

**Food delivery management system**

Student: Zilai Denis

Grupa: 30238

Contents

**No table of contents entries found.**

# Obiectivul temei

Scopul principal al acestei teme este proiectarea si implementarea unui sistem de management al livrarii produselor pentru o companie de catering. Clientii care vor folosi aceasta aplicatie vor avea posibilitatea de a cauta anumite produse in functie de mai multe criterii si de a le comanda din meniul companiei. Sistemul ar trebui sa aiba 3 tipuri de utilizatori care se vor conecta folosind un nume de utilizator si o parola: administrator, angajat obisnuit si client. Administratorul va gestiona produsele din meniu si va genera un set de rapoarte specifice,iar angajatii vor fi notificati de fiecare data cand se va efectua o noua comanda de catre un client, astfel incat acestia vor putea pregati livrarea produselor comandate. Aplicatia va trebui sa implementeze o interfata grafica asociata care sa includa 4 ferestre:

* O fereastra de autentificare a utilizatorului
* O fereastra pentru operatiunile administratorului
* O fereastra pentru operatiunile clientilor
* O fereastra pentru procesarea comenzilor de catre angajati

Pentru a putea indeplini cu succes acest obiectiv va trebui realizam urmatoarele etape:

1. **Analiza problemei** (detaliata in [capitolul 2](file:///C:\Users\DZ\Desktop\PT2021_30228_Scarisoreanu_Cristian_Assignment_4\Documentatie.docx#_Analiza_problemei)), unde vom obtine cazurile de utilizare (use-cases) si cerintele functionale ale aplicatiei.
2. **Proiectarea aplicatiei** (detaliata in [capitolul 3](file:///C:\Users\DZ\Desktop\PT2021_30228_Scarisoreanu_Cristian_Assignment_4\Documentatie.docx#_Proiectare)), unde vom prezenta proiectarea orientata pe obiecte a aplicatiei, diagramele UML de clase si pachete ce definesc aceasta proiectare si structurile de date folosite.
3. **Implementarea aplicatiei** (detaliata in [capitolul 4](file:///C:\Users\DZ\Desktop\PT2021_30228_Scarisoreanu_Cristian_Assignment_4\Documentatie.docx#_Implementare)), unde vom descrie fiecare clasa utilizata cu campuri si metodele importante, modul de implementare al interfetei utilizator si functionarea aplicatiei.
4. **Testarea aplicatiei** (detaliata in [capitolul 5](file:///C:\Users\DZ\Desktop\PT2021_30228_Scarisoreanu_Cristian_Assignment_4\Documentatie.docx#_Rezultate)), unde vom verifica corectitudinea functionarii aplicatiei

# Analiza problemei

Conform cerintei problemei, in functie de rolul utilizatorului si operatiunile pe care acesta le poate face, putem deduce **cerintele functionale** ale aplicatiei. Prin urmare, aplicatia firmei de catering trebuie sa:

* Sa permita tuturor utilizatorilor sa se inregistreze si sa utilizeze numele de utilizator si parola inregistrate pentru a se conecta in sistem
* Sa permita administratorului sa:
  + importe setul initial de produse care vor completa meniul dintr-un fisier .csv
  + gestioneze produsele din meniu: sa adauge , sa stearga sau sa modifice produse si sa creeze produse noi, compuse din mai multe produse de baza (un exemplu de produs compus ar putea fi denumit „meniul zilei 1”, care sa fie compus dintr-o supa, o friptura , o garnitura si un desert)
  + genereze rapoarte despre comenzile efectuate pe baza mai multor criterii:
    - intervalul de timp al comenzilor – ar trebui generat un raport cu comenzile efectuate intr-un anumit interval de timp
    - produsele comandate de mai mult de un numar specificat de ori
    - clientii care au comandat mai mult de un numar specificat de ori si ale caror comenzi au fost mai scumpe decat o valoare/suma specificata
    - produsele comandate intr-o zi specificata, impreuna cu numarul de comenzi plasate pentru aceste produse
* Sa permita clientului sa:
  + Caute produse pe baza unuia sau mai multor criterii, cum ar fi un cuvant cheie (de ex. „Supa”), evaluarea produsului, numarul de calorii , cantitatea de proteine /grasimi/sodium, sau pretul produsului.
  + Creeze o comanda formata din mai multe produse (de baza sau compuse)
* Sa genereze o factura pentru fiecare comanda plasata. Factura va trebui sa contina data,ora,lista produselor comandate si pretul total.
* Sa informeze angajatul de fiecare data cand este creata o comanda noua
* Sa foloseasca serializarea pentru a persista produsele din meniu, comenzile efectuate si informatiile utilizatorilor, astfel incat acestea sa fie disponibile la viitoarele executii ale sistemului (prin deserializare)
* Sa foloseasca expresii lambda si „stream processing” pentru generarea rapoartelor specifice adminstratorului si pentru implementarea functionalitatilor de cautare disponibile pentru client

Daca vrem sa ne referim la cerintele **non-functionale**, una dintre cele mai importante ar fi proprietatea aplicatiei de a fi usor de utilizat. De aici intelegem ca aplicatia trebuie sa aiba o interfata grafica simpla, astfel incat operatiunile specifice fiecarui tip de utilizator sa se poata realiza cu usurinta.

In continuare este prezentata diagrama **cazurilor de utilizare** ale aplicatiei:

Diagram

Description automatically generated

Dupa cum se poate observa, exista mai multe cazuri de utilizare ale aplicatiei, in functie de rolul utilizatorului:

1. Administratorul:
   1. Importa produsele de baza din fisierul .csv.
   2. Gestioneaza produsele prin adaugarea, stergerea sau modificarea produselor de baza si crearea de produse compuse.
   3. Genereaza rapoartele specificate in cerintele functionale.
2. Clientul:
   1. Vizualizeaza lista de produse de baza / compuse din meniu.
   2. Cauta produse in functie de criteriile amintite in cerintele functionale.
   3. Plaseaza o comanda cu produsele selectate, in urma caruia se genereaza o factura.
3. Angajatul
   1. Este notificat de fiecare data cand un client plaseaza o comanda noua.

Un scenariu in care procesul de plasare si livrare a unei comenzi se **realizeaza cu succes** implica respectarea de catre client a urmatorilor pasi:

1. Acesta se inregistreaza prin alegerea unui nume de utilizator si a unei parole.
2. Se autentifica cu credentialele alese si intra in aplicatie
3. Vizualizeza intreaga lista de produse din meniul companiei
4. Cauta produse specifice prin completarea campurilor destinate criteriilor de cautare (unul sau mai multe)
5. Acesta obtine produsele cautate
6. Selecteaza produsele dorite si apasa pe butonul **ORDER**
7. Comanda este salvata si factura acesteia se genereaza automat intr-un fisier .txt
8. Angajatul este notificat cu privire la noua comanda efectuata si pregateste livrarea produselor

Clientul nu este obligat sa realizeze o cautare specifica a produselor.Acesta poate selecta produsele dorite direct din lista cu toate produsele din meniu.

**Erorile** care pot aparea in scenariul de mai sus sunt urmatoarele:

* Clientul incearca sa inregistre un nume de utilizator deja existent, caz in care este atentionat si obligat sa reintroduca un nume unic.
* Acesta nu alege niciun produs din meniu, caz in care comanda este invalida, iar pe ecran se afiseza un mesaj de eroare

# Proiectare

Pentru a realiza proiectarea POO a aplicatiei folosind un model arhitectural stratificat (Layered Arhitecture), avem nevoie de subpachete urmatoare:

1. **presentation –** cuprinde clasele care implementeaza interfetele grafice
2. **bll (business logic layer) –** cuprinde clasele care implementeaza functionalitatea aplicatiei
3. **data –** grupeaza clasele care modeleaza datele aplicatiei
4. **start –** contine clasa **App** care porneste aplicatia

Diagram

Description automatically generated

Pachetul **presentation** contine 4 ferestre:

* **LoginPage –** fereastra de autentificare/inregistrare
* **AdminPage –** fereastra pentru operatiunile administratorului
* **ClientPage –** fereastra pentru operatiunile clientului
* **EmployeePage –** fereastra de notificare a angajatului

Pe langa aceste ferestre, aici intalnim si clasa **Controller** care sta la baza functionarii acestei aplicatii.

In pachetul **bll** gasim urmatoarele clase, care vor fi descrie in amanunt in capitolul urmator, fiind cele mai importante clase ale aplicatiei.

* + - * + **BaseProduct**
        + **CompositeProduct**
        + **DeliveryService**
        + **IDeliveryServiceProcessing** (interfata)
        + **MenuItem** (clasa abstracta)
        + **Order**

Pachetul **data** grupeaza clasele:

* **Authenticator**
* **FileWriter**
* **Serializator.**

Relatiile dintre clasele prezentate sunt ilustrate **in diagrama UML** de clase:

A screenshot of a computer screen

Description automatically generated with medium confidence

# Implementare

## MenuItem

Pentru a defini produsele din meniu a fost aplicat **Composite Design Pattern,** astfel clasa **MenuItem** este o clasa abstracta ce descrie comportamentul comun al tuturor elementelor din compozitie. Ca si metode regasite in aceasta clasa, avem metodele abstracte de determinare a atributelor unui produs ( de baza / compus):

public MenuItem() {}  
public abstract double computePrice();  
public abstract String getTitle();  
public abstract double getRating();  
public abstract int getCalories();  
public abstract int getProtein();  
public abstract int getFat();  
public abstract int getSodium();

## BaseProduct

Aceast clasa mosteneste clasa **MenuItem** si modeleaza un produs de baza, produs care are ca si atribute urmatoarele campuri:

private String title;

private double rating;

private int calories;

private int protein;

private int fat;

private int sodium;

private double price;

Metodele abstracte din clasa MenuItem au fost implementate aici, ca niste simple metode accesoare. De exemplu, metoda **getPrice()** returneaza pretul unui produs de baza.

## CompositeProduct

Clasa **CompositeProduct,** clasa ce mosteneste de asemenea clasa **MenuItem,** modeland produsele compuse , are ca si atribute numele/titlul produslui si pretul acestuia, precum si o lista cu obiecte de tipul MenuItem. Astfel, un produs poate fi compus atat din produse de baza, cat si din alte produse compuse. Pentru a descrie modul in care metodele sunt implementate vom lua metoda **computePrice():**

public double computePrice() {

price=0;

for(MenuItem i:products)

{

price=price+i.computePrice();

}

return price; }

Aceasta metoda obtine pretul unui produs compus, adunand fiecare pret al fiecarui produs (poate sa fie atat de baza, cat si compus) din care acesta este alcatuit. La fel se procedeaza si pentru obtinerea cantitatii totale de calorii/proteine/grasimi/sodium. In schimb, evaluarea pentru un produs compus se va calcula ca medie a tuturor evaluarilor produselor din care acesta este alcatuit.

## Order

In aceasta clasa se descriu caracteristicile unei comenzi. O comanda trebuie sa aiba un client, un numar de comanda, o data la care s-a efectuat comanda, precum si pretul acesteia.(atributele **clientID** <String-se retine numele de utilizator al clientului>, **orderId**<int>**, orderDate** <de tipul LocalDateTime>si price <de tipul double>).

Majoritatea metodelor folosite in aceasta clasa sunt metode accesoare si mutatoare. Metoda **hashCode()** , suprascrisa din clasa **Object** a fost implementata pentru a se putea face legatura dintre o comanda si produsele comandate ( vezi sectiunea 4.6 ).

## IDeliveryServiceProcessing (interfata)

**IDeliveryServiceProcessing** declara metodele abstracte corespunzatoare functionalitatilor la care au acces administratorul, respectiv clientul. Aceasta interfata a fost proiectata folosind tehnica „**Design by Contract”** : s-au definit formal **pre** si **post** conditii in comentarii corespunzatoare **JavaDoc,** urmand sa se verifice apoi in clasa **DeliveryService** prin instructiuni **assert.** Exemplu pentru metoda de adaugare a unui produs in meniu:

*/\*\**

*\* Adds a new product to the menu*

*\**

*\** ***@param*** *i the product to be added to the menu*

*\** ***@pre*** *i!=null && menu.indexOf(i)!=-1*

*\** ***@post*** *menu.size()==menu.size()@pre+1*

*\*/*

void addProduct(MenuItem i);

Pentru ca aceste tag-uri sa fie vizibile in documentatia **JavaDoc,** documentatia trebuie sa se genereze cu parametrii : -tag pre:a:"Pre conditions:" -tag post:a:"Post conditions:" . **a-** ne spune ca putem plasa tag-urile inaintea oricarui obiect Java (clasa, metoda, constructor), desi nu avem nevoie de ele decat la metode.

## DeliveryService

Aceasta clasa implementeaza interfata prezentata anterior. Pe langa metodele din interfata, aceasta prezinta si o structura de date care realizeaza legatura dintre o comanda si produsele comandate aferente respectivei comenzi.

private final Map<Order, List<MenuItem>> orderInfo;

Tot aici se pastreaza si o lista cu toate produsele ce repezinta meniul companiei.

private List<MenuItem> menu;

Deoarece s-a folosit tehnica „Design by Contract” pentru a se defini in interfata pre si post conditiile, in aceasta clasa s-au verificat aceste conditii prin utilizarea instructiunii assert. De exemplu, pentru metoda **addProduct()** s-au verificat conditiile prezentate anterior, in sectiunea 4.5.

public void addProduct(MenuItem i) {

assert i != null;

assert isWellFormed();

int size = menu.size();

menu.add(i);

menu = menu.stream().filter(*distinctByKey*(MenuItem::getTitle)).collect(Collectors.*toList*());

menu = menu.stream().sorted().collect(Collectors.*toList*());

assert size + 1 == menu.size();

assert isWellFormed();

}

Dupa cum se poate observa, pe langa conditiile pre si post, s-a testat si valoarea functiei isWellFormed(). Aceasta functie defineste invariantii clasei , mai precies verifica daca clasa este „well-formed”. Acest lucru impune ca meniul sa fie ordonat in ordine lexicografica si totodata, sa nu existe produse duplicate.

Pentru a defini formal invariantii, sa introdus un comentariu aferent documentatiei JavaDoc, inainte de declararea clasei.

*/\*\**

*\* Declares the abstract methods corresponding to the functionalities to which the administrator and the client have access.*

*\**

*\** ***@invariant*** *isWellFormed()*

*\*/*

public class DeliveryService extends Observable implements IDeliveryServiceProcessing, Serializable {

Metodele utilizate in aceasta clasa folosesc **expresii lambda** si **stream processing** pentru generarea raporturilor si pentru realizarea operatiilor de cautare specifice clientului.

## Serializator

Se ocupa de partea de serializare si deserializare pentru a persista produsele din meniu, comenzile efectuate si totodata pentru a persista informatiile despre utilizatori. Aici se regasesc 2 metode specifice pentru serializare, o metoda care serializeaza un obiect de clasa **DeliveryService** si o metoda care serializeaza un obiect de clasa **Authenticator**.Analog, pentru deserializare.

## Authenticator

Contine informatiile de autentificare ale administratorului, ale clientilor si ale angajatilor.S-a folosit o structura de tipul HashMap pentru fiecare rol, pentru a face posibila pastrarea legaturii nume de utilizator-parola. Se pot inregistre noi clienti / angajati, informatiile persistand si dupa inchiderea aplicatiei (Serializare).

## FileWriter

Contine metodele care furnizeaza rapoartele specifice administratorului si factura comenzilor. Toate acestea se genereaza in forma unui fisier „.txt”. Formatul este unul si acelasi pentru fiecare raport in parte. (avem 4 metode diferite pentru cele 4 rapoarte).

## GUI

Interfata grafica este formata din 4 ferestre:

1. Fereastra de autentificare / inregistreare:

Graphical user interface, application

Description automatically generated

1. Fereastra pentru operatiunile clientului

A screenshot of a computer

Description automatically generated

1. Graphical user interface, application

   Description automatically generatedFereastra pentru operatiunile administratorului
2. Fereastra de notificare / Monitorul angajatului – Notificarea angajatului se realizeaza prin utilizarea **Observer Design-Pattern**

Text

Description automatically generated

# Rezultate

In urma testarii aplicatiei pentru mai multe cazuri de utilizare, se poate observa ca aceasta isi indeplineste scopul final, functionand fara probleme.

# Concluzie

In urma realizarii acestui sistem de mangement al comenzilor de mancare pentru o firma de catering,am invatat sa folosesc mai multe tehnici de programare cum ar fi Observer Design Pattern, Composite Design Pattern si Design By Contract. Totodata am invatat sa lucrez cu procesare de stream-uri si cu expresii lambda. Consider ca acest sistem v-a fi foarte util intr-o implementare reala.

# Bibliografie

1. <https://stackoverflow.com/>
2. <https://www.baeldung.com/>
3. <https://docs.oracle.com/javase/7/docs/technotes/guides/language/assert.html>
4. <https://www.diagrams.net/>
5. <https://howtodoinjava.com/java8/java-stream-distinct-examples/>
6. <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>
7. <https://www.tutorialspoint.com/java/java_serialization.htm>
8. <https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag>
9. <https://intellij-support.jetbrains.com/hc/en-us/community/posts/207014815-How-to-enable-assert>
10. <https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/>