DOCUMENTATION

Assignment 4

Jucan Rares Tudor
30422 – 1

Table of contents

1. Assignment Objectives

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.

Secondary Objectives:

I) Analyze the problem and identify requirements – how the application should work, what it should do and what are the use cases.
II) Design the Food Delivery Management System – which architectural pattern should be used, how the packages, classes and methods are divided
III) Implement the Food Delivery Management System – the Java code written for the implementation
IV) Test the Food Delivery Management System – testing the application to check if it works properly

2. Problem analysis, modelling and use case

Here are some functional requirements:

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

And some non-functional requirements:

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

The administrator can:

I) Import the initial set of products which will populate the menu from a .csv file
II) 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).
III) Generate reports about the performed orders considering the following criteria:

   a) 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
   b) the products ordered more than a specified number of times so far
   c) the clients that have ordered more than a specified number of times and the value of the order was higher than a specified amount
   d) the products ordered within a specified day with the number of times they have been ordered

The client can:

I) Register and use the registered username and password to log in within the system
II) View the list of products from the menu
III) Search for products based on one or multiple criteria such as keyword (e.g., "soup"), rating, number of calories/proteins/fats/sodium/prices.
IV) 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.

The use cases are:

i) 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

ii) 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

iii) 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).

1. The title of the product is already existent
2. A pop-up with an error message is displayed
3. The user presses the “Ok” button or the close button
4. The scenario returns to step 2

iiii) 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

1. 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)
2. The title of the product is not existent
3. A pop-up with an error message is displayed
4. 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

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

3. Design

Design Decisions:

I) Define the interface IDeliveryServiceProcessing containing the main operations that can be executed by the administrator and client, as follows

II) Administrator: import products, manage the products from the menu, generate reports

III) 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

IV) Use the Composite Design Pattern for defining the classes MenuItem, BaseProduct and CompositeProduct

V) Use the Observer Design Pattern to notify the employee each time a new order is created

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

VII) Define a method of type "well formed" for the class DeliveryService.

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

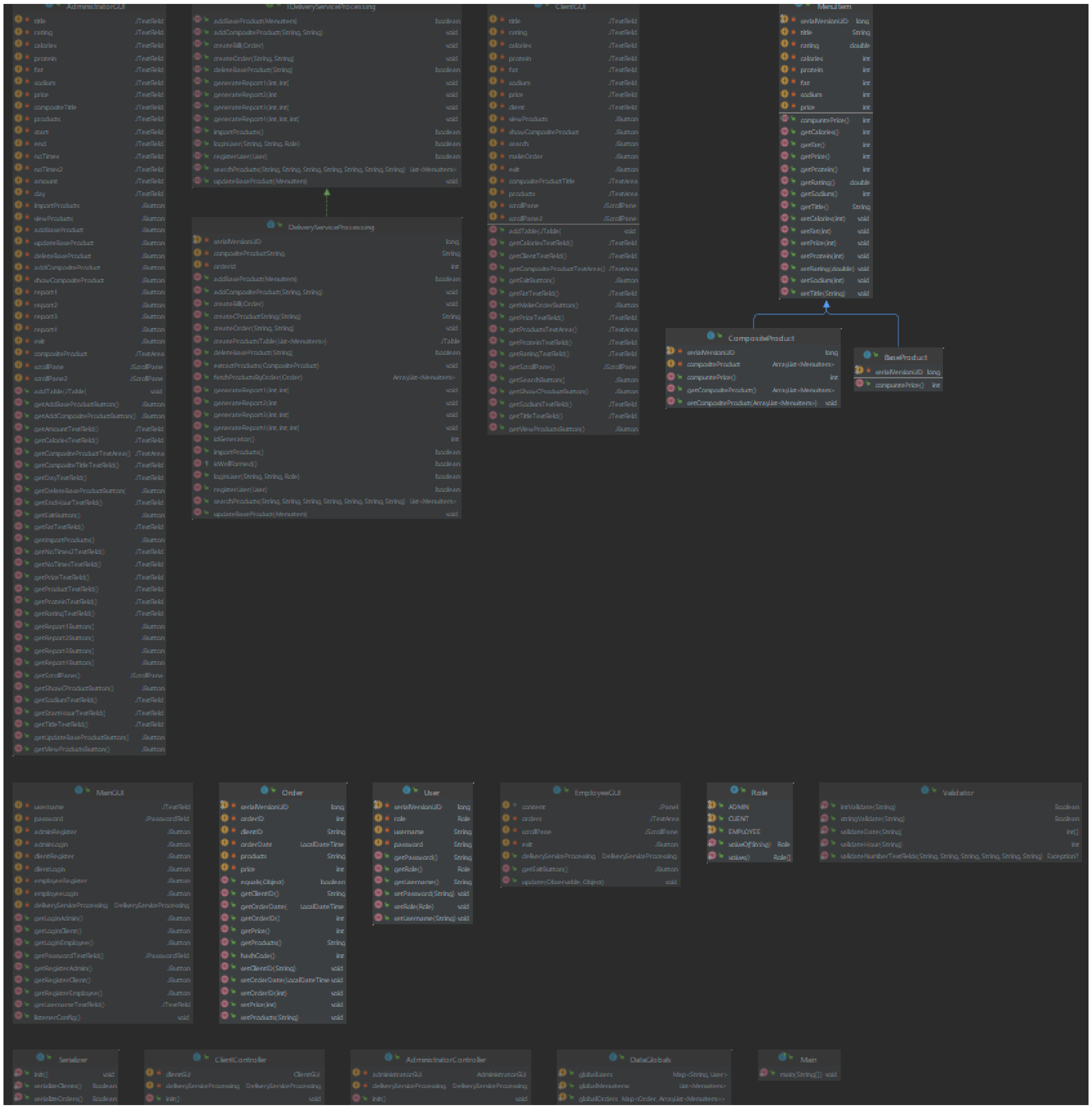
IX) The base products used initially for populating the DeliveryService object can be loaded from the products.csv file. Note: the administrator can manually add other base products as well. 9

X) 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

Some data structures I used are:

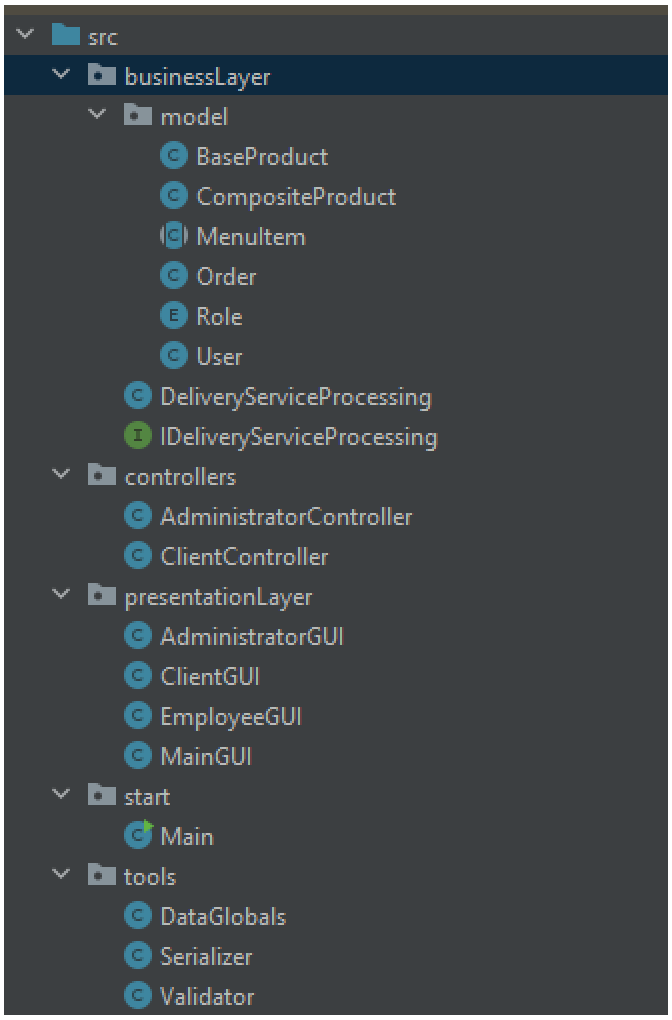
I) ArrayList – to store the list of menu items in the CompositeProduct and in DeliveryService – to store the list of menu items in the CompositeProduct

II) Map – HashMap to store the users uniquely – HashMap> to store the orders uniquely associated with the list of ordered products. For Order the equals and hashCode methods were overridden.


AdministratorGUI
IDeliveryServiceProcessing
ClientGUI
MenuItem
DeliveryServiceProcessing
CompositeProduct
BaseProduct
MainGUI
Order
User
EmployeeGUI
Role
Validator
Serializer
ClientController
AdministratorController
DataGlobals
Main
serialVersionUID long
title String
rating double
calories int
protein int
fat int
sodium int
price int
computePrice() int
getCalories() int
getFat() int
getPrice() int
getProtein() int
getRating() double
getSodium() int
getTitle() String
setCalories(int) void
setFat(int) void
setPrice(int) void
setProtein(int) void
setRating(double) void
setSodium(int) void
setTitle(String) void
compositeProduct ArrayList<MenuItem>
computePrice() int
getCompositeProduct() ArrayList<MenuItem>
setCompositeProduct(ArrayList<MenuItem>) void
addBaseProduct(MenuItem) boolean
addCompositeProduct(String, String) void
createBill(Order) void
createOrder(String, String) void
deleteBaseProduct(String) boolean
generateReport1(int, int) void
generateReport2(int) void
generateReport3(int, int) void
generateReport4(int, int, int) void
importProducts() boolean
loginUser(String, String, Role) boolean
registerUser(User) boolean
searchProducts(String, String, String, String, String, String, String) List<MenuItem>
updateBaseProduct(MenuItem) void
compositeProductString String
orderId int
createCProductString(String) String
createProductsTable(List<MenuItem>) JTable
extractProducts(CompositeProduct) void
fetchProductsByOrder(Order) ArrayList<MenuItem>
idGenerator() int
isWellFormed() boolean
ADMIN
CLIENT
EMPLOYEE
valueOf(String) Role
values() Role[]
intValidator(String) Boolean
stringValidator(String) Boolean
validateDate(String) int[]
validateHour(String) int
validateNumberTextFields(String, String, String, String, String, String) Exception?
init() void
serializeClients() Boolean
serializeOrders() Boolean
clientGUI ClientGUI
deliveryServiceProcessing DeliveryServiceProcessing
administratorGUI AdministratorGUI
globalUsers Map<String, User>
globalMenuItems List<MenuItem>
globalOrders Map<Order, ArrayList<MenuItem>>
main(String[]) void
orderID int
clientID String
orderDate LocalDateTime
products String
equals(Object) boolean
hashCode() int
role Role
username String
password String
contest JPanel
orders JTextArea
scrollPane JScrollPane
exit JButton
update(Observable, Object) void

4. Implementation

I used 5 packages to implement this application: start, tools, controllers, business layer and presentation layer.



```
src
  businessLayer
    model
      BaseProduct
      CompositeProduct
      MenuItem
      Order
      Role
      User
    DeliveryServiceProcessing
    IDeliveryServiceProcessing
  controllers
    AdministratorController
    ClientController
  presentationLayer
    AdministratorGUI
    ClientGUI
    EmployeeGUI
    MainGUI
  start
    Main
  tools
    DataGlobals
    Serializer
    Validator
```

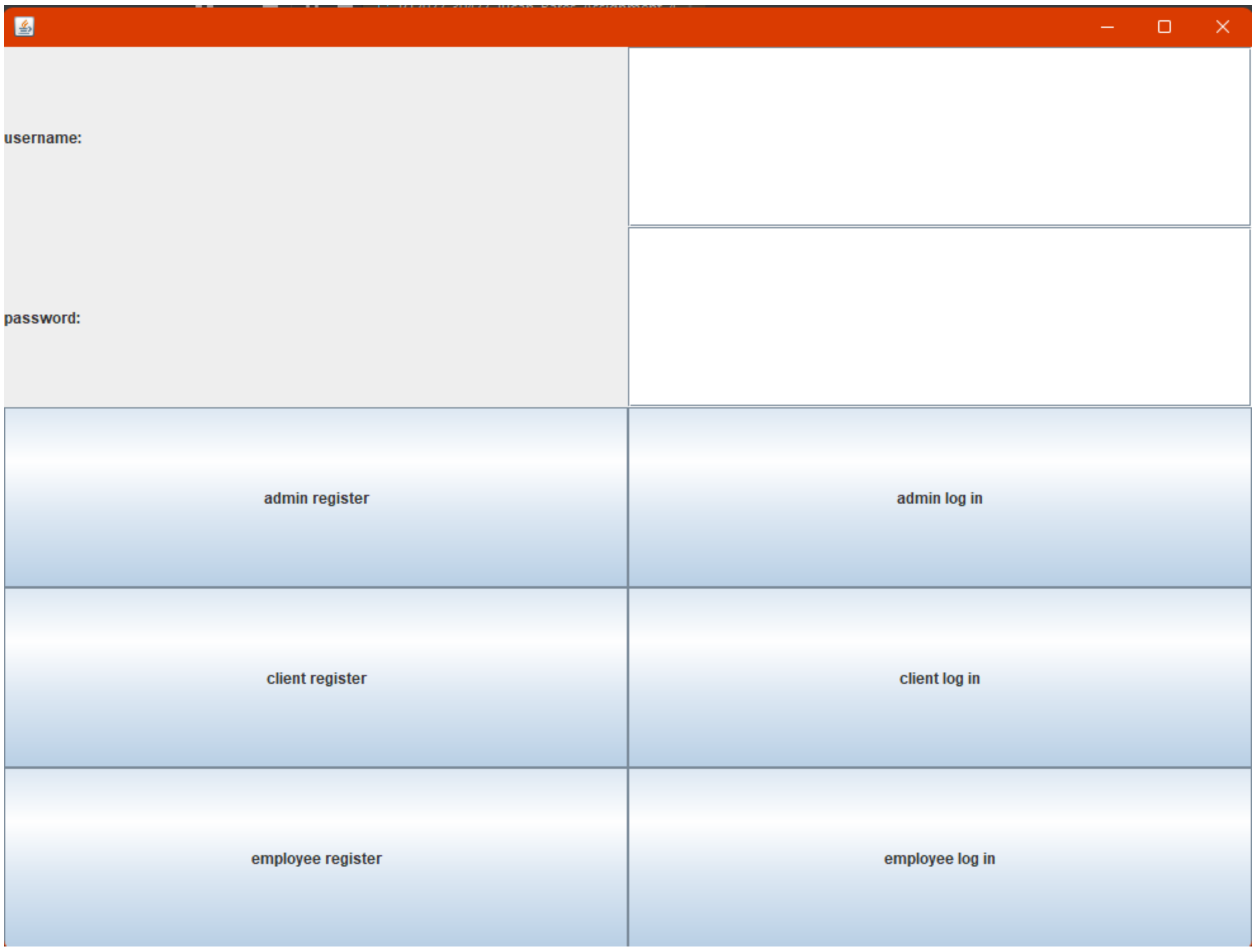
Start package:

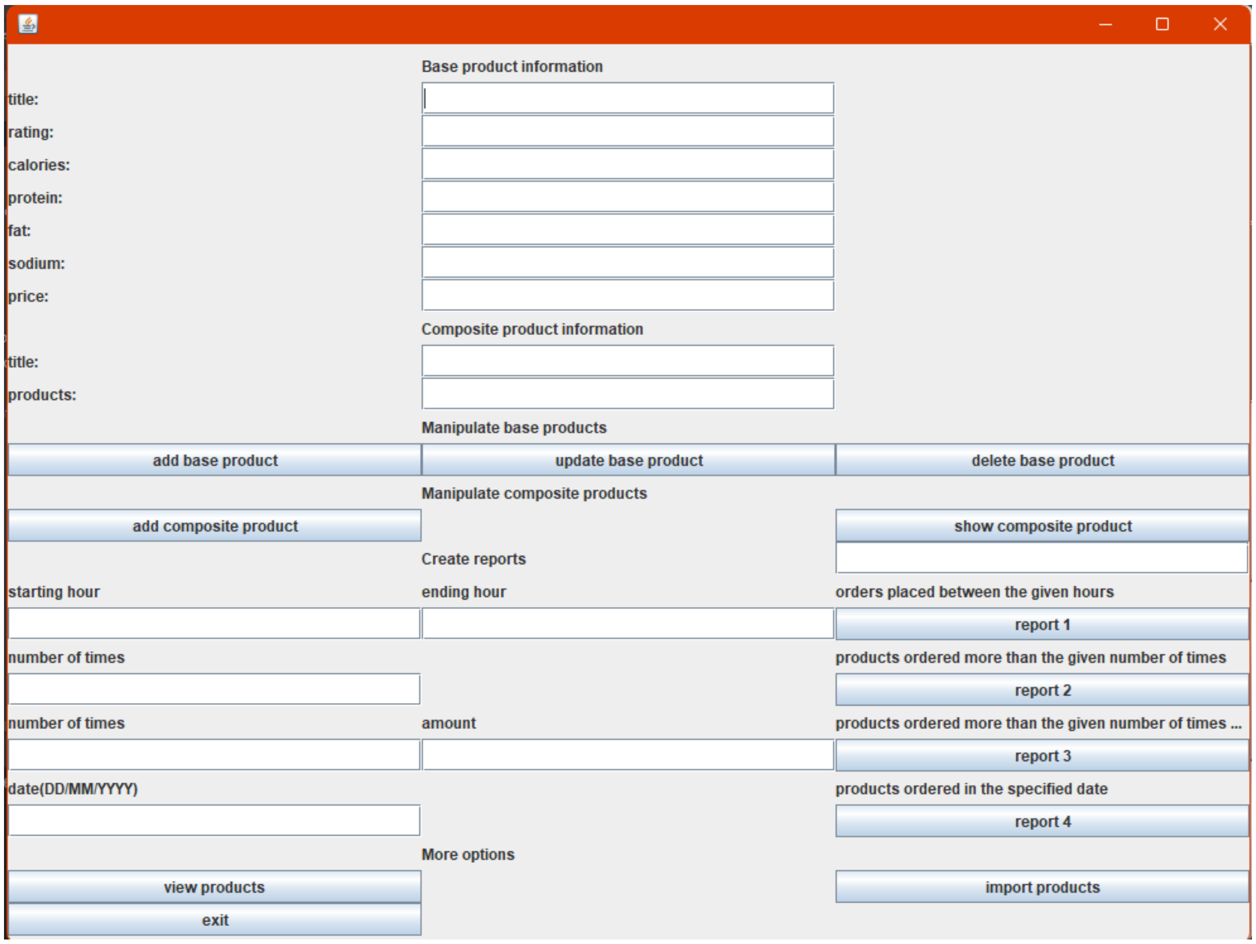
It contains the Main class which also holds the main method. In this method we display the main window of the GUI.

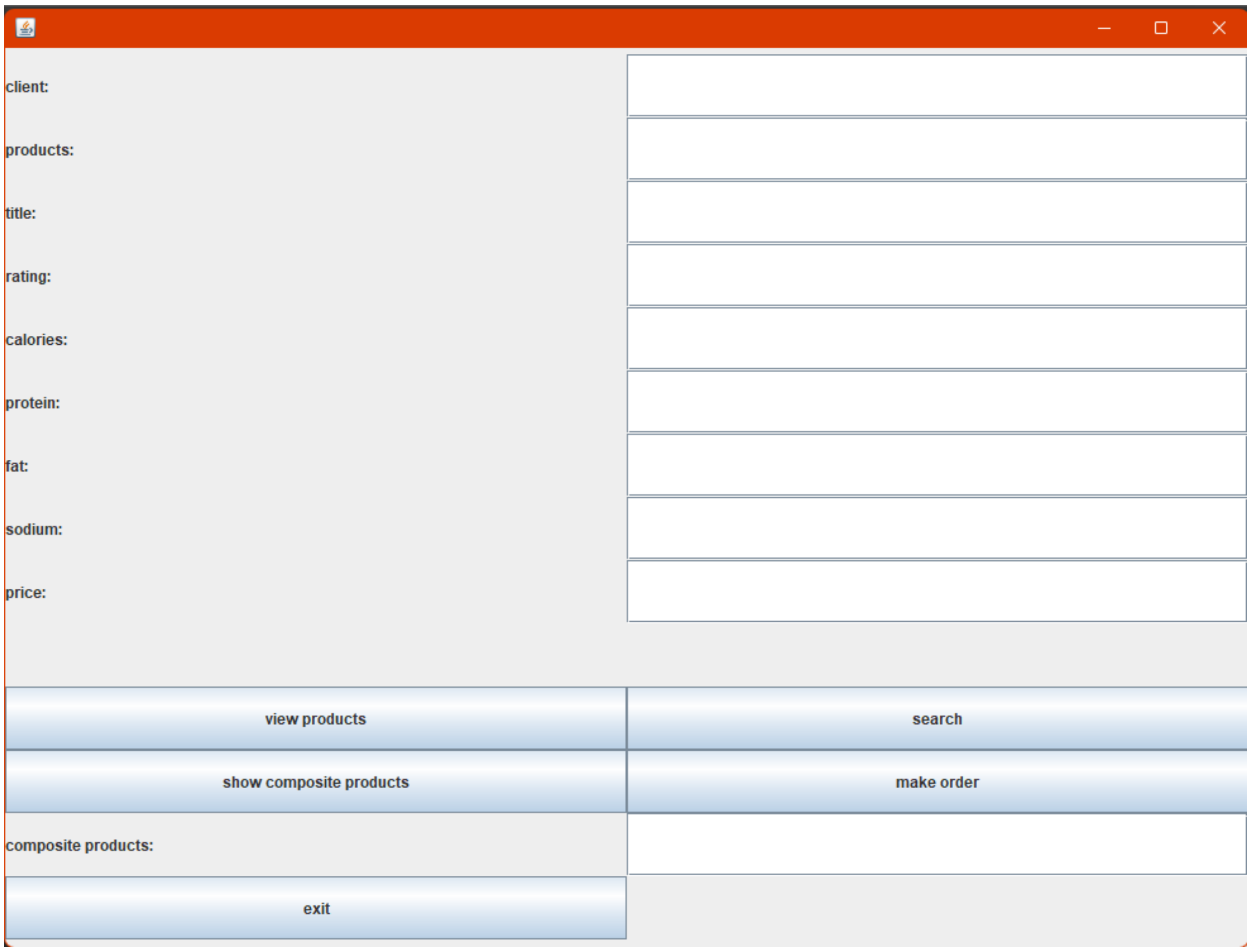
Tools package:

5. Results

We got a working delivery service application! Yey!




Base product information
title:
rating:
calories:
protein:
fat:
sodium:
price:
Composite product information
title:
products:
Manipulate base products
add base product
update base product
delete base product
Manipulate composite products
add composite product
show composite product
Create reports
starting hour
ending hour
orders placed between the given hours
report 1
number of times
products ordered more than the given number of times
report 2
number of times
amount
products ordered more than the given number of times ...
report 3
date(DD/MM/YYYY)
products ordered in the specified date
report 4
More options
view products
import products
exit


client:
products:
title:
rating:
calories:
protein:
fat:
sodium:
price:
view products
search
show composite products
make order
composite products:
exit

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

-assignment presentation support