**Documentation Homework 4**

**Food Delivery Management System**

**Name: Popa Adriana**

**Group: 30423**

**Laboratory assistant: Antal Marcel**

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 (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: o 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. o the products ordered more than a specified number of times so far. o the clients that have ordered more than a specified number of times and the value of the order was higher than a specified amount. o 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. **Dimensions of the problem**
2. **Problem analysis**

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.

Define and implement the classes from the class diagram shown above:

* 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 structure of type Map> for storing the order related information in the DeliveryService class. The key of the Map will be formed of objects of type Order, for which the hashCode() method will be overwritten to compute the hash value within the Map from the attributes of the Order (OrderID, date, etc.).
* Define a structure of type Collection which will save the menu (i.e. all the products) provided by the catering company. Choose the appropriate collection type for your implementation.
* 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 (adapted from Link) using lambda expressions and stream processing. 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.

1. **Scenarios and use cases**

The scenario and the use cases are strongly connected to the user’s actions.

Use Case Name: Administrator

Actors:

* User logs in and selects an action
* Actions (executed by the system, not visible to the user)
* Result (Shown on the interface or printed in a txt file)

Triggers:

* The user indicates that he wants to do with the menu(add /delete /modify /generate reports).

Preconditions:

* User has chosen the action.
* User has introduced the correct data.

Post-conditions:

* The action will be performed.
* The incorrect introduced data will result in an alert box telling the error and the action will not be performed
* The result will be shown to the user.

Normal Flow:

1. The user will log in.
2. The user will introduce the data .
3. The user will chose an action.
4. The system will read the data.
5. The system will validate the data.
6. The system will execute the action.
7. The user will see the updates.
8. The user can chose another action or
9. The user will log of.

Alternate Flows:

1A1. The user introduces the data in a wrong/unaccepted way.

1. He will need to reintroduce the data in the accepted/correct form
2. The system will continue with the operation.

2.A1 The user selects another operation.

1. The selected operation will be executed instead.

For the client user the usecase is the same exept he has a sign up option too.

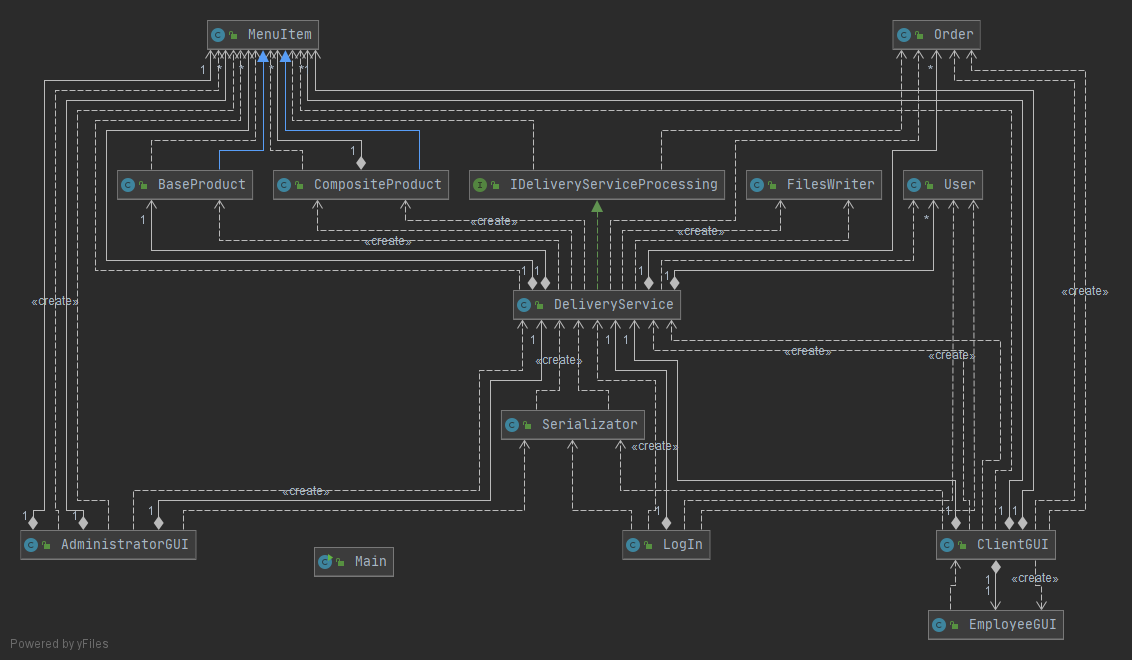
For the employee he only logs in an gets notified about clients order.

**3. Project design**

1. **Design decisions:**

I followed the class diagram presented in the requirements of the assignment. I added the necessary classes to improve the implement of the problem. The structure and the implementation strictly follows the requirements.

1. **UML Diagram**

****

**Diagram

Description automatically generated**

1. **Packages**

Java packages help in organizing multiple modules and group together related classes and interfaces.

My project uses the principle of layered architectures:

* Business Layer – contains the “brain” of the project
* Data Layer – contains the classes which saves and reads he
* Presentation layer – contains the GUIs of the project

1. **Data Structures**

The Map interface

•There are several data structures in Java that rely on the hash table: HashMap, Hashtable, LinkedHashMap, HashSet

•In order to implement the associative array structure, the Map interface was created.

•A Map in Java holds a collection of pairs key (K) and value (V) defined as: Entry<K,V>

•The various Map implementations differ through the underlying data structures:

◦Hash table: HashMap, Hashtable, LinkedHashMap

Hash Map in Java

•To understand Hashing in Java , we should understand the following terms :

◦*Hash Function*

◦*Hash Value*

◦*Bucket*

•According to the theory, an associative array/ map contains key-valuepairs. When implementing a hash table, the key is used to compute an index

•Java is an OOP language. The key is an Object.

In order to determine the bucket where to store the Entry<K,V>, two steps are required:

1. Compute a code from the K object

◦The Object class defines the method: public inthashCode()

◦This method has to be overriddenfor the K object to return an integer computed based on the object’s fields

◦The hashCodemethod should return the same integer for two equal, and different integers for different values

2. Apply the hash function on the code to determine the index

◦The code computed by the hashCodemethod is passed to the internal Java hash function that will compute the index of the bucket that stores the pair

◦Index = hash(hashCode(K))

◦The hash function implementation is kept internal in the java Collection framework for several reasons:

◦Performance

◦Automatic resize of the hash

Each bucket in Java contains a LinkedList.

•The Java implementation of Hashtablesolves collisions by chaining.

•After Java 1.8, the linked list was replaced by a binary search tree, so the worst case complexity was reduced from O(n) to O(log(n)).

Retrieving an element from the Map

•The hash stores pairs of form Entry<K,V>

•The get method returns the corresponding value V to the given key K

•If key is null (line 1) it returns the element from position 0 (HashMap allows only one null key but several null values)

•Otherwise, it applies the internal static hash function to the hash code of the key to obtain the index of the bucket

•Each bucket[index] contains a list with the elements Entry<K,V>with the property hash(hashCode(K))=index

•When returning the value V corresponding to the key K, the list is iterated and each key stored is compared with K using the equals method defined in K’s class.

•If a match is determined, the value V is returned. Otherwise, null is returned.

Adding an element to the Map

The method has the following signature: V put(K key, V value)

1.The index of the bucket is computed as

◦index = hash(hashCode(K))

2.The linked list from bucket[index] is traversed and each element K is compared using equalswith key

◦If a stored element equalswith the keythen the corresponding value is overriden

◦If no element is found, the pair <key, value> is added to the list stored in bucket[index]

Implementing the Key Object

•Taking into account the mechanisms for get and put, the class that is the type of the Key must override the following methods:

◦hashCode–in order to generate a number for each Object. This number will be the input of the internal hash function of the HashMap.

◦equals-in order to compare key equality for get operationor to check if the object exists in the map, in case of put operation.

**4. Implementation**

The problem is divided into smaller problems and then you solve these little, simple and well-known problems.

1. Business Layer – contains the logic of the application

* User Class

There are three types of users, each having a username, a password and a function declared as Strings.

Constructors:

public User(String username, String password, String function)

-initialize the user with the corresponding parameters

Getters and setters:

public String getFunction() -getter for the function of the user  
public void setFunction(String function) – setter for the function of the user  
public String getUsername() - getter for the username  
public void setUsername(String username) – setter for the username  
public String getPassword() – getter for the password   
public void setPassword(String password) – setter for the password

* Order Class

This class contains the OrderID, ClientID, OrderDate and OrderTime which is our polynom.

Constructors:

public Order(int orderID, int clientID, LocalDate orderDate, LocalTime orderTime)

-initialize the user with the corresponding parameters

Getters and setters:

public int getOrderID() – getter for order id  
public void setOrderID(int orderID) – setter for order id  
public int getClientID() – getter for order id  
public void setClientID(int clientID) – setter for order id   
public LocalDate getOrderDate() – getter for order date  
public void setOrderDate(LocalDate orderDate) – setter for order date  
public LocalTime getOrderTime() – getter for order time  
public void setOrderTime(LocalTime orderTime) – setter for order time

Methods:

public int hashCode() – Overrides the hashCode function for the Hash Map

* MenuItem Class

This class contains:

protected String title;  
protected double rating;  
protected int calories;  
protected int protein;  
protected int fat;  
protected int sodium;  
protected int price;

Constructors:

public MenuItem(String title, double rating, int calories, int protein, int fat, int sodium, int price)

-initializes the instances with the corresponding values.

Getters and setters:

public int getFat()  
public void setFat(int fat)   
public String getTitle()  
public void setTitle(String title)   
public double getRating()  
public void setRating(double rating)   
public int getCalories()  
public void setCalories(int calories)   
public int getProtein()  
public void setProtein(int protein)   
public int getSodium()  
public void setSodium(int sodium)   
public int getPrice()  
public void setPrice(int price)

* BaseProduct class

- extend MenuItem class

Constructors:

public BaseProduct(String title, double rating, int calories, int protein, int fat, int sodium, int price)

Methods:

public int computePrice()

* CompositeProduct class

-this class extend MenuItem class

private ArrayList<MenuItem> menuItems ;

Constructors:

public CompositeProduct(String title, double rating, int calories, int protein, int fat, int sodium, int price)

public CompositeProduct(String title,ArrayList<MenuItem> menuItem)

Methods:

public int computePrice()

* IDeliveryServiceProcessing interface

Methods:

void importProducts();  
void addMenuItem(MenuItem menuItem);  
void deleteMenuItem(MenuItem menuItem);  
void modifyMenuItem(MenuItem menuItem);  
void createCompositeProduct(ArrayList<MenuItem> menuItems, String name);  
void createOrder(Order order, ArrayList<MenuItem> menuItem);  
void finishOrder(Order order);  
int computeOrderPrice(Order order);  
void generateReports();  
ArrayList<MenuItem> searchKeyword(String keyword);  
ArrayList<MenuItem> sortRating(double min,double max,ArrayList<MenuItem> myMenu);  
ArrayList<MenuItem> sortCalories(int min,int max,ArrayList<MenuItem> myMenu);  
ArrayList<MenuItem> sortProtein(int min,int max,ArrayList<MenuItem> myMenu);  
ArrayList<MenuItem> sortFat(int min,int max,ArrayList<MenuItem> myMenu);  
ArrayList<MenuItem> sortSodium(int min,int max,ArrayList<MenuItem> myMenu);  
ArrayList<MenuItem> sortPrice(int min,int max,ArrayList<MenuItem> myMenu);

* DeliveryService class

private ArrayList<MenuItem> menu ;  
private Map<Order, ArrayList<MenuItem>> orderMenu ;  
private ArrayList<Order> orders;  
private ArrayList<User> users;  
public static PropertyChangeSupport *support*;  
public static String *messageE*;

Constructors:

public DeliveryService()

Getters and Setters:

public ArrayList<User> getUsers()  
public void setUsers(ArrayList<User> users)   
public ArrayList<MenuItem> getMenu()  
public void setMenu(ArrayList<MenuItem> menu)   
public Map<Order, ArrayList<MenuItem>> getOrderMenu()  
public void setOrderMenu(Map<Order, ArrayList<MenuItem>> orderMenu)   
public ArrayList<Order> getOrder()  
public void setOrders(ArrayList<Order> orders)

Methods:

public <T> Predicate<T> distinctByKey(Function<? super T, String> keyExtractor)

public void importProducts()

public void addMenuItem(MenuItem menuItem)

public void deleteMenuItem(MenuItem menuItem)

public void modifyMenuItem(MenuItem menuItem)

public void createCompositeProduct(ArrayList<MenuItem> menuItems,String name)

public void createOrder(Order order, ArrayList<MenuItem> menuItem)

public void finishOrder(Order order)

public int computeOrderPrice(Order order)

public void generateReports()

public ArrayList<MenuItem> searchKeyword(String keyword)

public ArrayList<MenuItem> sortRating(double min,double max,ArrayList<MenuItem> myMenu)   
public ArrayList<MenuItem> sortCalories(int min,int max,ArrayList<MenuItem> myMenu)   
public ArrayList<MenuItem> sortProtein(int min,int max,ArrayList<MenuItem> myMenu)   
public ArrayList<MenuItem> sortFat(int min,int max,ArrayList<MenuItem> myMenu)   
public ArrayList<MenuItem> sortSodium(int min,int max,ArrayList<MenuItem> myMenu  
public ArrayList<MenuItem> sortPrice(int min,int max,ArrayList<MenuItem> myMenu)   
private void writeObject(java.io.ObjectOutputStream stream) throws IOException   
private void readObject(java.io.ObjectInputStream stream) throws IOException, ClassNotFoundException

1. Data Layer

* Serializator class

This class serializes and deseriazises the informations used by the application such that it it not lost when switching users and closing the application

Methods:

public static void serialize(DeliveryService deliveryService) - serialize the data from deliveryService

public static DeliveryService deserialize() – deserialize the data from deliveryService

* FileWriter Class

This class creates and updates the .txt files used by the application

Methods:

public void generateBill(String s) – write the bills in a txt file

public void generateReports(String s) – write admin reports in a txt file

1. Presentation Layer

* LogIn Class

Controlls the log in interface:

Methods:

public void initialize()

public void showError(String s,String s1)

public boolean validateAdministrator(String user,String password)

public boolean validateEmployee(String user,String password)

public boolean validateClient(String user,String password)

public void logInButton(ActionEvent event) throws IOException

public boolean validateClientForSign(String user)

public void signUpButton()

public void exit()

* AdministratorGUI Class

Controlls the Administrator interface.

Methods:

public void initialize()  
public void showError(String s)   
public void show()  
public void importProduct()  
public boolean validateData()  
public void add()  
public void modify()  
public void delete()  
public void addComposite()  
public void generateReports()  
public void logOff(ActionEvent event) throws IOException

* EmployeeGUI Class

Methods:

public void initialize  
public void logOff(ActionEvent event) throws IOException

* ClientGUI Class

Methods:

public void initialize()

public void show( ArrayList<MenuItem> menu)

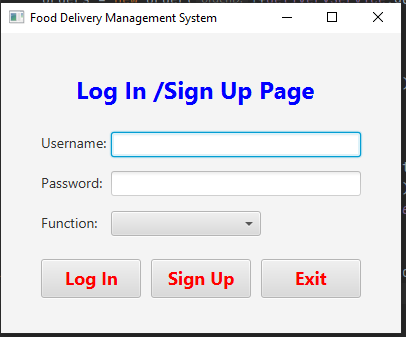
public void search()

public void orderButton()

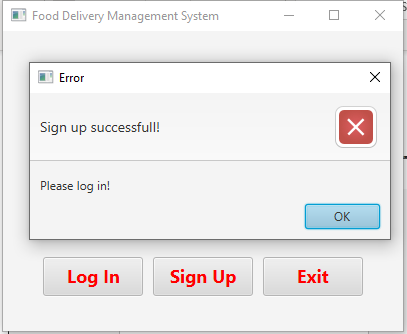
public void logOff(ActionEvent event) throws IOException

**5 .Results**

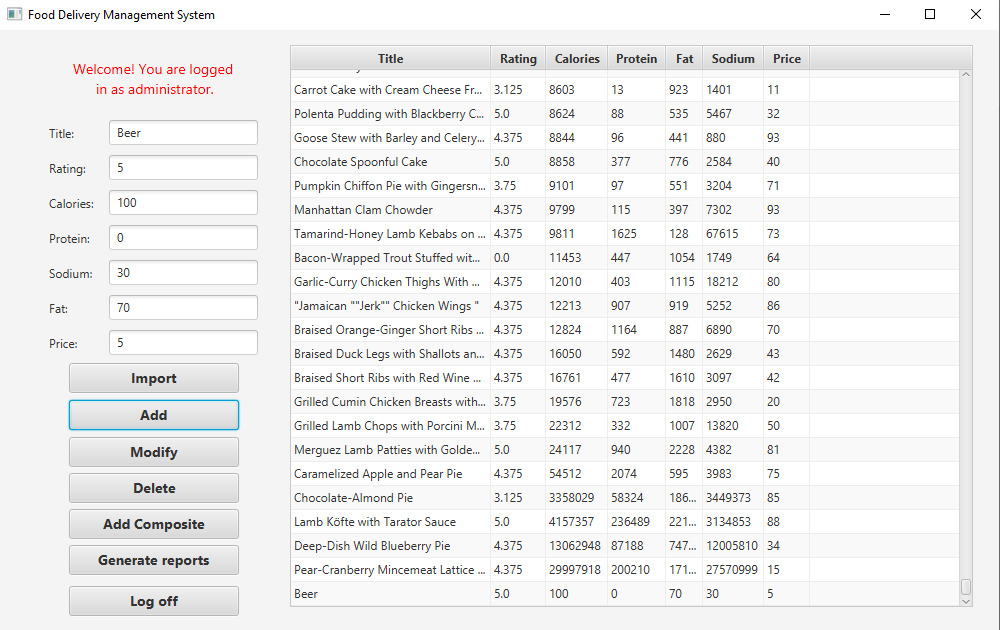
Log In window



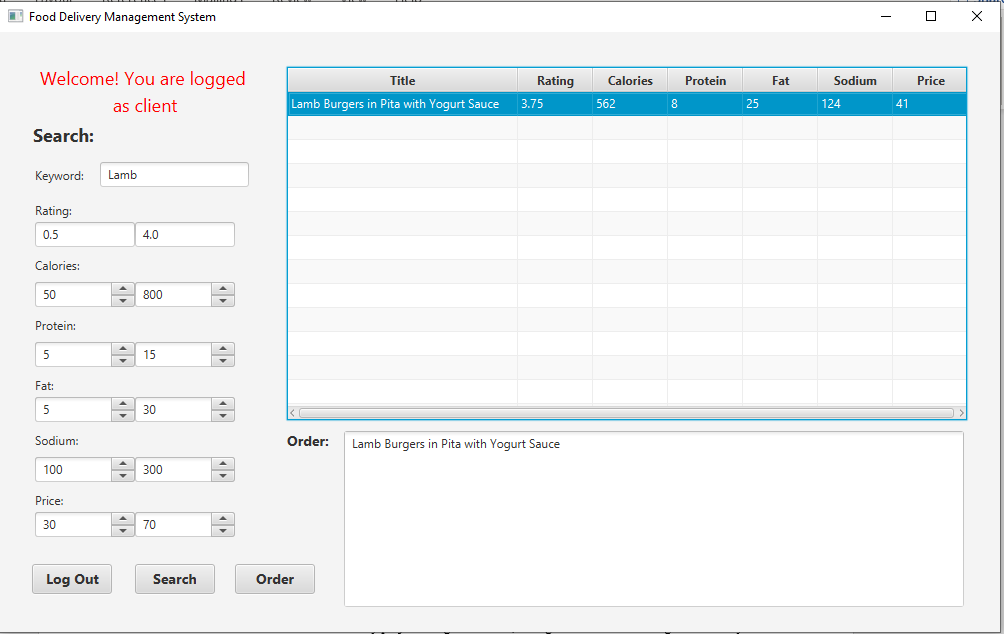
Successful sign in:



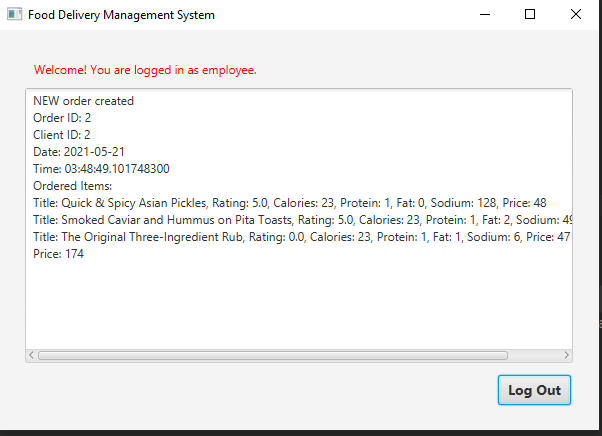
Administrator window:



Client window:



Employee window:



**6.Conclusions**

First of all, the project was a good opportunity to remember the OOP concepts learned this semester. I learned how to use maven and how to test my project using Junit. Also, it taught me that time management is very important and even if I had a very bad time management, I tried my best to finish this this project and for it to work properly.

Secondly, there are a few improvements, more like new operations that can be added to the project:

- creating an interface for the delivery guy, which tells when he pics up the order and when it is delivered

- upgrading the employee such that he decides which delivery guy takes which command and the order the orders are processes

-allowing the clients to propose their own daily menu

**7.Bibliography**

* <https://stackoverflow.com/questions/19490868/how-to-unselect-a-selected-table-row-upon-second-click-selection-in-javafx>
* <https://docs.oracle.com/javase/8/docs/api/java/time/LocalTime.html>
* <https://stackoverflow.com/questions/13777936/java-property-change-listener>
* <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>
* <https://medium.com/@KonfHub/how-to-process-csv-files-in-java-with-streams-a69221ae46e6>
* <https://docs.oracle.com/javase/tutorial/uiswing/events/propertychangelistener.html>
* <https://stackoverflow.com/questions/20879242/get-checkbox-value-in-a-table-in-javafx>
* <https://stackoverflow.com/questions/46380073/observer-is-deprecated-in-java-9-what-should-we-use-instead-of-it>
* <https://stackoverflow.com/questions/17702479/a-well-formed-java-string>
* Programming Techniques – Lectures of prof. Ioan SALOMIE
* Object-Oriented Programming - Lecture Slides of prof. Marius JOLDOS