

**FOOD DELIVERY MANAGEMENT SYSTEM**

Contents

[1. Obiectivul temei 2](#_Toc72970078)

[2. Analiza problemei, modelare, șcenarii, cazuri de utilizare 2](#_Toc72970079)

[3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator) 3](#_Toc72970080)

[4. Implementare 5](#_Toc72970081)

[a. Data Layer 5](#_Toc72970082)

[i. Clasa FileWriter 5](#_Toc72970083)

[ii. Clasa Serializațion 5](#_Toc72970084)

[b. Business Layer 6](#_Toc72970085)

[i. Clasa MenuItem 6](#_Toc72970086)

[ii. Clasa BaseProducts 6](#_Toc72970087)

[iii. Clasa CompositeProduct 6](#_Toc72970088)

[iv. Clasa IdPaswoord 6](#_Toc72970089)

[v. Clasa Users 6](#_Toc72970090)

[vi. Clasa Order 6](#_Toc72970091)

[vii. Clasa abstractă Observable 7](#_Toc72970092)

[viii. Interfața IDeliveryService 7](#_Toc72970093)

[ix. Clasa DeliveryService 8](#_Toc72970094)

[c. Presentation Layer 10](#_Toc72970095)

[i. Clasa Administrator 10](#_Toc72970096)

[ii. Clasa Client 10](#_Toc72970097)

[iii. Clasa Employee 10](#_Toc72970098)

[iv. Clasa LogIn 10](#_Toc72970099)

[v. Clasa NewClient 10](#_Toc72970100)

[vi. Interfața Observer 10](#_Toc72970101)

[d. Pachetul Model 11](#_Toc72970102)

[i. Clasa Controller 11](#_Toc72970103)

[5. Rezultate 11](#_Toc72970104)

[6. Concluzii 12](#_Toc72970105)

[7. Bibliografie 12](#_Toc72970106)

# Obiectivul temei

În cadrul acestei teme, principalul obiectiv a fost crearea unui sistem de managemant al livrărilor produselor alimentare pentru o companie de catering, astfel încât clientul poate comanda produse din meniul oferit. Astfel se pot enumera și anumite obiective secundare precum:

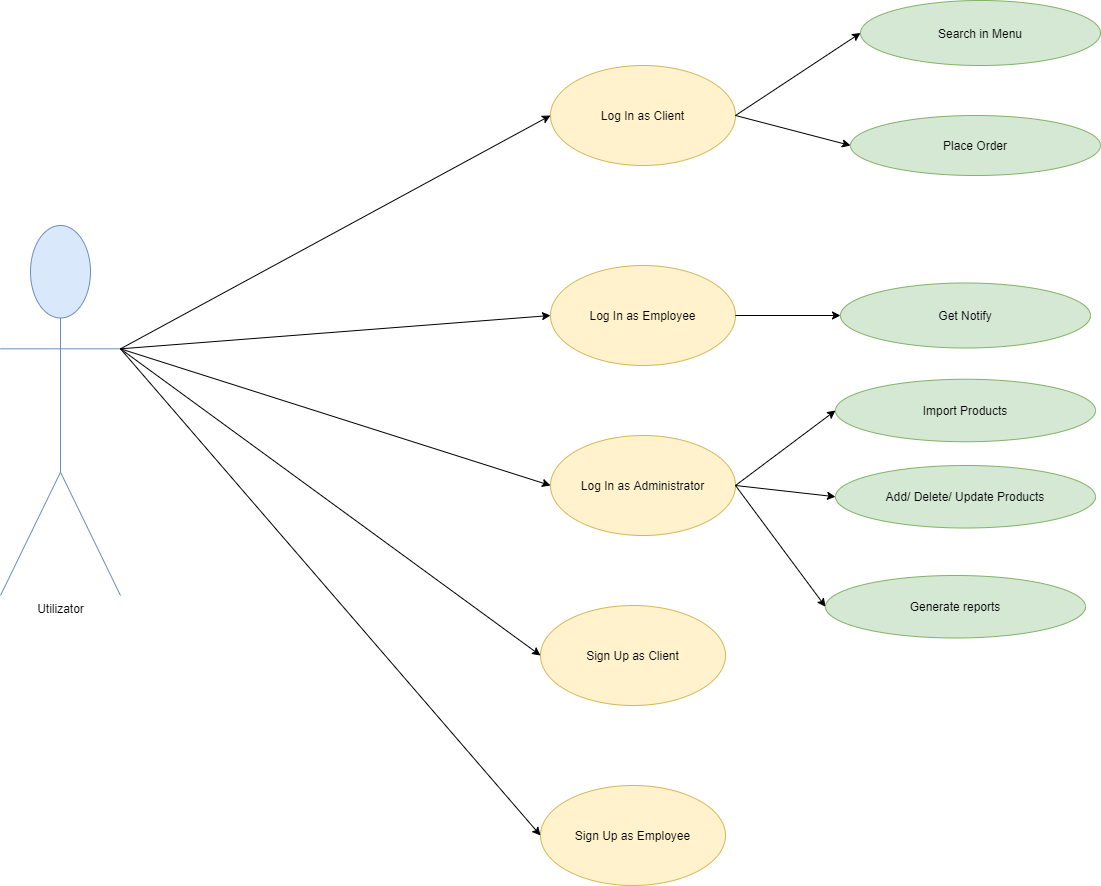
* Realizarea unei interfețe grafice dedicată fiecărui tip de utilizator
* Plasarea unor comenzi de către client
* Administrarea meniului de către administrator
* Notificarea angajaților atunci când eset plasată o comandă

# Analiza problemei, modelare, șcenarii, cazuri de utilizare

Având în vedere complexitatea sistemului, este necesară realizarea unei separarea utilizatorilor în trei categorii distinct:

* Utilizatorii de tip client care vor putea căuta produse pe baza unuia sau mai multe criterii și efectua comenzi de pentru produsele alese din meniul oferit, iar în cazul unui nou client va exista posibilitatea creări unui nou cont specific acestui tip de utilizator
* Utilizatorul de tip administrator poate importa produsele în meniu, efectua operații asupra produselor (de exemplu: ștergerea unui produs, crearea unui produs, modificarea unui produs, compunerea de noi meniuri) și generarea unor rapoarte referitoare la comenzile efectuate, rapoarte bazate pe mai multe criterii;
* Utilizatorii de tip angajat vor fi notificați la fiecare plasare a unei comenzi de către un client pentru a pregăti comanda;

Acest proiect are rolul de a simula functionarea unei firme de catering, pe baza cerințelor funcționale enumerate mai sus. Având în vedere complexitatea sistemului este necesară o modelare obiectuală, astef utilizatorii vor fi categorizați după permisiunile de acces și obiectivul accesului. Noul client va putea accesa optiunea creării unui nou cont pe baza unui nume de utilizator unic si a unei parole, atunci când acest cont există sa va putea loga pe baza credentialelor pentru a plasa o comandă. Clientul v-a putea alege pe baza unor criterii (de exempluȘ tipul produsului, pretul produsului, numărul de calorii pe care îl conține produsul etc) unul sau mai multe produse din meniul furnizat și va plasa comanda, în urma acestei operații se va furniza o chitanță une se va preciza numărul comenzii, produsele comandate și prețul. Administratorul se va putea loga pe baza credențialelor specifice și va putea importa produse, crea noi produse (produse de bază sau chiar produse complexe de tipul unor meniuri), șterge produse din meniu sau modifica produse. Pe lângă aceste operațiuni, administratorul are capacitatea de a genera rapoarte privind comenzile efectuate de către clienți. Noul angajat are opțiunea de a crea un nou cont și de a se loga în sistem pentru a fi notificat la fiecare plasare a unei noi comenzi.

*Diagrama use-case a sistemului:*

# Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

Proiectarea sistemului a fost realizată după modelul **layered** **architecture** **pattern**. Componentele din această arhitectură sunt organizate în layere orizontale, fiecare layer indeplinind un rol specific în cadrul aplicației: presentation layer (are rolul de a manipula interfața utilizatorului), business layer (face legătura între presentațion layer si logic layer, în acest pachet se vor implementa regulile de funcționare ale aplicației), logic layer (va realiza operațiile tehnice ale aplicației precum scrierea în fișiere).

Pentru a categorisi fiecare produs (produs de baza sau produs compus) am utilizat tehnica Composite Design Pattern. Această tehnică de programare descrie un grup de obiecte care este tratat cao singură instanță a aceluiași tip de obiect. Astfel, scopul este de a compune obiecte în structuri de arbori.

Deoarece se întâlnesc trei tipuri de utilizatori diferițicare vor avea permisiuni diferite de acces în aplicație a fost necesară stocarea informațiilor de tip user si parolă într-un HashMap.

Am ulilizat modelul Observer pentru a realiza relaționarea între client și angajat, atunci când este plasată o comandă. Acest model definește o dependență unu-la-mai-mulți între obiecte , astfel încât atunci când un obiect își modifică starea, toate dependențele sale sunt notificate și actualizate automat.

Pentru ca aplicația să funcționeze cât mai optim este necesasă salvarea în conturilor noub create pentru fiecare tip de utilizator și starea actuală a serviciului, pentru a nu fi pierdută informția furnizată de către utilizatorul de tip administrator (produsele noi adăugate în meniu). Această operație a fost realizată folosind serializarea. Serializarea reprezintă un mecanism de scriere a unui obiect într-un byte-stream. Operațiunea inversă de serializare se numește deserializare și presupune convertirea unui byte-stream într-un obiect. Procesul de serializare și deserializare este independent de platformă.

Stocarea informațiilor referitoare la comenzi (comanda și produsele comandate) au fost stcate într-un HashMap.

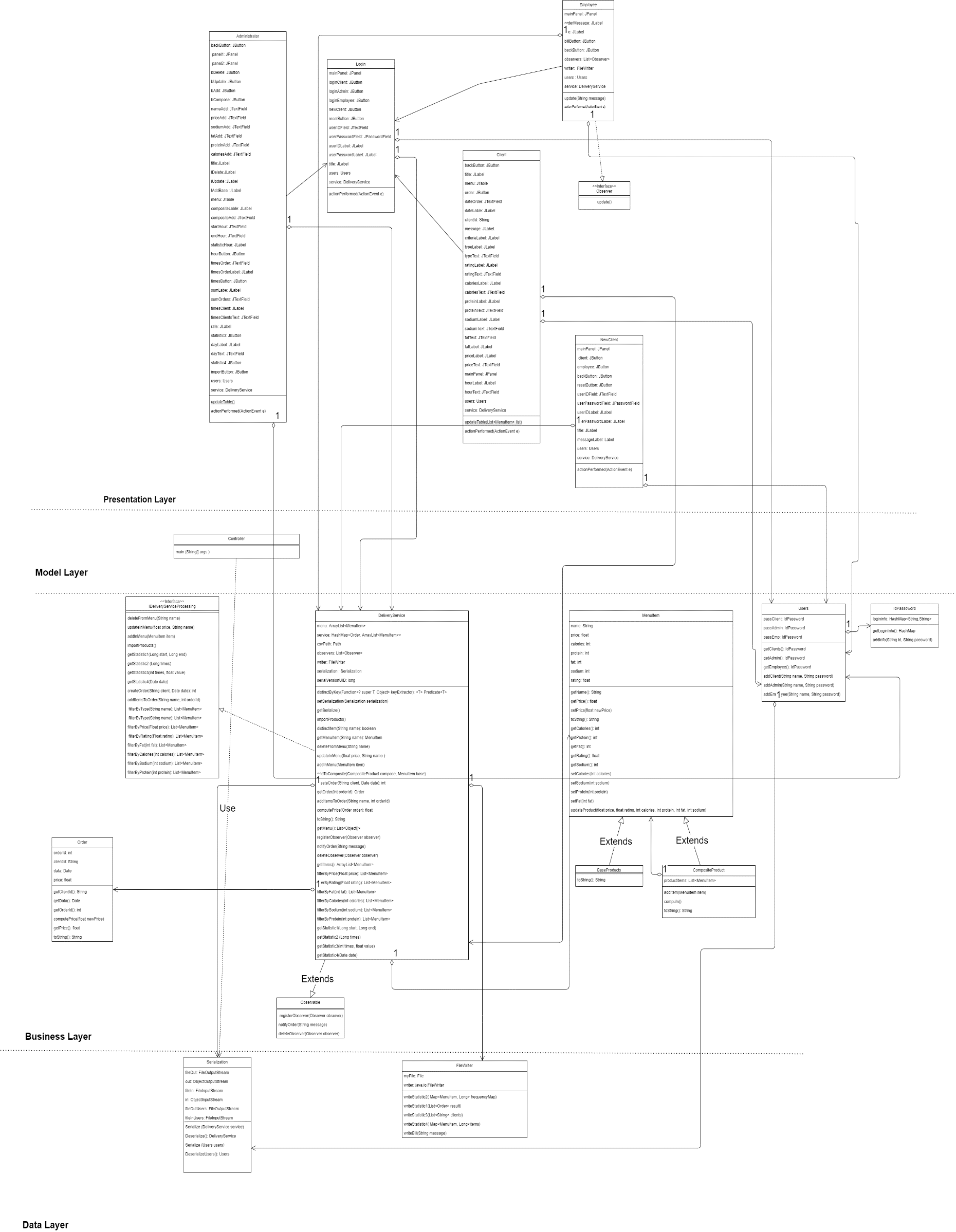
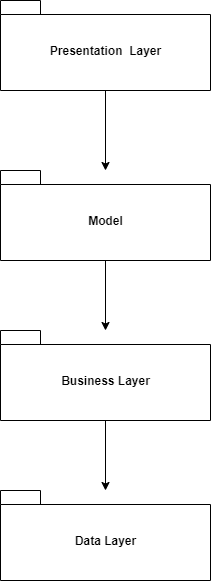


Diagrama de clase:

Diagrama de pachete:

# Implementare

## Data Layer

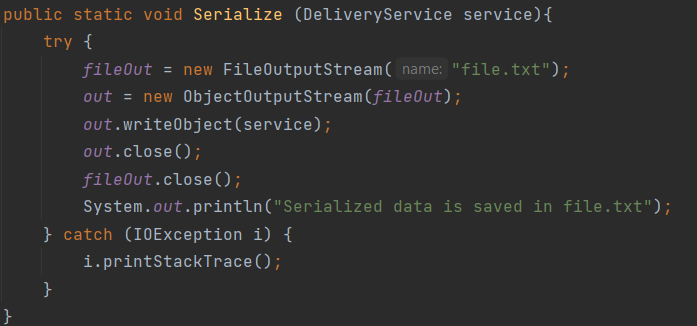
Acest pachet conține clasele care vor realiza operații tehnice ale programului precum scrierea și citirea din fișiere.

### Clasa FileWriter

În această clasă se va realiza scrierea rapoartelor în fișiere, dar și pentru generarea chitanței pentru client. Această clasă conține câte o metodă dedicată fiecărui tip de raport ce va urma a fi scris și o metodă de scriere a chitanței.

### Clasa Serializațion

În această clasă se va realiza serializarea și deserializare obiectelor de tipu DeliveryService și User. Metodele de serializarevor primi ca parametru câte un obiect de tipul DeliveryService, respectiv User și vor scrie într-un fișier de tipul .txt datele serializate. Cu ajutorul unui obiect de tipul  ObjectOutputStream se va realiza convertire obiectului într-un byte-stream prin apelarea metodei writeObject(Object object), această metodă convertește obiectul primit ca parametru și îl va scrie în fișierul stocat în variabila de tipul FileOutputStream.



Deserializarea obiectelor se realizează asemănător cu Serializare, doar că în acest caz metoda va returna un obiect de tipul clasei deserializate din fisier. Utilizând un obiect de tipul ObjectInputStream se va apela metoda readObject() care va regăsi primul obiect din stream, îl vadeserializa și va returna un obiect de tipul Object, deci este necesară efectuarea unui cast asupra obiectului returnat pentru a-l converti la clasa dorită.



## Business Layer

În acest pachet este implementată funcționarea logică a aplicației pe baza regulilor de funcționare. Sunt create clasele pentru obiectele care formează meniul (MenuItem, BaseProducts și CompositeProduct), se implementează funcționarea sistemului de managemant (în cadrul clasei DeliveryService), sunt create clasele destinate stocării datelor pentru utilizatori și sunt definite clasele pentru administrarea obiectelor de tip Order (Order).

### Clasa MenuItem

În această clasă sunt salvate datele despre produsele din meniu, numele produsului, prețul produsului, numărul de calorii, numărul de proteine, numărul de grăsimi, cantitatea de sodiu și ratingul produsului. Printre metodele clasei se regăsesc metode care returnează fiecare atribut al produsului și metode care vor schimba valorile atributelor, înlocuindu-le cu numărul primit ca parametru. Metoda updateProduct va updata produsul cu datele furnizate.

### Clasa BaseProducts

Această clasă moștenește clasa MenuItem și este utilizată pentru stocarea informațiilor despre obiectele de tipul produselor de bază (ex: supă).

### Clasa CompositeProduct

Această clasă moștenește la rândul său clasa MenuItem și este utilizată pentru stocarea informațiilor despre obiectele compuse (ex: meniul zilei). Dat fiind faptul că un obiect compus este format din mai multe produse, acestea vor fi stocate într-o structură de tipul ArrayList<MenuItems>. Această clasă implementează metoda prin care se adaugă un nou produs în lista de produse conținute (addItem(MenuItem item)) si metoda prin care pentru fiecare produs compus se va realiza compunerea datelor caracteristice fiecărui produs din meniu prin însumarea caracteristicilor produselor ce îl compun.

### Clasa IdPaswoord

În cadrul acestei clase se vor salva perechile de userId și parolă într-un HashMap pentru fiecare tip de utilizator. Metodele conținute sunt: metoda prin care se va adăuga o nouă pereche (atunci când va fi creat un nou cont) și metoda prin care se vor furniza toate perechile de acest tip.

### Clasa Users

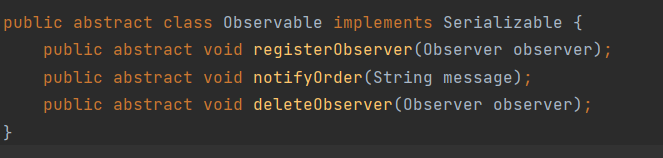
Această clasă are rolul de a implementa operațiile efectuate de către utilizatori la intrarea în aplicație (crearea unui cont nou). În această clasă se vor instanția trei atribute de clasă corespunzătoare fiecărui tip de utilizator de tipul IdPassword. Metodele din această clasă sunt: metode prin care se vor returna credențialele utilizatorilor, metode prin care sunt adăugați noi utilizatori. Pentru a serializa informația din această clasă este necesară crearea unui atribut de clasă de tipul Serialization care va apela metoda Serialize(Users user) prin metoda etSerialize(). Clasa Users implementează interfața Serializable pentru posibilitatea de serializare și deserializare a datelor.

### Clasa Order

În această clasă sunt implementate obiectele de tipul comenzilor, fiecare comandă având specific un număr (orderId), numele de utilizator al clientului care a efectuat comanda (clientId), data pentru care a fost efectuată comanda (data) și prețul comenzii (price). Se regăsesc metode prin care sunt furnizate date referitor la comandă precum data (getData()), numărul comenzii (getOrderId()), numele de utilizator al clientului care a plasat comanda (getClientId()) și prețul comenzii (getPrice()). Metoda computePrice primește ca parametru un număr de tip float care specifică prețul produsului adăugat în comandă, preț care va fi adăugat la prețul comenzii.

### Clasa abstractă Observable

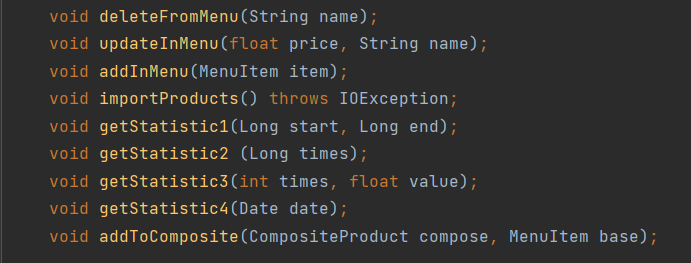
Această clasă definește metodele abstracte care vor fi utilizate pentru comunicarea între angajat și client.



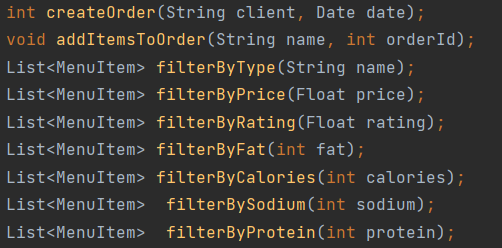
### Interfața IDeliveryService

În această interfață definește principalele operațiuni care vor putea fi efectuate de către administrator și client.

Operațiunile care vor putea fi utilizate de administrator reprezintă importarea de produse, modificarea meniului (ștergere de produse, adăugare de produse de bază sau compuse, modificare de produse) și generarea rapoartelor.



Operațiunile care vor putea fi realizate de către clienți reprezintă crearea unei comenzi, adăugarea unor produse în comandă, filtrarea produselor din meniu pe baza unor criterii.

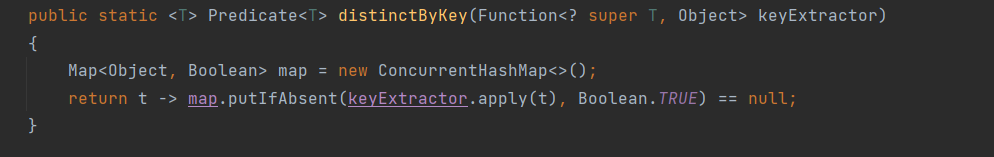


### Clasa DeliveryService

Această clasă moștenește clasa IDeliveryService și implementează interfețele Observable și Serializable. În cadrul acestei clasă este implementată funționalitatea serviciului. Informațiile sunt stocate in atribute de clasă: meniul este stocat într-o structură de tipul ArrayList<MenuItem>, comenzile sunt stocate într-un HashMap<Order, ArrayList<MenuItem>> unde vor fi salvate toate comenzile împreună cu lista de produse selectate, o structură de tipul List<Observer> , atributele de tipul FileWriter și Serialization sunt utilizate pentru accesarea metodelor din pachetul Data Layer.

Metodele setSerialize(Serialization serialization) și getSerialize() au rolul de a seta atributul de clasa serialization și de a apela funcția de serializare din clasa Serialization destinată obiectelor de tipul DeliveryService.

Metoda importProducts va deschide fișierul .csv une se află lista produselor de bază și utilizând streamuri și expresii lambda va parcurge liniile fișierului și va returna obiecte de tipul BaseProducts care vor fi mai apoi adăugate în lista de produse. Pentru a nu apărea duplicate în lista de produse am realizat metoda distinctByKey.



Metoda getMenuItem va returna parcurge întreaga listă de produse și va returna produsul al cărui nume a fost primit ca parametru.

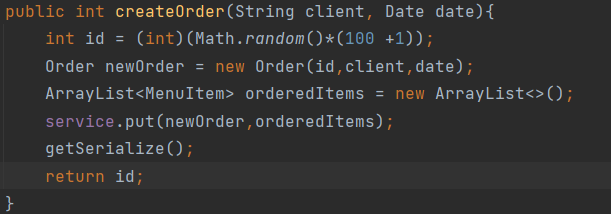
Metoda deleteFromMenu va parcurge lista de produse și va șterge produsul al cărui nume a fost primit ca parametru.

Metoda updateInMenu va parcurge lista de produse și va seta noul preț pentru produsul al cărui nume a fost transmis ca parametru.

Metoda addInMenu va adăuga produsul primit caparametru.

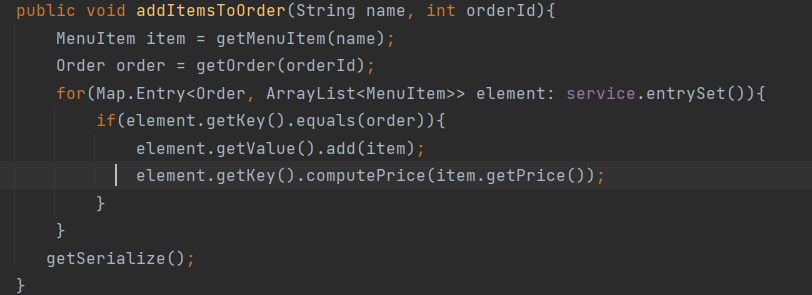
Metoda addToComposite va adăuga un nou produs primit ca parametru în lista de produse ale produsului compus primit ca parametru.

Metoda createOrder va crea o nouă comandă pentru utilizatorul al cărui Id a fost primit ca parametru și cu data primită ca parametru, Id-ul comenzii va fi generat random ca fiind un număr între 0 și 100. După crearea unei noi comenzi se va adăuga in hahmap comanda împreună cu o listă unde vor urma a fi stocate produsele comandate.



Metoda getOrder va parcurge hahmap-ul și dacă se va regăsi elementul cu id-ul transmis ca parametru, acesta se va returna Order.

Metoda addItemsToOrder va adăuga în lista corespunzătoare orderid-ului primit ca parametru produsul al cărui nume a fost primit ca parametru.



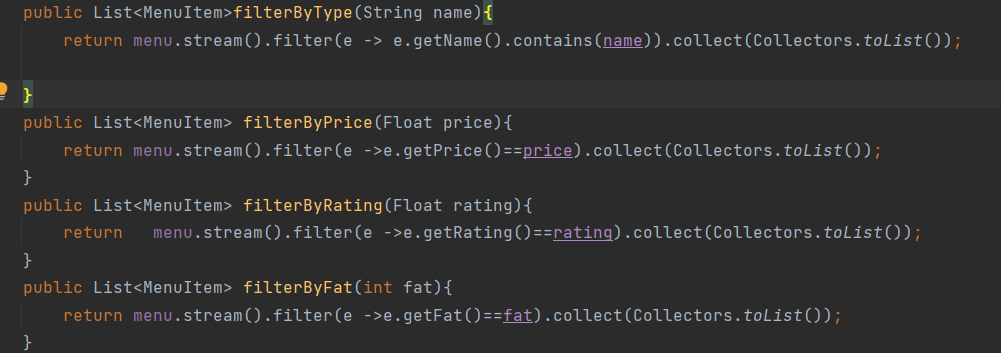
Metoda getMenu va returna sub forma unei liste de siruri de obiecte elementele meniului.

Metoda registerObserver va adăuga un nou obiect de tipul Observer în listă.

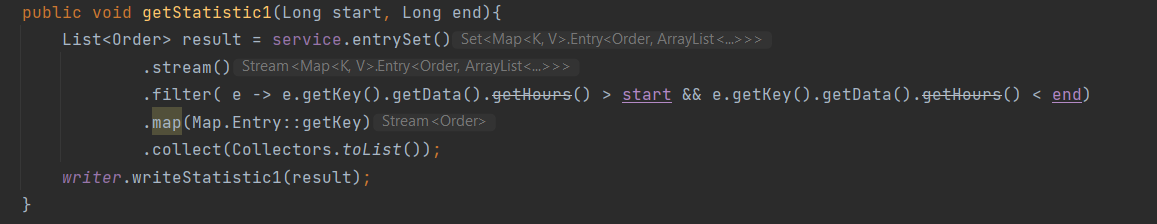
Metoda notifyOrder va apela metoda update din interfața Observer care va notifica fiecare angajat plasarea unei noi comenzi.

Metoda deleteObserver va șterge obiectul de tip Observer din listă.

Metodele filterByType, filterByPrice, FilterByRating, filterbyFat, filterbyCalories, filterBySodium și filterByProtein vor filtra elementele din meniu utilizând streamuri și expresii de tip lambda astfel încăt vor fi selecționate doar produsele care vor avea specificațiile dorite.



Metodele getStatistic1, getStatistic2, getStatisic3, getStatistic4 vor utiliza streamuri si expresii lambda pentru a selecționa din hashmap informațiile dorite pentru fiecare dintre rapoarte.



## Presentation Layer

În cadrul acestui pachet se realizează operațiile asupra interfața grafică.

### Clasa Administrator

În această clasă va fi creat fram-ul pentru utilizatorul de tip administrator unde utilizatorul va putea efectua operațiunile corespunzătoare prin introducerea de date în câmpuri de tipul JtextField și apăsasarea unor butoane.

Metoda updateTable va șterge componentele de tipul Jtable, dacă există, și va crea un nou astfel de table populându-l cu datele din meniu actuale.

### Clasa Client

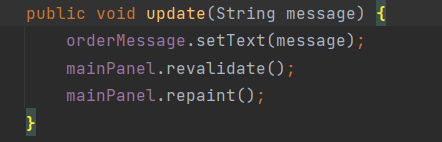
În această clasă va fi creat Jframeul pentru utilizatorii de tip clienți unde vor putea efectua operațiunile corespunzătoare prin introducerea de date în câmpurile JtextField si apăsarea unor butoane.

Metoda updateTable va șterge componentele de tipul Jtable, dacă există, și va crea un nou astfel de table populându-l cu datele din meniu actuale.

### Clasa Employee

Va fi creat JFrameul pentru utilizatorii de tip employee unde vor fi notificați atunci când va fi plastă o comandă și vor putea genera factura.

Această clasă implementează interfața Observable, astfel este necesară implementarea metodei update care va afișa în interfață noua comandă plasată.



### Clasa LogIn

În această clasă se va crea JFrame-ul de unde fiecare utilizator se va putea autentifica sau în cazul unui client sau angajat nou se vor putea accesa opțiuni de creare de noi conturi.

### Clasa NewClient

În această clasă se va crea JFrame-ul pentru optiunea de creare de noi conturi, unde noul utilizator va putea introduce noi credențiale și va fi adăugat în hashmap-ul corespunzător.

### Interfața Observer

Pentru a notifica fiecare plasare a unei comenzi în interfața dedicată angazaților este necesară metoda update() care este definită în această interfață.

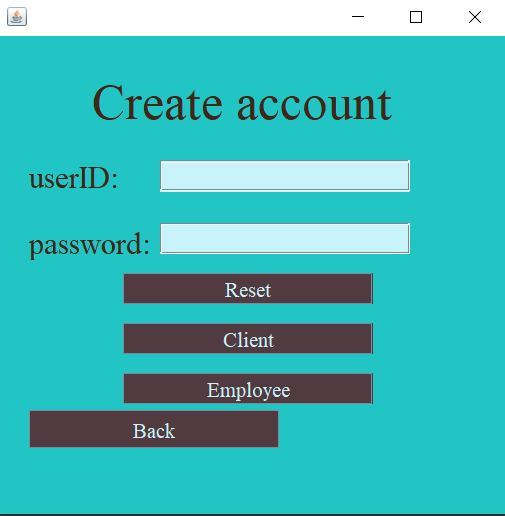
## Pachetul Model

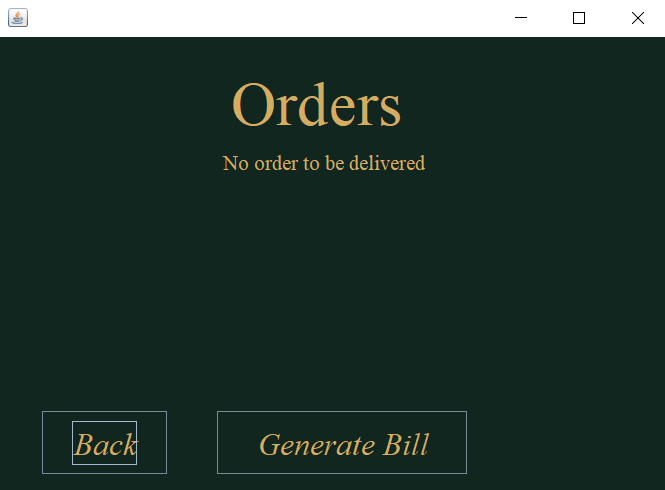
### Clasa Controller

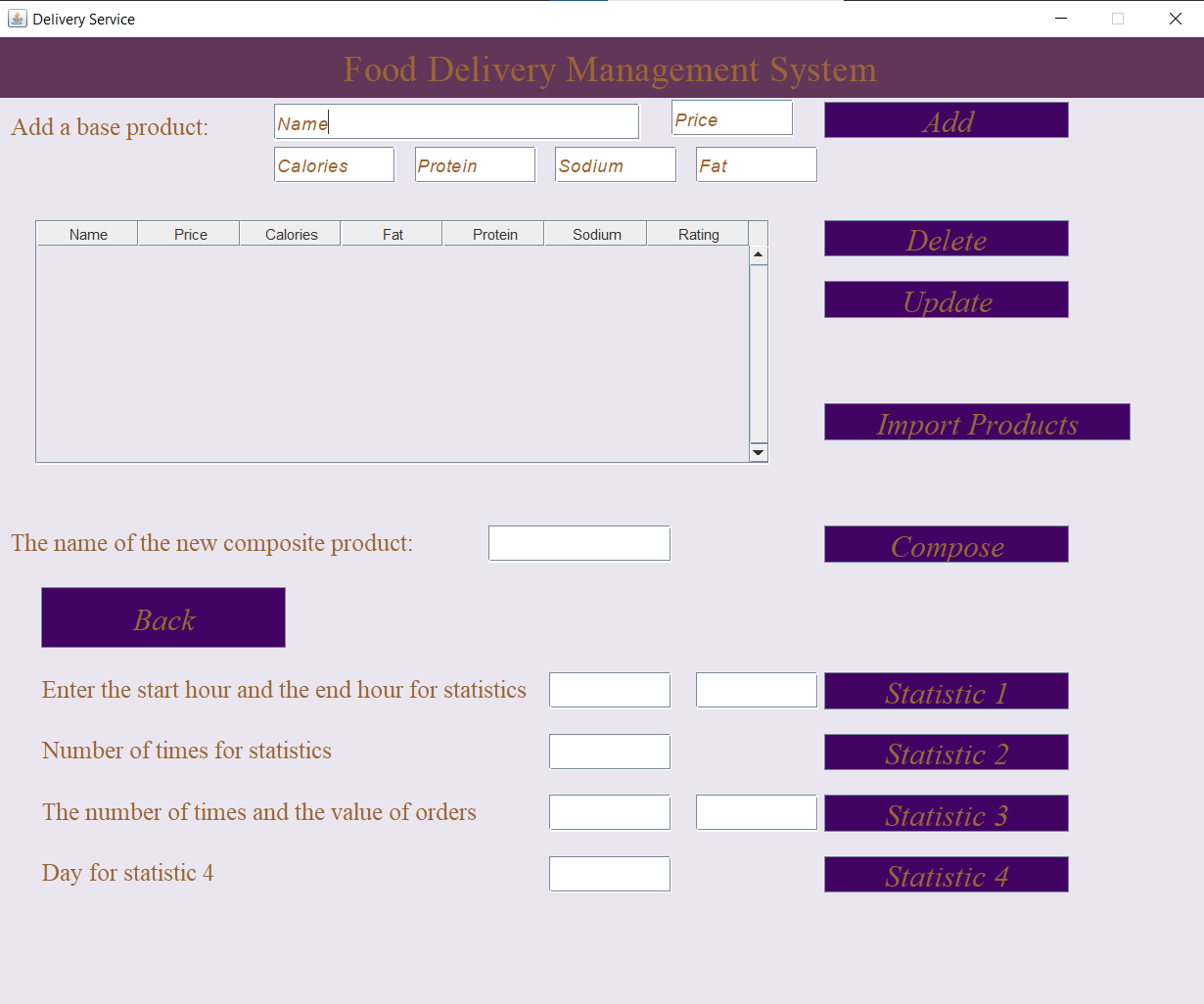
În cadrul acestei clase este definită doar metoda mei care va controla functionarea sistemului.

# Rezultate

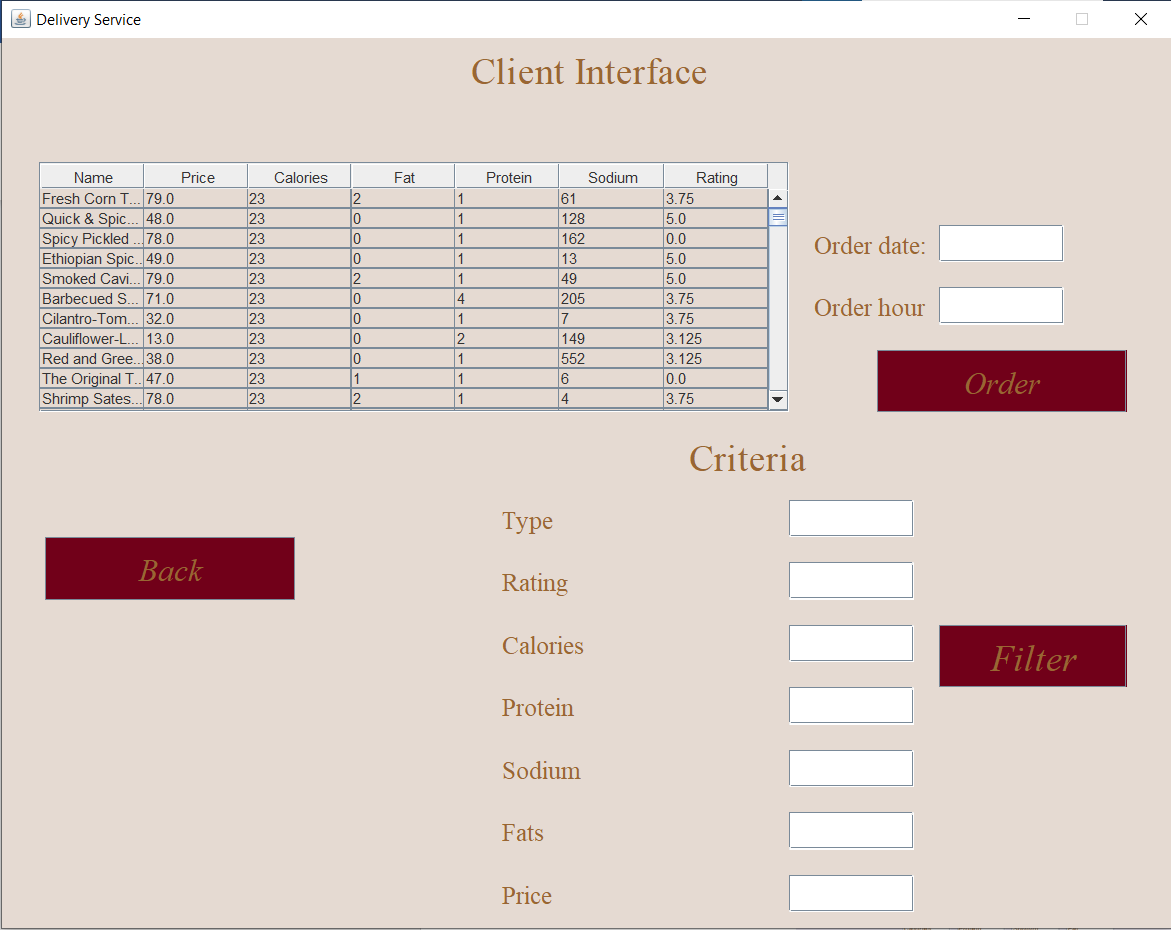
Interfața grafică pentru LogIn:

Interfața grafică pentru crearea unui nou cont:

Interfața grafică a angajatului:



Interfața grafică a Administratorului:



Interfața grafică a Clientului:

## Concluzii

În cadrul acestei teme s-au clarificat notiuni precum Lambda expressions and stream processing, Java serialization, Composite Design Pattern, Observer Design Pattern.

# Bibliografie

Lambda expressions and stream processing:

[Method References (The Java™ Tutorials > Learning the Java Language > Classes and Objects) (oracle.com)](https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html)

[Processing Data with Java SE 8 Streams, Part 1 (oracle.com)](https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html)

[Java 8 Stream Tutorial (winterbe.com)](https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/)

[Java Stream distinct() with Examples - HowToDoInJava](https://howtodoinjava.com/java8/java-stream-distinct-examples/)

Serialization

[Java - Serialization - Tutorialspoint](https://www.tutorialspoint.com/java/java_serialization.htm)

Assert

[Programming With Assertions (oracle.com)](https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html)

[javadoc-The Java API Documentation Generator (oracle.com)](https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag)