**Technical University of Cluj-Napoca**

Programming Techniques

FOOD DELIVERY MANAGEMENT SYSTEM

ASSIGNMENT 4

Text

Description automatically generated with medium confidence

Student: Deac Dan Cristian

## Overview

* Language: Java 100%
* IDE: Intellij
* GUI: JavaFX

Here is a short overview of the GUI:

Graphical user interface, application

Description automatically generated

On the left side there is the employee side of the application and on the right, it’s the client side. The client can add items to the order by searching from the menu, selecting the item and pressing add. The total order will be updated and the client can submit the order or clear items form the order by selecting them and clicking clear.

Once the order is submitted by the client, a bill is generated on the client side and the employee GUI gets updated with the new order.

This project implements the observer design pattern.

The communication is not done through a server because this is more of a demo to showcase the implementation of the design pattern, everything is done locally and those views are part of the same instance of the application.

The admin has a special view where he can generate reports based on the orders the app received.

Group:30423

# CONTENT

[1. Objective 3](#_Toc95297885)

[2. Problem analysis, scenarios, use cases 3](#_Toc95297886)

[3. Design 3](#_Toc95297887)

[4. Implementation 3](#_Toc95297888)

[5. Results 3](#_Toc95297889)

[6. Conclusions 3](#_Toc95297890)

[7. Bibliography 3](#_Toc95297891)

## Assignemnt Objective

The objective of this assignment is to implement a food delivery management system for a restaurant/catering company. The main functionalities of the application are to enable clients to register and log in into the application, and choose from the company’s menu their desired meal, and then submit the order. The client will receive a bill containing his information and the order will be sent to the employees of the business. They will be notified in real time about incoming orders. The app will also have accounts with administrator access. They can update, add, and delete products from the app’s menu in order to maintain everything updated. They can also generate several reports to gather some information about how certain products are performing.

Sub-objectives:

* Understanding the requirements and performing the required actions to fulfill them.
* Mapping out the inner-working of the application – Design stage
* Implementation of the main Classes and then the service that performs the actions.
* Creating an intuitive user interface that also as a decent aspect because otherwise it will not be mine.
* Testing the application and finding bugs and fixing them or turning them into features.

## Problem Analysis

The application will be split into 5 main parts:

* Controller – which will eventually be the login form stage
  + - Enables clients to log in and register
    - Save new users and using serialization to store the data
* ClientScene – will represent the client’s graphical user interface and perform the required action with the help of the delivery service
  + - Enable clients to search the products
    - Help the client by sorting the data by its attributes(Rating, calories etc…)
    - Place and order
* EmployeeScene – this will be a new stage which will work congruently with the other stage
  + - this class will take the role of the observer, because it needs to be updated in real time while changes are made in the client side
* AdminScene – this class handles the admin specific actions.
  + - Enable the administrator to make changes on the application data
    - Help generate reports
    - Import items from a csv file
* DeliveryService – this will be the engine that will be accessible to the other main parts above. This will help with the tracking of the data and performing several operations amongst classes.
  + - Implements IDeliveryService
    - Implements Observable
    - Singleton

## Use case diagram:

Diagram

Description automatically generated

## Handling unintended scenarios

The application is not a deployable ready application and has more or less the purpose of the enabling students to practice and learn. Therefore some of the functionalities are not expected to work in the real world. This being said, the application still tries it’s best to handle unexpected/ unwanted scenarios. It tries to imitate real life features like a search feature.

The client is always expected to enter maybe not so expected data. Therefore the application should be able to do it’s best to handle scenarios where the client leaves required fields empty. Its shows several alerts to inform the user when such a situation occurs

Graphical user interface, application

Description automatically generated

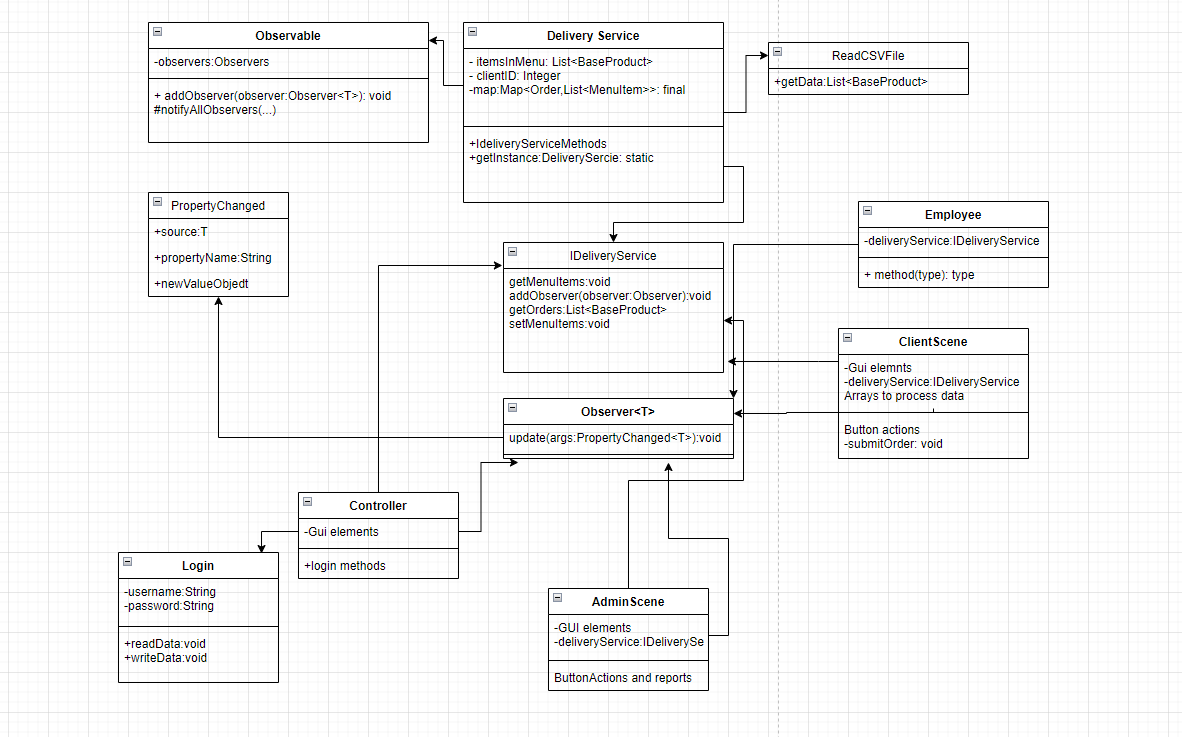
## 3 Design

Package diagram:

Diagram

Description automatically generated

UML Class diagram:



## Implementation

In my case, the BaseProducts class represents the base of any product, holding all the attribures:

Title, rating, calories, sodium, protein, price, fats. The MenuItem represents the base class and it holds the Title and the Price of the objects that extend it : BaseProduct and CompositeProduct;

This Class uses Composite Design Pattern in order to generate a menu items’prices.

These classes are used to hold the required data which will be used in the DeliveryService.

This project also uses the Observer Design Pattern.

public interface Observer<T> {  
 void update(PropertyChanged<T> args);  
}

There are several objects: Employee, Client, Administrator that make use of the observer to update themselves when important changes happen to the DeliveryService class which is Observable. The Delivery Service class holds the main Arrays used to hold the data and represent it.

public class Observable<T> {  
  
 private List<Observer<T>> observers = new ArrayList<>();  
  
 public void addObserver(Observer<T> observer) {  
 observers.add(observer);  
 }  
  
 protected void notifyObservers(T source, String propertyName, Object newValue) {  
 final PropertyChanged<T> changedEventArgs = new PropertyChanged<>(source, propertyName, newValue);  
 for (Observer<T> observer : observers) {  
 observer.update(changedEventArgs);  
 }  
 }  
}

The class Login implements Serializable in order to store username/password information in byte format.

It is capable of reading back the data and providing the Controller with the login information.

public class Login implements java.io.Serializable{  
 private String username;  
 private String password;  
 private String filename = "users.txt";  
  
 public Login(String username, String password){  
 this.password = password;  
 this.username = username;  
 }  
 public void writeData(){  
 try {  
 FileWriter myWriter = new FileWriter(filename, true);  
  
 myWriter.append(username).append(":").append(password).append("\n");  
 myWriter.close();  
 // System.out.println("success");  
 } catch (IOException e) {  
 System.*out*.println("An error occurred.");  
 e.printStackTrace();  
 }  
 }  
 public List<Login> readData() throws IOException{  
 Path path = Path.*of*("users.txt");  
 List<Login> list;  
 list = Files.*lines*(path)  
 .map(line-> {  
 String[] fields = line.split(":");  
 return new Login(fields[0],fields[1]);  
 }).toList();  
 return list;  
 }…

The Order class stores the orderID and the ClientId along with the Data. IT is used as the Key in the map where the main data is stored about the MenuItems and Orders currently present in the application.

It is used by several Classes to go through HashMaps that hold important data, like the menu items.

The IDeliveryService Interface holds all the main operations and most important. It helps with populating the Map with the Orders and the List with the available menu items. Whenever such a method is called, we also notify all the observers in the DeliveryService to make sure everything works as expected.

public interface IDeliveryService {  
 List<BaseProduct> getMenuItems();  
  
 void addOrder(Order order, List<MenuItem> list);  
  
 Map<Order, List<MenuItem>> getOrders();  
  
 void addObserver(Observer<IDeliveryService> observer);  
  
 Integer getOrderTotal(Order order);  
  
 void removeOrder(Order order);  
  
 void removeBaseProduct(BaseProduct baseProduct);  
  
 void addBaseProduct(BaseProduct baseProduct);  
  
 void setMenuItems(List<BaseProduct> list);  
  
 void removeItem(BaseProduct baseProduct);  
  
 void setClientID(String name);  
  
 Integer getClientID();  
}

The DeliveryService is built upon the singleton pattern. The class extends the Observable class and implements the IDeliveryService. It is called once using the getInstance Method.

This class also populates the MenuItem lists with the products.csv file by calling a static method .getData from the ReadCSVFile class. This by default reads the products.csv sent to us by the lab teacher, but it also can work with data if imported through the admin interface. There is products2.csv of smaller size meant for a better visualization of the updates present in the admin interface upon the menu items.

public class ReadCSVFile {  
 public static List<BaseProduct> getData(Path path) {  
 List<BaseProduct> products;  
  
 Set<String> titles = new HashSet<>();  
 try {  
 products = Files.*lines*(path)  
 .skip(1)  
 .filter(line -> !titles.contains(line.split(",")[0]))  
 .map(line -> {  
 String[] fields = line.split(",");  
 titles.add(fields[0]);  
 return new BaseProduct(fields[0], Double.*parseDouble*(fields[1]), Integer.*parseInt*(fields[2]),  
 Integer.*parseInt*(fields[3]), Integer.*parseInt*(fields[4]), Integer.*parseInt*(fields[5]), Integer.*parseInt*(fields[6]));  
  
 }).distinct().toList();  
 } catch (IOException e) {  
 e.printStackTrace();  
 return new ArrayList<>();  
 }  
 return new ArrayList<>(products);  
 }

There has been a lot of use of Lambda expressions in this project, here is an example where I used it to generate the orders in the Employees:

@Override  
public void update(PropertyChanged<IDeliveryService> args) {  
 //System.out.println(args.toString());  
 staffOrderList.setItems(FXCollections.*observableArrayList*(  
 deliveryService.getOrders().entrySet().stream()  
 .map(order -> {  
 String menuItems = order.getValue().stream()  
 .map(MenuItem::getTitle)  
 .collect(Collectors.*joining*("\n"));  
 String total = deliveryService.getOrderTotal(order.getKey()).toString();  
  
 String result = String.*format*("======Order %s:======\n%s\nTotal: %s$", order.getKey().getOrderID(), menuItems, total);  
  
 deleter.put(result, order.getKey());  
  
 return result;  
 })  
 .collect(Collectors.*toList*()))  
 );  
  
}

This kind of structure was used in several parts in the project and make processing data a bit easier. To be noted is the fact that some ListViews(used for presenting data to the user in the GUI) are declared to contain String objects instead of BaseProducts. It made life a bit more complicated so I needed those expressions in order to work with data. It was a bit confusing at the beginning but then it came more and more natural. It was a steep learning curve.

GUI – Here comes the fun part

This project uses JAVAFX framework aided by Scenebuilder.

Text

Description automatically generated with medium confidence

The Icon was made using several icons and ideas from other businesses. It was build using Paint3D, an unexpectedly easy and simple image editing software.

As you can see there are several prompts to provide the user with instant information about what kind of information is expected and how to better format it. It is especially useful in the AdminScene where the user has multiple textboxes.

Graphical user interface, text, application

Description automatically generated

The user is able to go through multiple windows and still keep the information due to the IDeliveryService and DeliveryService classes. This is how the information about orders persists through different stages. This is useful for generating the reports later on. Login information is stores indefinetly, even after restarting the application.

Graphical user interface, application

Description automatically generated

The UI is easy to use because the user can easily search his desiered items, then just select them from the ListView and click Add. You have a list of your order and the price total. After clicking submit, the information will be available for the employee in his own window. The submit also generates a bill in .txt format for the user, in similar format as the employee’s list view.

Here is an example of a Bill.txt:

========= Bill0 =========

Smoked-Salmon Quesadillas with Warm Tomatoes and Arugula

My Favorite Roast Turkey

My Favorite Roast Turkey

Date: 23/07/2022 20:49:13

========= Total: 62$ =========

## Results:

This application required a lot of knowledge that had to be learned and it had a lot of information about how it wanted to be done, where some things were “unclear”, or could be interpreted differently. I believe that it is good because we won’t then have the same project and it also makes you more engaged into thinking of finding solutions to your bad code that you wrote for 10 hours, and you don’t want to start over, and somehow make it work.

This project was definitely a challenge but I believe that I managed to deliver and make something that is worth checking out. I would consider in my free time working on these projects and maybe uploading them to my personal github, and further develop them into something that can mean something to maybe even future Employers.

The application behaves as long as it is ran normally. I ofcourse tried some weird scenarios and tried to limit test it as much as possible and I am pretty happy of how it turned out. I believe it is steady enough for how much I managed to play with it while trying to figure out while it breaks in one spot. It was usually a chain reaction of trying to fix or make something, then finding out while testing that thing that other parts don’t work, but in the end it turned out as stable as it could get.

## Conclusions

In conclusion, I was expecting this project to be easy compared to others, but it turned out to be the hardest one yet. It required to implement several design patterns and workflows like lambda expressions, and it demanded a lot of trial and error. Everything seemed so fragile and could break so easily. I also learned some interesting tricks in Intellij IDE which made working a bit easier as I progressed.

At the end, it was quite the experience, in simple words, it was stressful given the deadline that has to be met. The code as always is full of surprises, and you never know what bug or error could pop up that makes you work several additional hours for something so simple. Also once gaining the experience, the speed of the work was faster and faster, so everything turned out well in the end. I hope the project will meet its expectations as I sacrificed as much time I could to get it done in a good and colorful way.

## Bibliography:

Stack overflow ofc.

GeeksforGeeks.com

[(2051) Passing information between Scenes with Javafx - YouTube](https://www.youtube.com/watch?v=Wc1a2KshJ4w)

•[1] http://javahungry.blogspot.com/2013/08/hashing-how-hash-map-works-in-javaor.html •[2] https://howtodoinjava.com/java/collections/hashmap/how-hashmap-works-in-java/ •[3] http://javahungry.blogspot.com/2014/03/hashmap-vs-hashtable-difference-withexample-java-interview-questions.html •[4] http://javahungry.blogspot.com/2013/08/how-sets-are-implemented-internallyin.html •[5] https://www.baeldung.com/java-serialization •[6] https://www.geeksforgeeks.org/serialization-in-java/ •[7] <https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.htm>

Lambda expressions and stream processing -https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html o https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html o https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8- streams.html o https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/ o https://howtodoinjava.com/java8/java-stream-distinct-examples/ • Java serialization o http://www.tutorialspoint.com/java/java\_serialization.htm o https://www.baeldung.com/java-serialization o https://www.geeksforgeeks.org/serialization-in-java/ o https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html • Java HashMap o http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html