

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE



UNIVERSITATEA TEHNICĂ

DIN CLUJ-NAPOCA

**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
CATEDRA CALCULATOARE**

foodDeliveryManagement System Documentație

Dimitriu David

Grupa 30228 | An II semestrul 2

Cuprins

1. Obiectivul temei.	
2. Analiza problemei, modelare, scenarii, cazuri de utilizare.	
3. Proiectare.	
4. Implementare.	
5. Rezultate.	
6. Concluzii	
7. Bibliografie.	

1. Obiectivul temei:

Obiectivul temei a fost sa proiectam și sa implementam un sistem de management al livrării de alimente pentru o companie de catering. Clientul poate comanda produse din meniul companiei. Sistemul ar trebui să aibă trei tipuri de utilizatori care se conectează folosind un nume de utilizator și o parolă: administrator, angajat obișnuit și client.

Programul va avea o interfață grafică dedicată, concepută pentru a fi intuitivă, atractivă pentru utilizatori, atât clienți, cât și angajați, administratori, iar pentru fiecare comandă i se va crea un fișier text cu rol de confirmare a comenzii și bon fiscal, pe care vor apărea datele clientului, numărul comenzii, data și ora comenzii, produsele comandate (cu cantitățile respective), prețul unitar și total al fiecărui produs produs dar și plata totală a comenzii.

Cerinta temei :

Requirement	Grading
Minimum to pass <ul style="list-style-type: none">• Object-oriented programming design, classes with maximum 300 lines, methods with maximum 30 lines, Java naming conventions• Implement the class diagram from Section 1. Choose appropriate data structures for saving the <i>Orders</i> and the <i>MenuItems</i>.• Define the class <i>BaseProduct</i> with the following fields: title, rating, calories, proteins, fats, sodium, price. Read the data from the file <i>products.csv</i> using streams and split each line in 7 parts: <i>title</i>, <i>rating</i>, <i>calories</i>, <i>protein</i>, <i>fat</i>, <i>sodium</i>, <i>price</i>, and create a list of objects of type <i>BaseProduct</i>. NOTE: the file contains duplicate dishes so make sure you select only one.• Graphical interface:<ul style="list-style-type: none">○ Log in window	5 points

<ul style="list-style-type: none"> ○ Window for Administrator operations (see the requirements in Section 1) ○ Window for Client operations (see the requirements in Section 1) • Use lambda expressions and stream processing for generating the administrator specific reports (see Section 1). • Use lambda expressions and stream processing to implement the search functionalities available to the client (see Section 1). • Good quality documentation covering the sections from the documentation template. 	
Use the Composite Design Pattern for modelling the classes MenuItem, BaseProduct, CompositeProduct.	1 point
Create bill in .TXT format.	0.5 points
Design by contract: preconditions and postconditions in the <i>IDeliveryServiceProcessing</i> interface. Implement them in the <i>DeliveryService</i> class using the assert instruction. Define an invariant for the class <i>DeliveryService</i> . Generate the corresponding JavaDoc files which should include the custom tags and descriptions associated to the defined pre, post conditions and invariants.	1.5 points
Window for the employee user: use Observer Design Pattern to notify each time a new Order is added.	1 point
Save the information from the <i>DeliveryService</i> class in a file (i.e., file.txt) using serialization. Load the information when the application starts.	1 point

2. Analiza problemei, modelare, scenarii, cazuri de utilizare:

Modelarea problemei se face în mare parte conform exemplului atașat prezentării temei. În cazul de admin, acesta poate importa un fisier csv, poate adauga, sterge, edita un produs din lista și poate crea un meniu format din produse. Generați rapoarte despre comenzile efectuate luând în considerare următoarele criterii:

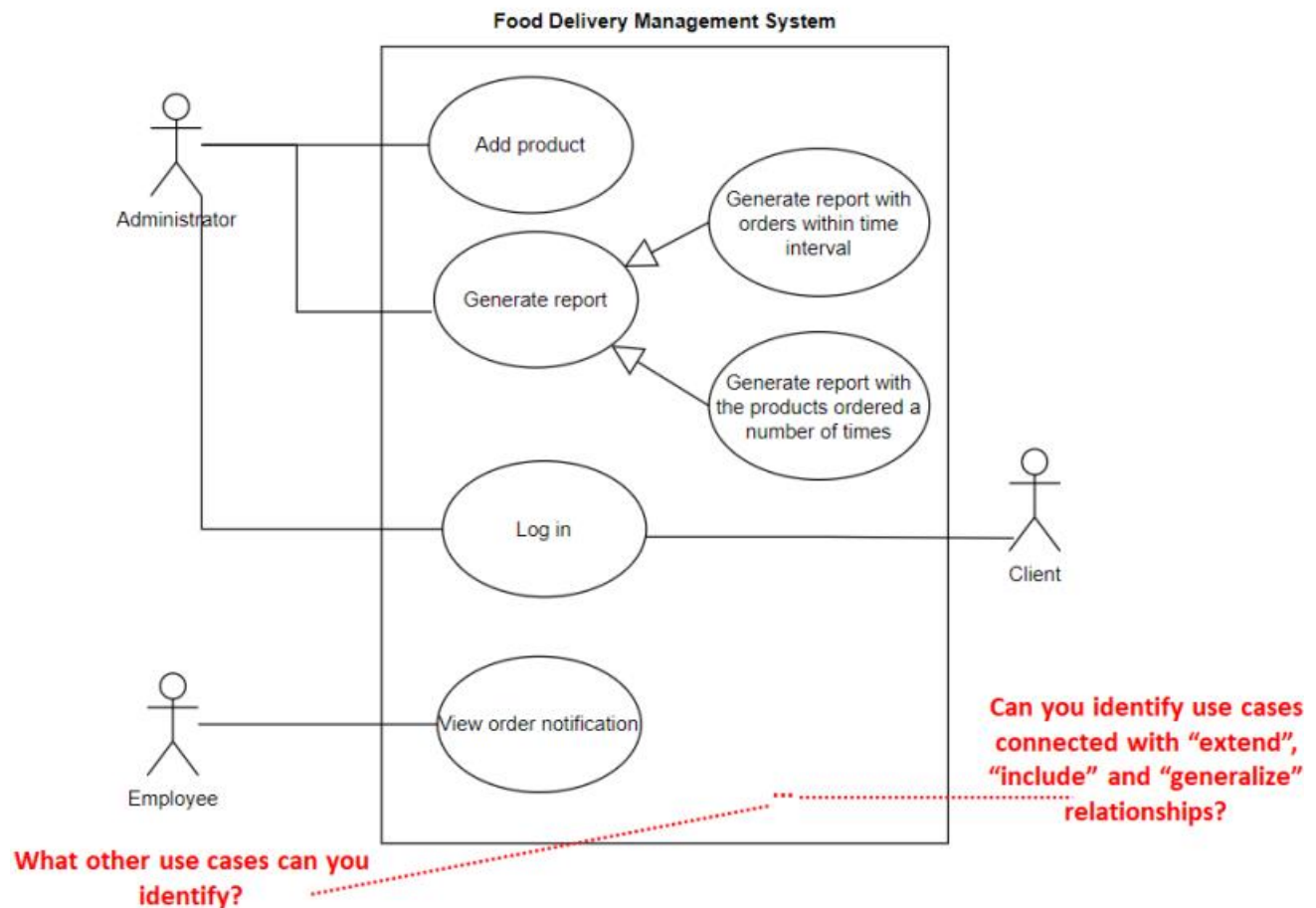
- 1) intervalul de timp al comenzilor – trebuie generat un raport cu comenzile efectuate între o anumită oră de început și o dată de sfârșit, indiferent de dată.
- 2) produsele comandate de mai mult de un anumit număr de ori până acum
- 3) clienții care au comandat mai mult de un anumit număr de ori până acum și valoarea comenzii a fost mai mare decât o sumă specificată.
- 4) produsele comandate într-o zi specificată cu numărul de ori pe care le au fost comandat.

Un client poate :

- 1) Să se autentifice cu un username și o parolă din fisierul users.csv

- 2) Sa vada lista de produse
- 3) Sa caute anumite produse dupa un keyword sau sa filtreze rezultatul in functie de rating, calorii, protein, grasimi, sodium sau pret.
- 4) Sa creeze o comandă formată din mai multe produse – pentru fiecare comandă va fi data și ora persistat și va fi generată o factură care va enumera produsele comandate și prețul totala ordinului.

Angajat : Angajatul este anunțat de fiecare dată când o nouă comandă este efectuată de către un client pentru a putea pregăti livrarea preparatelor comandate.

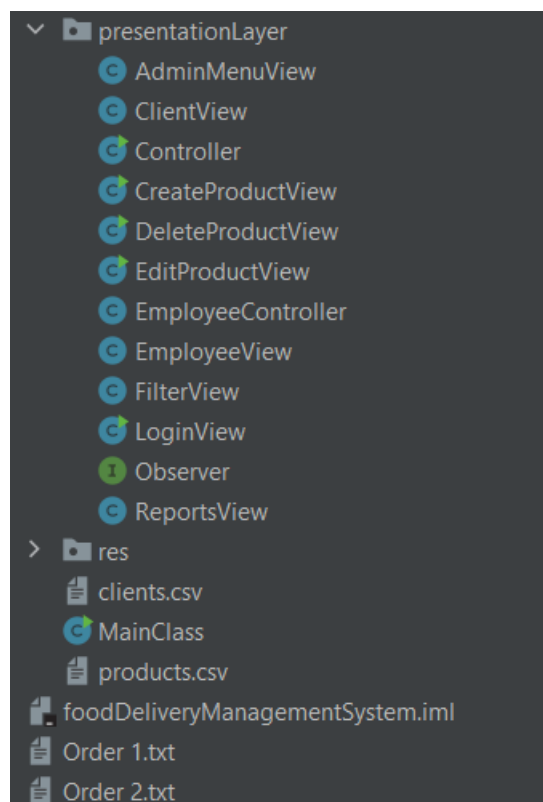
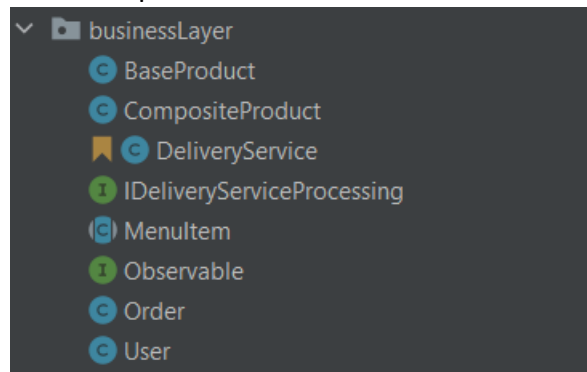


3. Proiectare (decizii de proiectare, diagrame UML, structure de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator).

Pentru proiectarea m-am ghidat dupa structura data in prezentare. Am proiectat programul in 3 pachete : bussinessLayer, dataLayer si presentationLayer. Pachetul

presentationLayer include clasele AdminMenuView , ClientView,Controller,CreateProductView,DeleteProductView, EditProductView, EmployeeView, EmployeeController, FilterView, LoginView,Observer si ReportsView care sunt responsabile cu interactiunea cu user-ul programului. In pachetul businessLayer avem clasele BaseProduct,CompositeProduct si MenuItem in care am folosit Composite Design Pattern.Am folosit de asemenea Observer Design Pattern care notifica angajatul de fiecare data cand primeste o noua comanda.

Structura proiectului :



Clasa *BaseProduct* reprezinta un produs simplu care extinde clasa abstracta MenuItem si continue un constror in care apelam cu super atributele clasei MenuItem.

Clasa *CompositeProduct* care de asemenea extinde clasa MenuItem in care avem metodele de computePriceList<MenuItem> menuItemList), public double

computeRating(List<MenuItem> menuItemList), public int
 computeCalories(List<MenuItem> menuItemList), public int
 computeProtein(List<MenuItem> menuItemList) care au rolul de aduna fiecare
 proprietate a unui baseProduct din meniu
 Clasa DeliveryService implementeaza interfata IDeliveryServiceProcessing si Observable
 in care avem un Map<Order,List<MenuItem>> hashMap, ArrayList<MenuItem>
 menuItemList care reprezinta lista de iteme din meniu, o lista de useri, List<User> si o lista
 de observer List<Observer>.. Aici avem metoda de a filtra un produs : public
 List<MenuItem> searchProductBy(String keyword, double minRating, double maxRating,
 int minPrice, int maxPrice, int minCalories, int maxCalories, int minProtein, int
 maxProtein, int minFat, int maxFat, int minSodium, int maxSodium) in functie de ce
 doreste clientul, get pt useri , public boolean validProduct(String name) care verifica ca
 un produs sa nu fie deja in meniu, metoda public MenuItem searchByName(String
 name) care cauta dupa nume in menuItemList un produs, metoda pentru delete a unui
 produs, pentru editarea acestuia. De asemenea aceste metode se regasesc si pentru
 produsele compuse, adica meniul. Aici avem metoda de a citi din CSV produsele :
 public List<MenuItem> readProductsFromCSV(String fileName) {
 String separator = ",";

```

    ArrayList<MenuItem> menuItemList1 = new ArrayList<>();

    Stream<String> lines = null;

    try {

        lines = Files.lines(Paths.get(fileName));

    } catch (IOException e) {

        e.printStackTrace();

    }

    assert lines != null;

    List<List<String>> items = lines.map(line ->
Arrays.asList(line.split(separator))).collect(Collectors.toList());

    items.remove(0); //remove la numele coloanelor

    items.forEach(value -> {

        MenuItem item = new BaseProduct(value.get(0),
(Double.parseDouble(value.get(1))), (Integer.parseInt(value.get(2))),
(Integer.parseInt(value.get(3))), (Integer.parseInt(value.get(4))),
(Integer.parseInt(value.get(5))), (Integer.parseInt(value.get(6))));

        try {

            menuItemList1.add(item);

        } catch (Exception exception) {

```

```

        exception.printStackTrace();
    }
});

menuItems = menuItems1;

return menuItems1;
}

```

Tot in clasa *DeliveryService* regasim metoda pentru a citi dintr-un fisier csv lista de user cu username,id,parola si numarul care reprezinta ce tip de user e (client,admin sau worker),metoda pentru plasare a unei comenzi,genereare a notei de plata si a celor 4 rapoarte din cerinta .Metoda `public JTable createTable(ArrayList<MenuItem> list)` primeste o lista de produse si creeaza un tabel de tipul *JTable* .Aici am suprascris metodele din *Observer*, `public void addObserver(Observer o)` si `public void notifyObservers(Order order)` ,inclusive get si set pentru lista de iteme din meniu

Interfata *IDeliveryServiceProcessing*, in care avem antetul tuturor functiilor pe care le vom suprascrie in alte clase,metodele enumerate mai sus.

Clasa *MenuItem*, care este o clasa abstracta ce continue attributele :

```

private String name;

private double rating;

private int calories;

private int protein;

private int fat;

private int sodium;

private int price;

```

De asemenea,continue si antetul functiilor compute ce sunt suprascrise in clasa *CompositeProduct*.Aici mai avem si get si set pentru fiecare atribut al clasei.

Constuctorul clasei :

```

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

    this.name = name;

    this.rating = rating;

    this.calories = calories;

    this.protein = protein;

```



```

        this.fat = fat;

        this.sodium = sodium;

        this.price = price;
    }

```

Interfata *Observable* care contine antetul metodelor de notifyObserver si addObserver pe care le vom folosi pentru Observer Design Pattern.

Clasa *Order* are attributele idOrder, idClient si orderDate .

Constructorul clasei Order :

```

public Order(int orderID, int clientID) {
    this.idOrder = orderID;
    this.clientId = clientID;
    this.orderDate = LocalDateTime.now();
}

```

Clasa User reprezinta un user universal care are id,username,parola si tip (0 pentru admin, 1 pentru angajat, 2 pentru client) ,constructorul clasei si get si set pentru attribute.De asemenea avem si metoda de toString() pentru a afisa in consola attributele .

In presentationLayer avem view-urile.Cele mai importante clase sunt :

Controller in care facem logica din spatele fiecarui buton si legam view urile de backend

.Aici avem de asemenea metoda Main :

```

public static void main(String[] args) {
    LoginView loginView = new LoginView();
    AdminMenuView adminMenuView = new AdminMenuView();
    ClientView clientView = new ClientView();
    EmployeeView employeeView = new EmployeeView();
    DeliveryService deliveryService = new DeliveryService();
    JTable productsTable = new JTable();
    CreateProductView createProductView = new CreateProductView();
    EditProductView editProductView = new EditProductView();
    DeleteProductView deleteProductView = new DeleteProductView();

    Controller controller = new Controller(loginView, adminMenuView,
    clientView, employeeView, createProductView, editProductView, deleteProductView,
    deliveryService, productsTable);
    EmployeeController employeeController = new
    EmployeeController(employeeView, deliveryService);
}

```

Interfata Observer care contine antetul metodei de update.

Metoda de update():

```
public void update(Order order) {  
    DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd/MM/yyyy  
HH:mm:ss");  
    String s = "";  
  
    for(Order o : deliveryService.getMap().keySet()){  
        s = "Order id : " + o.getOrderld() + "Client id : " + o.getClientld() +  
"Order date : " + dtf.format(order.getOrderDate());  
    }  
    employeeView.setTextArea(s);  
  
}
```

EmployeeController in care folosim Observer Design Pattern sa notificam
worker-ul cu comanda plasata :


Diagrama UML:


Interfata grafica:

Interfata pentru log in:


Technical University of Cluj-Napoca

X


food delivery



Password



LOGIN

Interfata pentru admin :

<-

Manage the products.

ADD PRODUCT

DELETE PRODUCT

EDIT PRODUCT

CREATE MENU

Menu Name

VIEW ALL

Products Meniu

X

GENERATE REPORTS

IMPORT CSV

Structuri de date folosite

Ca structuri de date s-au folosit *List* și *ArrayList* pentru a defini liste de produse, cantități, comenzi. S-au folosit *Set* și *HashSet* pentru reținerea de produse și utilizatori. S-a folosit *HashMap* pentru reținerea de comenzi (chei) și lista de produse comandate specifică fiecărei comenzi.

S-au folosit *Stream*-uri pentru extragerea datelor din fișierele .csv, pentru filtrarea produselor de către client și pentru extragerea datelor ce se vor reda în rapoartele generate de administratori.

Implementare:

4.1. Clasele pachetului *business*

Acesta este pachetul în care s-a definit și interfața *IDeliveryService*, explicată anterior, în secțiunea 3.3. Interfețe definite

4.1.1. *DeliveryService*:

Clasa folosită la realizarea anumitor operații caracteristice pentru aplicația de catering.

4.1.2. *Order*

Clasa pentru modelarea java a unei comenzi.

4.2. Clasele pachetului *data*

4.2.1. *FileWriter*

Clasa pentru importarea de produse și utilizatori.

4.2.2. *SerializatorMenuItem*

Clasa pentru serializarea și deserializarea produselor din meniu.

4.2.3. *SerializatorOrder*

Clasa pentru serializarea și deserializarea comenzilor.

4.2.3. *SerializatorUser*

Clasa pentru serializarea și deserializarea utilizatorilor.

4.3. Clasele pachetului *model.products*

4.3.1. *BaseProduct*

Clasa pentru modelarea java a unui produs de bază.

4.3.2. *CompositeProduct*

Clasa pentru modelarea java a unui produs compus.

MenuItem

Clasa pentru modelarea java a unei intrari in meniu.

.Clasele pachetului *model.users*

.*User*

Clasa pentru modelarea java a unui utilizator.

Clasele pachetului *presentation.admin*

AddProductsView

Clasa pentru crearea interfetei grafice de adaugare a unui nou produs in baza de date.

AdministratorView

Clasa pentru crearea ferestrei principale a unui admin

DeleteProductsView

Clasa pentru crearea interfetei grafice de stergere produse

EditProductsView

Clasa pentru crearea interfetei grafice de editare produse

EditProductsTableView

Clasa pentru crearea modelului abstract de tabel al JTable-ului din interfata grafica de editare produse

GenerateReportsView

Clasa pentru crearea interfetei grafice de selectare a unui tip de raport care sa fie generat

Clasele pachetului *presentation*

Clasele pachetului *presentation*

ClientView

Clasa pentru crearea ferestrei principale a unui client

FilterView

Clasa pentru crearea interfetei grafice de filtrare a produselor

OrderProductsTableModel

Clasa pentru crearea modelului abstract de tabel al JTable-ului de produse din interfata grafica pentru client

Clasele pachetului *presentation.controller*

Controller

Clasa pentru realizarea conexiunii dintre Model si View

Clasele pachetului *presentation.employee*
EmployeeView
Clasa pentru crearea ferestrei unui angajat
Clasele pachetului *presentation*
LoginView

1. Clasa pentru crearea interfetei grafice de adaugare a unui nou client

2. Rezultate.

După finalizarea proiectului, au fost efectuate numeroase teste pentru a verifica corectitudinea și consistența aplicației Java. indică numărul de comenzi plasate în baza de date curentă, având în vedere, în acest caz, comanda cu câte produse.

3. Concluzii:

Tema de față a prezentat elemente de noutate în primul rând prin folosirea de streamuri și definirea de invarianți, precondiții și postcondiții, dar și prin structurile de date folosite (HashSet și HashMap), prin extinderea Observable și implementarea Observer și prin serializare și deserializare.

Elementele de grafică au fost, de asemenea, o provocare în realizarea proiectului, în special prin realizarea actualizării instanțe a tabelelor la intervenția unei modificări, ori la filtrarea produselor.

4. Bibliografie:

- https://dsrl.eu/courses/pt/materials/PT2021-2022_Assignment_4.pdf
- https://dsrl.eu/courses/pt/materials/A4_Support_Presentation.pdf

Lambda expressions and stream processing:

- <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>
- <https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html>
- <https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html>

Java serialization :

- https://www.tutorialspoint.com/java/java_serialization.htm
- <https://www.baeldung.com/java-serialization>