double
CompositeProduct
- products: ArrayList<BaseProduct>
+ computePrice(): double
UserService
+ s: Serializator<User>
+ users: ArrayList<User>
+ checkUsers(userName: String, password: String, status: Status): boolean
+ createUser(userName: String, password:String, status: Status): booelan
+ fundUser(userName: String, password:String, status: Status): User
+ findUserByID(id: UID): User
<<Enumeration>>
Status
ADMIN, CLIENT, EMPOYEE
Use
User
- userID: UID
- userName: String
- password: String
- status: Status
getters & setters
<<Interface>>
DeliveryServiceProcessing
+ importProducts(filename: String): void
+ generateTimeIntervalReport(start: Integer, end: Integer): ArrayList<Order>
+ generateNumberOfTimesReport(limit: Integer): ArrayList<MenuItem>
+ generateClientReport(orderLimit: Integer, priceLimit: Double): ArrayList<User>
+ generateMenuItemReport(day: Integer): HashMap<MenuItem, Integer>
+ addProduct(title: String, rating: Float, calories:integer, ...): boolean
+ deleteProduct(title: String): boolean
+ modifyProduct(title: String, rating: Float, calories: Integer, ...): boolean
+ createComposedProduct(title: String, products: ArrayList<BaseProduct>): boolean
+ createOrder(products: ArrayList<MenuItem>, quantities: ArrayList<Integer>): void
+ search(...): ArrayList<MenuItem>
DeliveryService
+ fileWriter: FileWriter
+ orders: ArrayList<Order>
+ menuItems: ArrayList<MenuItems>
+ menuItemsToString(): List<String>
+ menuItemsToString(): String[]
+ checkDuplicate(b: BaseProduct): boolean

~Class diagrams~

3. Description

Design and implement a food delivery management system for a catering company. The client can order products from the company's menu. 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 which will populate the menu from a .csv file.

• Manage the products from the menu: add/delete/modify products and create new products composed of several products from the menu (an example of composed product could be named “daily menu 1” composed of a soup, a steak, a garnish, and a dessert).

• Generate reports about the performed orders considering the following criteria: o time interval of the orders – a report should be generated with the orders performed between a given start hour and a given end hour regardless the date. o the products ordered more than a specified number of times so far. o the clients that have ordered more than a specified number of times so far and the value of the order was higher than a specified amount. o the products ordered within a specified day with the number of times they have been ordered.

The client can:

• 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 one or multiple criteria such as keyword (e.g., “soup”), rating, number of calories/proteins/fats/sodium/price.

• Create an order consisting of several products – for each order the date and time will be persisted and a bill will be generated that will list the ordered products 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.

4. Implementation

About every class in details:

Business-logic package

BaseProduct

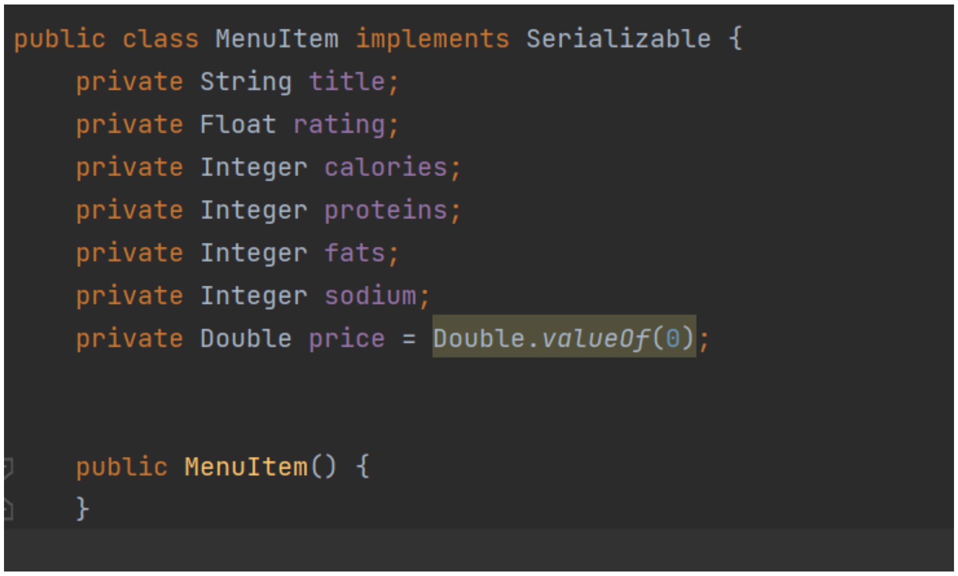
This class extends the MenuItem class and provides a constructor for all the fields declared in its superclass. It also overrides the conputePrice() method.

CompositeProduct

This class extends the MenuItem class and has a private field of arrayList type, where the BaseProducts are stored, that belong to the same composite product. Therefore, the client can order a daily menu for example, and get multiply base products in a single menu item.
It provides getter and setter methods, and overrides the computePrice() method of its superclass.

MenuItem

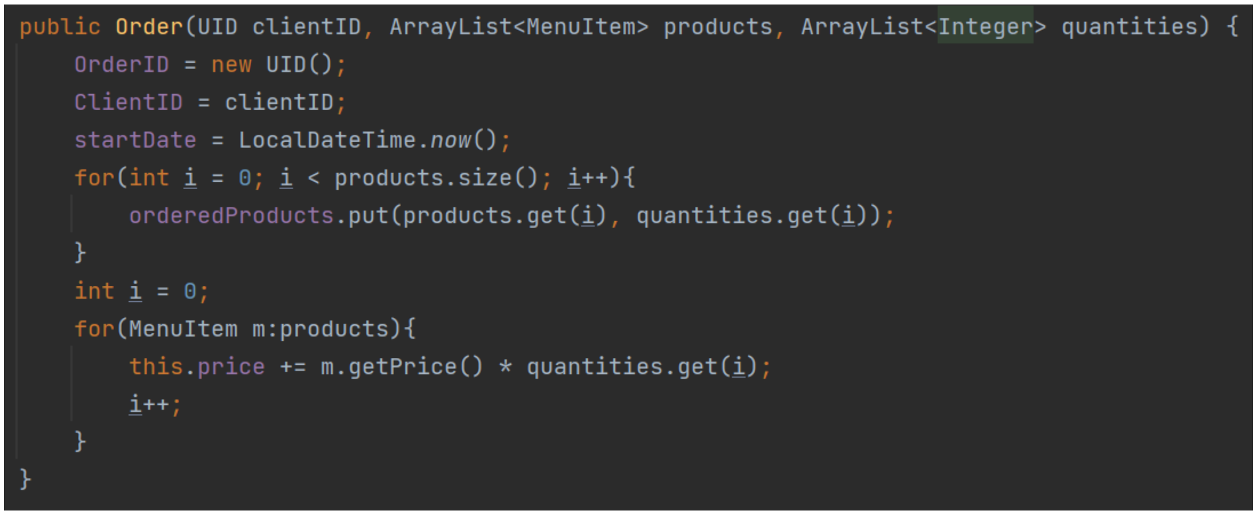
The super class of BaseProduct and CompositeProduct. It has private fields that are instantiated by the subclasses’ constructors. This class provides only an empty constructor. This class implements the Serializable interface, there the MenuItems before closing the application, will be serialized and every time the application is run, they are desrialized. Therefore, the information that was imported from the file is saved after closing the app.



```
public class MenuItem implements Serializable {
    private String title;
    private Float rating;
    private Integer calories;
    private Integer proteins;
    private Integer fats;
    private Integer sodium;
    private Double price = Double.valueOf(0);


    public MenuItem() {
    }
```

Order

This class has an unique ID field, for that I have used the UID class. Every order has a client ID associated to it, which is the current ID that is saved every time a user logs into the application. The hash map structure is used for the association between a menu item and the quantity selected by the user. It also provides getter and setter methods. This class also implements Serializable, therefore, the orders get saved until an employee deletes them, until they get finalized.



```
public Order(UID clientID, ArrayList<MenuItem> products, ArrayList<Integer> quantities) {
    OrderID = new UID();
    ClientID = clientID;
    startDate = LocalDateTime.now();
    for(int i = 0; i < products.size(); i++){
        orderedProducts.put(products.get(i), quantities.get(i));
    }
    int i = 0;
    for(MenuItem m:products){
        this.price += m.getPrice() * quantities.get(i);
        i++;
    }
}
```

User

This class has an unique ID field, for that I have used the UID class. Every user has a username and a password associated to it. The status field shows if the user is an admin/client/employee. It shows different user interfaces for all of these three categories. This class also provides getter and setter methods.
This class implements Serializable, therefore, the orders get saved until an employee deletes them, until they get finalized.

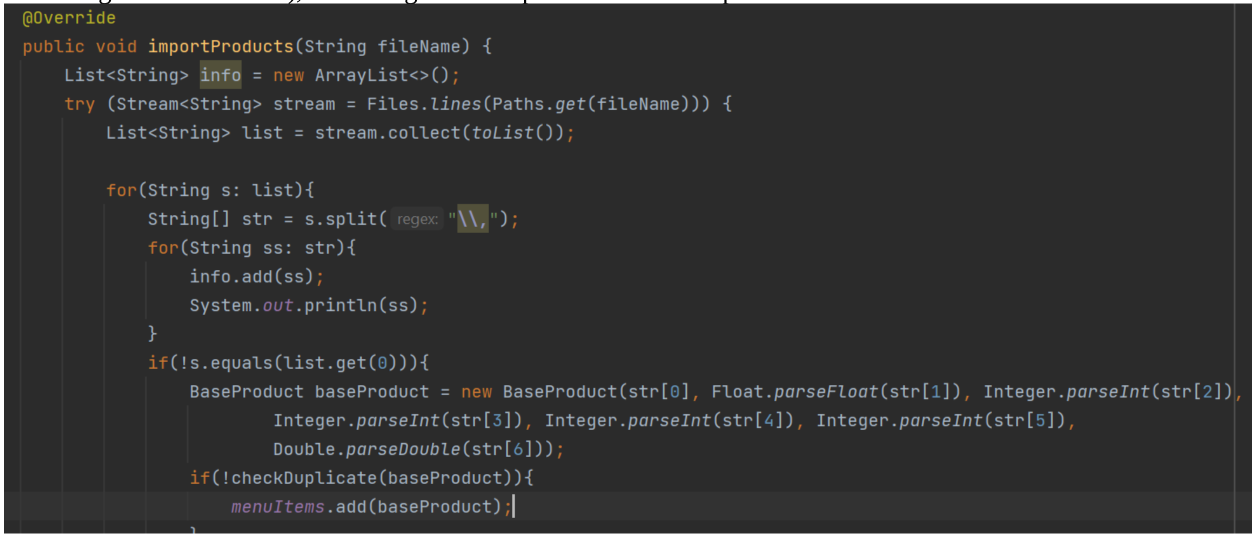
UserService

The UserService class provides methods to manipulate user data. It creates new users, checks for duplicates, if a user is already registered, it can find a user by their ID or by their username and password.
The users are saved in the static array list field of this class.

DeliveryService

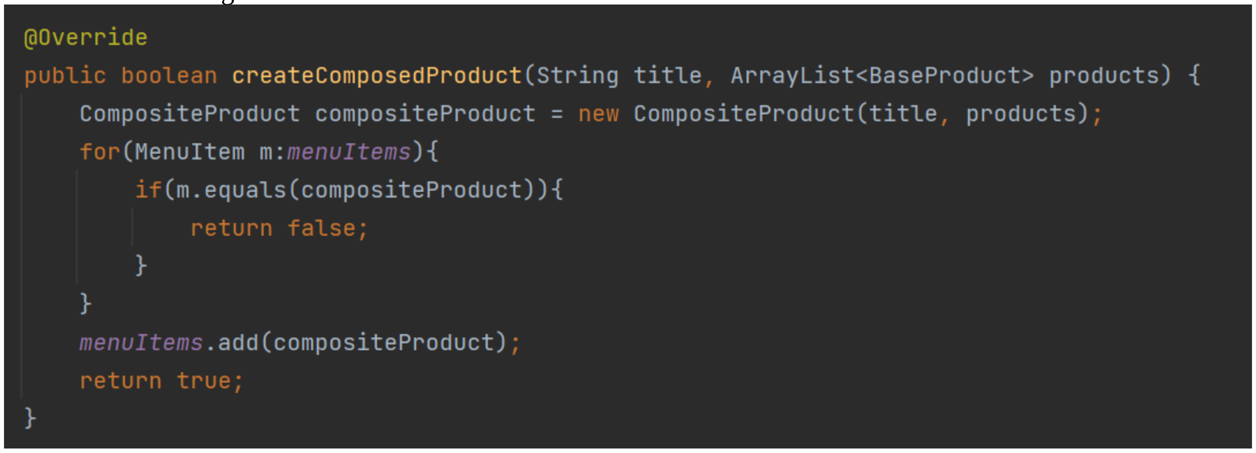
This class implements the DeliveryServiceProcessing interface It provides an implementation for all the methods that manipulate the data for the administrator and the client.

For the administrator: they can add a new base product into the menu, delete any base or composite product from the menu, modify the details of the products, import products from a file(I have hardcoded the given “products.csv” file with the given menu items), and also generate reports based on the previous orders.



```java
@Override
public void importProducts(String fileName) {
    List<String> info = new ArrayList<>();
    try (Stream<String> stream = Files.lines(Paths.get(fileName))) {
        List<String> list = stream.collect(toList());

        for(String s: list){
            String[] str = s.split( regex: "\\,");
            for(String ss: str){
                info.add(ss);
                System.out.println(ss);
            }
            if(!s.equals(list.get(0))){
                BaseProduct baseProduct = new BaseProduct(str[0], Float.parseFloat(str[1]), Integer.parseInt(str[2]),
                        Integer.parseInt(str[3]), Integer.parseInt(str[4]), Integer.parseInt(str[5]),
                        Double.parseDouble(str[6]));
                if(!checkDuplicate(baseProduct)){
                    menuItems.add(baseProduct);
```

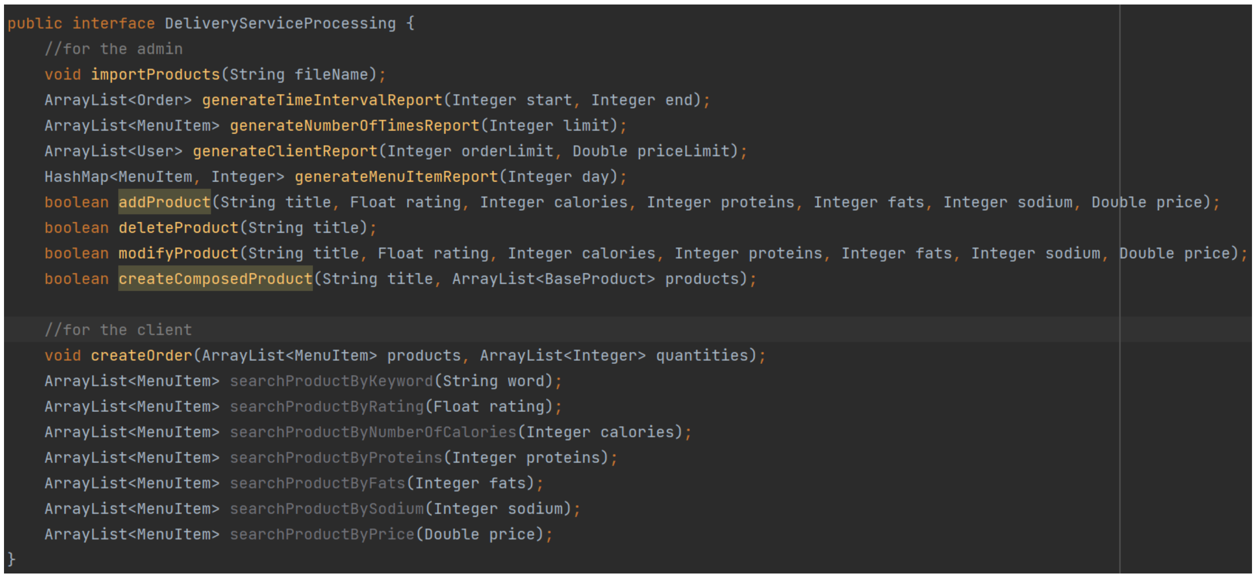
The importing of the products is done by streams. It also checks for duplicates before adding a new item to the menu. The generating of the reports uses the technique of streams as well.
Another method in this class is the createComposedProduct method, which instantiates a new menu item of type composite product. It gets an array list of base products and adds it to the array list field of a composite product. Then the new items gets added to the menuItem list.



```java
@Override
public boolean createComposedProduct(String title, ArrayList<BaseProduct> products) {
    CompositeProduct compositeProduct = new CompositeProduct(title, products);
    for(MenuItem m:menuItems){
        if(m.equals(compositeProduct)){
            return false;
        }
    }
    menuItems.add(compositeProduct);
    return true;
}
```

DeliveryServiceProcessing

This interface has all the methods’ signatures that the administrator and the client will need. It defines all the methods such as generate report, search by different fields, import products and others alike. The DeliveryService class implements this interface. That is why their names show the relationship between them.



```
public interface DeliveryServiceProcessing {
    //for the admin
    void importProducts(String fileName);
    ArrayList<Order> generateTimeIntervalReport(Integer start, Integer end);
    ArrayList<MenuItem> generateNumberOfTimesReport(Integer limit);
    ArrayList<User> generateClientReport(Integer orderLimit, Double priceLimit);
    HashMap<MenuItem, Integer> generateMenuItemReport(Integer day);
    boolean addProduct(String title, Float rating, Integer calories, Integer proteins, Integer fats, Integer sodium, Double price);
    boolean deleteProduct(String title);
    boolean modifyProduct(String title, Float rating, Integer calories, Integer proteins, Integer fats, Integer sodium, Double price);
    boolean createComposedProduct(String title, ArrayList<BaseProduct> products);

    //for the client
    void createOrder(ArrayList<MenuItem> products, ArrayList<Integer> quantities);
    ArrayList<MenuItem> searchProductByKeyword(String word);
    ArrayList<MenuItem> searchProductByRating(Float rating);
    ArrayList<MenuItem> searchProductByNumberOfCalories(Integer calories);
    ArrayList<MenuItem> searchProductByProteins(Integer proteins);
    ArrayList<MenuItem> searchProductByFats(Integer fats);
    ArrayList<MenuItem> searchProductBySodium(Integer sodium);
    ArrayList<MenuItem> searchProductByPrice(Double price);
}
```

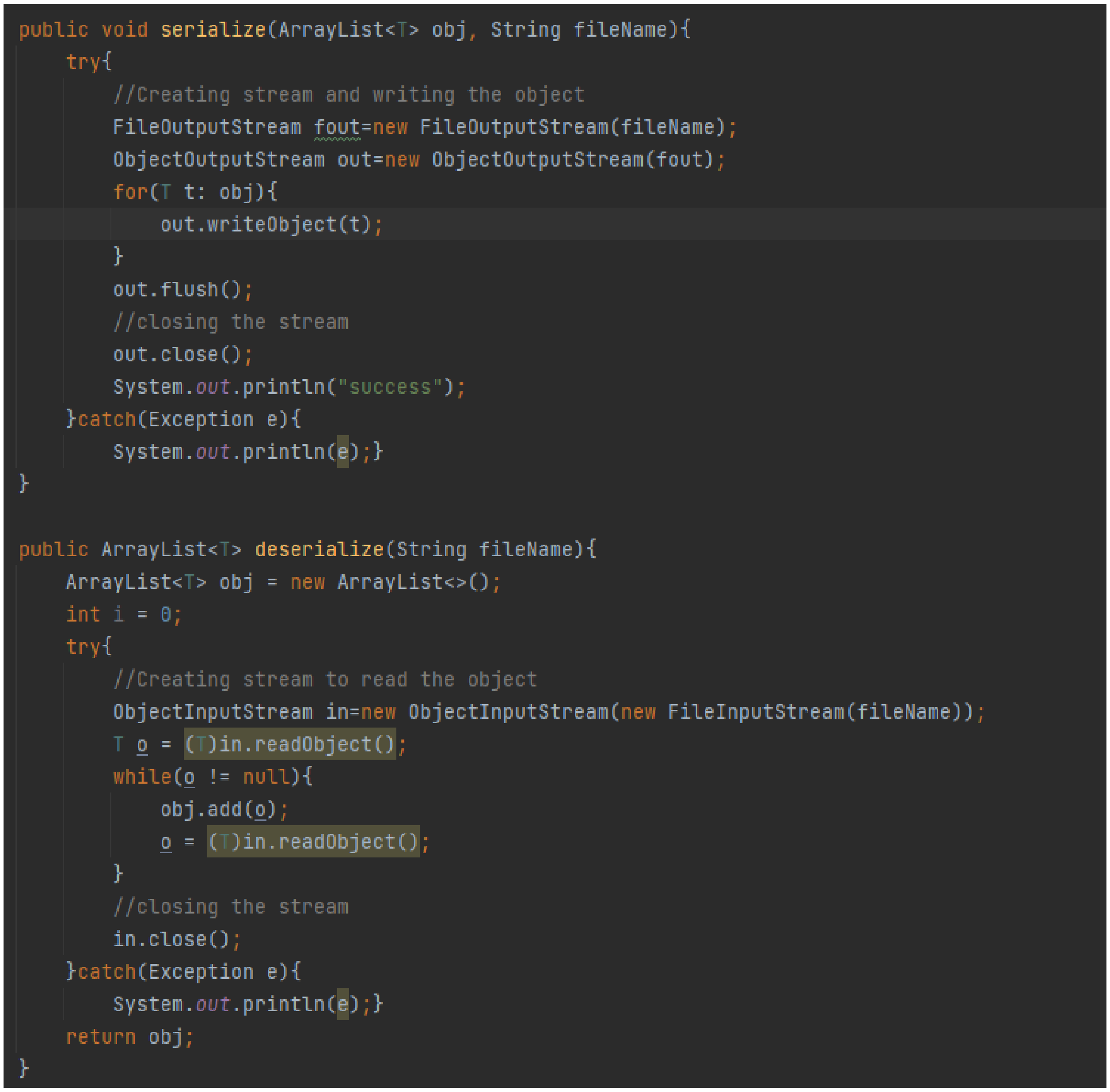
Data-access package

FileWriter

This class is responsible for generating all the .txt files, including the bill and the different reports. It has methods for every type of report because they all show different information about the orders, the clients or the menu items. It could have not been generalized. The open() and close() methods are used by all the writing methods in this class.

Serializator

This class has two main methods that are serialize() and deserialize(). These methods are called from the LoginView class, from the gui package. Every time the application starts, the menu items, the orders and the users are deserialized, therefore, the application saves the previous information stored about them. And in the same manner, when the login window is closed, before closing the app the information is saved in different txt files: the clients are in users.txt, the orders in orders.txt and the menu items in file.txt, as it was specified in the requirements of the problem.
This class is a generic class, the same methods are called for all three types of fields mentioned above, only the generic parameter changes.



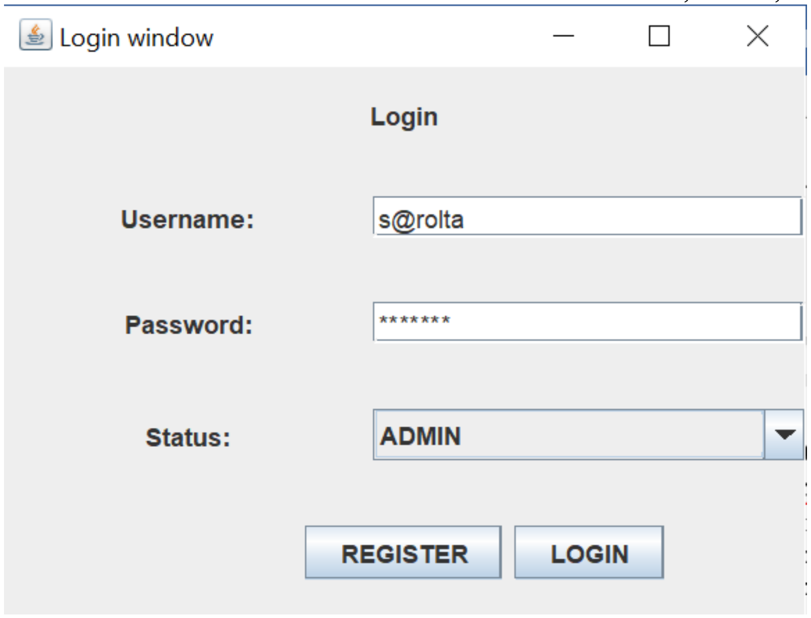
```
public void serialize(ArrayList<T> obj, String fileName){
    try{
        //Creating stream and writing the object
        FileOutputStream fout=new FileOutputStream(fileName);
        ObjectOutputStream out=new ObjectOutputStream(fout);
        for(T t: obj){
            out.writeObject(t);
        }
        out.flush();
        //closing the stream
        out.close();
        System.out.println("success");
    }catch(Exception e){
        System.out.println(e);}
}

public ArrayList<T> deserialize(String fileName){
    ArrayList<T> obj = new ArrayList<>();
    int i = 0;
    try{
        //Creating stream to read the object
        ObjectInputStream in=new ObjectInputStream(new FileInputStream(fileName));
        T o = (T)in.readObject();
        while(o != null){
            obj.add(o);
            o = (T)in.readObject();
        }
        //closing the stream
        in.close();
    }catch(Exception e){
        System.out.println(e);}
    return obj;
}
```

Gui package

LoginView

This is the first JFrame that pops up when starting the application. The login window requires users to write their name and password to be able to access the data in the app. If the user is new to the application, he/she can register at first and then continue with the login.

There are three statuses that the user can choose: he could be an administrator, a client, or an employee.


Login window

Login

Username: s@rolta

Password: *******

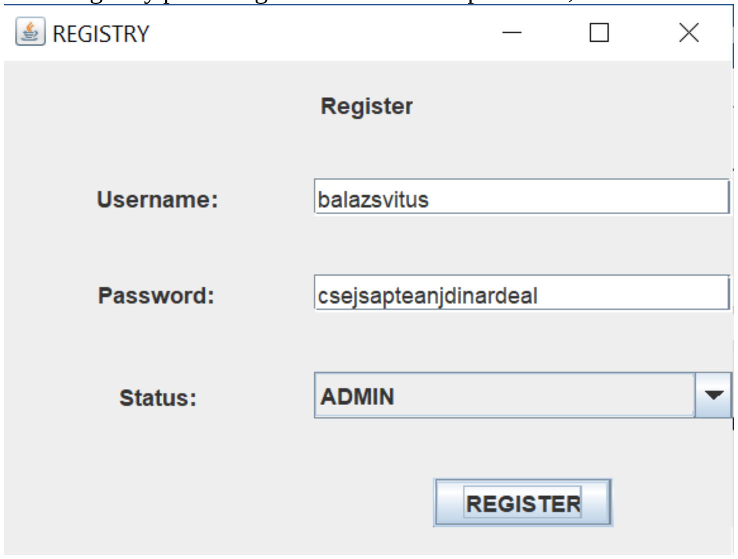
Status: ADMIN

LoginController

This class implements the ActionListener interface. When the login or the register buttons are pressed, it creates a new JFrame corresponding to the information provided by the user. It also sets the current user when the login is performed.

RegisterView

This class extends JFrame and provides the option to register for a new user. The information will be saved in the system, so he/she can later on login by providing the username and password, and also the status.


REGISTRY

Register

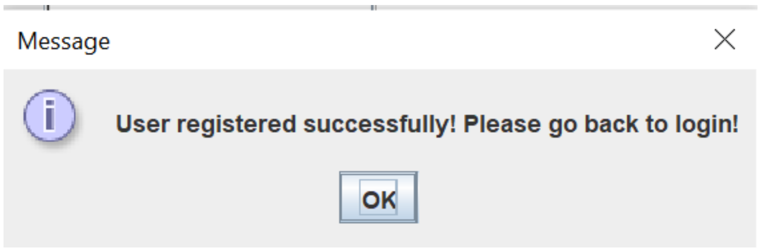
Username: balazsvitus

Password: csejsapteanjdinardeal

Status: ADMIN

RegisterController

This class implements the ActionListener interface. When the register button is pressed there are two messages that can appear on the screen: the user has registered successfully, as shown below on the screenshot, or the user is already registered, and he can proceed to the login window. In both cases, the window disappears and the user is redirected to the login window.


Message

User registered successfully! Please go back to login!

Admin package ~ inside gui package

AdminView

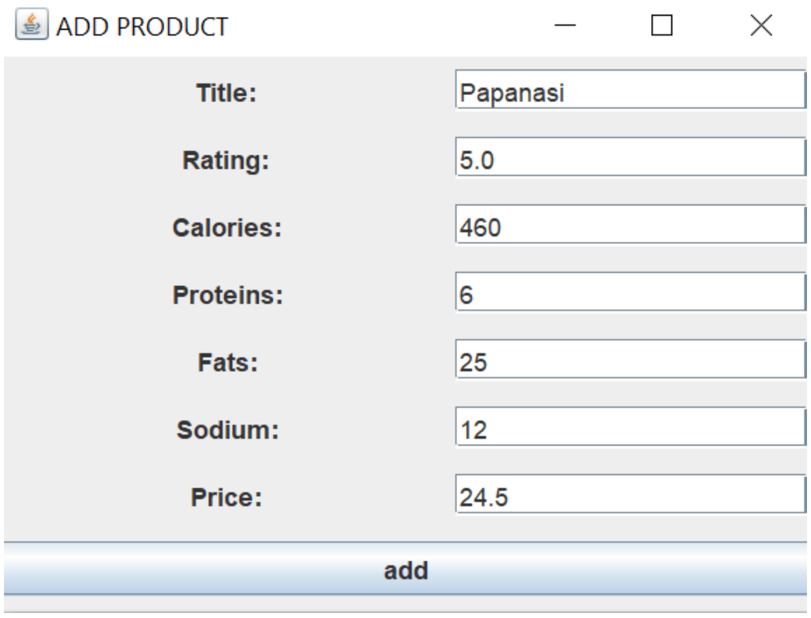
This class extends JFrame. It provides different functionalities that a system administrator would need: import products from a hard-coded file, add new base products to the menu, delete products or modify them, and create new composite products from the existing menu items. On the bottom part of the frame the admin can see the menu items.
The administrator can generate different reports, based on the category selected. This would be useful to see who their customers are, what are the most beloved products in the menu and what are the products that are seldom ordered. It can help improve the catering company. These reports are generated as txt files and will be found under the names of: ClientReport.txt, MenuItemReport.txt, NumberOfTimesReport.txt and TimeIntervalReport.txt.

AdminController

This class implements the ActionListener interface. It is responsible for opening the corresponding frames that the administrator requests, and it also calls the methods from DeliveryService to generate reports, to add/delete/modify and create products.

AddProductView

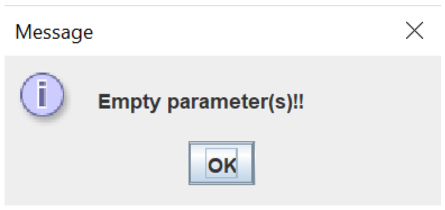
This class extends JFrame. The admin can specify all the details about the new product he wants to add. The DeliveryService class methods doublecheck if a product with the same name already exists, if yes, then the user is notified about it, if not, the user gets a message that the new product has been added to the menu items.


ADD PRODUCT

Title:	Papanasi
Rating:	5.0
Calories:	460
Proteins:	6
Fats:	25
Sodium:	12
Price:	24.5

AddProductController

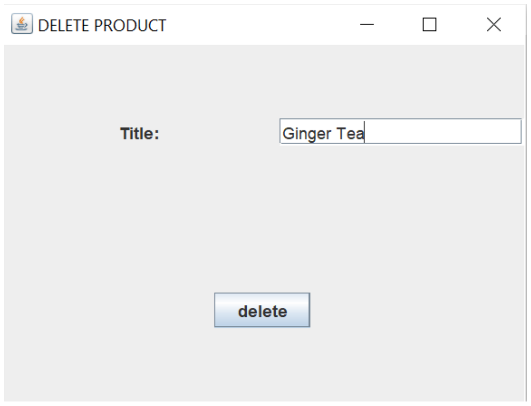
This class implements the ActionListener interface. Whenever the add button is pressed, this class calls the addProduct() method from DelieryService class. But before that, it also checks if the parameters given by the user are in the correct range.


Message

Empty parameter(s)!!

DeleteProductView

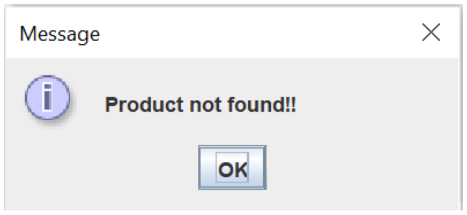
This class extends JFrame. The admin can provide the name of the menu item that he/she wishes to delete and then press delete.


DELETE PRODUCT

Title: Ginger Tea

DeleteProductController

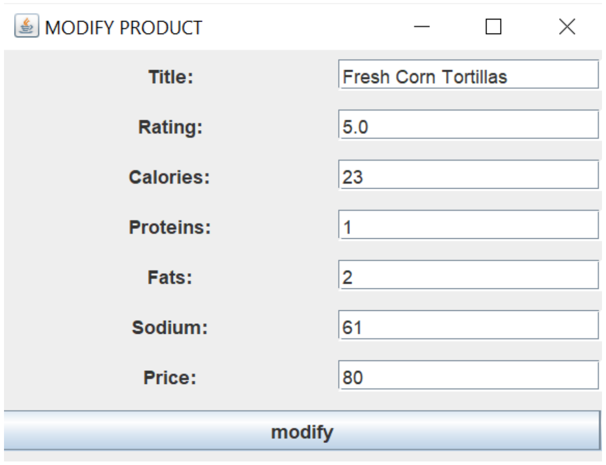
This class implements the ActionListener interface. It checks if the given name exists in the menu item list. If not, the user gets a warning message, as shown below. If the product was found, then it will get deleted and the menu item list is updated.


Message

Product not found!!

ModifyProductView

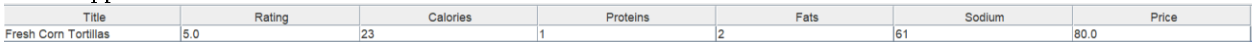
This class extends JFrame. The admin must provide all the fields of the given product. The title cannot be modified, that's how the app checks the ecxistance of the product in the menu item list. All the other parameters however can be changed if the user wants it to. Otherwise, just keep the values as they are in the table. Then the admin must press modify.


MODIFY PRODUCT

Title:	Fresh Corn Tortillas
Rating:	5.0
Calories:	23
Proteins:	1
Fats:	2
Sodium:	61
Price:	80

ModifyProductController

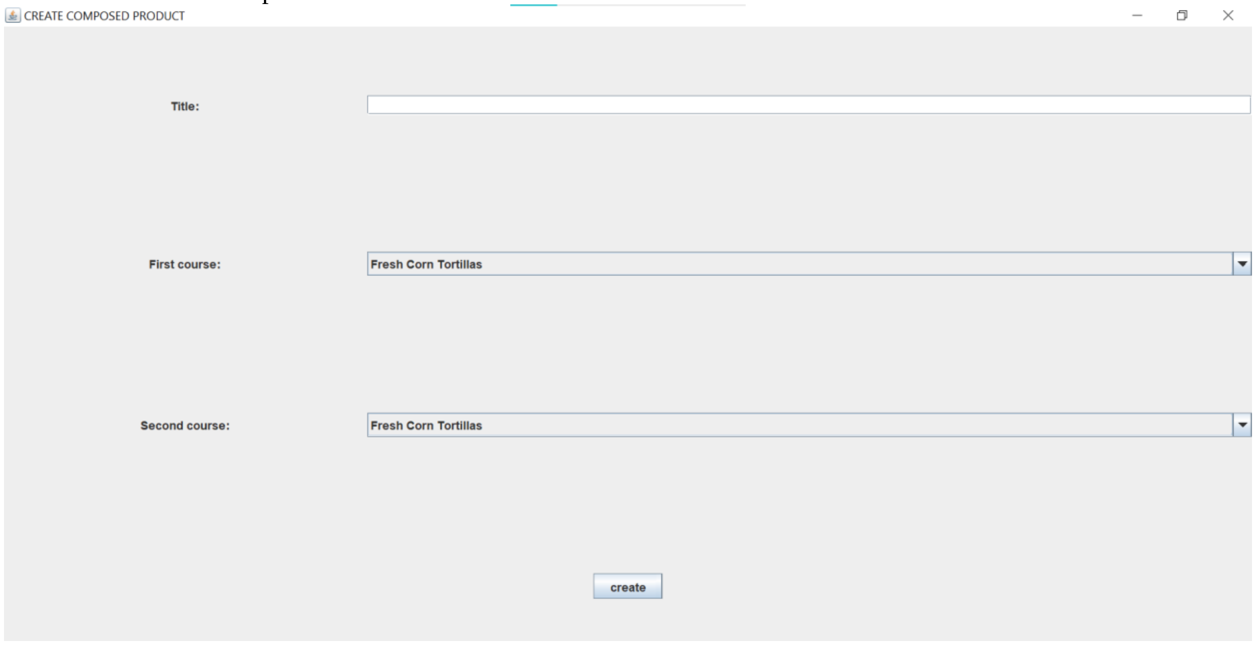
This class implements the ActionListener interface. When the modify button is pressed, the DeliveryService class searches through the menu item list and if it finds the product, then it modifies the values and updates the table. If the title is not found, then it prints a warning message to the screen. On the screenshot it is shown how the new values appear in the table below.



Title	Rating	Calories	Proteins	Fats	Sodium	Price
Fresh Corn Tortillas	5.0	23	1	2	61	80.0

CreateCompositeProductView

This class extends JFrame. The admin can combine two menu items to create a daily menu for example. From the droplist he can select from the already existing menu items two, and combine them in a new menu item, giving it a new title. Then he must press create.

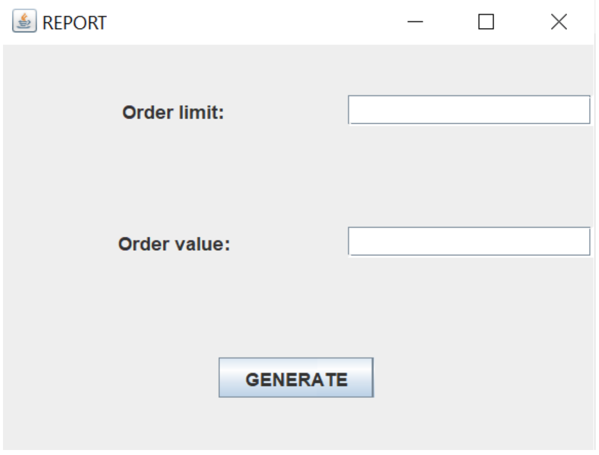


CreateCompositeProductController

This class implements the ActionListener interface. If the title field is not empty, then a new product gets created and added to the menu items. The table gets updated as well. If the title field is left empty, the user gets a warning message to provide a name for the new menu item.

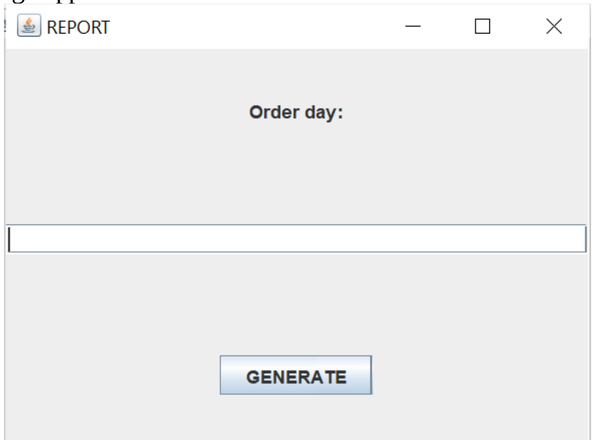
ClientReport

This class extends JFrame. If the admin provides a minimum number of orders and a minimum value for these orders from a client, then he will get a report stating all the clients that have ordered more than the minimum number of times, and their order had a total value more than the minimum amount specified. The admin must press generate after providing the values. If the report was generated successfully, a message appears to inform the user.



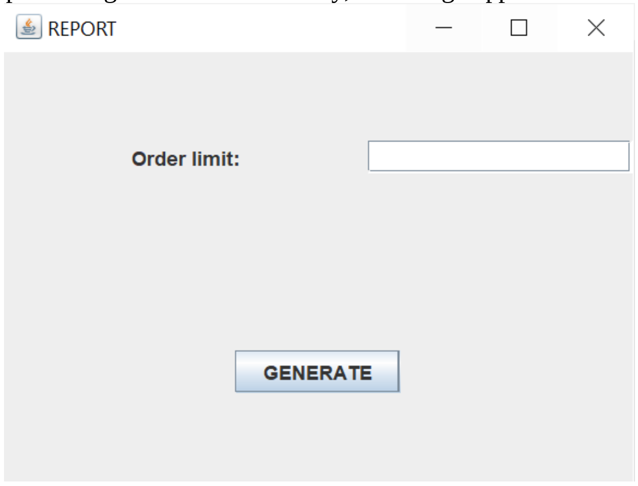
MenuItemReport

This class extends JFrame. The admin must specify a day of the year (from 0 to 365) to get the product ordered on the specified day. He will also get the number of times these products have been ordered on that day. If the day is given an incorrect value, then the user gets a warning message to specify a correct day of the year. If the report was generated successfully, a message appears to inform the user.



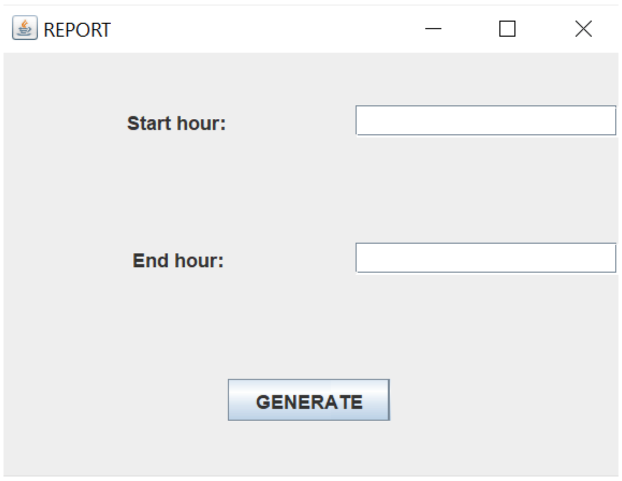
NumberOfTimesReport

This class extends JFrame. The admin must provide an order limit to get all the products that have been ordered more than a specified number of times. If this limit is a negative number, the user gets a warning message to provide a correct order limit. If the report was generated successfully, a message appears to inform the user.



TimeIntervalReport

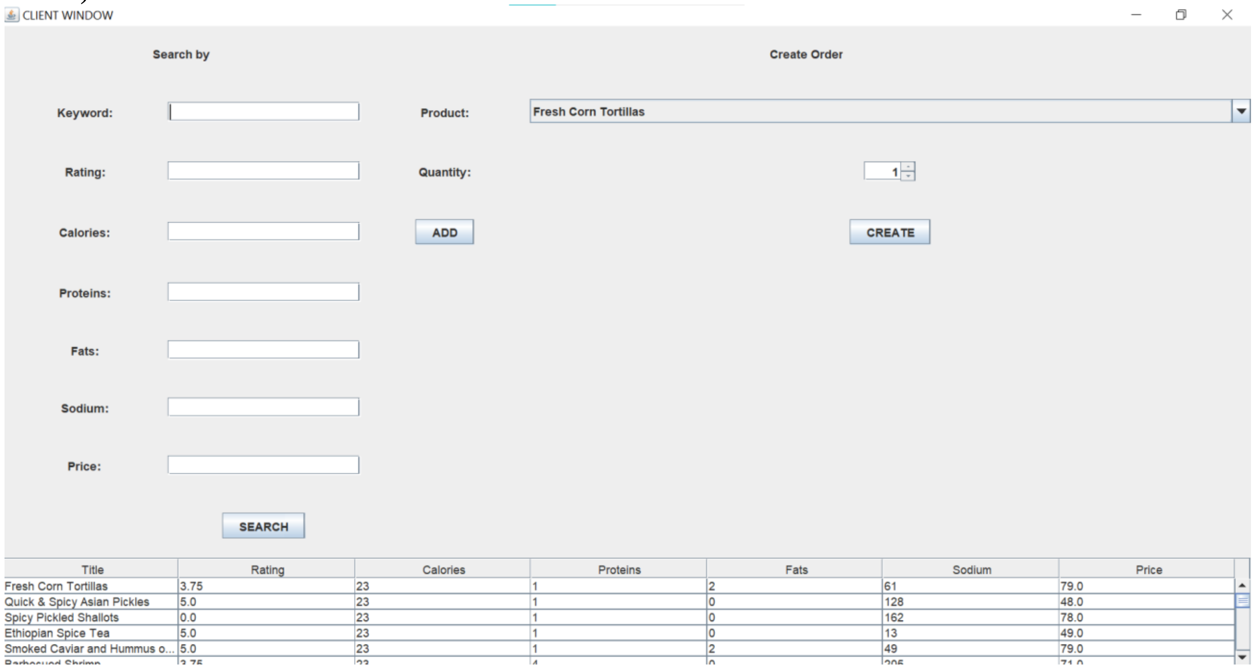
This class extends JFrame. The admin must provide a starting and a finishing hour to get all the products ordered between these limits. If this limits are negative numbers or the start time is greater than the finishing time, the user gets a warning message to provide correct limits. If the report was generated successfully, a message appears to inform the user.



Client package ~ inside gui package

ClientView

This class extends JFrame. If a client logs into the system, he will arrive to this client window. From here he can see all the menu items available. He can place orders with multiple menu items and choosing the desired quantity for each of them. He can also search specific items based on different categories (all the fields of the items can be a criteria).



Product: Fresh Corn Tortillas

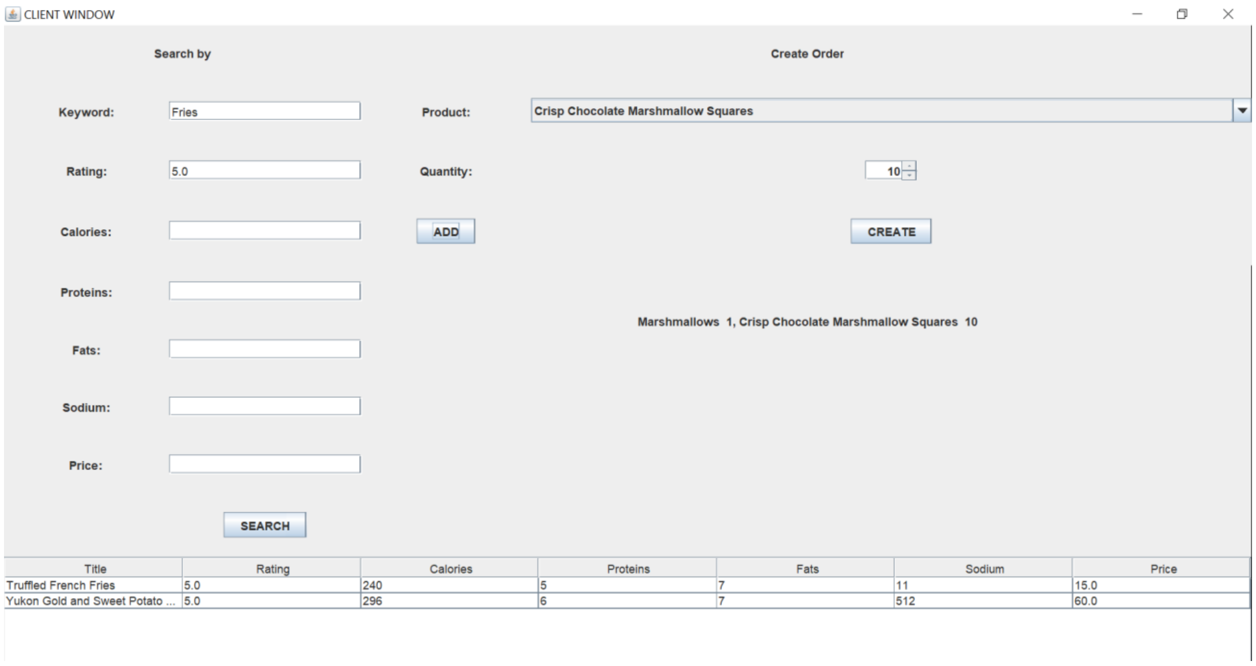
Quantity: 1

Title	Rating	Calories	Proteins	Fats	Sodium	Price
Fresh Corn Tortillas	3.75	23	1	2	61	79.0
Quick & Spicy Asian Pickles	5.0	23	1	0	128	48.0
Spicy Pickled Shallots	0.0	23	1	0	162	78.0
Ethiopian Spice Tea	5.0	23	1	0	13	49.0
Smoked Caviar and Hummus o...	5.0	23	1	2	49	79.0
Barbecued Shrimp	3.75	23	4	0	206	71.0

ClientController

This class implements the ActionListener interface. By typing in some of the fields of the searching criterion, and then pressing search, the client can see in the table at the bottom of the frame the items that correspond to the provided criterion.

If he wishes to place an order, he can select the item from the dropdown list, then specify the quantity (at least 1), and then pressing add is like adding it to the current order. When all the products have been added, the create button creates the new order and notifies the user with a message that the order has been created. The selected items disappear from the screen as well.



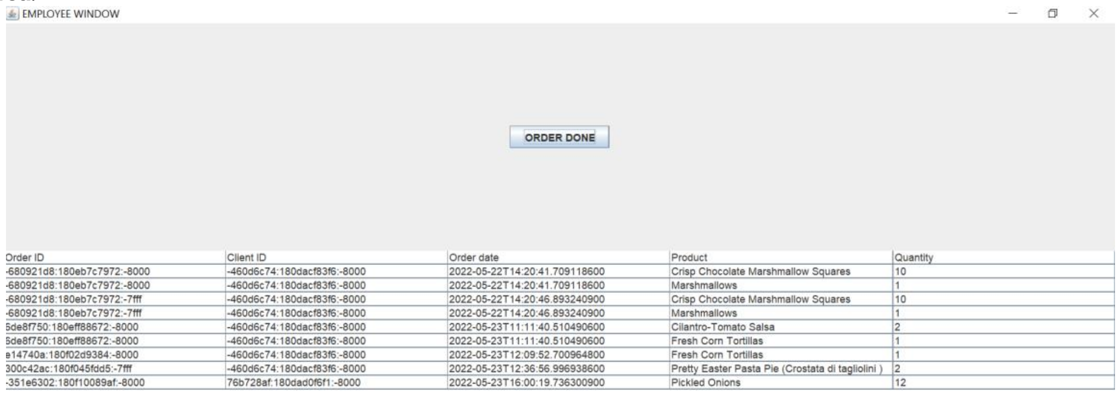
Marshmallows 1, Crisp Chocolate Marshmallow Squares 10

Title	Rating	Calories	Proteins	Fats	Sodium	Price
Truffled French Fries	5.0	240	5	7	11	15.0
Yukon Gold and Sweet Potato ...	5.0	296	6	7	512	60.0

Employee package

EmployeeView

This class extends JFrame. The employee can see all the orders that have been placed in the table below. He can see all the products from the orders on a separate line of the table so he knows in which order should the items be prepared.



Order ID	Client ID	Order date	Product	Quantity
-680921d8:180eb7c7972:-8000	-460d6c74:180dacf83f6:-8000	2022-05-22T14:20:41.709118600	Crisp Chocolate Marshmallow Squares	10
-680921d8:180eb7c7972:-8000	-460d6c74:180dacf83f6:-8000	2022-05-22T14:20:41.709118600	Marshmallows	1
-680921d8:180eb7c7972:-7fff	-460d6c74:180dacf83f6:-8000	2022-05-22T14:20:46.893240900	Crisp Chocolate Marshmallow Squares	10
-680921d8:180eb7c7972:-7fff	-460d6c74:180dacf83f6:-8000	2022-05-22T14:20:46.893240900	Marshmallows	1
5de8f750:180eff88672:-8000	-460d6c74:180dacf83f6:-8000	2022-05-23T11:11:40.510490600	Cilantro-Tomato Salsa	2
5de8f750:180eff88672:-8000	-460d6c74:180dacf83f6:-8000	2022-05-23T11:11:40.510490600	Fresh Corn Tortillas	1
e14740a:180f02d9384:-8000	-460d6c74:180dacf83f6:-8000	2022-05-23T12:09:52.700964800	Fresh Corn Tortillas	1
300c42ac:180f045fdd5:-7fff	-460d6c74:180dacf83f6:-8000	2022-05-23T12:36:56.996938600	Pretty Easter Pasta Pie (Crostata di tagliolini)	2
-351e6302:180f10089af:-8000	76b728af:180dad0f6f1:-8000	2022-05-23T16:00:19.736300900	Pickled Onions	12

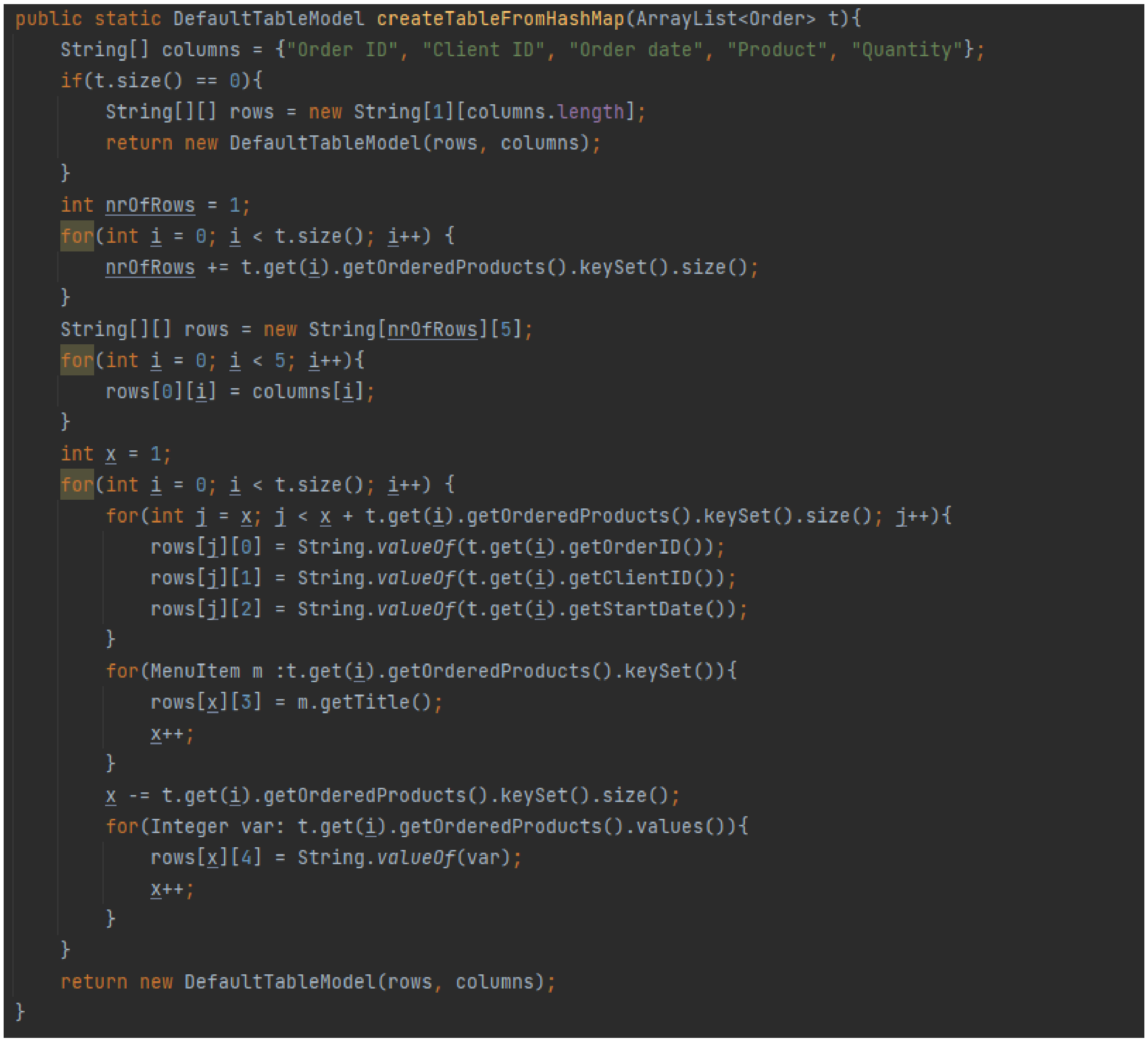
EmployeeController

This class implements the ActionListener interface. When the employee presses order done button, the first row of the table gets deleted, therefore, the current order is deleted. The employee has finished it and he starts the next one. If all the orders have been deleted and the employee presses the button, he gets a warning message that there are no more orders to be prepared.

Utils package

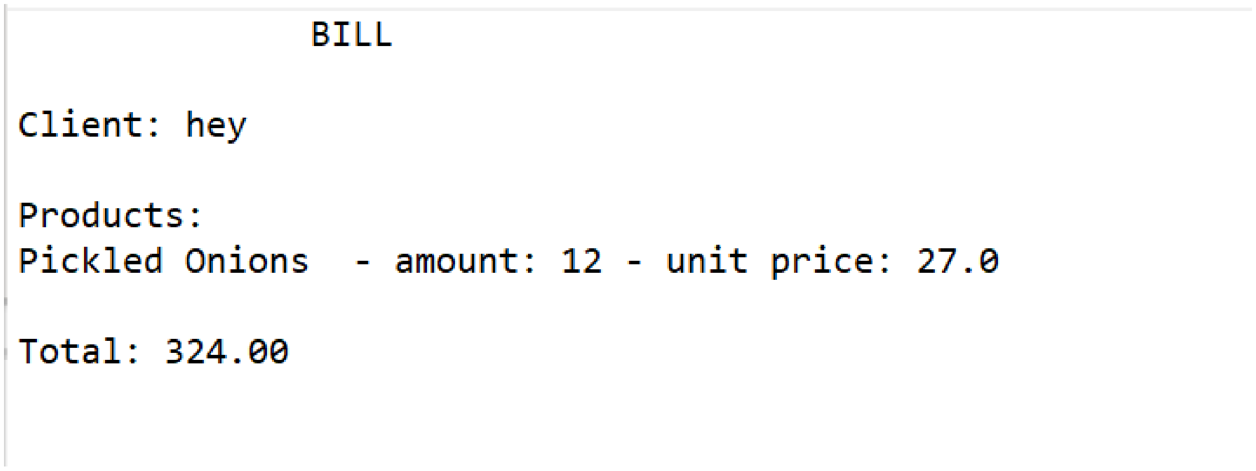
TableFactory

This class has two methods: the createTable that returns a DefaultTableModel and the createTableFromHashMap which returns the same type of object. The only difference between them is the fact that the second one is used for the orders to be visualized in a table form, while the first one is used for menu items. The order class has a field that is a hash map, so to get all the products and their quantities from the hash map, a different strategy is used to create the table.

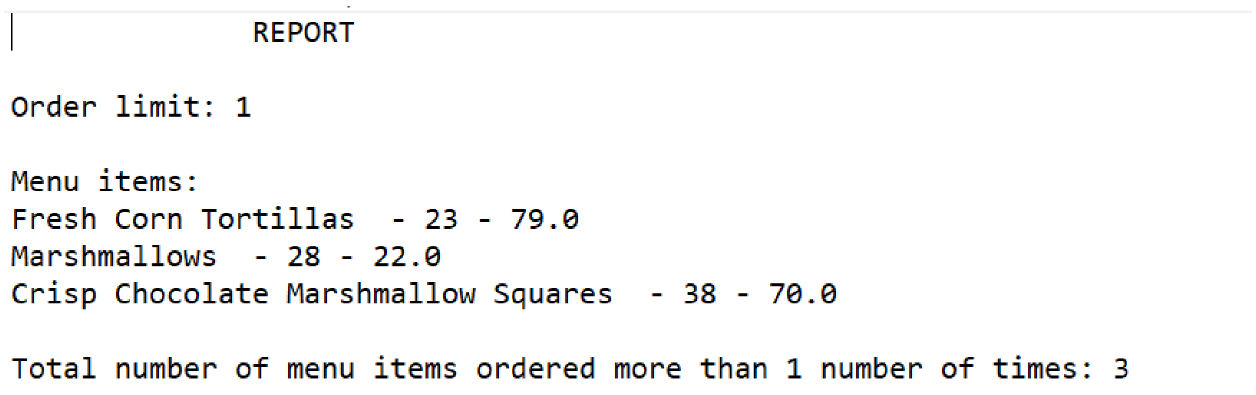


```java
public static DefaultTableModel createTableFromHashMap(ArrayList<Order> t){
    String[] columns = {"Order ID", "Client ID", "Order date", "Product", "Quantity"};
    if(t.size() == 0){
        String[][] rows = new String[1][columns.length];
        return new DefaultTableModel(rows, columns);
    }
    int nrOfRows = 1;
    for(int i = 0; i < t.size(); i++) {
        nrOfRows += t.get(i).getOrderedProducts().keySet().size();
    }
    String[][] rows = new String[nrOfRows][5];
    for(int i = 0; i < 5; i++){
        rows[0][i] = columns[i];
    }
    int x = 1;
    for(int i = 0; i < t.size(); i++) {
        for(int j = x; j < x + t.get(i).getOrderedProducts().keySet().size(); j++){
            rows[j][0] = String.valueOf(t.get(i).getOrderID());
            rows[j][1] = String.valueOf(t.get(i).getClientID());
            rows[j][2] = String.valueOf(t.get(i).getStartDate());
        }
        for(MenuItem m :t.get(i).getOrderedProducts().keySet()){
            rows[x][3] = m.getTitle();
            x++;
        }
        x -= t.get(i).getOrderedProducts().keySet().size();
        for(Integer var: t.get(i).getOrderedProducts().values()){
            rows[x][4] = String.valueOf(var);
            x++;
        }
    }
    return new DefaultTableModel(rows, columns);
}
```

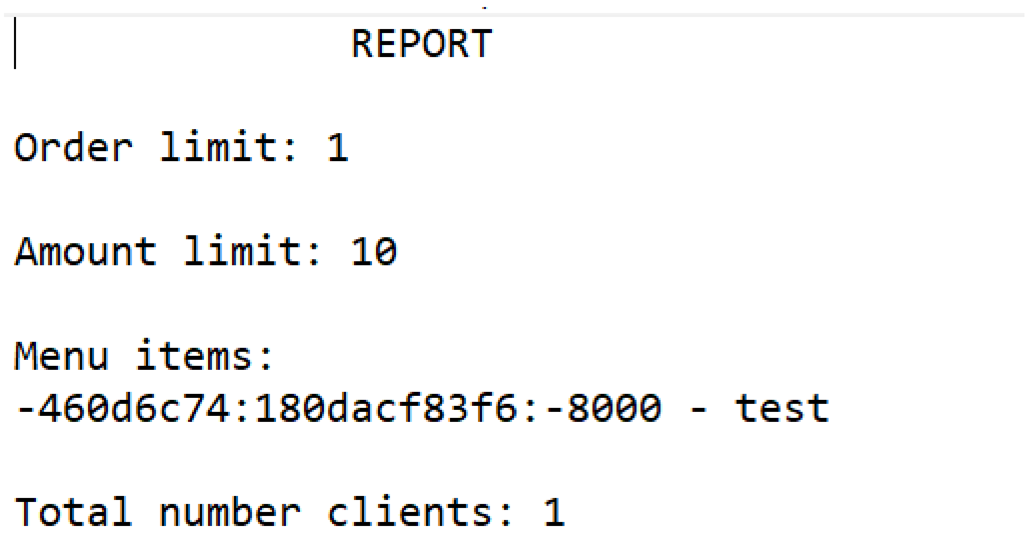
5. Results



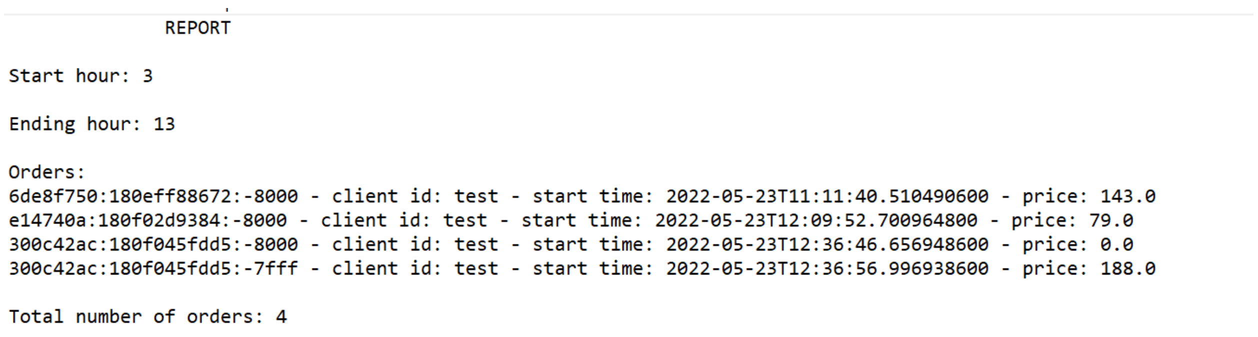
```
               BILL

Client: hey

Products:
Pickled Onions  - amount: 12 - unit price: 27.0

Total: 324.00
```



```
              REPORT

Order limit: 1

Menu items:
Fresh Corn Tortillas  - 23 - 79.0
Marshmallows  - 28 - 22.0
Crisp Chocolate Marshmallow Squares  - 38 - 70.0

Total number of menu items ordered more than 1 number of times: 3
```



```
              REPORT

Order limit: 1

Amount limit: 10

Menu items:
-460d6c74:180dacf83f6:-8000 - test

Total number clients: 1
```


```
                REPORT

Start hour: 3

Ending hour: 13

Orders:
6de8f750:180eff88672:-8000 - client id: test - start time: 2022-05-23T11:11:40.510490600 - price: 143.0
e14740a:180f02d9384:-8000 - client id: test - start time: 2022-05-23T12:09:52.700964800 - price: 79.0
300c42ac:180f045fdd5:-8000 - client id: test - start time: 2022-05-23T12:36:46.656948600 - price: 0.0
300c42ac:180f045fdd5:-7fff - client id: test - start time: 2022-05-23T12:36:56.996938600 - price: 188.0

Total number of orders: 4
```

6. Conclusion

This project represents the basics of a catering company's management system. The basic tools are provided to communicate between the different users, to be able to manipulate the menu and to add new items. The report form the txt files could help the company with statistics to improve their business plan and update the menu from time to time. The application is however not complete. It could have many more features that are not incorporated in the current design.

First, the user interface could be made much more interactive, from the design point of view there is room for improvement. Some colors would make it a lot more fun to use.

Second, the registering is a bit faulty. At present, any user has a right to register as administrator. In a normal case, the administrators have a hard-coded user object associated to them and they cannot simply register with a new username and password every time. This would present a great danger in security and the system could be manipulated with by the wrong people. This would be a major security issue that the project would have to improve on.

In conclusion, there are many more objectives that could be accomplished from this first scratch. However, the main objectives are met and in the process of making the app I have learnt about so many different aspects regarding Java. I have learnt about the Observer pattern, streams and how can lambda expressions be used together with them. I have incorporated Javadoc comments for the project to be easily understood in the future. Also, the pre- and post-conditions were a new field for me, this is the first time that I have used them to verify the parameters provided to methods. Overall, the project is a demonstration for a management system that would we improved in the future for commercial use.

7. Bibliography

The resources for this project were the pdf-s provided by the professor, the presentation of the project and the pdf guide.
I have also tackled several problems with the help of different websites explaining different Java concepts.