**TEHNICI DE PROGRAMARE FUNDAMENTALA**

**TEMA 4**

**FOOD DELIVERY MANAGEMENT SYSTEM**

Nume: Vultur Ana-Maria

Grupa: 30223

**Cuprins:**

1. Obiectivul temei........................................................................................3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare.......................3
3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)...................................................................................................5
4. Implementare.............................................................................................9
5. Rezultate...................................................................................................11
6. Concluzii...................................................................................................12
7. Bibliografie...............................................................................................13
8. **Obiectivul temei**

**Obiectivul principal**

Obiectivul principal al temei este de a implementa un sistem de management a livrarii mancarii pentru o companie de catering, unde un client poate comanda produse din meniul companiei. Sistemul contine 3 tipuri de utilizatori care se logheaza utilizand un nume si o parola: administrator, angajat si client.

Fiecare din cei 3 utilizatori are diferite functionalitati:

-Administratorul poate sa importe setul initial de produse si sa genereze meniul, poate sa adauge, stearga, modifice produse sau sa creeze noi produse compuse din mai multe produse si sa genereze diferite rapoarte legate de comenzile facute in functie de anumite criterii.

-Clientul poate sa se inregistreze si sa utilizeze numele inregistrat pentru a se loga, sa vada lista de produse din meniu, sa caute produsele din meniu dupa diferite criterii, si sa creeze comenzi formate din mai multe produse, iar pentru fiecare comanda se va genera o factura cu data si ora creearii, lista de produse comandate si pretul total al comenzii.

-Angajatul este atentionat de fiecare data cand o noua comanda este creata de un client, pentru a sti cand sa incepa sa pregateasca livrarea produselor comandate.

**Obiective secundare**

|  |  |
| --- | --- |
| **Obiectiv Secundar** | **Descriere** |
| Dezvoltarea de use case-uri si scenarii | Într-un sistem software un ‘use-case’ este o listă de acțiuni sau pași de eveniment care definesc în mod obișnuit interacțiunile dintre un rol ( cunoscut ca un actor în limbajul unificat de modelare ( UML ) ) și un sistem in atingerea unui obiectiv.  In computere, un scenariu este o naratiune a interactiunilor dintre roluri ( actori in UML ) si sistem. In cazul nostru rolul - un utilizator al aplicatiei. |
| Alegerea structurilor de date | Structurile de date folosite pentru a duce la capat obiectivul principal |
| Impartirea pe clase | Clasele se pot imparti in 3 pachete pe baza arhitecturii de tipul Layered. |
| Dezvoltarea algoritmilor | Vor fi descrise structurile de date necesare pentru atingerea obiectivului principal, schema UML ( diagrama de clase ) precum si strategia folosita pentru a gestiona livrarea produselor. |
| Implementarea solutiei | Vor fi descrise fiecare clasa cu campurile si metodele importante precum si descrierea interfetei utilizator; |
| Testare | Rezultatele vor fi scrise in fisiere .txt |

1. **Analiza problemei**

Prin analiza problemei, ne referim la un prim set abstract de operații și proprietăți prin care încercăm să depistăm eventualele însușiri și comportamente ale proceselor necunoscute.

Aplicatia trebuie sa simuleze procesarea comenzilor clientilor unei companii de catering. Problema se împarte în mai mulți pași după cum ne și sugerează cerința. Pentru o astfel de aplicatie avem de nevoie de o clasa care contine produsele din meniu. Trebuie sa definim operatiile corespunzatoare pentru adaugarea, stergerea, modificarea produselor, creearea comenzilor, generarea facturilor.

Interfata grafica cu utilizatorul este diferita pentru fiecare tip de utilizator.

**Use Case-uri**

Pentru fiecare tip de utilizator exista un nume de utilizator si o parola, care se introduc in fereastra de log in pentru a putea accesa fereastra corespunzatoare fiecaruia.

**Use Case 1**: Adauga, creeaza, actualizeaza meniul

Actor principal: administrator

Principalele scenarii de succes:

1. Administratorul introduce in fereastra de log in username-ul: admin si parola: admin, dupa care apasa butonul Administrator si i se deschide fereastra corespunzatoare.
2. Administratorul introduce date noi pentru un produs de tipul BaseProduct
3. Administratorul poate alege sa adauge sa modifice, sa adauge sau sa stearga produsul cu datele introduse, apasand butonul corespunzator operatiei.
4. Sistemul preia informatia, iar in cazul in care datele sunt corecte, se va efectua operatia.

Scenarii de esec: - utilizatorul introduce date gresite, in acest caz se va afisa un mesaj de eroare.

* utilizatorul introduce nume si parola gresite la logare.

**Use Case 2**: Creaza meniu

Actor principal: client

Principalele scenarii de succes:

1. Clientul introduce in fereastra de log in username-ul: client si parola: client, dupa care apasa butonul Client si i se deschide fereastra corespunzatoare.
2. Clientul introduce in textField numele produsului pe care doreste sa il adauge la comanda.
3. Daca doreste sa adauge mai multe porduse, apasa pe butonul Add Product si introduce numele urmatorului produs. Aceasta operatie se repeta pana cand clientul nu mai doreste sa adauge si alte porduse la comanda.
4. Cand a terminat de adaugat toate produsele la comanda, clientul apasa pe butonul de Finish Order.
5. Comanda este salvata, este transmisa angajatului si o factura in format .txt este generata.

Scenarii de esec: - utilizatorul introduce un nume de produs care nu se regaseste in meniu, in acest caz se va afisa un mesaj de eroare

* utilizatorul introduce nume si parola gresite la logare.

**Use Case 3**: Vizualizeaza comenzi

Actor principal: angajat

Principalele scenarii de succes:

1. Angajatul vizualizeaza in fereastra corespunzatoare comenzile, pe masura ce acestea sunt create de clienti.

**Scenarii**

Se alege din interfata grafica fereastra corespunzatoare utilizatorului pe care vrem sa il folosim in aplicatie. Fiecare utilizator are o interfata diferita si are functionalitati diferite. Angajatul este doar un observator care nu face nicio actiune, doar observa comenzile care sunt create de clienti, primind cate o notificare la fiecare comanda inregistrata in sistem. Administratorul poate modifica meniul si genera rapoarte in legatura cu comenzile create, iar clientul poate doar sa vada meniul, sa creeze comenzi si sa se inregistreze in sistem cu un nume de utilizator si o parola.

1. **Proiectare**

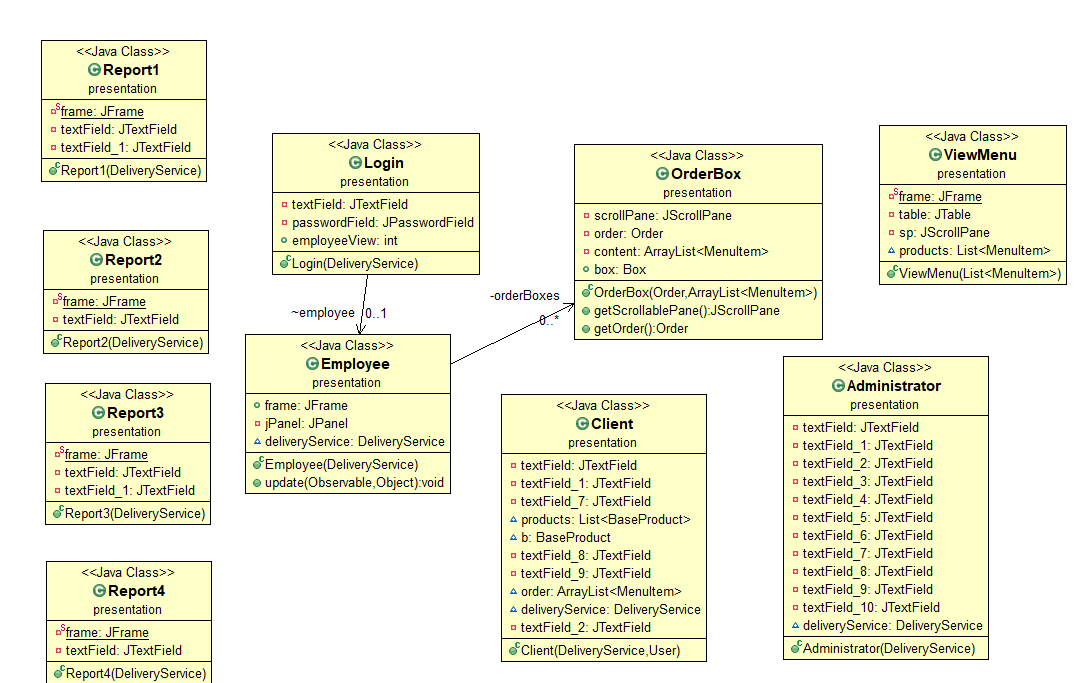
Proiectul este construit folosind arhitectura Layered, reprezentata de 3 pachete importante :

* Presentation layer care contine clasele necesare implementarii interfetei grafice,
* Business layer care contine clase precum: Order, DeliveryService, MenuItem. Pentru design-ul clasei MenuItem este folosit Composite Design Pattern, unde am BaseProducts si CompositeProducts compuse din base products. Ambele categorii: BaseProducts si CompositeProducts, reprezinta un MenuItem care este o clasa abtracta extinsa de cele 2. DeliveryService implementeaza operatiiile descrise in IDeliveryServiceProcessing.
* Data layer care contine clasa Serializator si FileWriter, care salveaza si incarca datele dintr un fisier. Clasa Serializator face serializarea si deserializarea datelor si clasa FileWriter genereaza factura intr-un fisier .txt.

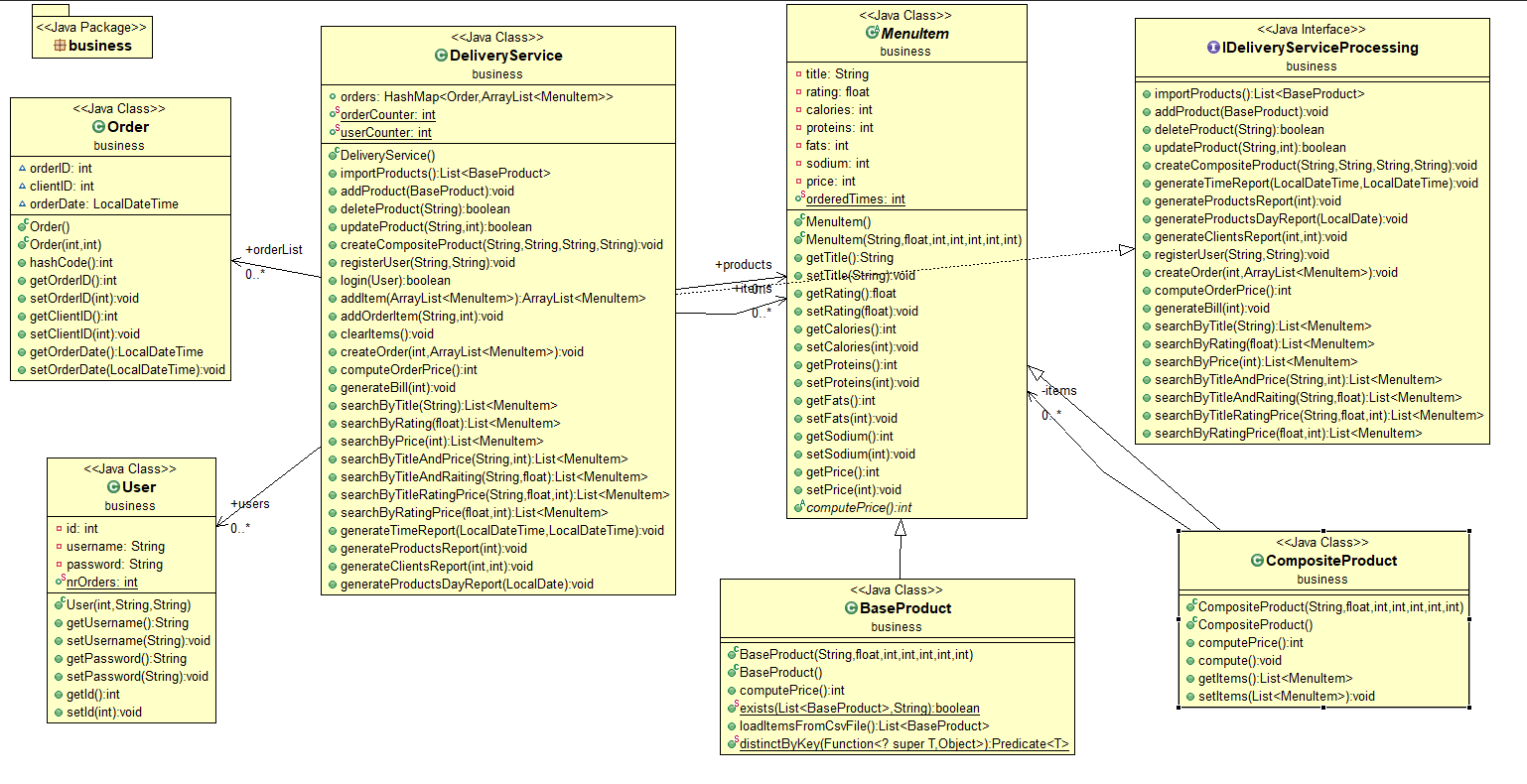
**Diagrama de clase de start:**

****

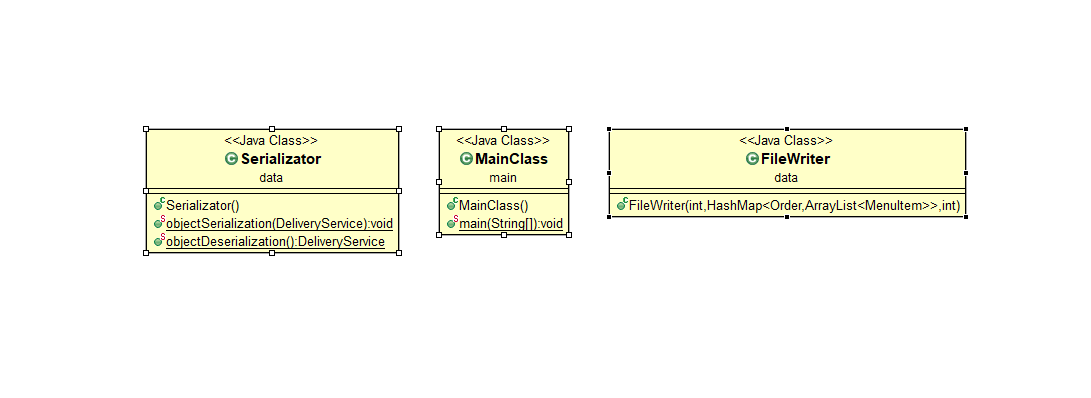
**Diagrama de clase din pachetul business:**



**Diagrama de clase din pachetul presentation:**



**Diagrama de clase din pachetele main si data:**



**Structuri de date**

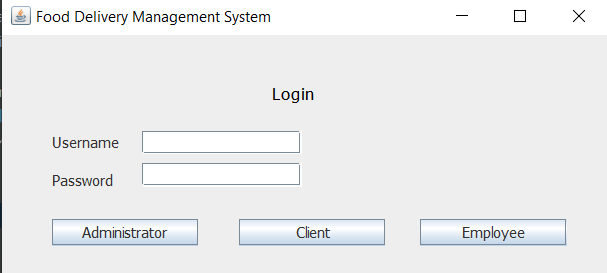
Există câteva structuri de date importante utilizate din Java Collections Framework. Pentru stocarea meniului companiei, se folosește ArrayList.

Comenzile sunt stocate într-un HashMap având ca cheie și obiect Order și ca valoare o listă (ArrayList a fost utilizată) care conține produsele selectate (HashMap <Order, List <MenuItem>>). Aici se folosește o listă în locul unui set, deoarece la această parte pot apărea duplicate.

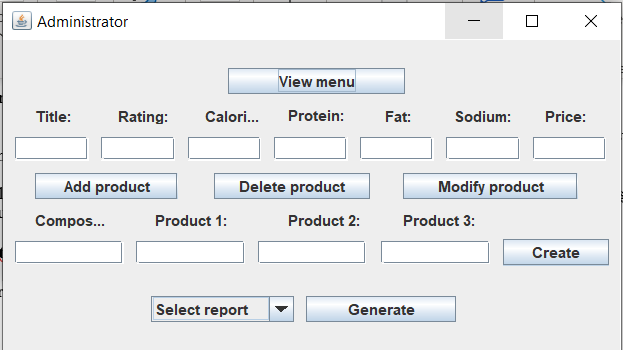
public HashMap<Order, ArrayList<MenuItem>> orders = new HashMap<>();  
public List<MenuItem> products = new ArrayList<>();

**Interfata grafica cu utilizatorul:**

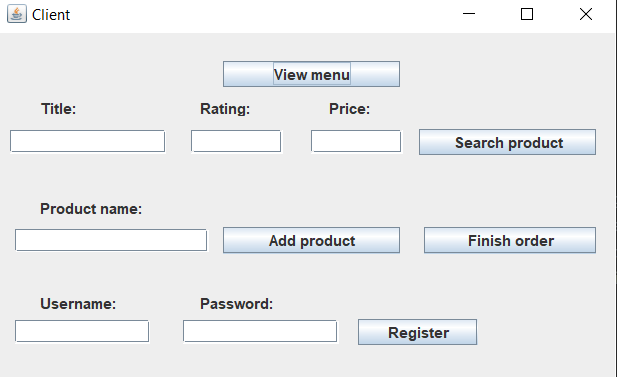
Fereastra de login:



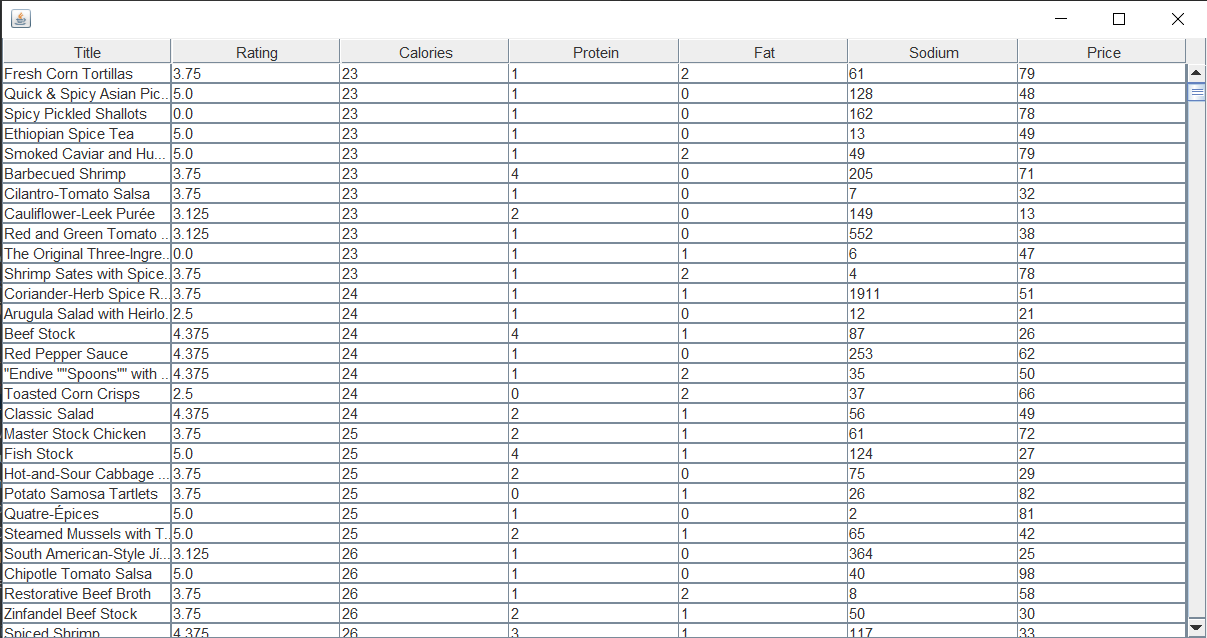
Fereastra pentru administrator:



Fereastra pentru client:



Vizualizarea meniului:



1. **Implementare**

Am folosit 4 pachete:

* business: contine clasele care incapsuleaza logica aplicatiei
* data: contine clasele de serializare si scriere in fisier
* presentation: contine clasele care definesc interfata utilizatorului
* main: contine clasa cu metoda principala de unde se porneste aplicatia

**Package main:**

Clasa **Main**:

-clasa principala de testare aplicatiei

-creeaza o instanta a clasei Login care porneste fereastra de inceput aplicatiei

**Package presentation:**

-contine clasele necesare pentru a realiza interfata grafica cu utilizatorul

-clasele Administrator, Client, Employee, Login sunt ferestrele principale

-clasa ViewMenu este fereastra de afisare a unui tabel cu produsele din meniu

-clasele Report1, Report2, Report3, Report4 sunt clasele de unde se genereaza rapoartele de catre administrator

-clasa OrderBox, reprezinta box-urile care sunt adaugate in Employee cand este creata o comanda

**Package data:**

Clasa **FileWriter**:

-genereaza o factura in format txt cand un client creeaza o comanda noua

Clasa **Serializator**:

-contine 2 metode: o metoda pentru serializare, salvarea datelor despre meniu si comenzi intr-un fisier .ser, si o metoda pentru citirea datelor de acolo.

**Package business:**

Clasa **BaseProduct**:

-pentru definirea acestei clase am folosit Composite Design Pattern

-extinde clasa MenuItem

-reprezinta un produs de baza care are atributele clasei MenuItem

-suprascrie metoda computePrice() care in aceasta clasa returneaza pur si simplu pretul produsului

Clasa **CompositeProduct**:

-pentru definirea acestei clase am folosit Composite Design Pattern

-extinde clasa MenuItem

-reprezinta un pordus compus din mai multe produse de baze (BaseProduct)

-suprascrie metoda computePrice() care aici returneaza suma preturilor tuturor produselor din care este compus produsul de tipul CompositeProduct.

Clasa **DeliveryService**:

-implementeaza interfata DeliveryService

-are ca instante:

public HashMap<Order, ArrayList<MenuItem>> orders = new HashMap<>();  
public List<MenuItem> products = new ArrayList<>();  
public ArrayList<Order> orderList = new ArrayList<>();  
public static int *orderCounter* = 1;  
public static int *userCounter* = 1;  
public List<User> users = new ArrayList<>();

-clasa DeliveryService este centrul aplicatiei, toate actiunile, functionalitatile se gasesc in aceasta clasa.

-este construita folosind Design by Contract, testand preconditiile si postconditiile folosind instructiuni assert in toate metodele implementate in aceasta clasa.

Interfata **IDeliveryServiceProcessing**:

-interfata care este implementata de DeliveryService

-contine metodele pentru implementarea functionalitatilor pentru client si administrator:

public List<BaseProduct> importProducts();  
public void addProduct(BaseProduct menuItem);  
public boolean deleteProduct(String menuItemName);  
public boolean updateProduct(String menuItemName, int price);  
  
public void createCompositeProduct(String title, String prod1, String prod2, String prod3);  
  
public void generateTimeReport(LocalDateTime start, LocalDateTime finish) throws FileNotFoundException;  
  
public void generateProductsReport(int nr) throws FileNotFoundException;  
  
public void generateProductsDayReport(LocalDate date) throws FileNotFoundException;  
  
public void generateClientsReport(int nr, int value) throws FileNotFoundException;  
  
//client  
public void registerUser(String username, String password);  
public void createOrder(int userID, ArrayList<MenuItem> items) throws FileNotFoundException;  
public int computeOrderPrice();  
public void generateBill(int orderID);  
  
public List<MenuItem> searchByTitle(String title);  
  
public List<MenuItem> searchByRating(float rating);  
  
public List<MenuItem> searchByPrice(int price);  
  
public List<MenuItem> searchByTitleAndPrice(String title, int price);  
  
public List<MenuItem> searchByTitleAndRaiting(String title, float rating);  
  
public List<MenuItem> searchByTitleRatingPrice(String title, float rating, int price);  
  
public List<MenuItem> searchByRatingPrice(float rating, int price);

Clasa abstracta **MenuItem**:

-contine urmatoarele atribute:

private String title;  
private float rating;  
private int calories;  
private int proteins;  
private int fats;  
private int sodium;  
private int price;  
public static int *orderedTimes*;

-contine metode de getters si setters pentru accesul la variabilele instanta ale clasei

-este o clasa abstracta care poate fi BaseProduct sau CompositeProduct

-pentru definirea acestei clase am folosit Composite Design Pattern

Clasa **Order**:

-contine variabilele orderID, clientID si orderDate, atributele specfice unei comenzi

int orderID;  
int clientID;  
LocalDateTime orderDate;

-contine metoda hashCode() care este suprascrisa

Clasa **User**:

-este clasa care contine un utilizator de tipul client, intrucat clientii se pot inregistra in sistem utilizand un nume de utilizator si o parola

-are ca si atribute:

private int id;  
private String username;  
private String password;  
public static int *nrOrders*;

1. **Rezultate**

La pornirea aplicatie se alege ce utilizator va folosi aplicatia, apoi se redirectioneaza utilizatorul spre fereastra corespunzatoare. Acestia pot introduce date in campurile disponibile si sa execute functionalitatile fiecaruia.

Pentru fiecare comanda se creaza un nou fisier si aceasta se introduce in fereastra angajatului.

Continutul unui fisier care contine o comanda:

Order: 4  
Date: 2021-05-18 19:55  
Client: 1  
Price: 130  
  
Products:   
Name: Quatre-Épices Price: 81 RON  
Name: Classic Salad Price: 49 RON

Fereastra pentru angajat dupa ce au fost facute cate comenzi:



1. **Concluzii**

Concluziile includ faptul că o planificare bine gândită a claselor ușurează lucrurile și economisește timp. Odată cu creșterea dificultății în proiecte, trebuie abordată problema cu o perspectivă inteligentă și nu doar să sari imediat la scrierea codului. Astfel, începând cu diagramele UML mi-am dat timp să reflectez pentru abordarea adecvată a implementării programului dorit.

In urma implementarii acestui proiect am invatat sa gestionez si sa prelucrez informatiile din tabele pentru a le putea afisa si pentru a putea interactiona utilizatorul cu ele, sa folosesc stream-uri si expresii lambda, care mi-au usurat foarte mult munca, ajuntandu-ma sa fac cautari scriind mai putin cod.

Alte aspecte pe care le-am învățat au fost implementarea Design by Contract, Composite Design Pattern, Observer Design Pattern, care m-au ajutat să înțeleg cum să folosesc un astfel de model, care sunt avantajele acestuia și de ce ar trebui să lucrez cu acesta.

Consider ca, o dezvoltare ulterioara ar putea fii aceea de a adauga si o simulare in timp real a comenzilor, masurarea timpului de livrare a practic o aplicatie de simulare apropiata cu realitatea, implementarea mai multor functionalitati, o interfata cu utilizatorul mai prietenoasa.

1. **Bibliografie**

<http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html>

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

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

https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/

https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html

http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html

https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag