Technical University of Cluj-Napoca

Programming Techniques

Assignment 4

Food Delivery Management System



Teacher: Prof. Ioan Salomie

Teacher Assistant: Dr. Viorica Chifu

Student: Domokos Robert - Ștefan

Group: 30421

Content

1. Objective
2. Analysis
3. Design
4. Implementation
5. Results
6. Conclusions
7. Bibliography
8. Objective

The main objective of this assignment is to design and implement a food delivery management system. The system allows customers to view the products and order the desired product. The administrator is allowed to see all the products, to import them and to perform many other operations on them, also the administrator can generate reports about the orders, clients and products. Finally the employee is notified when a customer makes an order and can see the notification.

The secondary objectives of the assignment are:

* Use as support the class diagram presented in the assignment’s PDF;
* Design 3 different GUI for the Employee, Administrator and Customer;
* Using the Composite and Observer Design Pattern for the items, respectively for the notifications
* Make use of the Design by contract method
* Use serialization for saving the information
* Creating a bill for each new Order
* Make use of the lambda expressions when generating the reports and the search functionalities
* Document the classes and methods using JavaDoc

1. Analysis

* Modeling and requirements

The requirements of this assignment consists of modeling the data in such ways that the data can be processed using Java. One way for modeling the data like presented above is by dividing the data in multiple data parts such as: User, Order, BaseProduct, CompositeProduct and MenuItem.

The input of the application is given as a .csv file representing the products, while other input can be introduced via the text fields of the GUIs such as: a new product, the title of the product that we want to delete and many others.

As for the output the application has the reports , the products, the filtered products and the notifications for the Employee.

The classes that divide the data and model it are modeled as:

User

* String username;
* String password;
* String type;
* int id;

Order

* int orderId;
* int clientId;
* LocalDate orderDate;
* GregorianCalendar hour;
* int price;

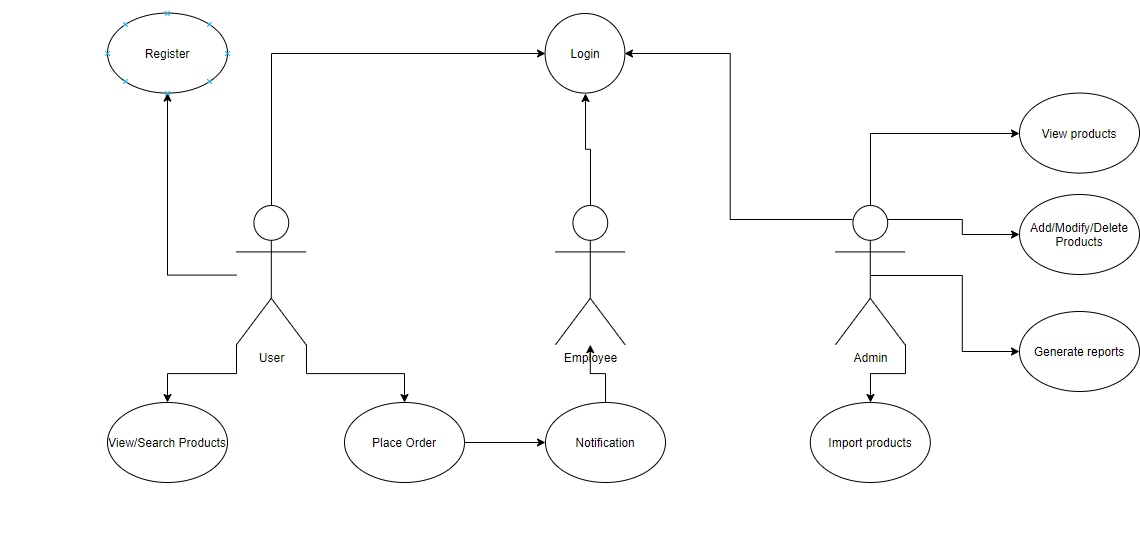
MenuItem

* String title;
* double rating;
* int calories;
* int proteins;
* int fats;
* int sodium;
* int price;

BaseProduct which extends the MenuItem abstract class;

CompositeProduct which extends the MenuItem abstract class

* List<MenuItem> itemList;
* Use case diagram



* Scenarios and use case

As presented above the use cases of the application has 3 different actors that can use the application in 3 different ways.

These 3 actors are:

* Employee that is waiting for a notification that notifies him if a new order is made;
* Client/User that he is allowed to register or to log in, can place an order and can view the products that he wants using the filter that he needs;
* Administrator that can see all the products, add products, modify products, delete products, import products, can see different reports made on orders, users and products;

1. Design

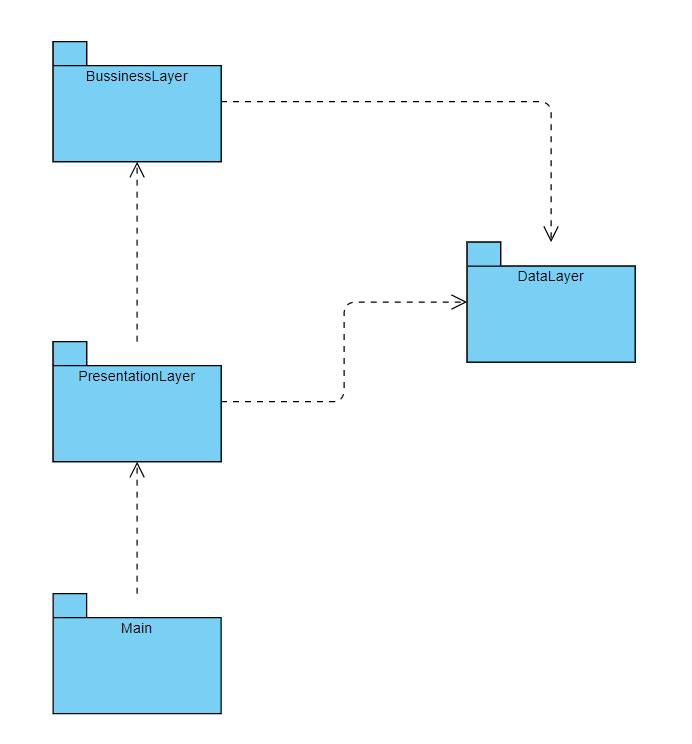
The design part of the project/assignment consists of the design decisions, UML diagrams, relationships, data structures, class design, packages and user interface;

* Design decision

As already mentioned in the requirements part and modeling part for this assignment, a Layered Architecture is used in the code to divide the created classes into main 4 packages:

* BusinessLayer package: where we model the data and implement the required operations;
* DataLayer package: where we have the Serializator which contains the required serialization methods;
* PresentationLayer: where the GUI are made;
* Main which starts the application and runs it;
* UML Diagram and relationships

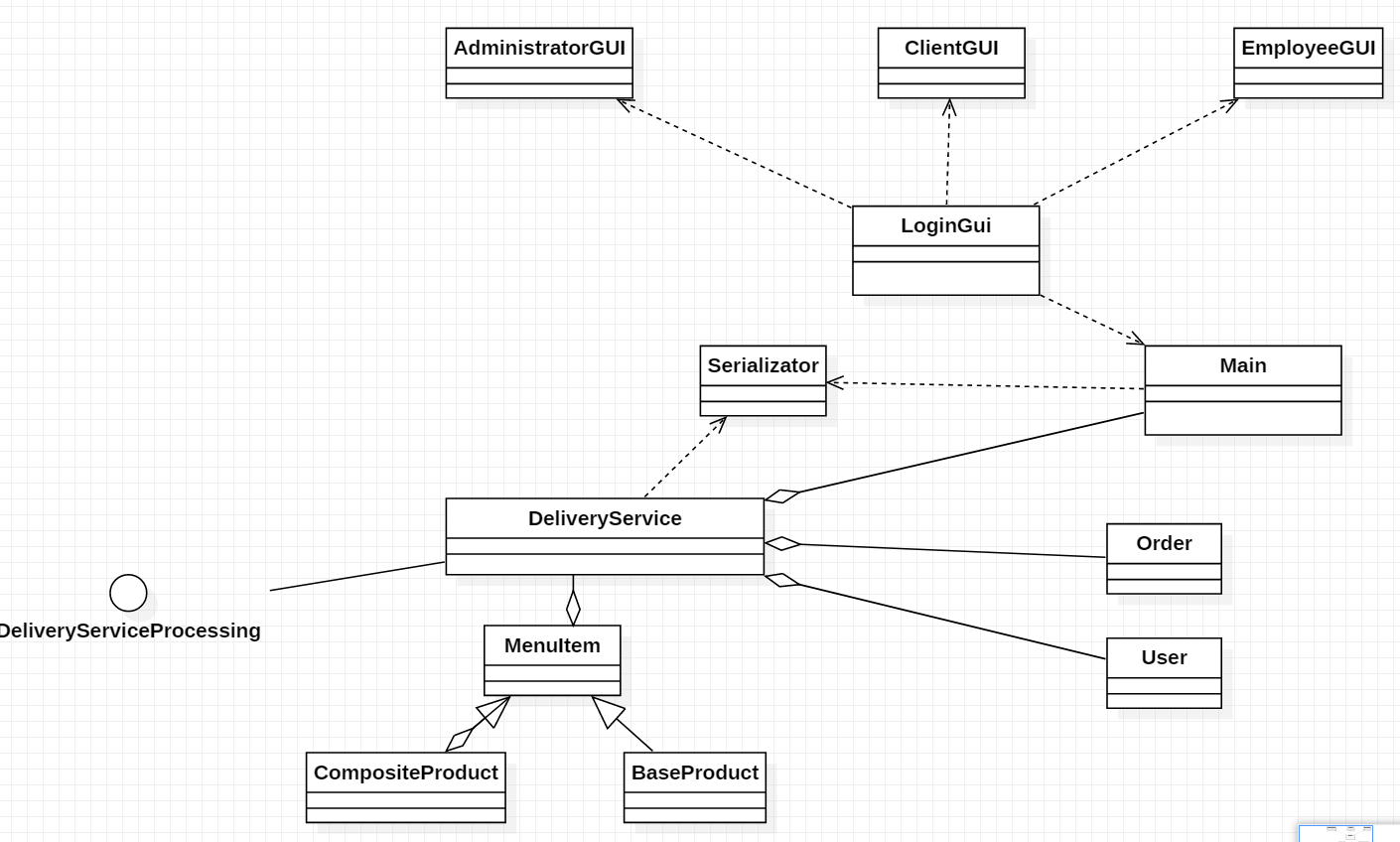
As presented in the next diagram anyone can see the main relationships present between the main packages of the application:



* Data structures

As for the data structures used in this application with such a large design, there aren’t many data structures that stand out of the ordinary, but some that could be mentioned are:

* HashMap used in order to keep all the orders with their according items persistently;
* LocalDate and GregorianCalendar which helped me in using the date and hour of the required orders;
* Class design



Some of the most important classes that contain the most important methods in the application can be broken down into but more detailed in the Implementation part of the documentation:

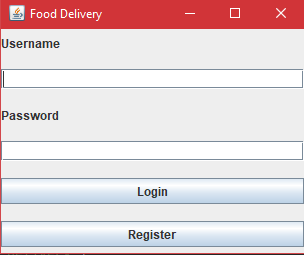
* BaseProduct, CompositeProduct and the abstract class MenuItem which represent the simple and combined products present on the menu;
* GUI Classes
* Serializator which uses the serialize and deserialize methods to save and write the data such that we have a persistent software;
* Main class which starts the application
* Order and User classes made for orders, respectively for users;
* DeliveryService class in which are implemented all the methods and function needed for having a persistent software;

* Interfaces

The assignment contains only one interface DeliveryServiceProcessing which contains and defines the functions that need to be implemented in the DeliveryService class;

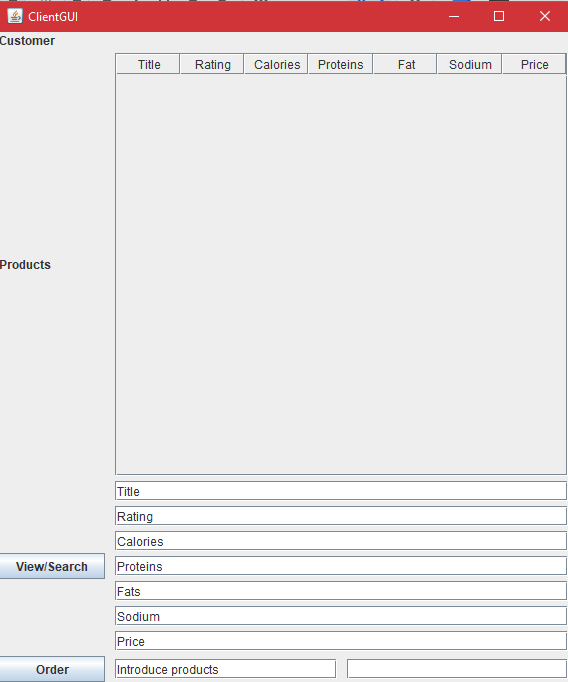
* User Interface

As the Main class runs the first interface that pops out is the Login GUI, which allows an administrator an employee or a client to log in the application, but the client can also register into it;



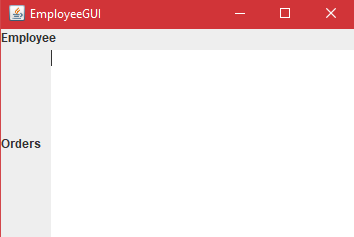
After logging in, a new interface opens up which can be detected by the type of the account introduced and we have 3:

* ClientGUI



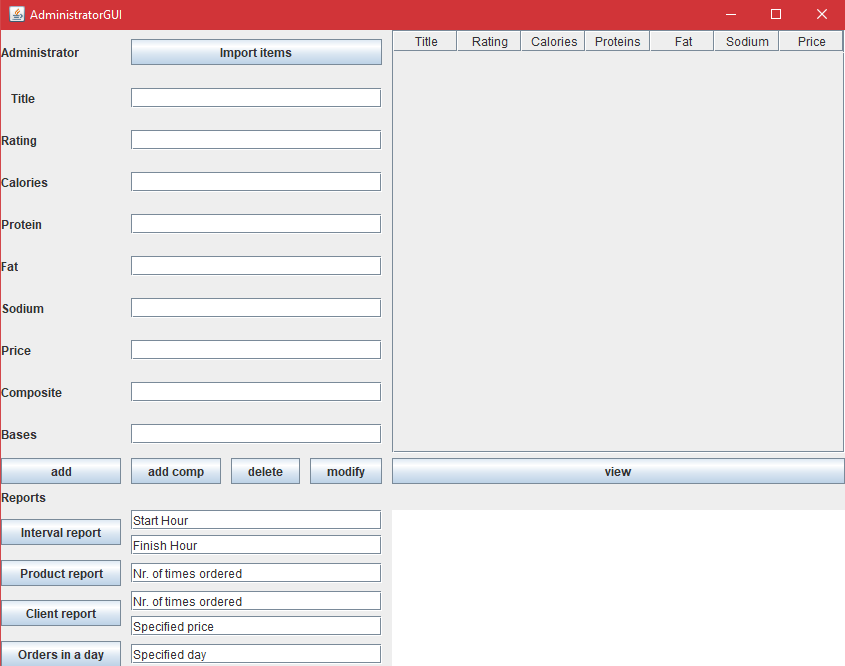
This interface has 2 buttons: View/Search button that allows the Client to view all the products or to apply one of the desired filters and the Order button which allows the Client to place a new order by introducing the titles of the products that he desires therefore the final price will be shown in the left bottom text field. It has also a JTable where all the products can be seen.

* EmployeeGUI



This interface has a simple JTextArea in which will be printed the notifications.

* AdministratorGUI



This interface has multiple buttons such as: add, add comp, delete, modify and view for operations on the products that can be made only by the input introduced in the text fields present; Interval report, Product report, Client report, Orders in a day for operations on the required reports that will appear as text in the left bottom JTextArea. It also has a JTable where all the products can be seen without any filters. Here the administrator can create different products, even composite ones.

1. Implementation

* Main.java

The main role of this class is to start the application by creating the LoginGUI;

* Login.java

This is the Login GUI where the user can log in as admin, employee or client. Thus opening the

required GUI;

* EmployeeGUI.java

This is the class representing the Employee GUI;

* ClientGUI.java

This is the class representing the Client GUI;

* AdministratorGUI.java

This is the class representing the Administrator GUI;

* Serializator.java

The main role of this class is to serialize and deserialize the data from the project by using the next 2 methods which are called in other classes;

For serialization:

public static void serializeDeliveryService(DeliveryService d)

{

FileOutputStream file = null;

try {

file = new FileOutputStream("Delivery.ser");

ObjectOutputStream out = new ObjectOutputStream(file);

out.writeObject(d);

out.close();

file.close();

} catch (IOException e) {

e.printStackTrace();

}

}

For deserialization:

public static DeliveryService deserializeDeliveryService(){

DeliveryService d = null;

try {

FileInputStream file = new FileInputStream("Delivery.ser");

ObjectInputStream in = new ObjectInputStream(file);

d = (DeliveryService)in.readObject();

in.close();

file.close();

return d;

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

d = new DeliveryService();

serializeDeliveryService(d);

}

return d;

}

* User.java

The main role of this class that implements Serializable is to represent the users and model their data. They can be clients, administrators or employees. This class also contains the required getters and setters and the main constructor. It has as parameters an username, a password, a type and an id;

* Order.java

The main role of this class that implements Serializable is to represent the orders and model their data. This class also contains the required getters and setters and the main constructor. It has as parameters an orderID, a clientID, an orderDate, an hour and a price. It also has a method hashCode() required for hashing which is overriden.

* MenuItem.java

This is an abstract class that implements Serializable is the class that represents the menu of the food delivery system. It has as protected parameters a title, a rating, the calories, the proteins, the fats, the sodium and a price. These parameters are inherited by the other 2 classes that extend this one: BaseProduct and CompositeProduct.

* DeliveryServiceProcessing.java

This is the interface of the application that defines the functions that are implemented in the DeliveryService.

* DeliveryService.java

This is the most important class of this application because most of the declared methods and functions that are necessary for processing the data are implemented here. It also holds the data which will be saved.

Here are implemented most of the operations that can be used by the Client and Administrator such as:

* Search methods
  + this methods are used for implementing the filter option for the Client;
  + this methods also use stream implementation and lambda expressions;

private List<MenuItem> searchTitle(String title){

return itemList.stream().filter(p->p.getTitle().contains(title)).collect(Collectors.toList());

}

private List<MenuItem> searchRating(double rating){

return itemList.stream().filter(p->p.getRating()>rating).collect(Collectors.toList());

}

private List<MenuItem> searchCalories(int cal){

return itemList.stream().filter(p->p.getRating()<cal).collect(Collectors.toList());

}

private List<MenuItem> searchProteins(int pro){

return itemList.stream().filter(p->p.getRating()<pro).collect(Collectors.toList());

}

private List<MenuItem> searchFats(int fat){

return itemList.stream().filter(p->p.getRating()<fat).collect(Collectors.toList());

}

private List<MenuItem> searchSodium(int s){

return itemList.stream().filter(p->p.getRating()<s).collect(Collectors.toList());

}

private List<MenuItem> searchPrice(int price){

return itemList.stream().filter(p->p.getRating()<price).collect(Collectors.toList());

}

* Create order method
  + this method creates an order after the Client introduces in the required field the title of the desired products;

@Override

public void createOrder(JTextField t0, JTextField t1,JTextField t2) {

assert(t0!=null);

int totalPrice = 0;

List<MenuItem> lis = new ArrayList<>();

for(String s :t0.getText().split(";")){

MenuItem mem = searchOne(s);

totalPrice += mem.getPrice();

lis.add(mem);

}

t1.setText(String.valueOf(totalPrice));

int idd = -1;

for(User u: userList)

{

if(u.getUsername().equals(t2.getText()))

{

idd = u.getId();

}

}

Order or= new Order(new Random().nextInt(1000),idd, LocalDate.now());

or.setPrice(totalPrice);

orderList.put(or,lis);

setChanged();

//notifyObservers(itemList);

clearChanged();

try {

generateBill(or);

} catch (IOException e) {

e.printStackTrace();

}

Serializator.serializeDeliveryService(this);

}

* CompositeProduct.java

This class models the data of a product that is composed of multiple base products. It also extends the menu class having a List<MenuItem> itemList;

* BaseProduct.java

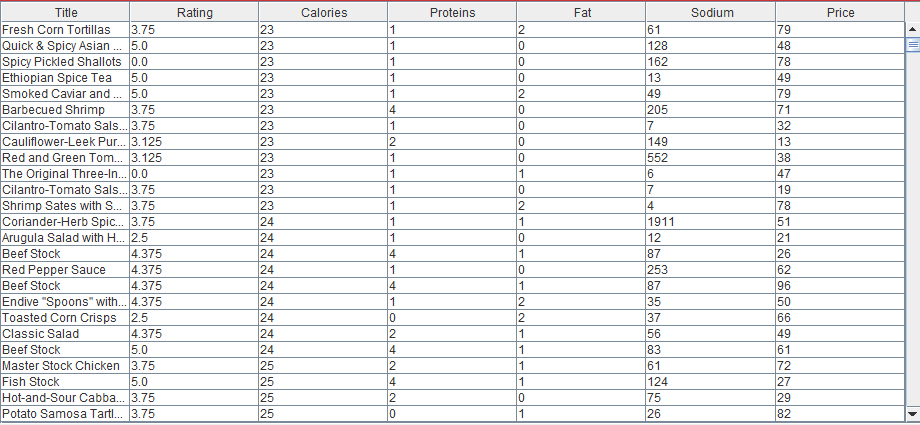
This class represents a base product that will be stored from the menu. It also extends the menu class.

1. Results

This part of the documentation is represented by several different outputs that the application provides.

* One output that can be considered as result is the table where the products can be seen by the Client

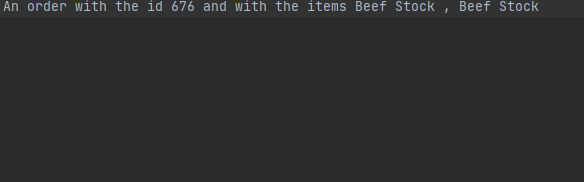
filtered or all.



* Another output that can be considered as result is the list of reports present in the AdministratorGUI;



The bill is also presented in a bill.txt file which stores the bill of a new order made by a Client:



1. Conclusions

In summary, the assignment served to learn about many powerful Java techniques, such as:

Serialization and Assertion. I believe that serialization is a faster and more flexible way of storing data compared to databases, even though it can be faster in some scenarios.

At the same time, it served as a very real example of an application that could actually end up being used by a real restaurant. I believe that the large amount of design patterns that were required in the project will allow all ideas tobe better written as clean code in the future.

Some of the improvements to the application that I think would be necessary could include:

* Expanding the data that can be modeled by adding additional classes that can be easily integrated through the existing serialization system.
* Adding network support that allows the application to be used on multiple computers.
* Enabling registration of all types of possible users.
* Providing a more user-friendly interface.
* Deploying the application in a real business that needs it.
* Enabling employees to also see orders placed in the past in case they fail to complete the last order in time.
* Create a web version of the application that is easier to use without having to install the application and Java on all computers.
* Create a more general version that can be used by other businesses, such as a bookstore or comic book store.

1. Bibliography

<https://online.visual-paradigm.com/diagrams/solutions/free-class-diagram-tool/>

<https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>

<https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html>

<http://www.tutorialspoint.com/java/java_serialization.html>

<https://intellij-support.jetbrains.com/hc/en-us/community/posts/207014815-How-toenable-assert>