# FOOD DELIVERY MANAGEMENT SYSTEM

Horvath Ariana-Cristine

## Table of Contents

# 1. Assignment objective

## 1.1 Main Objective

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 (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:
    - 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;
    - 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 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.

## 1.2 Sub-objectives

- **Analyze** the problem and identify requirements – how the application should work, what it should do and what are the use cases (described in chapter 2. Analysis)
- **Design** the Food Delivery Management System – which architectural pattern should be used, how the packages, classes and methods are divided (described in chapter 3. Design)
- **Implement** the Food Delivery Management System – the Java code written for the implementation (described in chapter 4.Implementation)
- **Test** the Food Delivery Management System – testing the application to check if it works properly (described in chapter 5.Results)

## 2. Analysis

### 2.1 Requirements

#### 2.1.1 Functional requirements

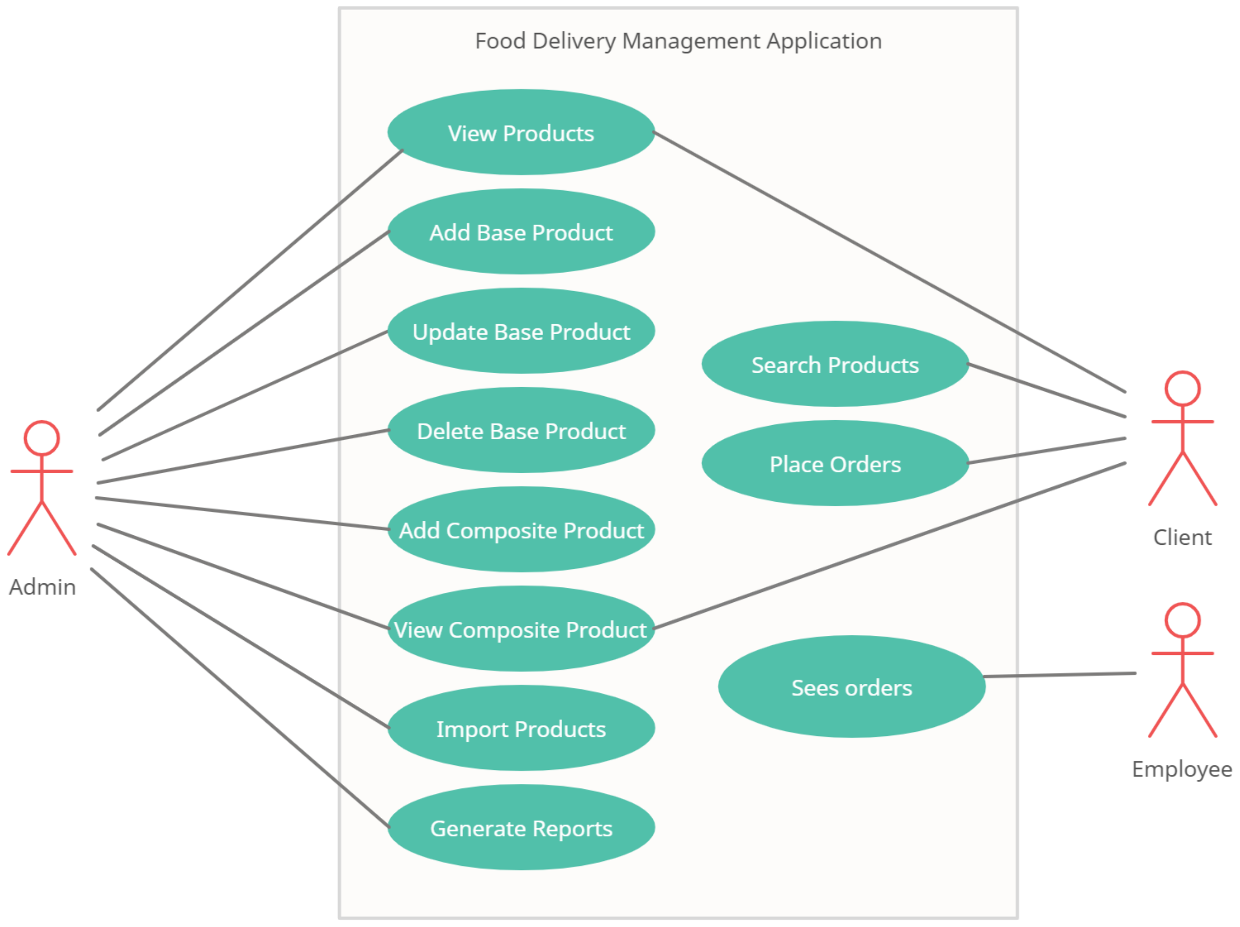
- The application should allow users to register and use the registered data to log in.
- **For each of these, the application should allow users to perform the required operations (described in the Main Objective).**
- The application should validate input and make modifications in the delivery service only if the data is valid.
- The application should create a bill (in a .txt file) for every order made.
- The application should create the required reports (in .txt files).
- The application should keep data updated between different types of users using the application in the same time.

#### 2.1.2 Non-Functional requirements

- The application should be intuitive and easy to use by the user.
- The application should not allow the users to introduce invalid input.
- The application should warn the user if the input is not valid or empty.
- The application should notify the users when the operations performed by them are successful.

### 2.2 Use-cases and scenarios

[2]



- **Use Case**: View Products

**Primary Actor**: Admin, Client

**Main Success Scenario**:

1. The user registers/logs in successfully.
2. The user selects “View Products”.
3. A table with the products appears on the screen, having the products’ title, rating, calories, protein, fat, sodium, price.

**Alternative Sequence:** none

- **Use Case**: Import Products

**Primary Actor**: Admin

**Main Success Scenario**:

1. The admin registers/logs in successfully.
2. The admin selects “Import Products”.
3. The list of base products is imported from the “products.csv” file.

**Alternative Sequence:** none

- **Use Case**: Add Base Product

**Primary Actor**: Admin

**Main Success Scenario**:

1. The user registers/logs in successfully.
2. The user inserts in the text fields the data for a base product: title, rating, calories, protein, fat, sodium, price.
3. The user selects “Add Product” by pressing the button.
4. The application saves the product.
5. An information message is shown to let the user know that the operation was performed successfully.

**Alternative Sequence:** Incorrect input

- The user leaves empty one or more of the text fields (title, rating, calories, protein, fat, sodium, price) or it inserts invalid data (rating is not a double between 0 and 5, calories, protein, fat, sodium, price are not integers).
- The title of the product is already existent.
- A pop-up with an error message is displayed;
- The user presses the “Ok” button or “x”(Close);
- The scenario returns to step 2.

- **Use Case**: Update Base Product

**Primary Actor**: Admin

**Main Success Scenario**:

1. The user registers/logs in successfully.
2. The user inserts in the text fields the data for a base product: title, rating, calories, protein, fat, sodium, price.
3. The user selects “Update Product” by pressing the button.
4. The application updates the product.

5. An information message is shown to let the user know that the operation was performed successfully.

**Alternative Sequence:** Incorrect input

- The user leaves empty one or more of the text fields (title, rating, calories, protein, fat, sodium, price) or it inserts invalid data (rating is not a double between 0 and 5, calories, protein, fat, sodium, price are not integers).
- The title of the product is not existent.
- A pop-up with an error message is displayed;
- The user presses the "Ok" button or "x"(Close);
- The scenario returns to step 2.

- **Use Case**: Delete Base Product

**Primary Actor**: Admin

**Main Success Scenario**:

1. The user registers/logs in successfully.
2. The user inserts in the text fields the title of the product to be deleted.
3. The user selects "Delete Product" by pressing the button.
4. The application deletes the product.
5. An information message is shown to let the user know that the operation was performed successfully.

**Alternative Sequence:** Incorrect input

- The user leaves empty the title text field or it inserts invalid data: the title of the product is not existent.
- A pop-up with an error message is displayed;
- The user presses the "Ok" button or "x"(Close);
- The scenario returns to step 2.

- **Use Case**: Add Composite Product

**Primary Actor**: Admin

**Main Success Scenario**:

1. The user registers/logs in successfully.
2. The user inserts in the text fields the data of the composite product: title and products of which it consists, separated by comma.
3. The user selects "Add Product" by pressing the button.
4. The application saves the product.
5. An information message is shown to let the user know that the operation was performed successfully.

**Alternative Sequence:** Incorrect input

- The user leaves empty the title or products text field or it inserts invalid data: the title of the product is already existent or one of the products from the list is not existent in the menu.
- A pop-up with an error message is displayed;
- The user presses the "Ok" button or "x"(Close);
- The scenario returns to step 2.

- **Use Case**: View Composite Product

**Primary Actor**: Admin, Client

**Main Success Scenario**:

1. The user registers/logs in successfully.
2. The user selects "Show Product" by pressing the button.
3. In a text area nearby, it appears a list of products of which the composite product consists.

**Alternative Sequence:** Incorrect input

- The user leaves empty the title text field or it inserts invalid data: the title of the product is not existent in the menu or the product is not a composite one.
- A pop-up with an error message is displayed;
- The user presses the “Ok” button or “x”(Close);
- The scenario returns to step 2.

- **Use Case**: Generate Reports

**Primary Actor**: Admin

**Main Success Scenario**:

1. The user registers/logs in successfully.
2. The user introduces data for report 1 (orders that are placed between start hour and end hour)
3. The user selects “Generate Report 1” by pressing the button.
4. The report is created in the Report1.txt file and An information message is shown to let the user know that the operation was performed successfully.
5. The user introduces data for report 2 (products ordered more than specified number of times).
6. The user selects “Generate Report 2” by pressing the button.
7. The report is created in the Report1.txt file and An information message is shown to let the user know that the operation was performed successfully.
8. The user introduces data for report 3 (clients that ordered more than specified number of times and the value of the order was higher than specified amount).
9. The report is created in the Report1.txt file and An information message is shown to let the user know that the operation was performed successfully.

**Alternative Sequence:** Incorrect input

- The user leaves empty one or more text fields or it inserts invalid data: the start hour and end hour are not of hour format or the start hour is after the end one, the number of times or amount are not integers or are negative, the date is not a valid date or is a date in the future.
- A pop-up with an error message is displayed;
- The user presses the “Ok” button or “x”(Close);
- The scenario returns to step 2.

- **Use Case**: Search Products

**Primary Actor**: Client

**Main Success Scenario**:

1. The user registers/logs in successfully.
2. The user inserts in one or more text fields the data for a base product to be searched: title, rating, calories, protein, fat, sodium, price.
3. The user selects “Search” by pressing the button.
4. A table with the products meeting the searching criteria appears on the screen, having the products’ title, rating, calories, protein, fat, sodium, price.

**Alternative Sequence:** Incorrect input

- The inserts invalid data (rating is not a double, calories, protein, fat, sodium, price are not integers).
- A pop-up with an error message is displayed;
- The user presses the “Ok” button or “x”(Close);
- The scenario returns to step 2.

- **Use Case**: Place Orders

**Primary Actor**: Client

**Main Success Scenario**:

1. The user registers/logs in successfully.
2. The user inserts in the corresponding text fields the data of an order: client username and products to be ordered, on separate lines.
3. The user selects “Place Order” by pressing the button.
4. The application saves the order.
5. An information message is shown to let the user know that the operation was performed successfully.
6. The bill is generated in a .txt file.

**Alternative Sequence:** Incorrect input

- The user leaves empty the client or products text field or it inserts invalid data: the client is not existent or one of the products from the list is not existent in the menu.
- A pop-up with an error message is displayed;
- The user presses the “Ok” button or “x”(Close);
- The scenario returns to step 2.

- **Use Case**: See Orders

**Primary Actor**: Employee

**Main Success Scenario**:

1. The user registers/logs in successfully.
2. The user sees the orders (products to be prepared) when they are added by the client. There can be more than one order.

**Alternative Sequence:** none

## 3. Design

### 3.1 Design Decisions

- Define the interface IDeliveryServiceProcessing containing the main operations that can be executed by the administrator and client, as follows:
  • **Administrator**: import products, manage the products from the menu, generate reports
  • **Client**: create new order which implies computing the price for an order and generating a bill in .txt format, searching for products based on several criteria.

- Use the Composite Design Pattern for defining the classes MenuItem, BaseProduct and CompositeProduct.

- Use the Observer Design Pattern to notify the employee each time a new order is created.

- Implement the class DeliveryService using a predefined JCF collection which uses a hashtable data structure. The hashtable key will be generated based on the class Order, which can have associated several MenuItems.

- Define a method of type “well formed” for the class DeliveryService.

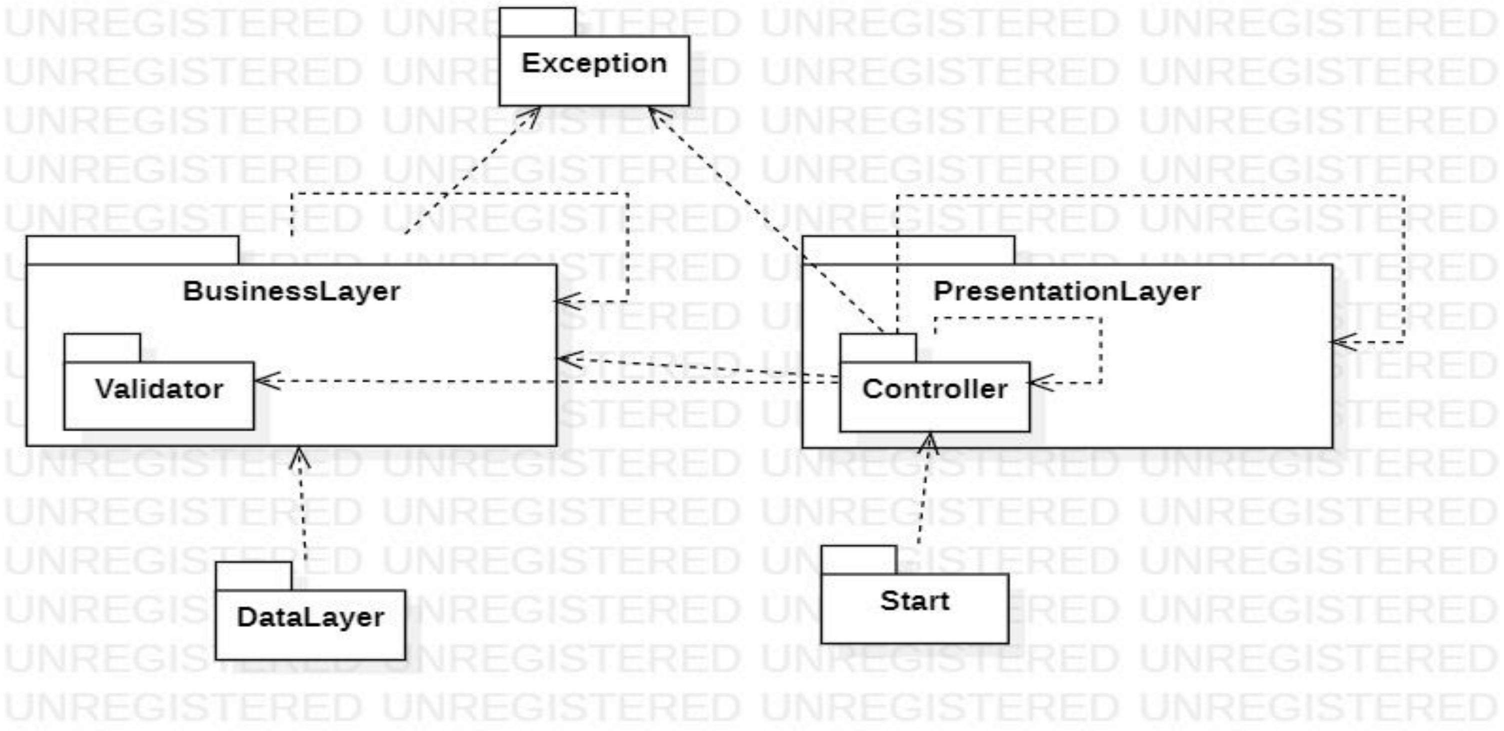
- Implement the class using Design by Contract method (involving pre, post conditions, invariants, and assertions).

- The base products used initially for populating the DeliveryService object can be loaded from the products.csv file. Note: the administrator can manually add other base products as well.

- The menu items, performed orders and user information will be persisted using serialization so as to be available at future system executions by means of deserialization.

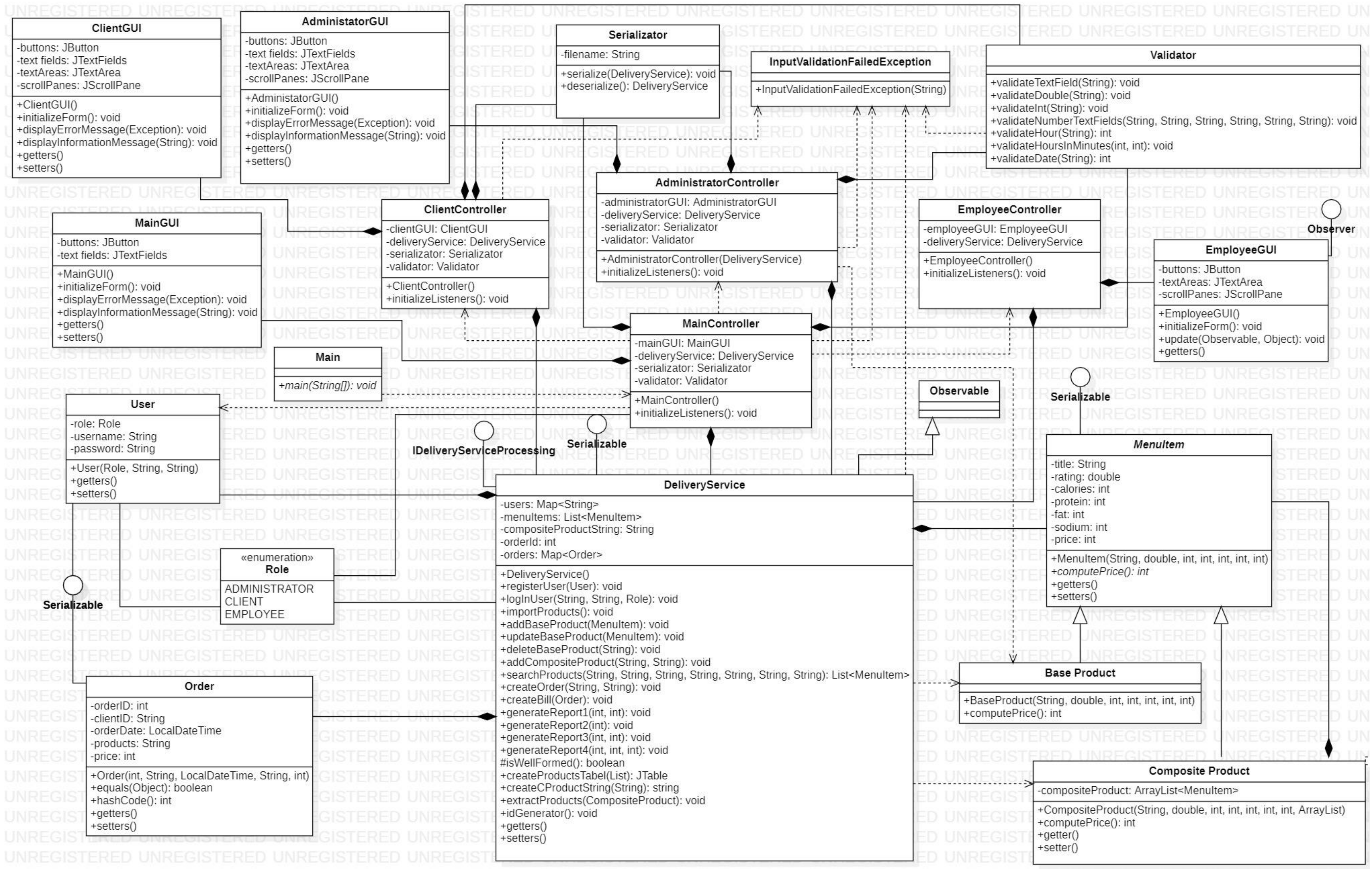
## 3.2 Diagrams

### 3.2.1 Package Diagram

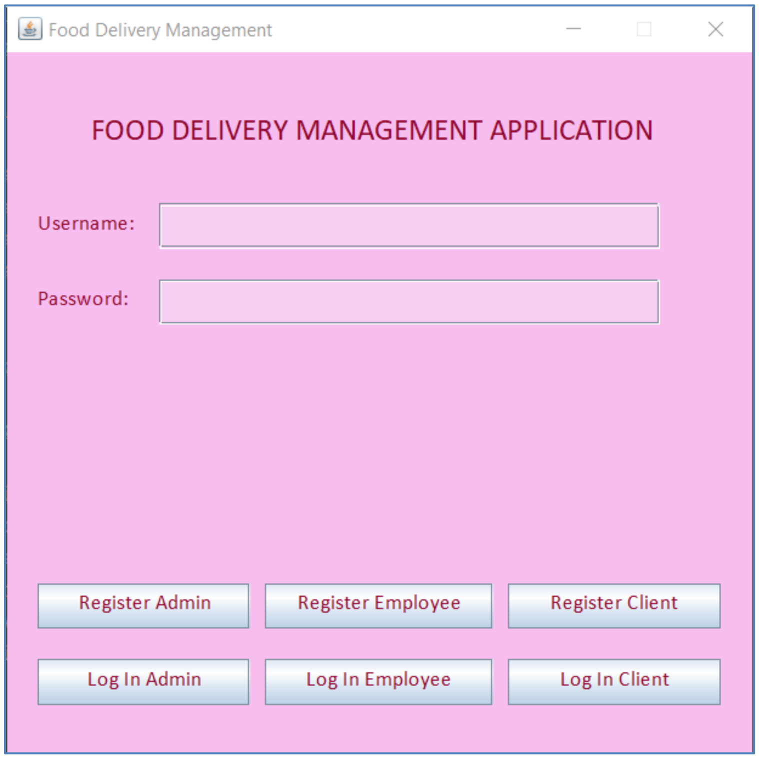


[3]

### 3.2.2 Class Diagram


ClientGUI
-buttons: JButton
-text fields: JTextFields
-textAreas: JTextArea
-scrollPanes: JScrollPane
+ClientGUI()
+initializeForm(): void
+displayErrorMessage(Exception): void
+displayInformationMessage(String): void
+getters()
+setters()
AdministatorGUI
-buttons: JButton
-text fields: JTextFields
-textAreas: JTextArea
-scrollPanes: JScrollPane
+AdministatorGUI()
+initializeForm(): void
+displayErrorMessage(Exception): void
+displayInformationMessage(String): void
+getters()
+setters()
Serializator
-filename: String
+serialize(DeliveryService): void
+deserialize(): DeliveryService
InputValidationFailedException
+InputValidationFailedException(String)
Validator
+validateTextField(String): void
+validateDouble(String): void
+validateInt(String): void
+validateNumberTextFields(String, String, String, String, String, String): void
+validateHour(String): int
+validateHoursInMinutes(int, int): void
+validateDate(String): int
AdministratorController
-administratorGUI: AdministratorGUI
-deliveryService: DeliveryService
-serializator: Serializator
-validator: Validator
+AdministratorController(DeliveryService)
+initializeListeners(): void
MainGUI
-buttons: JButton
-text fields: JTextFields
+MainGUI()
+initializeForm(): void
+displayErrorMessage(Exception): void
+displayInformationMessage(String): void
+getters()
+setters()
ClientController
-clientGUI: ClientGUI
-deliveryService: DeliveryService
-serializator: Serializator
-validator: Validator
+ClientController()
+initializeListeners(): void
EmployeeController
-employeeGUI: EmployeeGUI
-deliveryService: DeliveryService
+EmployeeController()
+initializeListeners(): void
Observer
EmployeeGUI
-buttons: JButton
-textAreas: JTextArea
-scrollPanes: JScrollPane
+EmployeeGUI()
+initializeForm(): void
+update(Observable, Object): void
+getters()
Main
+main(String[]): void
MainController
-mainGUI: MainGUI
-deliveryService: DeliveryService
-serializator: Serializator
-validator: Validator
+MainController()
+initializeListeners(): void
Observable
Serializable
User
-role: Role
-username: String
-password: String
+User(Role, String, String)
+getters()
+setters()
IDeliveryServiceProcessing
Serializable
MenuItem
-title: String
-rating: double
-calories: int
-protein: int
-fat: int
-sodium: int
-price: int
+MenuItem(String, double, int, int, int, int, int)
+computePrice(): int
+getters()
+setters()
DeliveryService
-users: Map<String>
-menuItems: List<MenuItem>
-compositeProductString: String
-orderId: int
-orders: Map<Order>
+DeliveryService()
+registerUser(User): void
+logInUser(String, String, Role): void
+importProducts(): void
+addBaseProduct(MenuItem): void
+updateBaseProduct(MenuItem): void
+deleteBaseProduct(String): void
+addCompositeProduct(String, String): void
+searchProducts(String, String, String, String, String, String, String): List<MenuItem>
+createOrder(String, String): void
+createBill(Order): void
+generateReport1(int, int): void
+generateReport2(int): void
+generateReport3(int, int): void
+generateReport4(int, int, int): void
#isWellFormed(): boolean
+createProductsTabel(List): JTable
+createCProductString(String): string
+extractProducts(CompositeProduct): void
+idGenerator(): void
+getters()
+setters()
«enumeration»
Role
ADMINISTRATOR
CLIENT
EMPLOYEE
Serializable
Base Product
+BaseProduct(String, double, int, int, int, int, int)
+computePrice(): int
Order
-orderID: int
-clientID: String
-orderDate: LocalDateTime
-products: String
-price: int
+Order(int, String, LocalDateTime, String, int)
+equals(Object): boolean
+hashCode(): int
+getters()
+setters()
Composite Product
-compositeProduct: ArrayList<MenuItem>
+CompositeProduct(String, double, int, int, int, int, int, ArrayList)
+computePrice(): int
+getter()
+setter()
[3]

## 3.3 GUI Design



Food Delivery Management

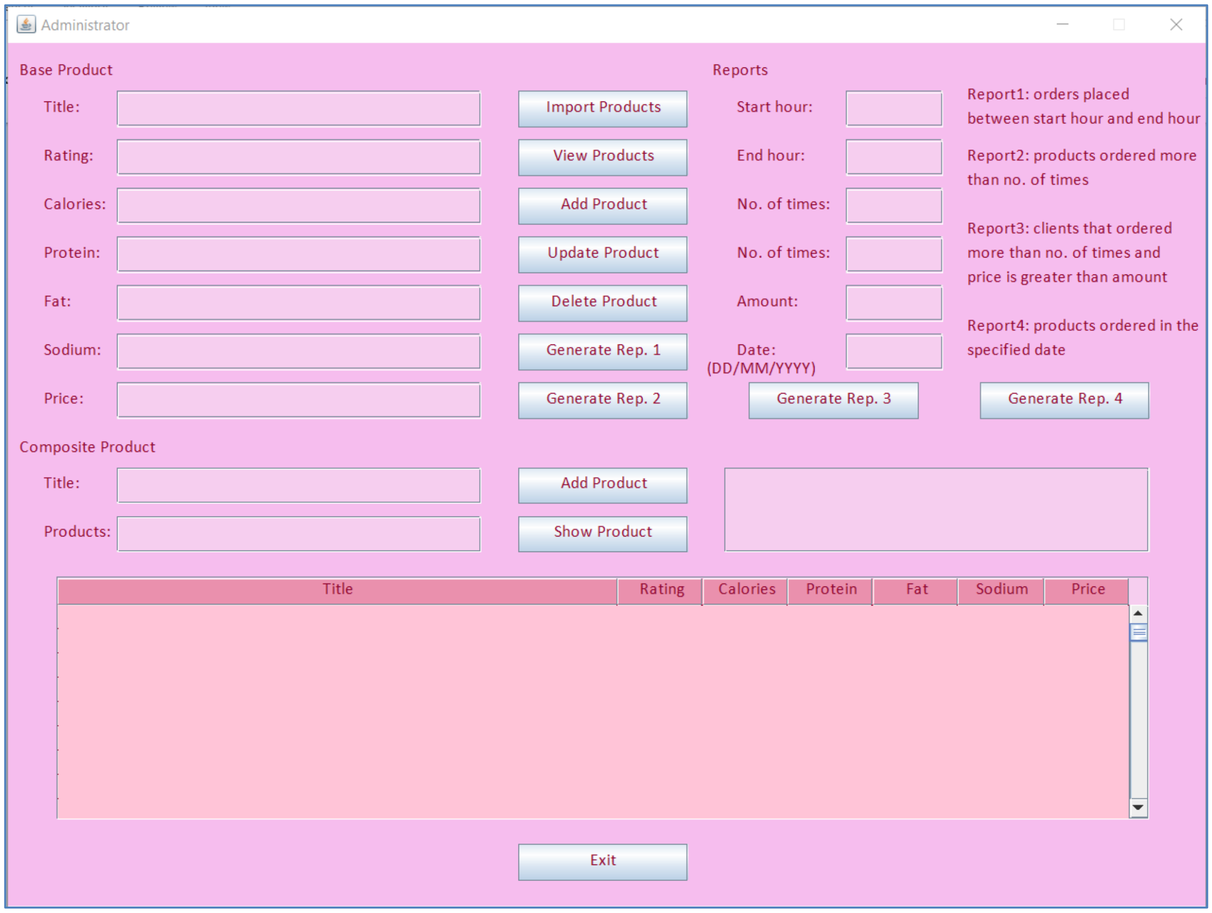
FOOD DELIVERY MANAGEMENT APPLICATION

Username:

Password:

Register Admin | Register Employee | Register Client

Log In Admin | Log In Employee | Log In Client



Administrator

Base Product

Title:

Rating:

Calories:

Protein:

Fat:

Sodium:

Price:

Import Products | View Products | Add Product | Update Product | Delete Product | Generate Rep. 1 | Generate Rep. 2

Reports

Start hour:

End hour:

No. of times:

No. of times:

Amount:

Date: (DD/MM/YYYY)

Report1: orders placed between start hour and end hour

Report2: products ordered more than no. of times

Report3: clients that ordered more than no. of times and price is greater than amount

Report4: products ordered in the specified date

Generate Rep. 3 | Generate Rep. 4

Composite Product

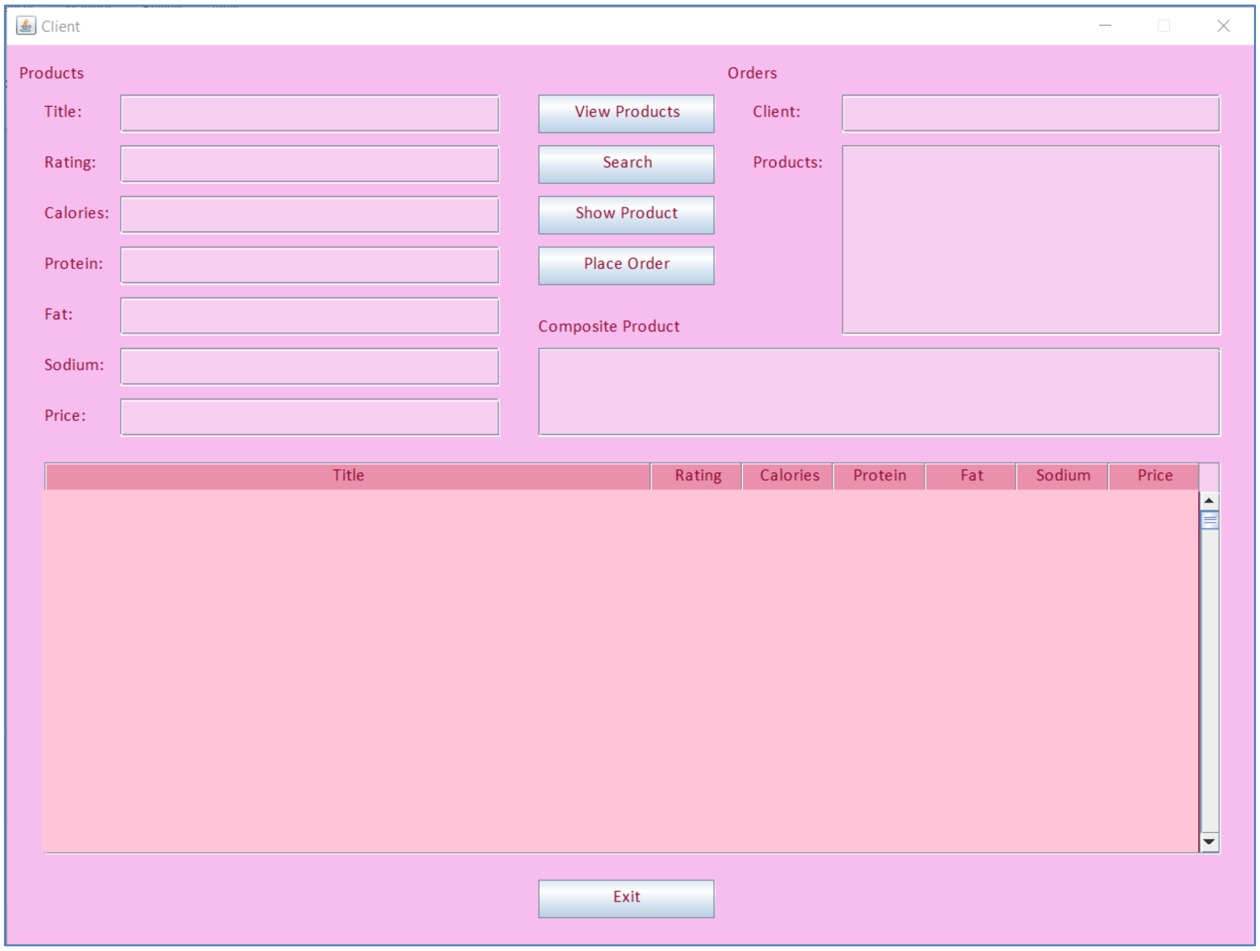
Title:

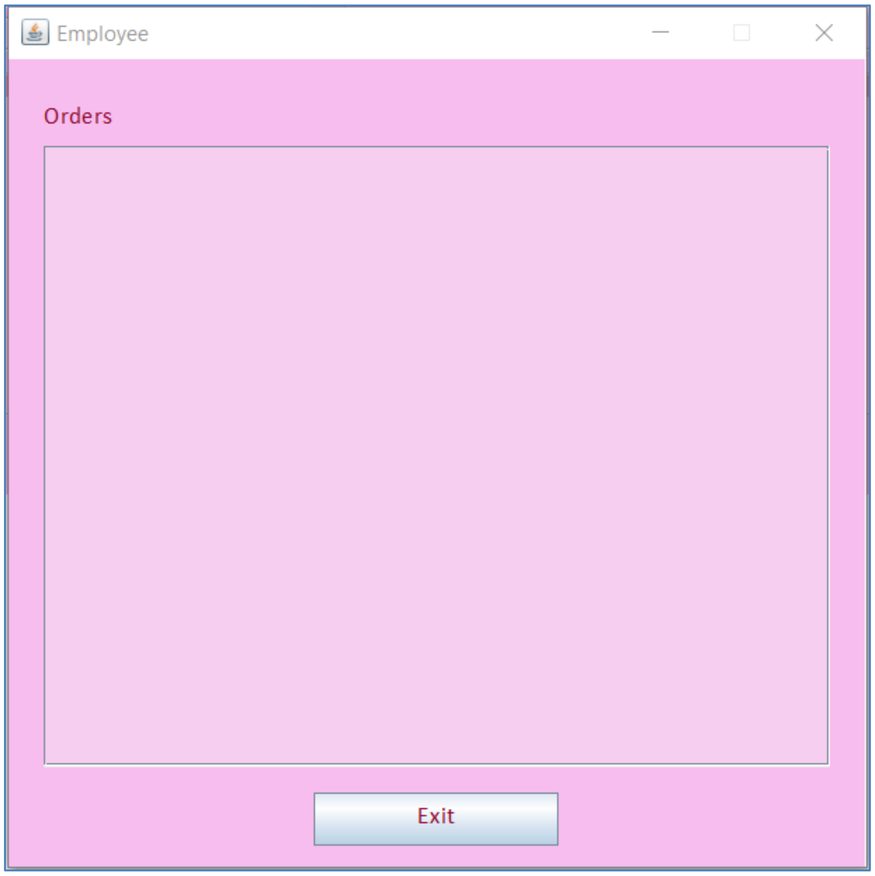
Products:

Add Product | Show Product

| Title | Rating | Calories | Protein | Fat | Sodium | Price |
|---|---|---|---|---|---|---|

Exit


Client
Products
Title:
Rating:
Calories:
Protein:
Fat:
Sodium:
Price:
View Products
Search
Show Product
Place Order
Orders
Client:
Products:
Composite Product
Title
Rating
Calories
Protein
Fat
Sodium
Price
Exit


Employee
Orders
Exit

## 3.4 Data Structures

- **ArrayList** – to store the list of menu items in the CompositeProduct and in DeliveryService
  – to store the list of menu items in the CompositeProduct
- **Map** – HashMap<String, User> to store the users uniquely
  – HashMap<Order, ArrayList<MenuItem>> to store the orders uniquely associated with the list of ordered products. For Order the *equals* and *hashCode* methods were overridden.

## 4. Implementation

- equals() and hashCode() for Order

```
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Order order = (Order) o;
    return orderID == order.orderID &&
            price == order.price &&
            Objects.equals(clientID, order.clientID) &&
            Objects.equals(orderDate, order.orderDate) &&
            Objects.equals(products, order.products);
}

@Override
public int hashCode() {
    return orderID*13 + orderDate.getDayOfMonth()*7 + orderDate.getHour()*11;
}
```

- Serializator

```
public void serialize(DeliveryService deliveryService) {
    try {
        FileOutputStream file = new FileOutputStream(filename);
        ObjectOutputStream out = new ObjectOutputStream(file);
        out.writeObject(deliveryService);
        out.close();
        file.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
public DeliveryService deserialize() {
    DeliveryService deliveryService = new DeliveryService();
    try {
        FileInputStream file = new FileInputStream(filename);
        if (file.available() > 0) {
            ObjectInputStream in = new ObjectInputStream(file);
            deliveryService = (DeliveryService) in.readObject();
            in.close();
            file.close();
        }
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }
    return deliveryService;
}
```

- Search products

```
@Override
public List<MenuItem> searchProducts(String title, String rating, String calories, String protein, String fat, String
sodium, String price) {
    assert isWellFormed();

    List<MenuItem> filteredItems = new ArrayList<>(menuItems);
    Stream<MenuItem> stream = filteredItems.stream();
    try {
        if (!title.isEmpty())
            stream = stream.filter(product -> product.getTitle().contains(title));
        if (rating.compareTo("") != 0)
            stream = stream.filter(product -> product.getRating() == Double.parseDouble(rating));
        if (calories.compareTo("") != 0)
            stream = stream.filter(product -> product.getCalories() == Integer.parseInt(calories));
        if (protein.compareTo("") != 0)
            stream = stream.filter(product -> product.getProtein() == Integer.parseInt(protein));
        if (fat.compareTo("") != 0)
            stream = stream.filter(product -> product.getFat() == Integer.parseInt(fat));
        if (sodium.compareTo("") != 0)
            stream = stream.filter(product -> product.getSodium() == Integer.parseInt(sodium));
        if (price.compareTo("") != 0)
            stream = stream.filter(product -> product.getPrice() == Integer.parseInt(price));

        assert isWellFormed();
        return stream.collect(Collectors.toList());
    } catch (NumberFormatException e) {
        throw new InputValidationFailedException("Input is not a number!");
    }
```

- Report 2: the products ordered more than a specified number of times so far.

```
@Override
public void generateReport2(int nbOfTimes) {
    assert nbOfTimes >= 0;
    assert isWellFormed();

    Stream<MenuItem> stream = Stream.of();
    for(Order order : orders.keySet()) {
        stream = Stream.concat(stream, orders.get(order).stream());
    }
    List<MenuItem> products = stream.collect(Collectors.toList());
    List<MenuItem> products2 = products.stream()
                                .filter(item -> Collections.frequency(products, item) > nbOfTimes)
                                .distinct()
                                .collect(Collectors.toList());
    FileWriter writer;
    try {
        writer = new FileWriter("Report2.txt");
        writer.append("Products ordered more then " + nbOfTimes + " times: \n\n");
        for(MenuItem item : products2) {
            writer.append(item.getTitle() + "\n");
        }
        writer.close();
        assert isWellFormed();
    } catch(IOException e) {
        e.printStackTrace();
    }
}
```

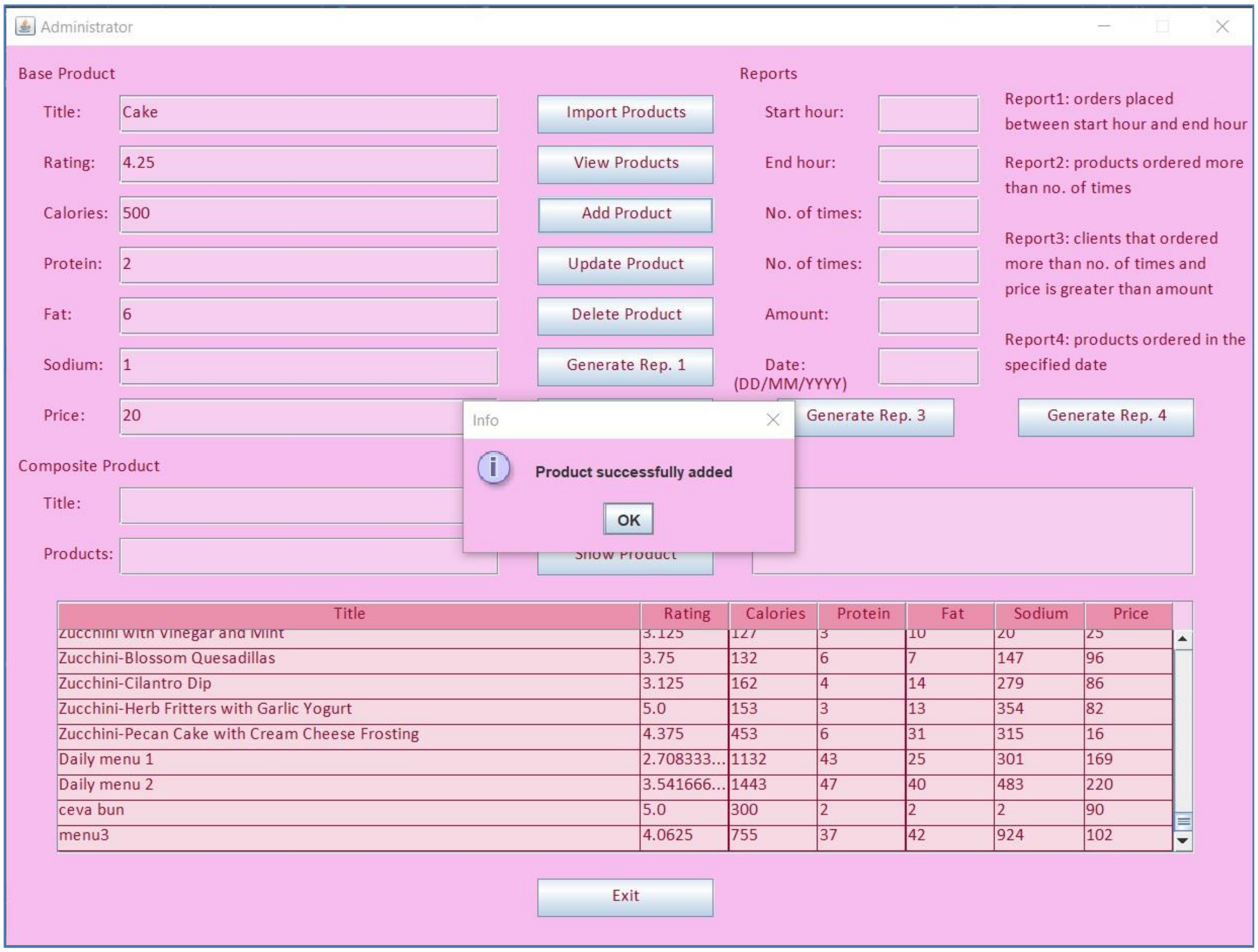
- Report 4: the products ordered within a specified day with the number of times they have been ordered

```
@Override
public void generateReport4(int day, int month, int year) {
    assert day >= 0 && day <= 31;
    assert month >= 0 && month <= 12;
    assert year >= 0 && year <= LocalDateTime.now().getYear();
    assert isWellFormed();

    Stream<MenuItem> stream = Stream.of();
    for(Order order : orders.keySet()) {
        if (order.getOrderDate().getDayOfMonth() == day && order.getOrderDate().getMonthValue() == month &&
                order.getOrderDate().getYear() == year)
        stream = Stream.concat(stream, orders.get(order).stream());
    }
    List<MenuItem> products = stream.collect(Collectors.toList());
    Map<MenuItem, Long> myProducts = products.stream().collect(Collectors.groupingBy(e -> e,
Collectors.counting()));
```

```
    FileWriter writer;
    try {
        writer = new FileWriter("Report4.txt");
        writer.append("Products ordered on "+day+"/"+month+"/"+year+"\n\n");
        for(MenuItem product : myProducts.keySet()) {
            writer.append(product.getTitle() + " " + myProducts.get(product) + "\n");
        }
        writer.close();
        assert isWellFormed();
    } catch(IOException e) {
        e.printStackTrace();
    }
}
```

## 5. <u>Results</u>

- Add Base Product



Administrator

Base Product

Title: Cake

Rating: 4.25

Calories: 500

Protein: 2

Fat: 6

Sodium: 1

Price: 20

Import Products | View Products | Add Product | Update Product | Delete Product | Generate Rep. 1

Reports

Start hour:

End hour:

No. of times:

No. of times:

Amount:

Date: (DD/MM/YYYY)

Report1: orders placed between start hour and end hour

Report2: products ordered more than no. of times

Report3: clients that ordered more than no. of times and price is greater than amount

Report4: products ordered in the specified date

Generate Rep. 3 | Generate Rep. 4

Composite Product

Title:

Products:

Show Product

Info

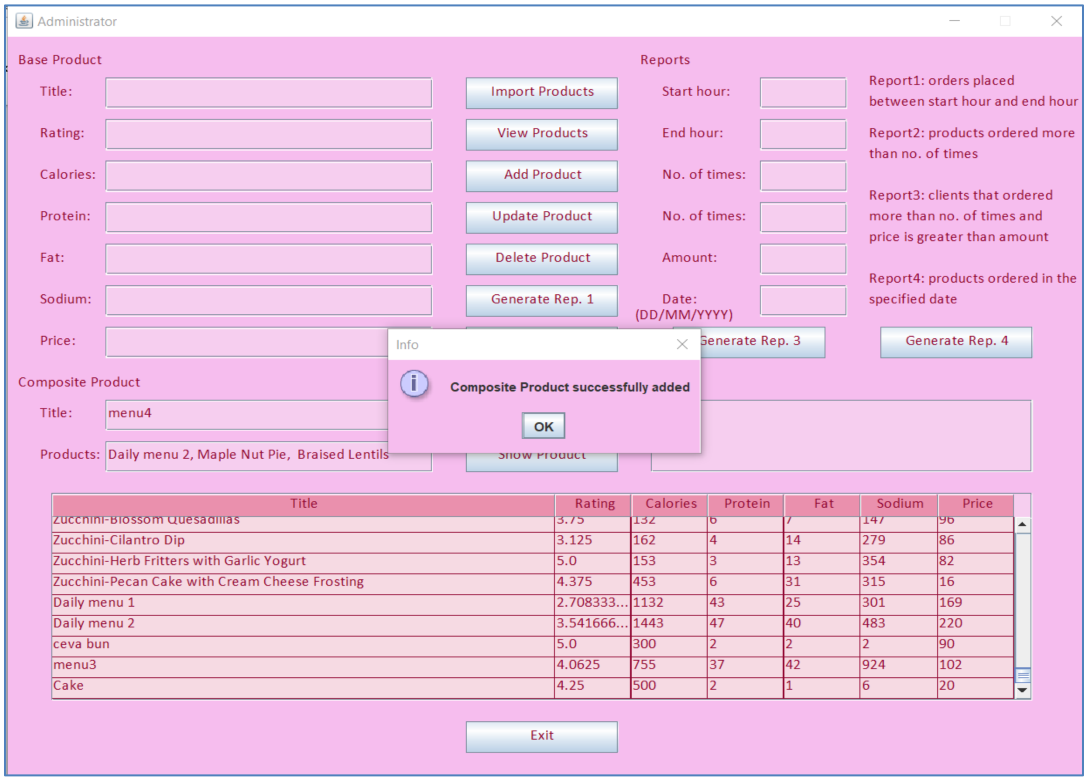
Product successfully added

OK

| Title | Rating | Calories | Protein | Fat | Sodium | Price |
|---|---|---|---|---|---|---|
| Zucchini with Vinegar and Mint | 3.125 | 127 | 3 | 10 | 20 | 25 |
| Zucchini-Blossom Quesadillas | 3.75 | 132 | 6 | 7 | 147 | 96 |
| Zucchini-Cilantro Dip | 3.125 | 162 | 4 | 14 | 279 | 86 |
| Zucchini-Herb Fritters with Garlic Yogurt | 5.0 | 153 | 3 | 13 | 354 | 82 |
| Zucchini-Pecan Cake with Cream Cheese Frosting | 4.375 | 453 | 6 | 31 | 315 | 16 |
| Daily menu 1 | 2.708333... | 1132 | 43 | 25 | 301 | 169 |
| Daily menu 2 | 3.541666... | 1443 | 47 | 40 | 483 | 220 |
| ceva bun | 5.0 | 300 | 2 | 2 | 2 | 90 |
| menu3 | 4.0625 | 755 | 37 | 42 | 924 | 102 |

Exit

- Add Composite Product



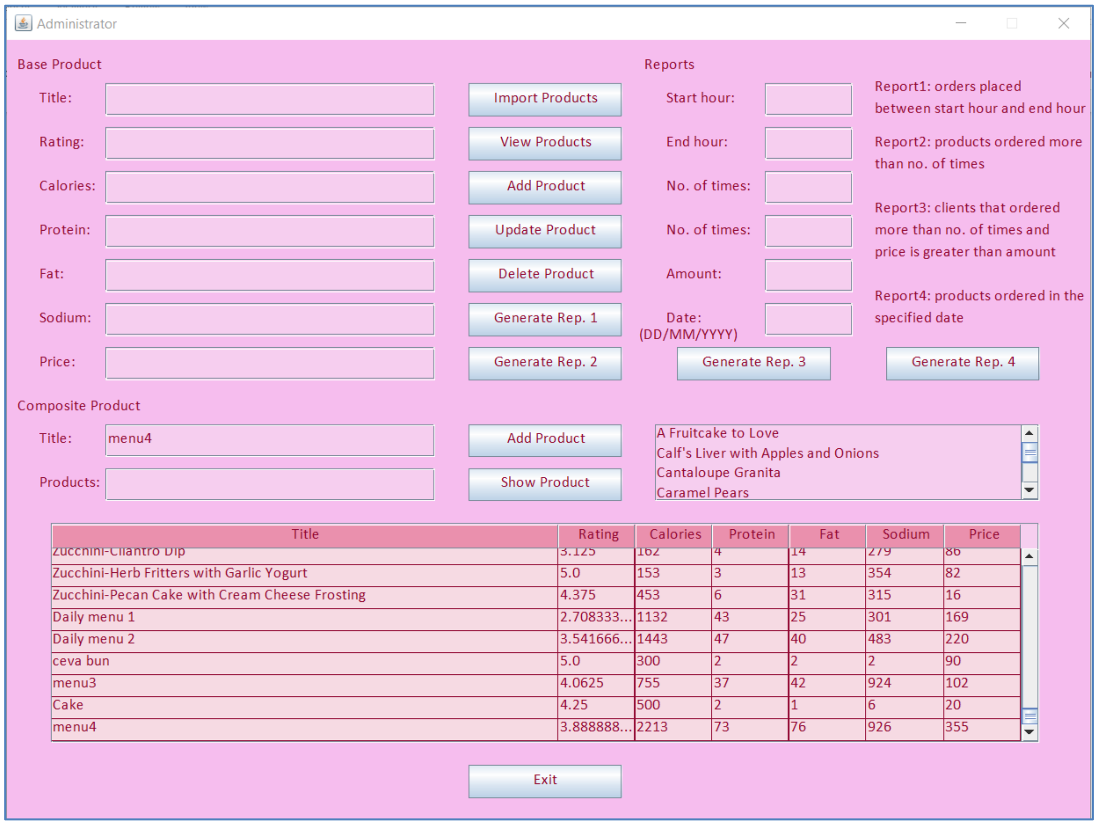
Title: menu4

Products: Daily menu 2, Maple Nut Pie, Braised Lentils

Composite Product successfully added

| Title | Rating | Calories | Protein | Fat | Sodium | Price |
|---|---|---|---|---|---|---|
| Zucchini-Blossom Quesadillas | 3.75 | 132 | 6 | 7 | 147 | 96 |
| Zucchini-Cilantro Dip | 3.125 | 162 | 4 | 14 | 279 | 86 |
| Zucchini-Herb Fritters with Garlic Yogurt | 5.0 | 153 | 3 | 13 | 354 | 82 |
| Zucchini-Pecan Cake with Cream Cheese Frosting | 4.375 | 453 | 6 | 31 | 315 | 16 |
| Daily menu 1 | 2.708333... | 1132 | 43 | 25 | 301 | 169 |
| Daily menu 2 | 3.541666... | 1443 | 47 | 40 | 483 | 220 |
| ceva bun | 5.0 | 300 | 2 | 2 | 2 | 90 |
| menu3 | 4.0625 | 755 | 37 | 42 | 924 | 102 |
| Cake | 4.25 | 500 | 2 | 1 | 6 | 20 |

- Show Product



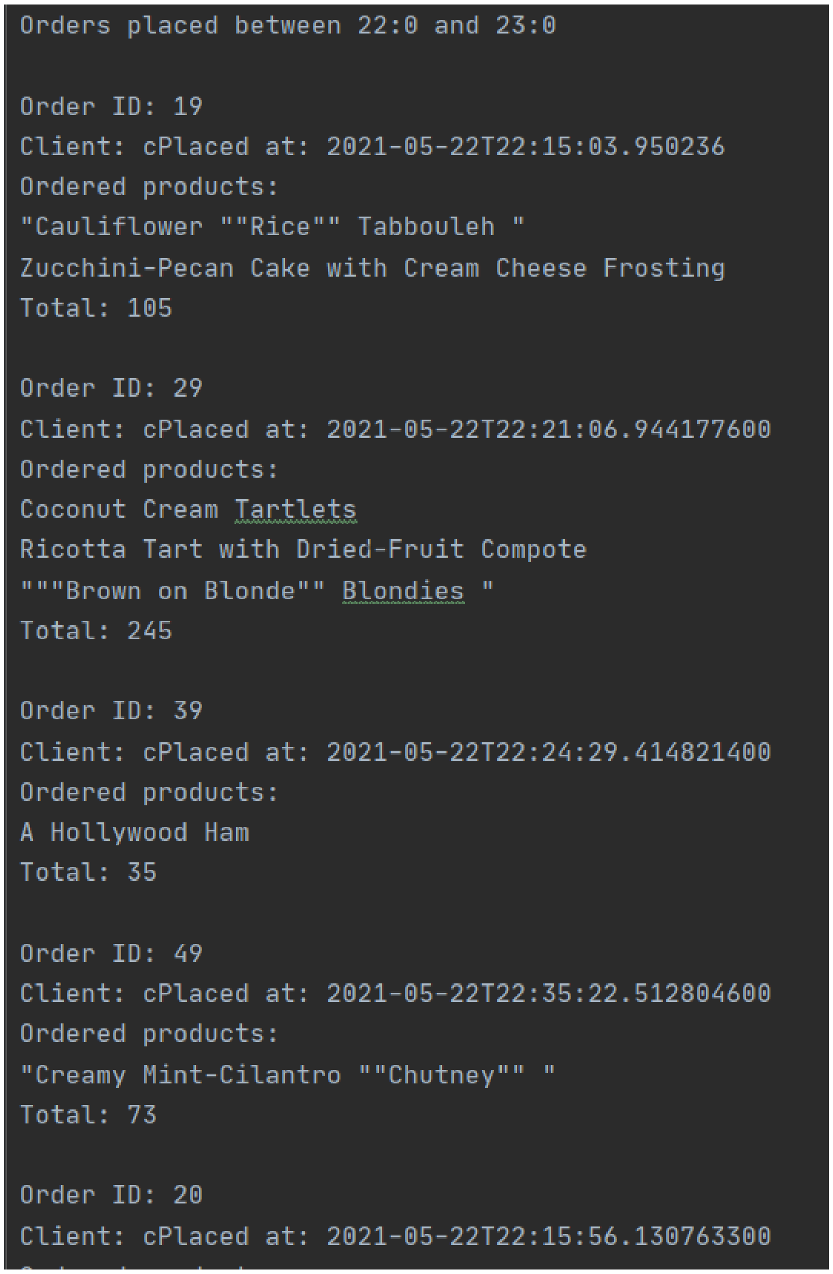
Title: menu4

Products:

A Fruitcake to Love
Calf's Liver with Apples and Onions
Cantaloupe Granita
Caramel Pears

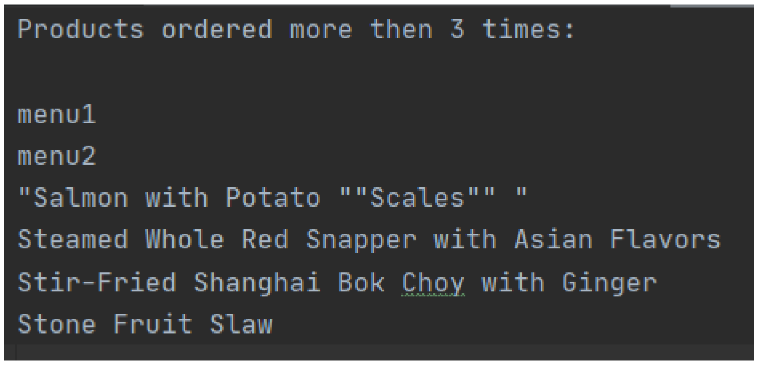
| Title | Rating | Calories | Protein | Fat | Sodium | Price |
|---|---|---|---|---|---|---|
| Zucchini-Cilantro Dip | 3.125 | 162 | 4 | 14 | 279 | 86 |
| Zucchini-Herb Fritters with Garlic Yogurt | 5.0 | 153 | 3 | 13 | 354 | 82 |
| Zucchini-Pecan Cake with Cream Cheese Frosting | 4.375 | 453 | 6 | 31 | 315 | 16 |
| Daily menu 1 | 2.708333... | 1132 | 43 | 25 | 301 | 169 |
| Daily menu 2 | 3.541666... | 1443 | 47 | 40 | 483 | 220 |
| ceva bun | 5.0 | 300 | 2 | 2 | 2 | 90 |
| menu3 | 4.0625 | 755 | 37 | 42 | 924 | 102 |
| Cake | 4.25 | 500 | 2 | 1 | 6 | 20 |
| menu4 | 3.888888... | 2213 | 73 | 76 | 926 | 355 |

- Report 1



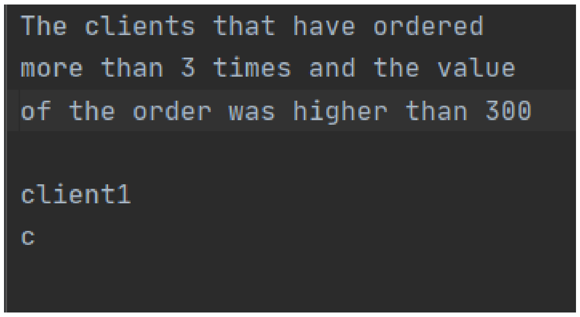
```
Orders placed between 22:0 and 23:0

Order ID: 19
Client: cPlaced at: 2021-05-22T22:15:03.950236
Ordered products:
"Cauliflower ""Rice"" Tabbouleh "
Zucchini-Pecan Cake with Cream Cheese Frosting
Total: 105

Order ID: 29
Client: cPlaced at: 2021-05-22T22:21:06.944177600
Ordered products:
Coconut Cream Tartlets
Ricotta Tart with Dried-Fruit Compote
"""Brown on Blonde"" Blondies "
Total: 245

Order ID: 39
Client: cPlaced at: 2021-05-22T22:24:29.414821400
Ordered products:
A Hollywood Ham
Total: 35

Order ID: 49
Client: cPlaced at: 2021-05-22T22:35:22.512804600
Ordered products:
"Creamy Mint-Cilantro ""Chutney"" "
Total: 73

Order ID: 20
Client: cPlaced at: 2021-05-22T22:15:56.130763300
```

Report 2



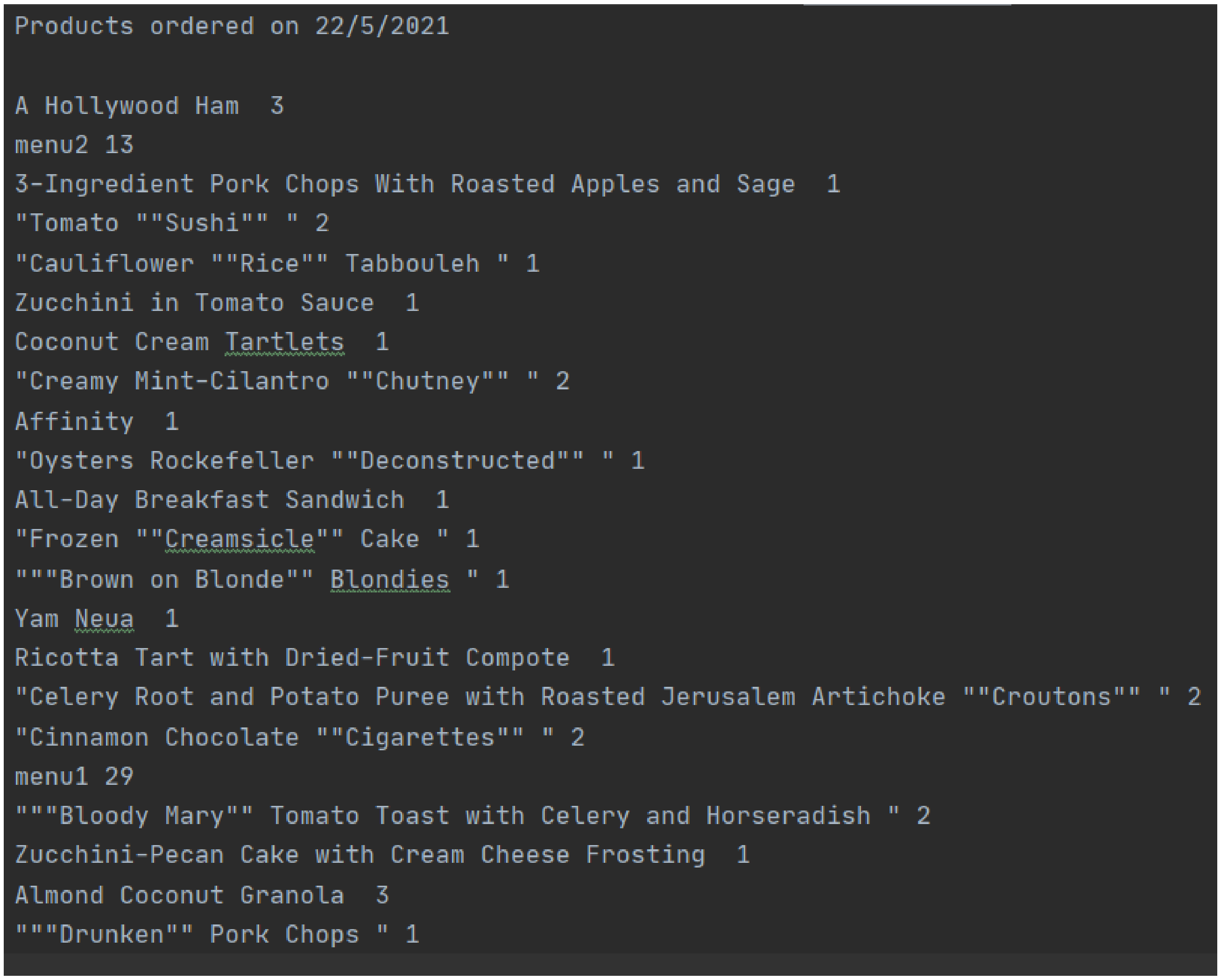
```
Products ordered more then 3 times:

menu1
menu2
"Salmon with Potato ""Scales"" "
Steamed Whole Red Snapper with Asian Flavors
Stir-Fried Shanghai Bok Choy with Ginger
Stone Fruit Slaw
```

Report 3



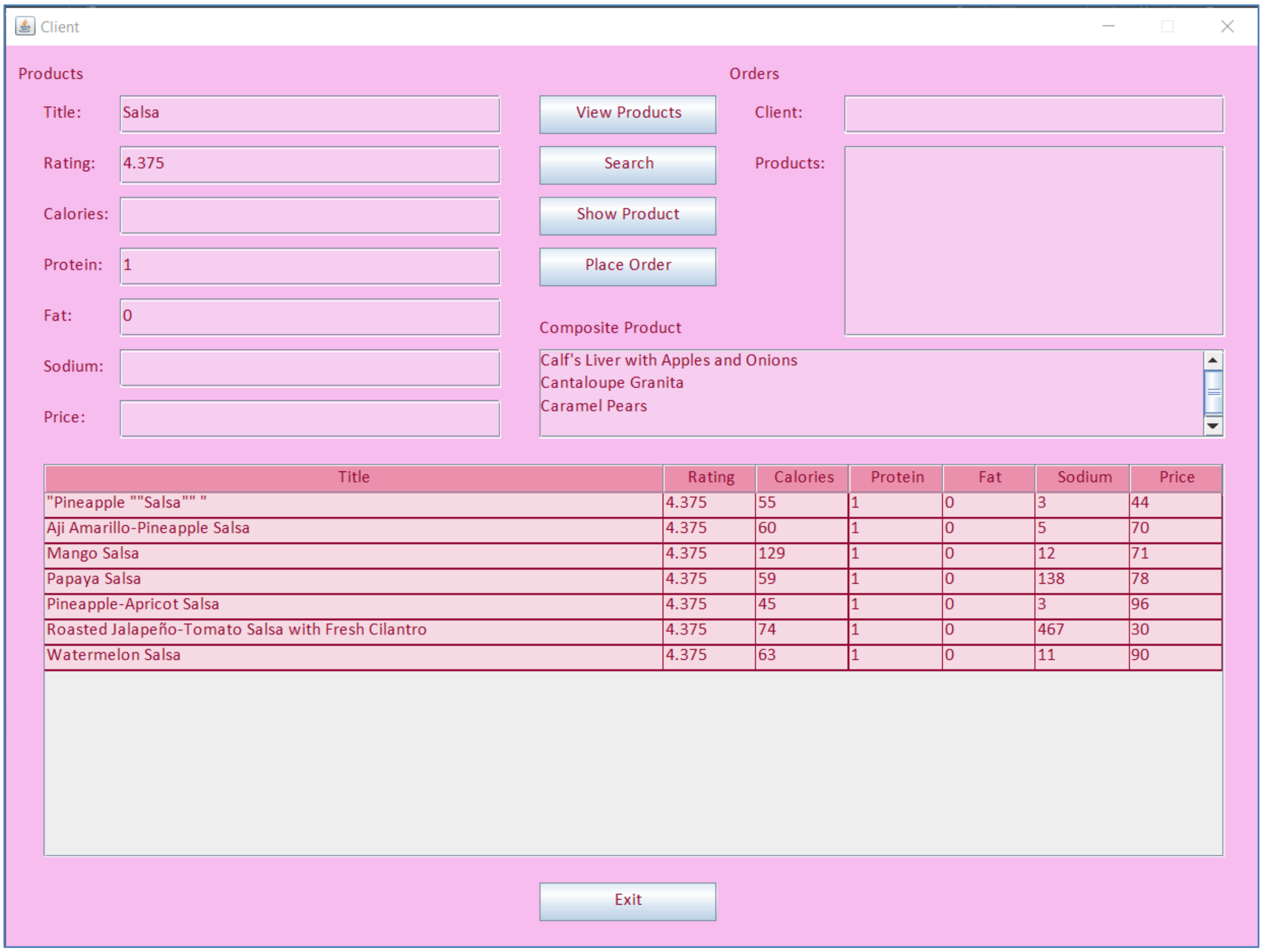
```
The clients that have ordered
more than 3 times and the value
of the order was higher than 300

client1
c
```

- Report 4



```
Products ordered on 22/5/2021

A Hollywood Ham  3
menu2 13
3-Ingredient Pork Chops With Roasted Apples and Sage  1
"Tomato ""Sushi"" " 2
"Cauliflower ""Rice"" Tabbouleh " 1
Zucchini in Tomato Sauce  1
Coconut Cream Tartlets  1
"Creamy Mint-Cilantro ""Chutney"" " 2
Affinity  1
"Oysters Rockefeller ""Deconstructed"" " 1
All-Day Breakfast Sandwich  1
"Frozen ""Creamsicle"" Cake " 1
"""Brown on Blonde"" Blondies " 1
Yam Neua  1
Ricotta Tart with Dried-Fruit Compote  1
"Celery Root and Potato Puree with Roasted Jerusalem Artichoke ""Croutons"" " 2
"Cinnamon Chocolate ""Cigarettes"" " 2
menu1 29
"""Bloody Mary"" Tomato Toast with Celery and Horseradish " 2
Zucchini-Pecan Cake with Cream Cheese Frosting  1
Almond Coconut Granola  3
"""Drunken"" Pork Chops " 1
```

- Search



Client

Products

Title: Salsa

Rating: 4.375

Calories:

Protein: 1

Fat: 0

Sodium:

Price:

Orders

Client:

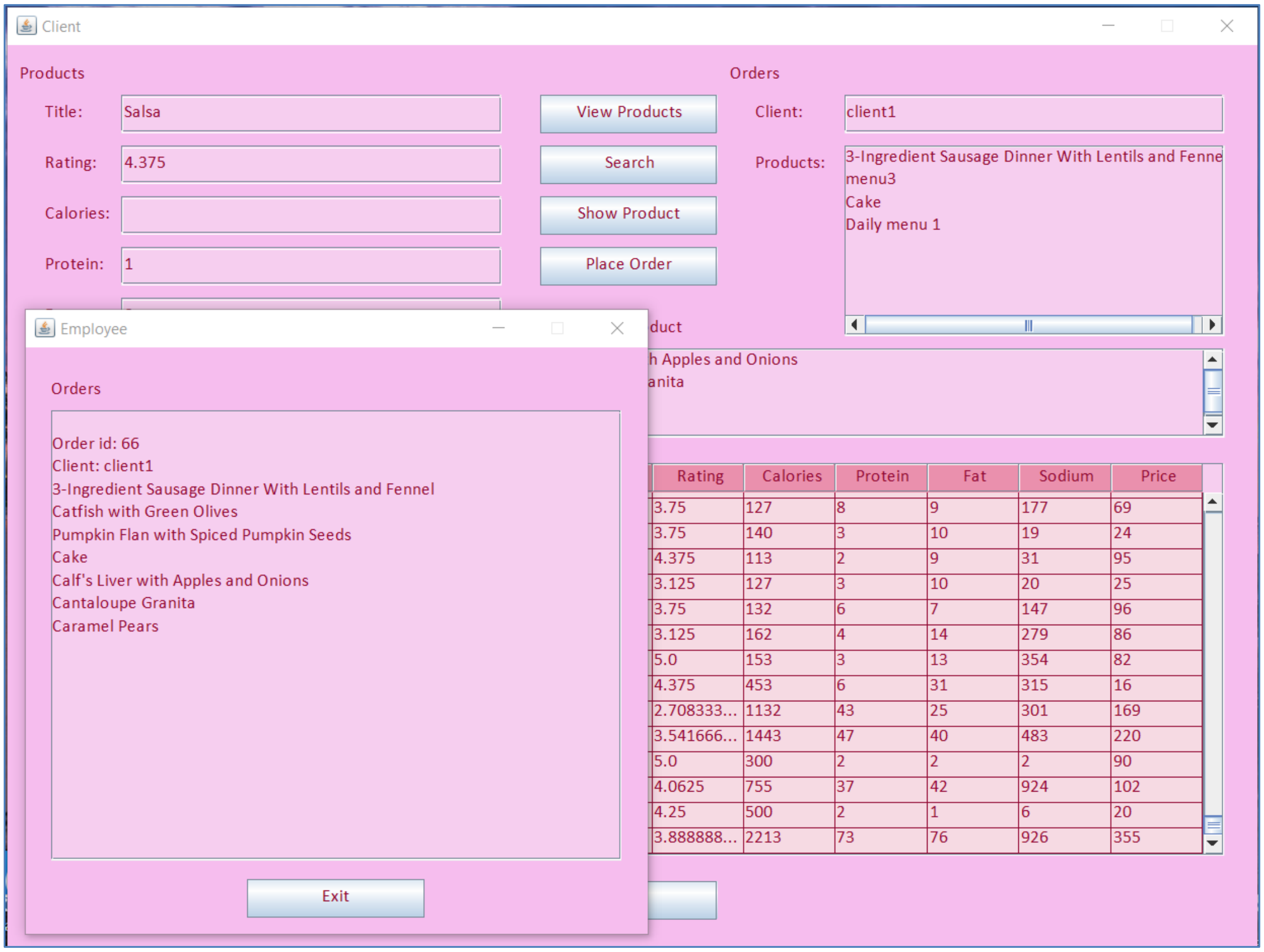
Products:

Composite Product

Calf's Liver with Apples and Onions
Cantaloupe Granita
Caramel Pears

| Title | Rating | Calories | Protein | Fat | Sodium | Price |
|---|---|---|---|---|---|---|
| "Pineapple ""Salsa"" " | 4.375 | 55 | 1 | 0 | 3 | 44 |
| Aji Amarillo-Pineapple Salsa | 4.375 | 60 | 1 | 0 | 5 | 70 |
| Mango Salsa | 4.375 | 129 | 1 | 0 | 12 | 71 |
| Papaya Salsa | 4.375 | 59 | 1 | 0 | 138 | 78 |
| Pineapple-Apricot Salsa | 4.375 | 45 | 1 | 0 | 3 | 96 |
| Roasted Jalapeño-Tomato Salsa with Fresh Cilantro | 4.375 | 74 | 1 | 0 | 467 | 30 |
| Watermelon Salsa | 4.375 | 63 | 1 | 0 | 11 | 90 |

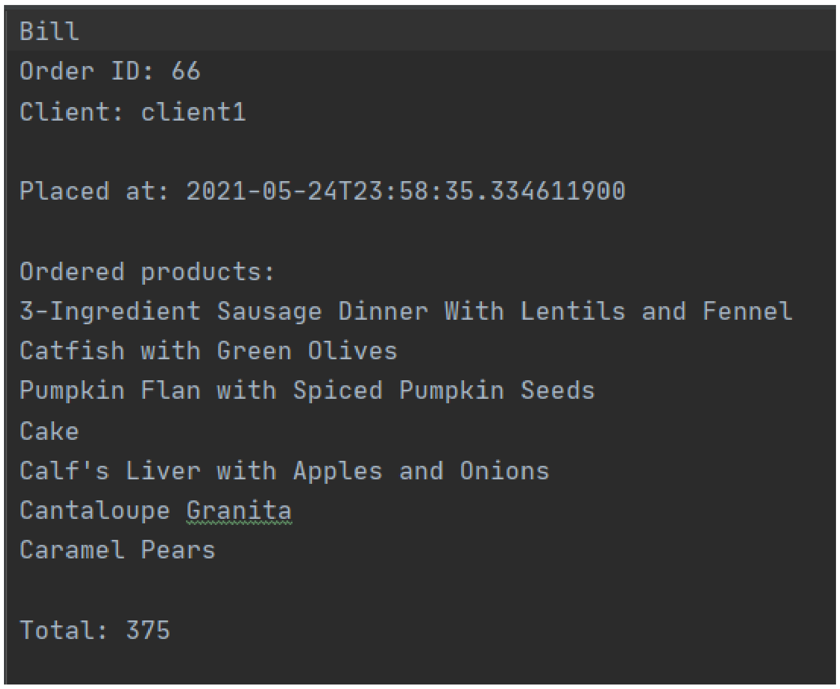
- Place Order



```
Order id: 66
Client: client1
3-Ingredient Sausage Dinner With Lentils and Fennel
Catfish with Green Olives
Pumpkin Flan with Spiced Pumpkin Seeds
Cake
Calf's Liver with Apples and Onions
Cantaloupe Granita
Caramel Pears
```

| Rating | Calories | Protein | Fat | Sodium | Price |
|---|---|---|---|---|---|
| 3.75 | 127 | 8 | 9 | 177 | 69 |
| 3.75 | 140 | 3 | 10 | 19 | 24 |
| 4.375 | 113 | 2 | 9 | 31 | 95 |
| 3.125 | 127 | 3 | 10 | 20 | 25 |
| 3.75 | 132 | 6 | 7 | 147 | 96 |
| 3.125 | 162 | 4 | 14 | 279 | 86 |
| 5.0 | 153 | 3 | 13 | 354 | 82 |
| 4.375 | 453 | 6 | 31 | 315 | 16 |
| 2.708333... | 1132 | 43 | 25 | 301 | 169 |
| 3.541666... | 1443 | 47 | 40 | 483 | 220 |
| 5.0 | 300 | 2 | 2 | 2 | 90 |
| 4.0625 | 755 | 37 | 42 | 924 | 102 |
| 4.25 | 500 | 2 | 1 | 6 | 20 |
| 3.888888... | 2213 | 73 | 76 | 926 | 355 |

- Bill.txt



```
Bill
Order ID: 66
Client: client1

Placed at: 2021-05-24T23:58:35.334611900

Ordered products:
3-Ingredient Sausage Dinner With Lentils and Fennel
Catfish with Green Olives
Pumpkin Flan with Spiced Pumpkin Seeds
Cake
Calf's Liver with Apples and Onions
Cantaloupe Granita
Caramel Pears

Total: 375
```

## 6. Conclusions

I consider that the user interface of the Food Delivery Management System is intuitive and easy to use and that the application does everything that is required and it can be useful for managing food orders.

The implementation had a lot of notions to consider, mostly relating to serialization, lambda expressions and stream processing, the composite design pattern, design by contract, the observer design pattern.

What I learned from this assignment was a deepening of the OOP concepts learned the last semester and working with GUIs, Maven etc. but also learning new things such as implementing serialization, managing lambda expressions and stream processing.

Future improvements of the application could be: a bigger GUI so that the longer data from the tables can be seen better; the possibility to manage composite products (update, delete) and the orders that were already placed (delete, update).

## 7. Bibliography

[1] Fundamental Programming Techniques – Lecture Slides, Assignment_4_Support_Presentation

[2] https://app.creately.com/diagram/

[3] https://staruml.io/

[4] https://www.baeldung.com/java-maps-streams

[5] Java - Serialization - Tutorialspoint

[6] https://stackoverflow.com/questions/505928/how-to-count-the-number-of-occurrences-of-an-element-in-a-list/2459753

[7] https://dzone.com/articles/how-to-read-a-big-csv-file-with-java-8-and-stream