DOCUMENTATATION

ASSIGNMENT 4: FOOD DELIVERY MANAGEMENT SYSTEM

STUDENT NAME: PELLE ANDREI

TABLE OF CONTENTS

1. Objectives

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:

  - 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 so far 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/prices.
- 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.

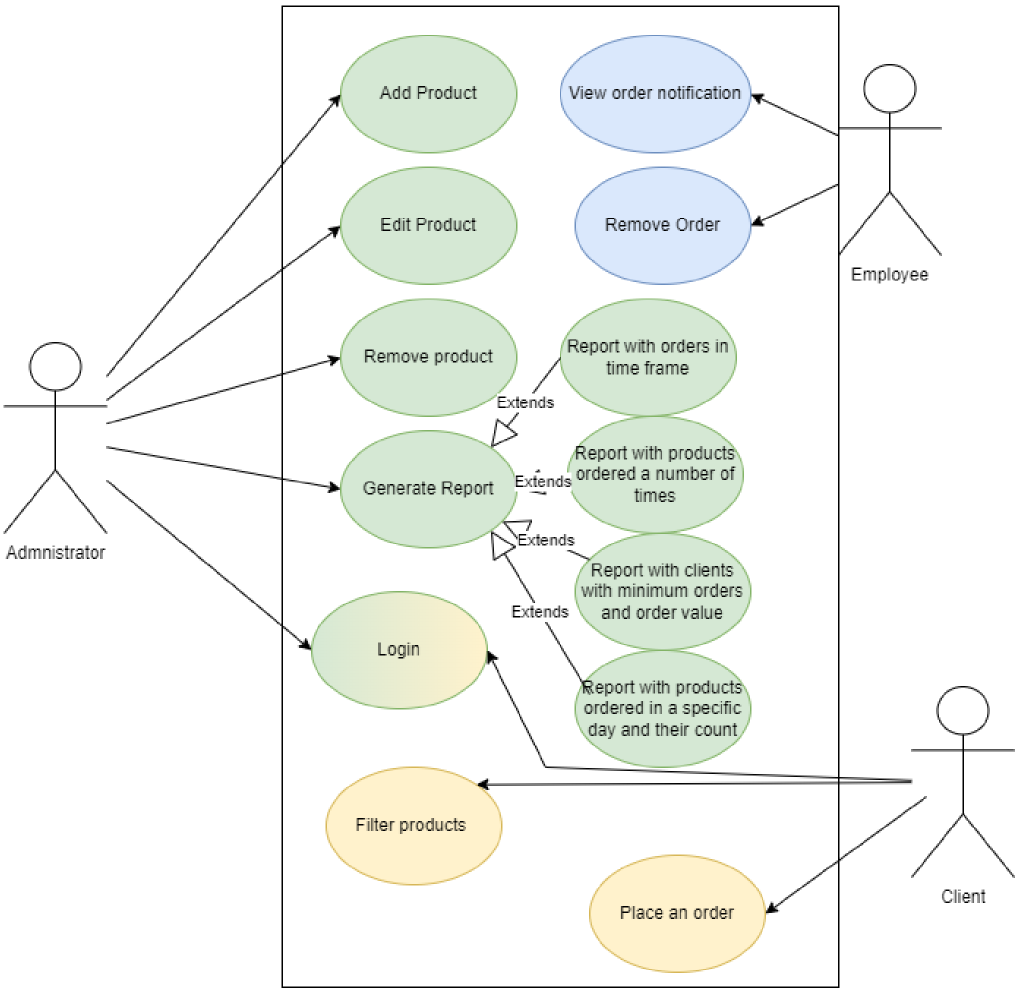
Breaking down the problem, I considered the following tasks:

- **Analyze and design an elegant solution** : One has to decide before starting work who is going to be using this application, what the use cases are and how all the components work together at a high level. (Chapter 2 and 3)
- **Building a user-friendly graphical user interface** that allows the client to register/login and use the application, the admin to manage products and the employee to be notified(Chapter 4)
- **Data modelling**: A food delivery management system is a complex task ;it must be modelled in a way that allows the operations to be performed easily and logically. (Chapter 3)
- **Input validation**: we must make sure the input is correct and display a message to the user if the input is somehow invalid or make sure that the input is not allowed to be entered at all in errored state. (Chapter 4)
- **Design by contract, streams**: The application must be built using design by contract and streams. These are not simple technologies, and their use requires know-how. (Chapter 4)
- **Test the application**: To ensure that the results are the ones we expect, the application needs to be tested. The results of some operations and bills have been appended to thus document. (Chapter 5)

2. Problem analysis, modelling, use cases

The following **functional requirements** can be defined:

- The food delivery management system should allow the administrator to add a new product/ menu item.
- The food delivery management system should allow the administrator to generate reports.
- The food delivery management system should allow the administrator to edit a product.
- The food delivery management system should allow the administrator to remove a product.
- The food delivery management system should allow the administrator and client to log in.
- The food delivery management system should allow the client to filter through the products based on some keywords and filters.
- The food delivery management system should allow the client to place an order.
- The food delivery management system should allow the employee to view order notifications.
- The food delivery management system should allow the employee to remove orders.


Add Product
View order notification
Edit Product
Remove Order
Employee
Remove product
Report with orders in time frame
Extends
Admnistrator
Generate Report
Extends
Report with products ordered a number of times
Extends
Report with clients with minimum orders and order value
Extends
Login
Report with products ordered in a specific day and their count
Filter products
Client
Place an order

The following **non-functional requirements** can be defined:

- The food delivery management system should be intuitive and easy to use by the user
- The food delivery management system should be easy to maintain
- The food delivery management system should be stable
- The food delivery management system be modifiable and allow for future development.

Use cases

Use Case 1: add product to menu

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 “Accept” button

5. The application saves the product's data

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

Alternative Sequence: Inserted name already exists
- The administrator inserts a name for the product that already exists
- The application displays an error message and requests the administrator to insert a name that is not already present in the table
- The scenario returns to step 3

Use case 2: edit product

Primary Actor: administrator
Main Success Scenario:
1. The administrator makes a selection from the table
2. The administrator selects the option to edit the product
3. The application will display a form in which the product details should be edited
4. The administrator inserts/changes the name of the product, the number of calories, proteins, fats, sodium, and its price
5. The administrator clicks on the "Accept" button
6. The application saves the product's data

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 4

Alternative Sequence: Edited name already exists
- The administrator inserts a name for the product that already exists
- The application displays an error message and requests the administrator to insert a name that is not already present in the table
- The scenario returns to step 4

Use case 3: delete product

Primary Actor: administrator
Main Success Scenario:
1. The administrator makes a selection from the table
2. The administrator selects the option to delete the product
3. The application will display a confirmation message if the admin is sure
4. The administrator clicks yes/ok
5. The application deletes the selected products

Alternative Sequence: No selection in table
- The administrator makes no selection in table
-The administrator clicks "remove"
- The application displays an error message and requests the administrator to select an entry from the table to delete
- The scenario returns to step 1

Alternative Sequence: Base product in composite error
- The administrator selects in table
-The administrator clicks "remove"
- The application displays an error message and informs the administrator that the current selection is invalid because one or more of the selected base products are currently part of one or more composite products that should be deleted first.
- The scenario returns to step 1

Use case 4: generate report

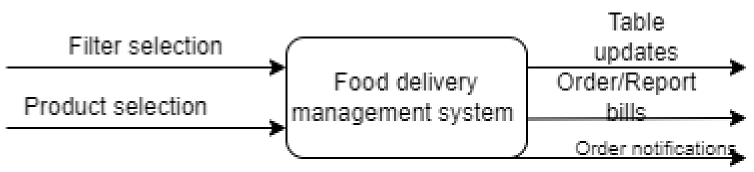
Primary Actor: administrator
Main Success Scenario:
2. The administrator selects the button to generate a report
3. The administrator enters the information for the filters
4. The administrator clicks generate report
5. The application updates the table
6.The application generates the corresponding report in a file.

Use case 5: add an order

Primary Actor: client
Main Success Scenario:
1. The client selects from the table
2. The administrator selects the option to create an order
3. The application will display a notification with the selected order contents
4. The application will notify the employee

Alternative Sequence: No selection in table
- The client makes no selection in table
-The client clicks create order
- The application displays an error message and requests the client to select an entry from the table to order
- The scenario returns to step 1

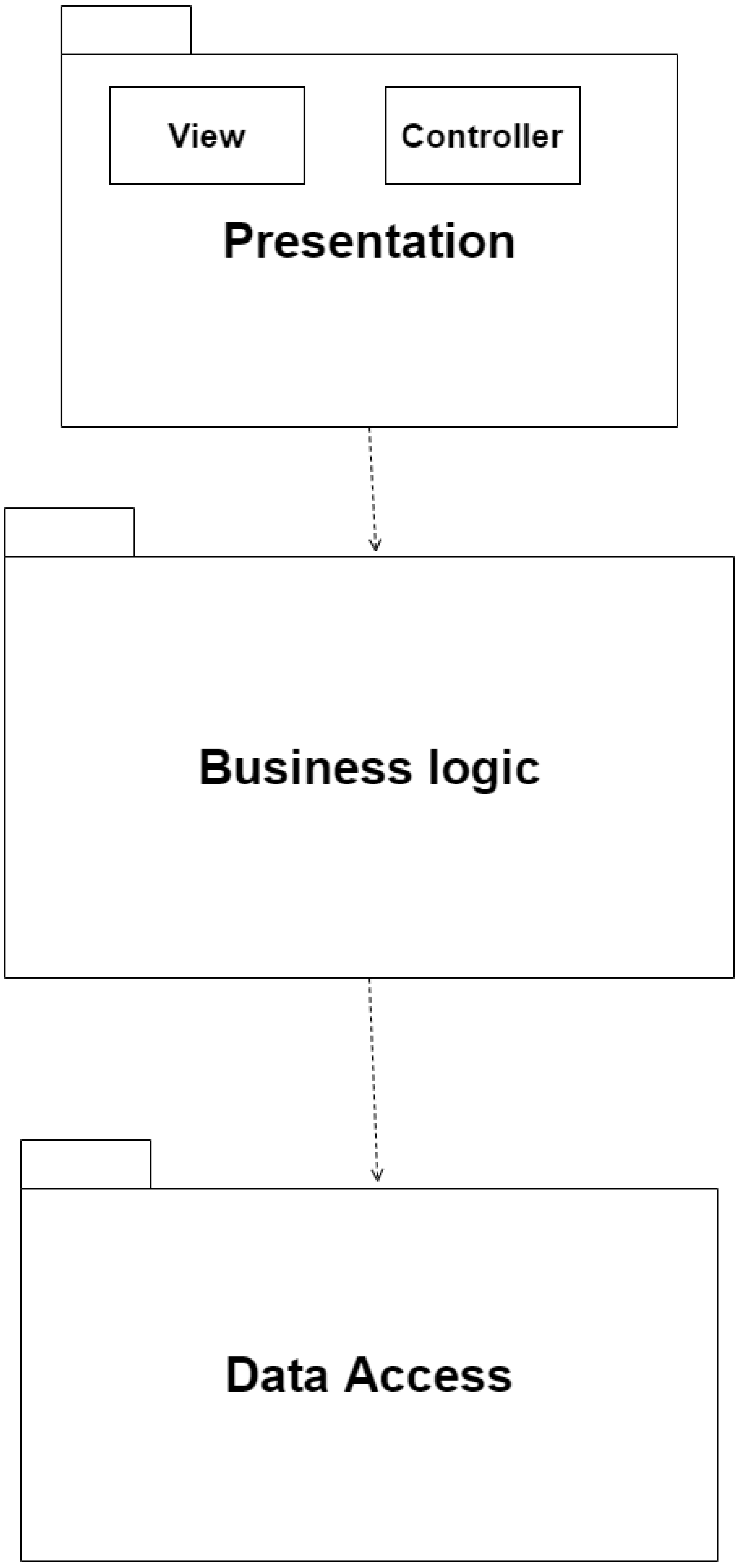
3. Design



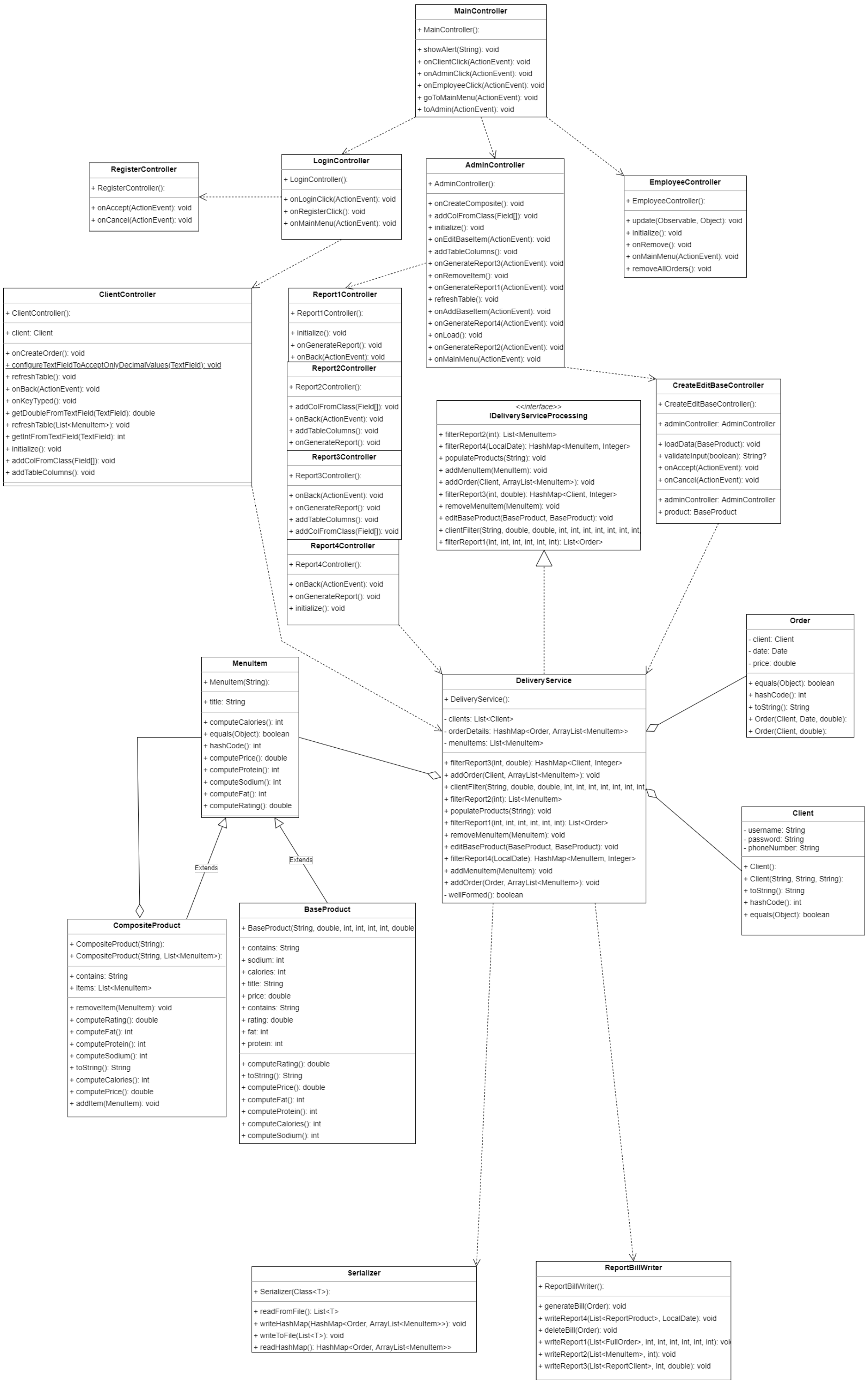
The food delivery management system: filter selection, product selection, table updates, report and order bills, order notifications.

Next, the design is broken down into packages:

- Controller: this package contains the classes that control the flow of the application. Because UI applications are event-driven, and JavaFX is no exception, this package decides what explicitly happens and waits for user input.
- Business_logic: Contains the business logic classes and the delivery service implementation. Only after the input is validated will the application proceed to make a request to the data access layer.
- Data: (or data access), this contains the classes that serialize data and write reports
- View (implicit): this package includes the definition for the UI elements written in Oracle's FXML language. It specifies the controller it is linked to for the action handlers.


View
Controller
Presentation
Business logic
Data Access

The packages are further broken down into classes. Such an illustration can be seen below:


MainController
+ MainController():
+ showAlert(String): void
+ onClientClick(ActionEvent): void
+ onAdminClick(ActionEvent): void
+ onEmployeeClick(ActionEvent): void
+ goToMainMenu(ActionEvent): void
+ toAdmin(ActionEvent): void
RegisterController
+ RegisterController():
+ onAccept(ActionEvent): void
+ onCancel(ActionEvent): void
LoginController
+ LoginController():
+ onLoginClick(ActionEvent): void
+ onRegisterClick(): void
+ onMainMenu(ActionEvent): void
AdminController
+ AdminController():
+ onCreateComposite(): void
+ addColFromClass(Field[]): void
+ initialize(): void
+ onEditBaseItem(ActionEvent): void
+ addTableColumns(): void
+ onGenerateReport3(ActionEvent): void
+ onRemoveItem(): void
+ onGenerateReport1(ActionEvent): void
+ refreshTable(): void
+ onAddBaseItem(ActionEvent): void
+ onGenerateReport4(ActionEvent): void
+ onLoad(): void
+ onGenerateReport2(ActionEvent): void
+ onMainMenu(ActionEvent): void
EmployeeController
+ EmployeeController():
+ update(Observable, Object): void
+ initialize(): void
+ onRemove(): void
+ onMainMenu(ActionEvent): void
+ removeAllOrders(): void
ClientController
+ ClientController():
+ client: Client
+ onCreateOrder(): void
+ configureTextFieldToAcceptOnlyDecimalValues(TextField): void
+ refreshTable(): void
+ onBack(ActionEvent): void
+ onKeyTyped(): void
+ getDoubleFromTextField(TextField): double
+ refreshTable(List<MenuItem>): void
+ getIntFromTextField(TextField): int
+ initialize(): void
+ addColFromClass(Field[]): void
+ addTableColumns(): void
Report1Controller
+ Report1Controller():
+ initialize(): void
+ onGenerateReport(): void
+ onBack(ActionEvent): void
Report2Controller
+ Report2Controller():
+ addColFromClass(Field[]): void
+ onBack(ActionEvent): void
+ addTableColumns(): void
+ onGenerateReport(): void
Report3Controller
+ Report3Controller():
+ onBack(ActionEvent): void
+ onGenerateReport(): void
+ addTableColumns(): void
+ addColFromClass(Field[]): void
Report4Controller
+ Report4Controller():
+ onBack(ActionEvent): void
+ onGenerateReport(): void
+ initialize(): void
<<interface>>
IDeliveryServiceProcessing
+ filterReport2(int): List<MenuItem>
+ filterReport4(LocalDate): HashMap<MenuItem, Integer>
+ populateProducts(String): void
+ addMenuItem(MenuItem): void
+ addOrder(Client, ArrayList<MenuItem>): void
+ filterReport3(int, double): HashMap<Client, Integer>
+ removeMenuItem(MenuItem): void
+ editBaseProduct(BaseProduct, BaseProduct): void
+ clientFilter(String, double, double, int, int, int, int, int, int,
+ filterReport1(int, int, int, int, int, int): List<Order>
CreateEditBaseController
+ CreateEditBaseController():
+ adminController: AdminController
+ loadData(BaseProduct): void
+ validateInput(boolean): String?
+ onAccept(ActionEvent): void
+ onCancel(ActionEvent): void
+ adminController: AdminController
+ product: BaseProduct
Order
- client: Client
- date: Date
- price: double
+ equals(Object): boolean
+ hashCode(): int
+ toString(): String
+ Order(Client, Date, double):
+ Order(Client, double):
MenuItem
+ MenuItem(String):
+ title: String
+ computeCalories(): int
+ equals(Object): boolean
+ hashCode(): int
+ computePrice(): double
+ computeProtein(): int
+ computeSodium(): int
+ computeFat(): int
+ computeRating(): double
DeliveryService
+ DeliveryService():
- clients: List<Client>
- orderDetails: HashMap<Order, ArrayList<MenuItem>>
- menuItems: List<MenuItem>
+ filterReport3(int, double): HashMap<Client, Integer>
+ addOrder(Client, ArrayList<MenuItem>): void
+ clientFilter(String, double, double, int, int, int, int, int, int,
+ filterReport2(int): List<MenuItem>
+ populateProducts(String): void
+ filterReport1(int, int, int, int, int, int): List<Order>
+ removeMenuItem(MenuItem): void
+ editBaseProduct(BaseProduct, BaseProduct): void
+ filterReport4(LocalDate): HashMap<MenuItem, Integer>
+ addMenuItem(MenuItem): void
+ addOrder(Order, ArrayList<MenuItem>): void
- wellFormed(): boolean
Client
- username: String
- password: String
- phoneNumber: String
+ Client():
+ Client(String, String, String):
+ toString(): String
+ hashCode(): int
+ equals(Object): boolean
Extends
Extends
CompositeProduct
+ CompositeProduct(String):
+ CompositeProduct(String, List<MenuItem>):
+ contains: String
+ items: List<MenuItem>
+ removeItem(MenuItem): void
+ computeRating(): double
+ computeFat(): int
+ computeProtein(): int
+ computeSodium(): int
+ toString(): String
+ computeCalories(): int
+ computePrice(): double
+ addItem(MenuItem): void
BaseProduct
+ BaseProduct(String, double, int, int, int, int, double
+ contains: String
+ sodium: int
+ calories: int
+ title: String
+ price: double
+ contains: String
+ rating: double
+ fat: int
+ protein: int
+ computeRating(): double
+ toString(): String
+ computePrice(): double
+ computeFat(): int
+ computeProtein(): int
+ computeCalories(): int
+ computeSodium(): int
Serializer
+ Serializer(Class<T>):
+ readFromFile(): List<T>
+ writeHashMap(HashMap<Order, ArrayList<MenuItem>>): void
+ writeToFile(List<T>): void
+ readHashMap(): HashMap<Order, ArrayList<MenuItem>>
ReportBillWriter
+ ReportBillWriter():
+ generateBill(Order): void
+ writeReport4(List<ReportProduct>, LocalDate): void
+ deleteBill(Order): void
+ writeReport1(List<FullOrder>, int, int, int, int, int, int): voi
+ writeReport2(List<MenuItem>, int): void
+ writeReport3(List<ReportClient>, int, double): void

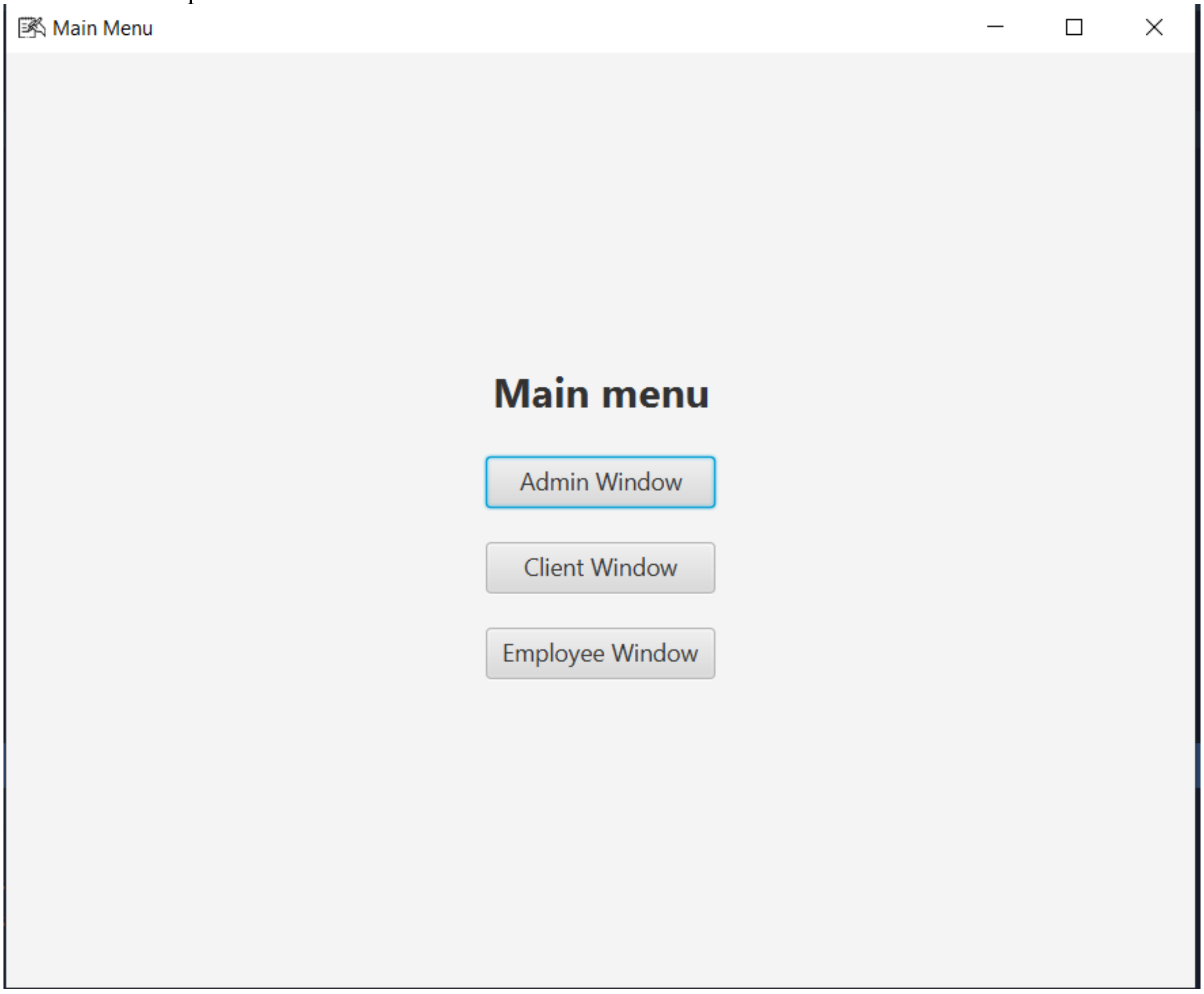
The data structure I considered for representing data in the application is an Order that has a client, date, and price. The key for the order is the date as it is very accurate. For the products, I chose to use the composite design pattern with the base product and composite product implementing the Menu Item interface. This common interface is the one we use when we have elements of that type and the calculation of based on the parameters.

The Controllers switch between the windows and also to filter selections and data from the table.

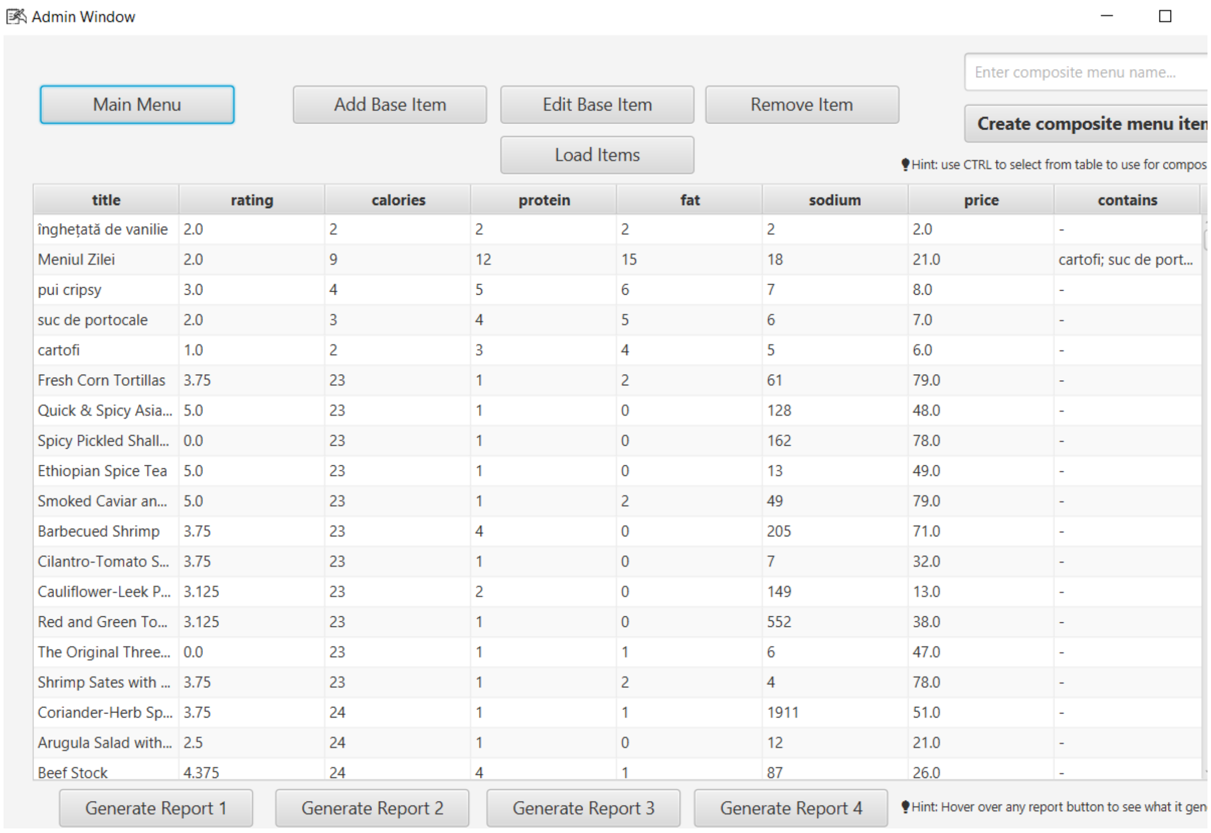
No special algorithms were used for this assignment. Calculating for menu items is done recursively in a tree-like manner, however this is handled by java.

4. Implementation

The main **GUI** window can be seen below. It consists of 3 simple buttons. Each button is tied to an action handler which will take the user to one of the 3 tables for operations: admin, client or employee. There is a TO-DO icon in the top left-hand corner.

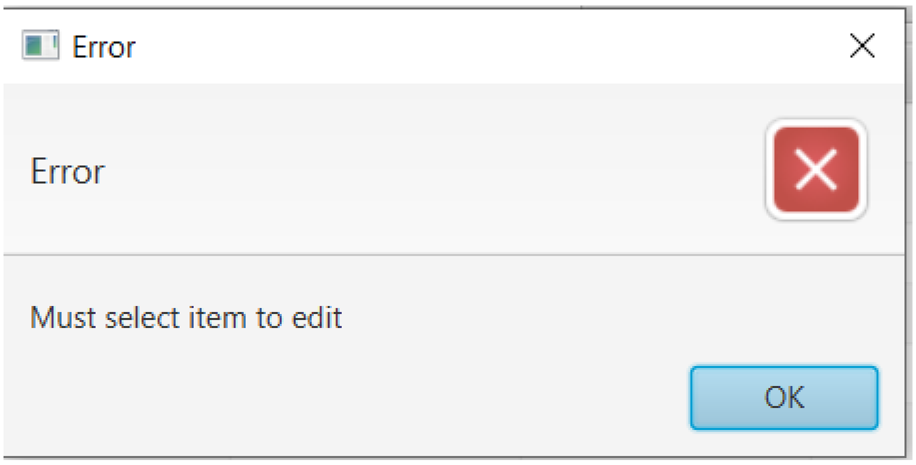


The administrator window can be seen below. Buttons in the top part do operations on products.


Admin Window
Enter composite menu name...
Main Menu
Add Base Item
Edit Base Item
Remove Item
Load Items
Create composite menu iten
Hint: use CTRL to select from table to use for compos
Generate Report 1
Generate Report 2
Generate Report 3
Generate Report 4
Hint: Hover over any report button to see what it gen

title	rating	calories	protein	fat	sodium	price	contains
înghețată de vanilie	2.0	2	2	2	2	2.0	-
Meniul Zilei	2.0	9	12	15	18	21.0	cartofi; suc de port...
pui cripsy	3.0	4	5	6	7	8.0	-
suc de portocale	2.0	3	4	5	6	7.0	-
cartofi	1.0	2	3	4	5	6.0	-
Fresh Corn Tortillas	3.75	23	1	2	61	79.0	-
Quick & Spicy Asia...	5.0	23	1	0	128	48.0	-
Spicy Pickled Shall...	0.0	23	1	0	162	78.0	-
Ethiopian Spice Tea	5.0	23	1	0	13	49.0	-
Smoked Caviar an...	5.0	23	1	2	49	79.0	-
Barbecued Shrimp	3.75	23	4	0	205	71.0	-
Cilantro-Tomato S...	3.75	23	1	0	7	32.0	-
Cauliflower-Leek P...	3.125	23	2	0	149	13.0	-
Red and Green To...	3.125	23	1	0	552	38.0	-
The Original Three...	0.0	23	1	1	6	47.0	-
Shrimp Sates with ...	3.75	23	1	2	4	78.0	-
Coriander-Herb Sp...	3.75	24	1	1	1911	51.0	-
Arugula Salad with...	2.5	24	1	0	12	21.0	-
Beef Stock	4.375	24	4	1	87	26.0	-

An error message when there is no selection in the table is generated.

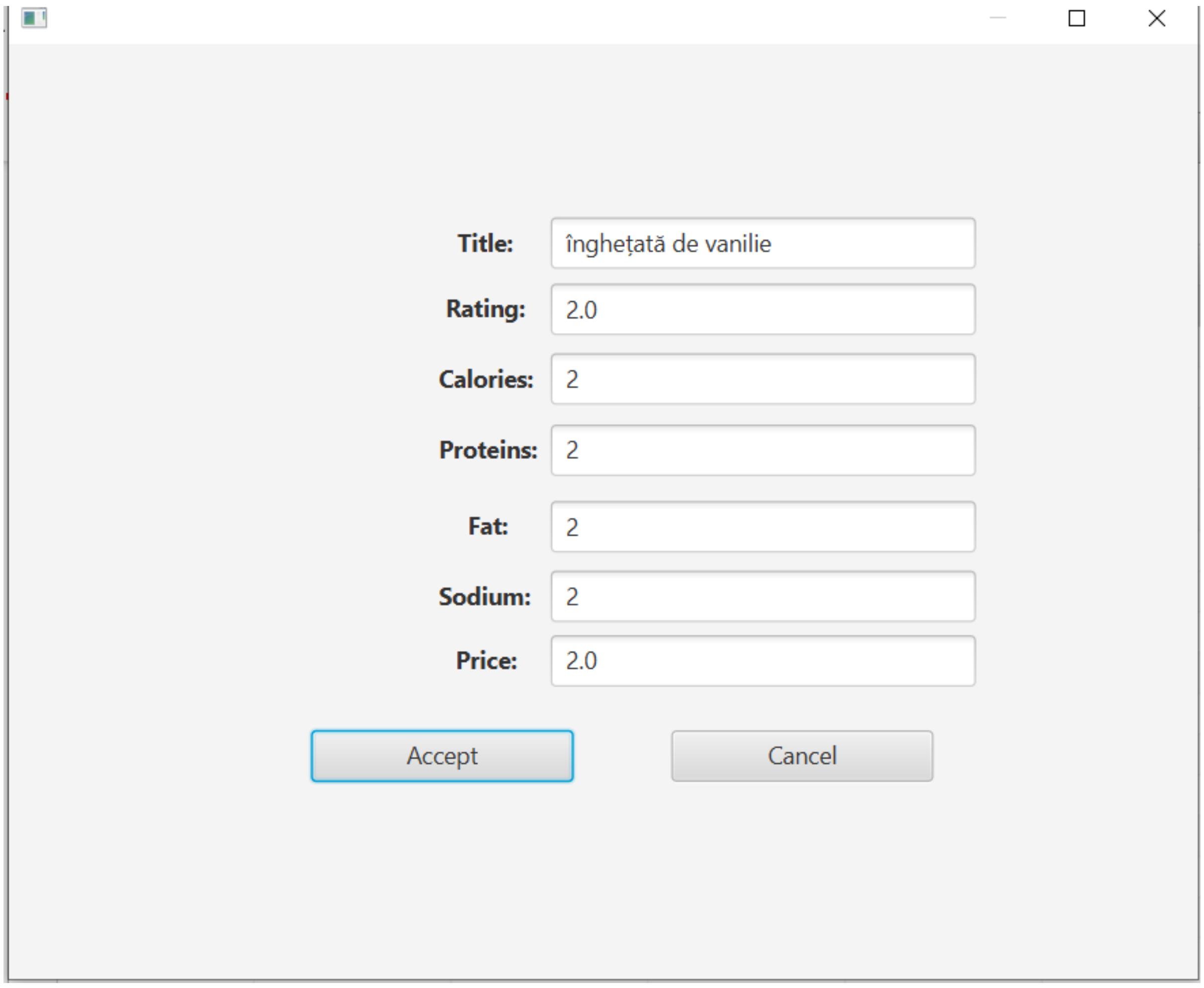


Error

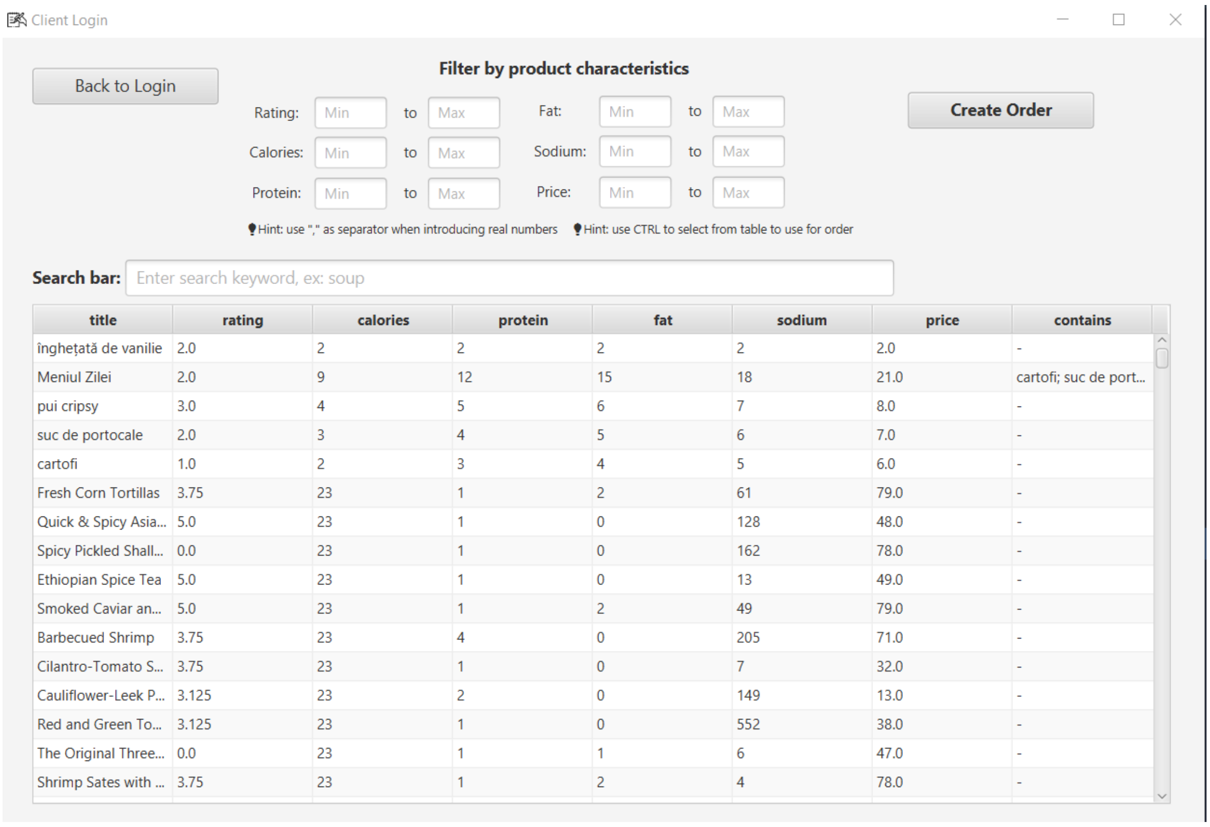
Must select item to edit

The buttons below the table open the GUI to report generation. Filter and information are introduced, and the reports are written in a file but also have their contents displayed on screen. Tooltips are present on the buttons so the administrator knows what the reports will generate before pressing the button.

The form that is loaded when the admin wants to create or edit base products can be seen below. Information is loaded into the fields when editing and they will be empty when the admin is creating. If a product already exists with that name, an error shall be generated.


Title:	înghețată de vanilie
Rating:	2.0
Calories:	2
Proteins:	2
Fat:	2
Sodium:	2
Price:	2.0

The client window looks like this. The client will write the filters in order to find the products and select them for the order. A notification is shown whenever an order is created.



Filter by product characteristics

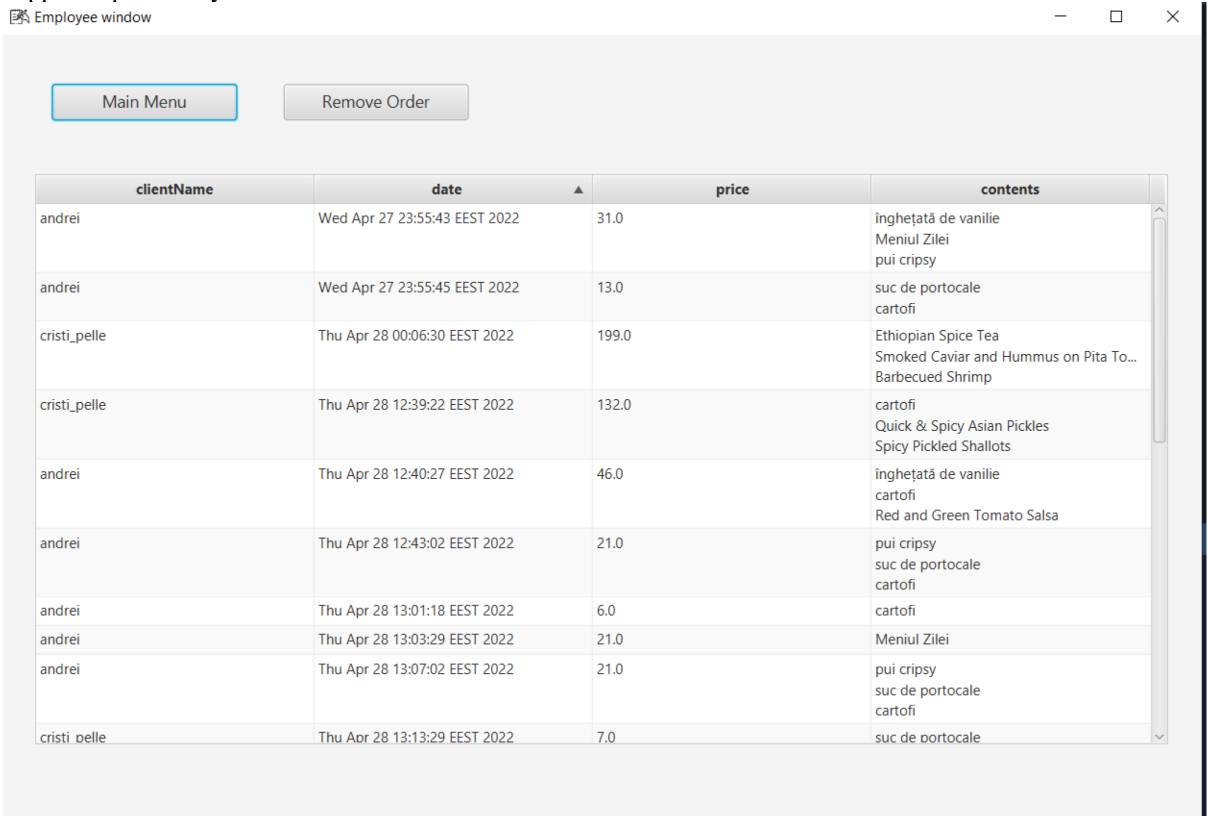
Rating:	Min	to	Max	Fat:	Min	to	Max
Calories:	Min	to	Max	Sodium:	Min	to	Max
Protein:	Min	to	Max	Price:	Min	to	Max

Hint: use "," as separator when introducing real numbers   Hint: use CTRL to select from table to use for order

Search bar: Enter search keyword, ex: soup

title	rating	calories	protein	fat	sodium	price	contains
înghețată de vanilie	2.0	2	2	2	2	2.0	-
Meniul Zilei	2.0	9	12	15	18	21.0	cartofi; suc de port...
pui cripsy	3.0	4	5	6	7	8.0	-
suc de portocale	2.0	3	4	5	6	7.0	-
cartofi	1.0	2	3	4	5	6.0	-
Fresh Corn Tortillas	3.75	23	1	2	61	79.0	-
Quick & Spicy Asia...	5.0	23	1	0	128	48.0	-
Spicy Pickled Shall...	0.0	23	1	0	162	78.0	-
Ethiopian Spice Tea	5.0	23	1	0	13	49.0	-
Smoked Caviar an...	5.0	23	1	2	49	79.0	-
Barbecued Shrimp	3.75	23	4	0	205	71.0	-
Cilantro-Tomato S...	3.75	23	1	0	7	32.0	-
Cauliflower-Leek P...	3.125	23	2	0	149	13.0	-
Red and Green To...	3.125	23	1	0	552	38.0	-
The Original Three...	0.0	23	1	1	6	47.0	-
Shrimp Sates with ...	3.75	23	1	2	4	78.0	-

The employee window can be seen below. Here, the employee is notified of every new order and the orders that happened previously have been serialized.

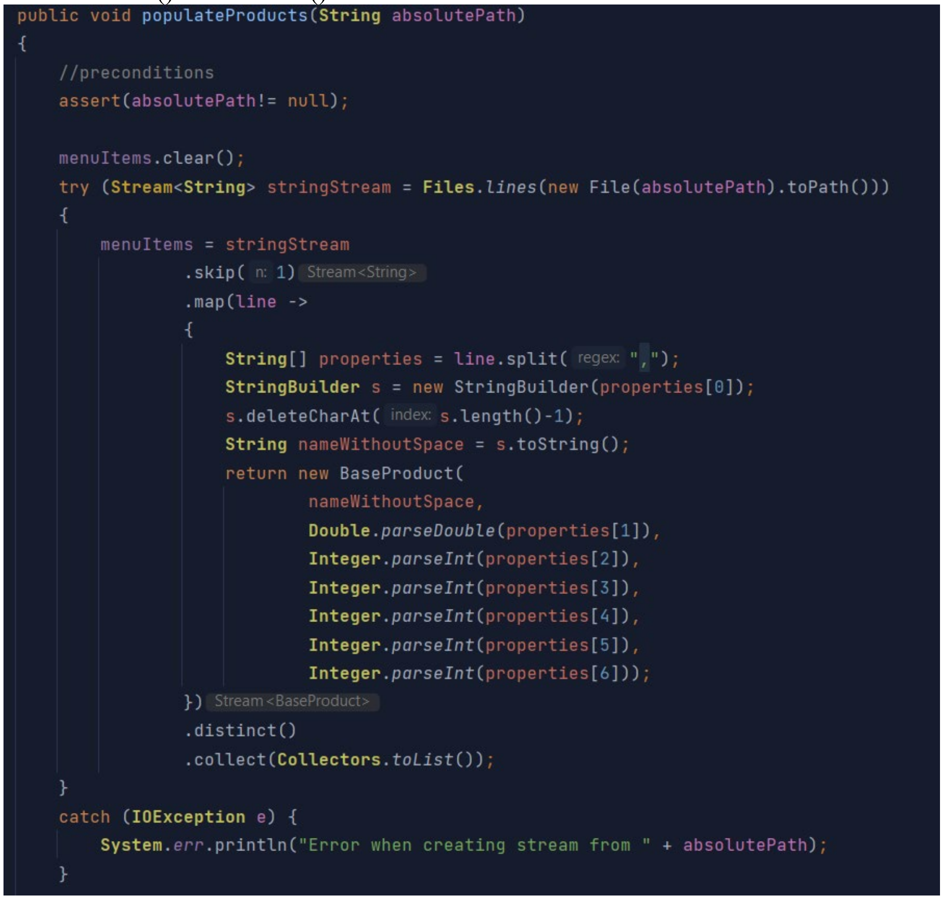


clientName	date	price	contents
andrei	Wed Apr 27 23:55:43 EEST 2022	31.0	înghețată de vanilie Meniul Zilei pui cripsy
andrei	Wed Apr 27 23:55:45 EEST 2022	13.0	suc de portocale cartofi
cristi_pelle	Thu Apr 28 00:06:30 EEST 2022	199.0	Ethiopian Spice Tea Smoked Caviar and Hummus on Pita To... Barbecued Shrimp
cristi_pelle	Thu Apr 28 12:39:22 EEST 2022	132.0	cartofi Quick & Spicy Asian Pickles Spicy Pickled Shallots
andrei	Thu Apr 28 12:40:27 EEST 2022	46.0	înghețată de vanilie cartofi Red and Green Tomato Salsa
andrei	Thu Apr 28 12:43:02 EEST 2022	21.0	pui cripsy suc de portocale cartofi
andrei	Thu Apr 28 13:01:18 EEST 2022	6.0	cartofi
andrei	Thu Apr 28 13:03:29 EEST 2022	21.0	Meniul Zilei
andrei	Thu Apr 28 13:07:02 EEST 2022	21.0	pui cripsy suc de portocale cartofi
cristi_pelle	Thu Apr 28 13:13:29 EEST 2022	7.0	suc de portocale

The delivery service class. Contains operations for clients, orders, and menu items. Implemented using the singleton design pattern, only a single instance of delivery service is present at any time. getInstance() will simply grab it and return it to the caller. It has a list of clients which is deserialized from a file in the constructor. This list contains all the information about the current registered clients and is queried whenever the interface wants to add a new client. It has a list of menu items that can be composite or base products. This list is also deserialized from a file at the beginning. Can also be loaded from a CSV using the populateProducts() method. Has a HashMap <Order, ArrayList>. Order is the key for the map, array list of menu item is the value. Many orders can hash to the same menu item list and saves space and time. Arraylist is used because it has to be serializable. This map is used whenever someone wants the details of an order, i.e., the ordered products. The class extends Observable and notifies its observers about any new orders that are inserted.
Invariant:
isWellFormed()

The constructor initializes data using serialization.

This method deletes all products from the table and replaces them with the products from the CSV file located at absolute path. Starts with a stream of strings generated by the Files.lines() method. First line is skipped because it contains just the names of the fields. The call to map will map that line to a new base product that parses each fields from the string array. The line is broken into pieces using string split on ",". For the name, the final character is removed since for some bizarre reason the csv file has a final space for the name. After the mapping, we now have a stream of base products. These are filtered to be distinct using the call to .distinct() which will internally call the hash code and equals function to check if the items are identical. The unique key is the name. Finally, the list is collected and stored.
Specified by:
populateProducts in interface IDeliveryServiceProcessing
Parameters:
absolutePath - -> Absolute path of file that contains information
Pre:
absolutePath != null
Post:
isWellFormed() && list.size() > 0

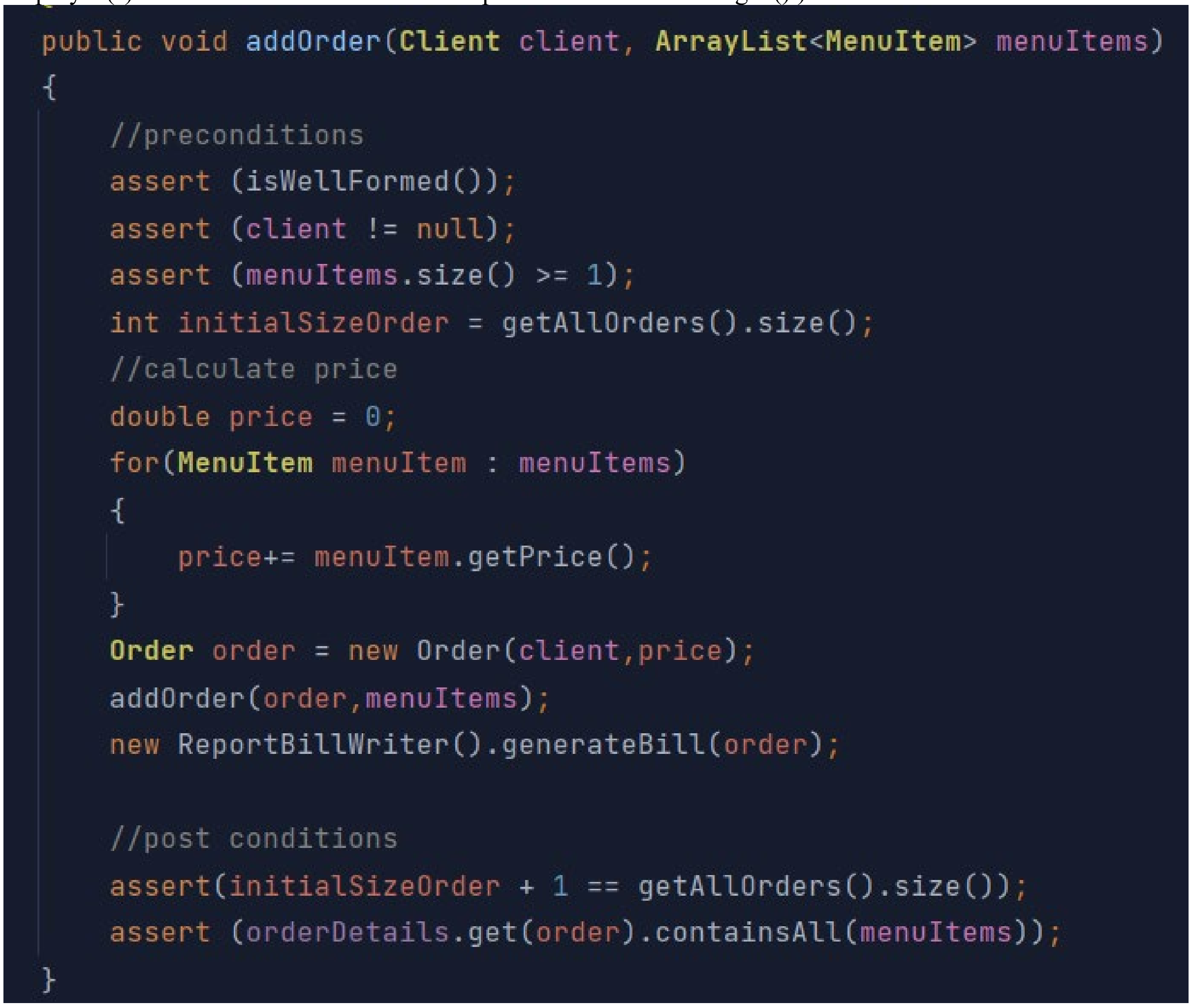


```
public void populateProducts(String absolutePath)
{
    //preconditions
    assert(absolutePath!= null);

    menuItems.clear();
    try (Stream<String> stringStream = Files.lines(new File(absolutePath).toPath()))
    {
        menuItems = stringStream
                .skip( n: 1) Stream<String>
                .map(line ->
                {
                    String[] properties = line.split( regex: ",");
                    StringBuilder s = new StringBuilder(properties[0]);
                    s.deleteCharAt( index: s.length()-1);
                    String nameWithoutSpace = s.toString();
                    return new BaseProduct(
                            nameWithoutSpace,
                            Double.parseDouble(properties[1]),
                            Integer.parseInt(properties[2]),
                            Integer.parseInt(properties[3]),
                            Integer.parseInt(properties[4]),
                            Integer.parseInt(properties[5]),
                            Integer.parseInt(properties[6]));
                }) Stream<BaseProduct>
                .distinct()
                .collect(Collectors.toList());
    }
    catch (IOException e) {
        System.err.println("Error when creating stream from " + absolutePath);
    }
```

Generates and order object with the required fields, calculating the total value of the order by iterating through the given menu items. When a call to new Order() is made, internally the date will be the current date precise to the millisecond. Once the order object is created, it is inserted into the map of order details and a bill is written for it.
Specified by:
addOrder in interface IDeliveryServiceProcessing
Parameters:
client - -> client that made the order
menuItems - -> list of menu items to be added to order

The private method with the same name will create an entry in the map with a link between the order and the menu items. Once the insertion is finished, the set changed and notify methods. Thus, the employee(s) shall be notified each time a new order is made. (Behind the scenes, notify calls the hasChanged method, so the employee(s) wouldn't be notified without the previous call to setChanged())

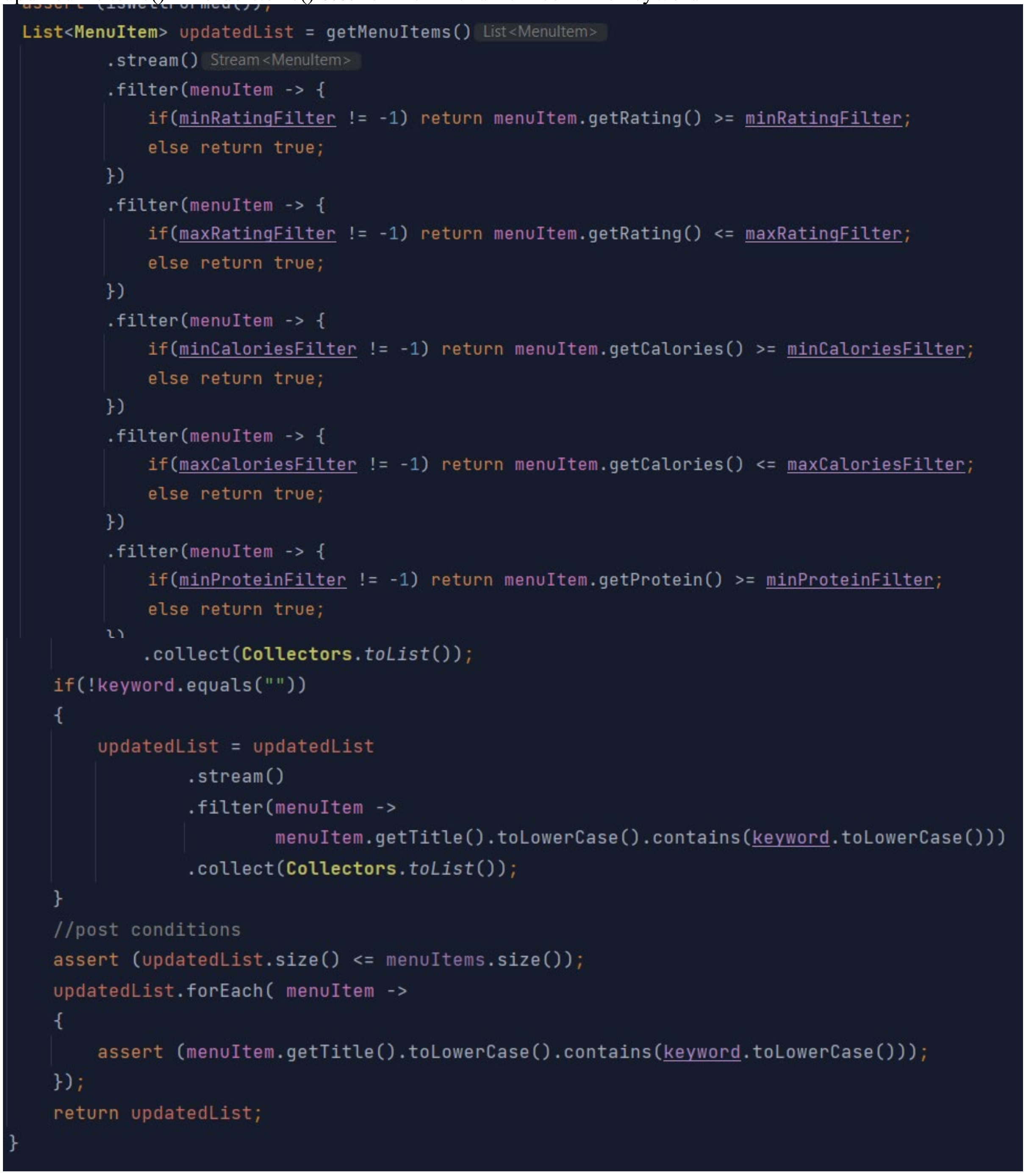


```java
public void addOrder(Client client, ArrayList<MenuItem> menuItems)
{
    //preconditions
    assert (isWellFormed());
    assert (client != null);
    assert (menuItems.size() >= 1);
    int initialSizeOrder = getAllOrders().size();
    //calculate price
    double price = 0;
    for(MenuItem menuItem : menuItems)
    {
        price+= menuItem.getPrice();
    }
    Order order = new Order(client,price);
    addOrder(order,menuItems);
    new ReportBillWriter().generateBill(order);

    //post conditions
    assert(initialSizeOrder + 1 == getAllOrders().size());
    assert (orderDetails.get(order).containsAll(menuItems));
}
```

Method to filter menu items for the first report. Filters are -1 if not applied. The list is first sorted according to the filters using streams and the .filter() method for the streams. The predicate is simple: checks if the filter is valid (different from -1) then checks if the details are within the requested bounds. Next, the list is sorted based on the keyword entered by the client.
Specified by:
clientFilter in interface IDeliveryServiceProcessing
Parameters:
keyword - -> search substring
Returns:
a list of orders conforming to the filters
Pre:
isWellFormed()

Post:
updatedList.size() <= orders.size() && forEach menu item contains keyword



```java
    List<MenuItem> updatedList = getMenuItems() List<MenuItem>
            .stream() Stream<MenuItem>
            .filter(menuItem -> {
                if(minRatingFilter != -1) return menuItem.getRating() >= minRatingFilter;
                else return true;
            })
            .filter(menuItem -> {
                if(maxRatingFilter != -1) return menuItem.getRating() <= maxRatingFilter;
                else return true;
            })
            .filter(menuItem -> {
                if(minCaloriesFilter != -1) return menuItem.getCalories() >= minCaloriesFilter;
                else return true;
            })
            .filter(menuItem -> {
                if(maxCaloriesFilter != -1) return menuItem.getCalories() <= maxCaloriesFilter;
                else return true;
            })
            .filter(menuItem -> {
                if(minProteinFilter != -1) return menuItem.getProtein() >= minProteinFilter;
                else return true;
            })
                .collect(Collectors.toList());
    if(!keyword.equals(""))
    {
        updatedList = updatedList
                .stream()
                .filter(menuItem ->
                        menuItem.getTitle().toLowerCase().contains(keyword.toLowerCase()))
                .collect(Collectors.toList());
    }
    //post conditions
    assert (updatedList.size() <= menuItems.size());
    updatedList.forEach( menuItem ->
    {
        assert (menuItem.getTitle().toLowerCase().contains(keyword.toLowerCase()));
    });
    return updatedList;
}
```

5. Results

For testing purposes, I used the GUI alongside some data from the delivery service to test things. Below you can see the example of a bill successfully created:

```
------------------------------------------------
BILL for order with on date Thu May 05 23:13:03 EEST 2022
------------------------------------------------
Client: Andrei
phone number:123456789
------------------------------------------------
Ordered products are:
Meniul Zilei
```

```
pui cripsy
cartofi
------------------------------------------------
Total order price is: 35.0
------------------------------------------------
```

Indeed, the price is calculated correctly as the products have 21, 8 and 6 as prices.

An example of report 1 is appended below:

```
------------------------------------------------
REPORT 1 for ADMINISTRATOR
------------------------------------------------
This report contains all the orders completed between different times of the day.
Filters applied:
minimum seconds: 0.
maximum seconds: 30.
minimum minutes: 0.
maximum minutes: 30.
minimum hour: 10.
maximum hour: 23.
------------------------------------------------
Orders that fit the criteria (empty means no orders fit criteria):
Client: andrei; Date: Thu Apr 28 13:01:18 EEST 2022; Value: 6.0.
Contents:
cartofi
Client: cristi_pelle; Date: Thu Apr 28 13:13:29 EEST 2022; Value: 7.0.
Contents:
suc de portocale
Client: andrei; Date: Thu Apr 28 13:07:02 EEST 2022; Value: 21.0.
Contents:
pui cripsy
suc de portocale
cartofi
Client: andrei; Date: Thu May 05 23:13:03 EEST 2022; Value: 35.0.
Contents:
Meniul Zilei
pui cripsy
cartofi
Client: andrei; Date: Thu Apr 28 13:03:29 EEST 2022; Value: 21.0.
Contents:
Meniul Zilei
------------------------------------------------
```

6. Conclusions

This homework was a good introduction to programming techniques with collections, lambda expressions and streams in java, and it helped me understand these mechanisms in java better, also it was a breeze to work with such mechanisms since they adhere to the java high level way of doing things. Streams and functional programming are indeed great mechanisms and allow the programmer to reduce the amount of boilerplate code that they have to write.

As a future improvement, I could stylize the GUI more using CSS and perhaps show the border in red for invalid input like I did in the first assignment, perhaps provide a way for users to edit composite products. Also, another improvement would be to allow clients to order multiple products at the same time.

7. Bibliography

1. https://dsrl.eu/courses/pt/ and everything contained within assignment4 support presentation, assignment 4 resources, lecture notes, etc.
2. Java Programming Masterclass by Tim Buchalka – a course available on Udemy about Java concepts and programming techniques. More specifically, chapter 17 deals with lambda expressions and operating on streams.
3. https://stackoverflow.com/questions/tagged/java
4. Stack overflow for information on streams and how to validate input in text fields
5. https://docs.oracle.com/javafx/2/api/javafx/scene/control/TableView.html
6. https://en.wikipedia.org/wiki/Non-functional_requirement#Examples
7. https://docs.oracle.com/javase/8/javafx/api/javafx/application/Application.html