Tehnici de Programare – Tema 4

*Gestionarea unui Sistem de Livrare a Comenzilor*

Popovici Ruxandra

*Grupa 30227*

1. Obiectivul Temei

Obiectivul principal al proiectului de fata a fost proiectarea si implementarea unui sistem de livrare a comenzilor unei companii de catering. Se foloseste o interfata grafica dedicata prin intermediul careia unul din cele 3 tipuri de utilizatori se va va putea loga cu un nume de utilizator si o parola (administratorul, angajatul si clientul) si va putea face operatiile specifice care vor fi descrise mai jos, fiecare in functie de rolul incredintat la prima logare (prin “Sign Up”).

Obiectivele secundare ale proiectului:

* Analizarea obiectivelor proiectului si a problemelor legate de implementare prin modelarea de scenarii corespunzatoare cazurilor de utilizare, astfel incat sa identificam corect cerintele;
* Proiectarea aplicatiei astfel incat toate necesitatile utilizatorului sa fie indeplinte;
* Implementarea solutiei optime descoperite prin scrierea de cod;
* Testarea aplicatiei, observandu-i si analizandu-i concret comportamentul pentru mai multe cazuri, luand in considerare scenarii diferite de folosire si diferite date de intrare.

2. Analiza problemei (*modelare, scenarii,cazuri de utilizare)*

Prin analiza problemei intelegem examinarea cerintelor, potentialelor probleme si a diferitelor modalitati in care aplicatia noastra ar putea fi folosita, pusa la incercare: introducerea de date (numele de utilizator, parola, meniul, datele cu care se doreste a efectua operatii de editare, realizare a rapoartelor) care va duce la generarea unui raspuns corespunzator din partea simulatorului in functie de corectitudinea lor, in functie de rolul si accesibilitatea utilizatorilor in functie de acesta.

Dorim ca utilizatorul sa fie atentionat cu privire la corectitudinea datelor, astfel incat va primi mesaj de eroare daca acestea nu respecta cerintele(sa fie numere intregi).

De asemenea, programul trebuie sa functioneze in mod adecvat si sa genereze raspunsul dorit, urmand principiul diagramei urmatoare:

Operatia pe care vrem sa o efectuam

(oricare din butoanele accesibile puse la dispozitie de catre aplicatie)

Rolul ales (prin username si parola)

Rezultat = Factura \*.txt si Rapoartele

+ interactiuni valide cu aplicatia

*FOOD DELIVERY MANAGEMENT SYSTEM*

*~Schema generala a proiectului~*

Date de intrare corespunzatoare fiecarui utilizator in parte

Diagrama Use Case demostreaza mai bine functionalitatea si ideea intregului proiect:

**User**

Dupa cum putem observa si mai sus, intregul sistem reprezinta aplicatia creata, in care User-ul reprezinta, de fapt, actorul / utilizatorul, care se asteapta sa primeasca un raspuns / rezultat dupa ce actioneaza asupra lui.

* Cerinte functionale ale aplicatiei, in functie de rolul utilizatorului:
* Administratorului:
* Trebuie sa permita utilizatorului sa importe produsele intiliale dintr-un fisier \*.csv;
* Trebuie sa permita utilizatorului sa administreze aceste produse care, mai apot, vor fi vizibile clientului prin meniu: adaugare, stergere, editare a produselor “de baza”, crearea a noi produse “de baza” si a produse / meniuri compuse;
* Trebuie sa permita utilizatorului sa genereze raporte despre comenzile efectuate in functie de mai multe criterii: intervalul de timp in care s-au efectual, numarul de care au fost comandate anumite produse, clientii care au comandat de un anumit numar de ori, iar valoarea comenzii a fost mai mare decat cea specificata, produsele comandate intr-o anumita zi a saptamanii, impreuna cu numarul de ori de care au fost comadate.
* Trebuie sa permita utilizatorului vizionarea rezultatului cerut la efectuarea unei actiuni prin care cere generarea de rapoarte – fisierul \*.txt, care va contine toate datele cautate.
* Clientului:
* Trebuie sa permita utilizatorului sa vizualizeze lista de produse pusa la dispozitie prin intermediul meniului de catre administrator;
* Trebuie sa permita utilizatorului ca caute produse in functie de anumite criterii: nume, rating, calorii, sare, etc..
* Trebuie sa permita utilizatorului sa aleaga din produsele repsective, sa plaseze o comanda si sa i se genereze, astfel, o factura cu ce a comadat (fisierul \*.txt), care va contine lista cu toate produsele din “cos” si pretul total.
* Angajatului:
* Trebuie sa permita utilizatorului sa vizualizeze ultima comada plasata pentru a “pregati” livrarea produselor specificate (va fi notificat de fiecare data cand o noua comada a fost plasata);
* Altele:
* Aplicatia trebuie sa fie intuitiva si usor de folosit de catre utilizator.
* Sa efectueze instructiunile cerute in caz de success sau sa afiseze un mesaj de eroare in caz ca datele primite nu au respectat cerintele sau formatul cautat, ori daca ceea ce s-a cautat nu a fost gasit.
* Scenariu de utilizare:

1. Utilizatorul intorduce numele de utilizator si parola: ex. “admin” si “admin”, avand rolul de aministrator;
2. In meniul deja existent alege sa introduca un nou produs din lista de produse initiale;
3. Utilizatorul introduce numele acetuia si apasa pe butonul “ADD PRODUCT”;
4. Se iese din fereastra specifica administratorului, iar un nou client se logheaza si doreste sa plaseze o comanda;
5. Cauta in functie de calorii, introducand valoarea 23 in caseta de cautare si selectand criteriul din tabela, dupa care apasa pe butonul corespunzzaoru, iar o noua tabela se populeaza cu rezultatul cautarii;
6. Clientul va alege ce produse doreste sa achizitioneze, le ”adauga in cos” si finalizeaza comanda;
7. Se va genera un fisier \*.txt, reprezentand factura comenzii, iar angajatul va primi o notificare cu detaliile comenzii.
8. Utilizatorul iese din aplicatie.

Un scenariu alternativ de utilizare:

1. Utilizatorul introduce numele unui client inexistent, ori o parola gresita, ori numele unui produs inexistent in meniu;
2. Aplicatia afiseaza un mesaj de eroare si sterge datele introduse;
3. Se revine la primii pasi din scenariul anterior.
4. Proiectare *(decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)*

Proiectarea in limbaj OOP a presupus mai multi pasi, in functie de deciziile luate: am urmarit implementarea unui model tip sistem interactiv si repartizarea claselor in pachete, asa cum se poate observa si in diagrama de mai jos:

**start**

Main

BaseProduct

CompositeProduct

MenuItem

Order

IDeliveryService

DeliveryService

**presentation**

UserGUI

AdministratorGUI

ClientGUI

EmployeeGUI

**data**

**bussiness**

BillGenerator

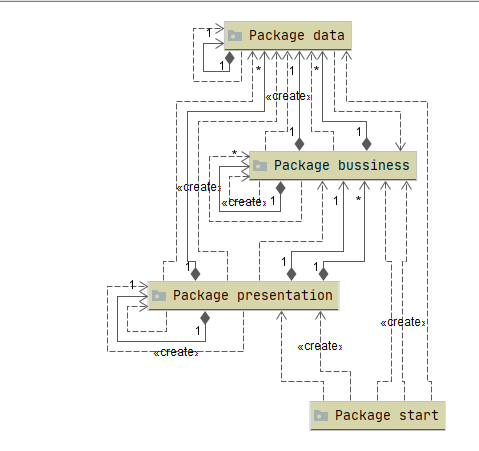
ReportGenerator

Serializer

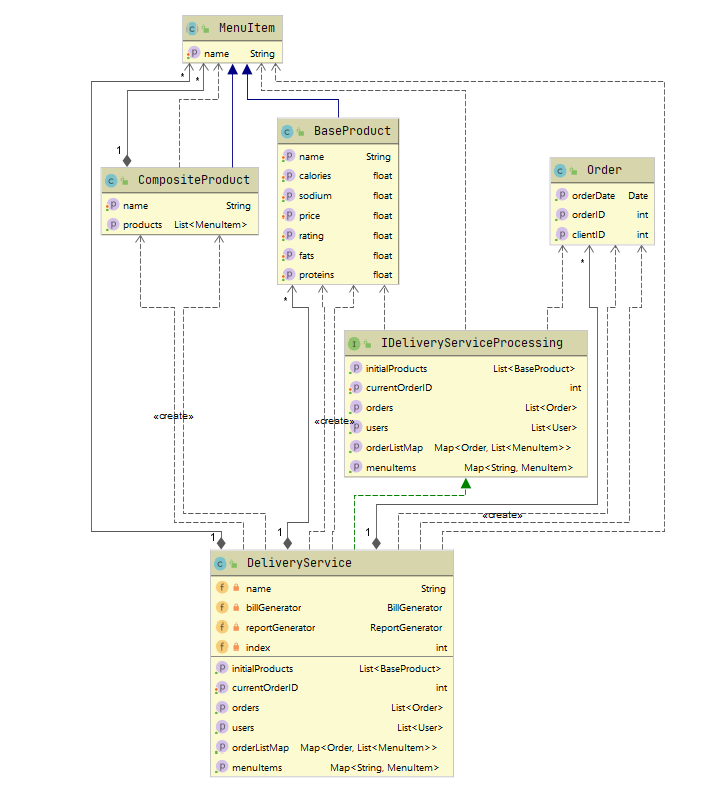
Role

User

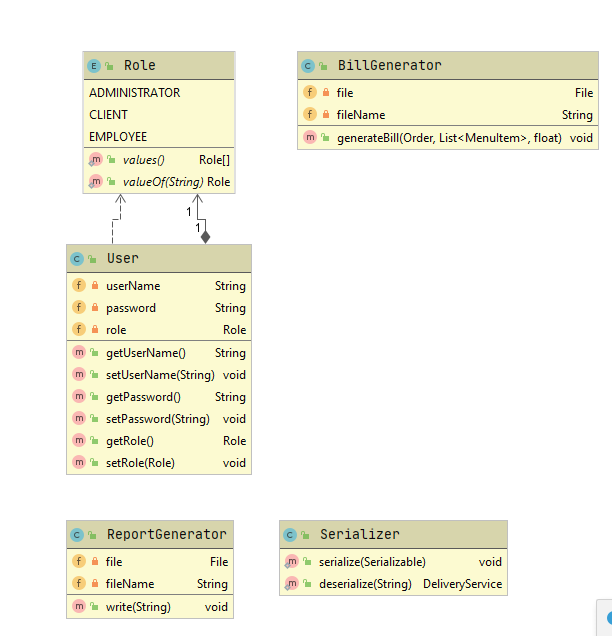
Diagrama UML a proiectului ajuta la o mai buna si amanuntita intelegere ai analiza a claselor, pachetelor, metodelor implementate in fiecare din acestea, cum se “leaga” si “comunica” intre ele:



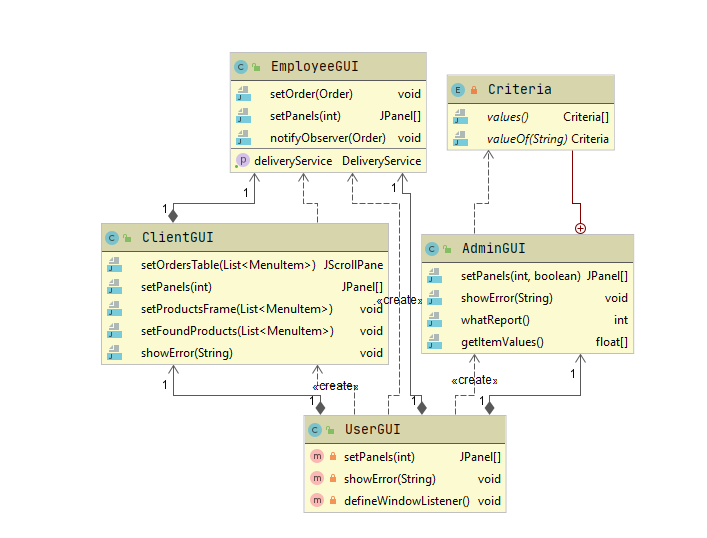
***Pachetul bussiness:***



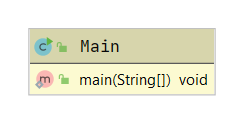
***Pachetul data:***

******

***Pachetul presentation:***

******

***Pachetul start:***

******

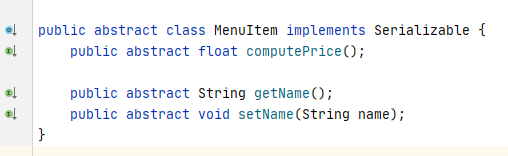
1. Implementare

* *Pachetul “bussiness”*

***Clasa abstracta MenuItem***

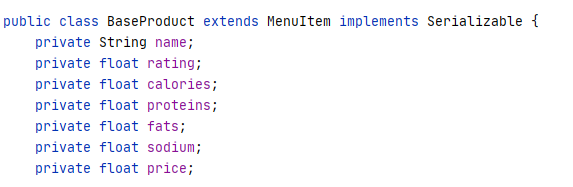
Va reprezenta produsele din meniul aplicatiei. Aleasa ca fiind abstracta, clasa va fi extinsa de cele alte doua clase descrise mai jos, pentru care comune vor fi metodele computePrice(), getName si setName, astfel incat ca, indiferent daca produsul este compus sau simplu, i se va pune la dispozitie utilizatorului numele si pretul.

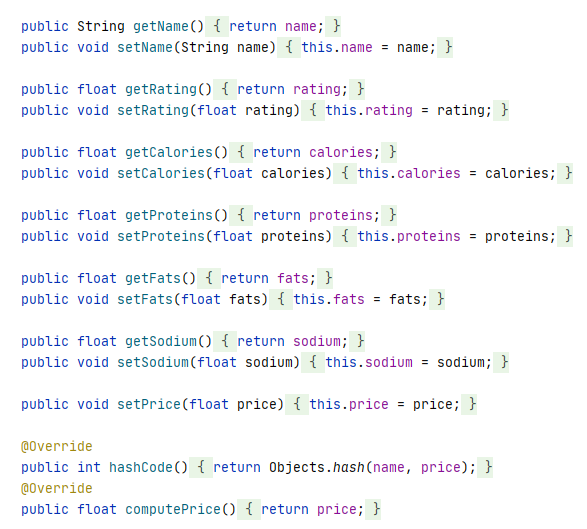
Clasa implementeaza interfata Serializable, pentru a putea realiza serializarea asupra acesteia, deci a-i salva valorile de la o rulare la alta.



***Clasa BaseProduct***

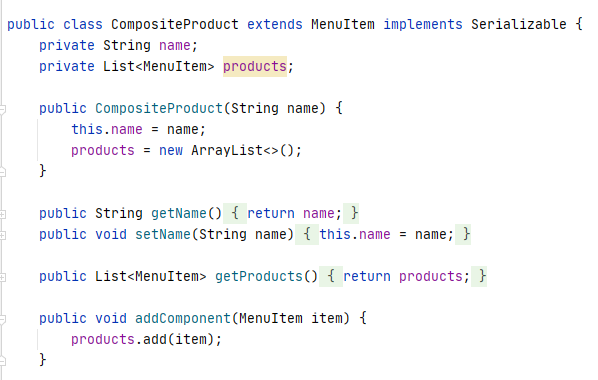
Va folosi si va avea rolul produselor de baza, a celor simple din meniu, care vor fi reprezentate initial, de lista a ce extragem din fisierul \*.csv dat la dispozitie. Specifice acestei clase sunt o serie de date de tip float care vor reprezenta atributele produselor (ex. rating, calorii, grasimi, sare, etc.), iar comune cu ale clasei “surori” vor fi numele si pretul. Implementeaza, astfel, metodele abstractei ale superclasei sale ca si simple getter-e si setter-e pentru valorile din aceasta.

**

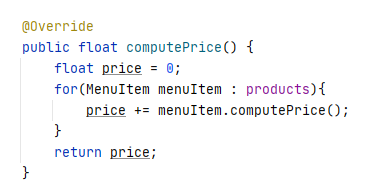


***Clasa CompositeProduct***

Reprezinta produsele compuse / meniuri formate din mai multe produse care vor ajunge sa fie create de catre utilizator dupa ce va avea la dispozitie produse simple de tip “BaseProduct” sau chiar si alte produse compuse. Asadar, pe langa numele care este un camp comun cu cel al produselor simple, clasa mai are specific o lista de obiecte de tip “MenuItem” in care se pot adauga componente diferite.

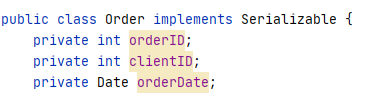


Astfel, metoda computePrice este modificata astfel incat sa poata returna suma preturilor tuturor componentelor din lista de obiecte de tip MenuItem:



***Clasa “Order”***

Pentru o buna organizare a comenzilor in aceasta aplicatie, clasa “Order” va pune la dispozitie obiecte cu urmaoarele atribute:



Pentru a sti cine, ce si cand a cumparat / efectuat o comanda.

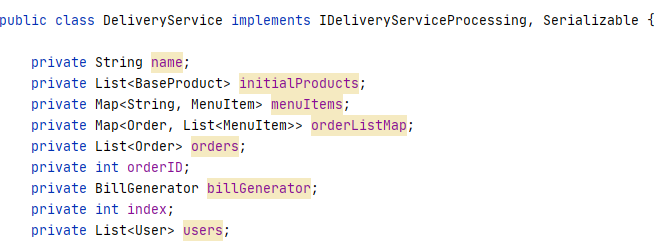
***Interfata “IDeliveryServiceProcessing”***

Aici vom gasi header-ele tuturor metodelor specifice prin care vom putea opera asupra meniului si aplicatiei in general.

***Clasa “DeliveryService”***

Va implementa atat metodele propuse de interfata IDeliveryService, cat si interfata Serializable si va fi obiectul proiectului asupra caruia se va face explicit operatia de serializare.

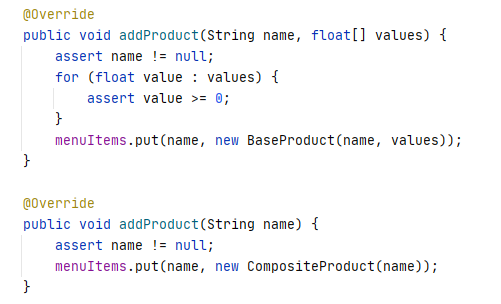
Parametrii sai specifici sunt declarati in functie de necesitatile aplicatiei.



Lista initiala de produse, reprezentata de List<BaseProduct> initialProducts va contine elementele extrase din fisierul \*.csv, adica numai produse simple, disponibile fiind numai administratorului aplicatiei. Popularea acesteia se face cu metoda importProducts(), apelata doar la prima pornire a aplicatiei de catre utilizatorul cu rol de admin:



Elementele din meniul ce va fi disponibil si clientului si pe care se vor face restul operatiilor sunt reprezentate de Map<String, MenuItem> menuItems, un HashMap in care se vor adauga ori obiecte simple, “BaseProduct”, ori obiecte compuse, “CompositeProduct”, in functie de numele acestora. In functie de tipul lor, avem doua metode prin care se face adaugarea:

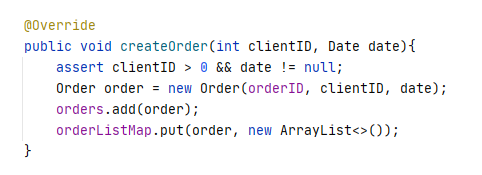


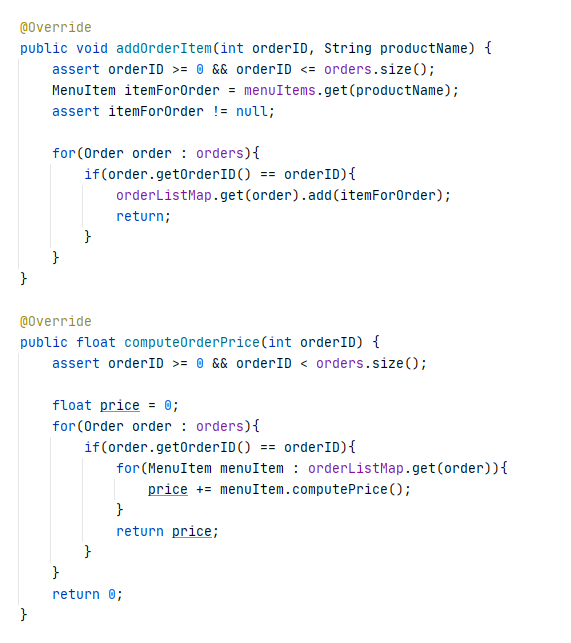
Si inca una prin intermediul careia se pot forma produsele compuse:



Evidenta comenzilor se va tine cu ajutorul celor doua structuri: Map<Order, List<MenuItem>> orderListMap si List<Order> orders;

Lista de comenzi va contine doar structuri in care se tine minte data plasarii comenzii si ID-urile acesteia si a clientului, iar HashMap-ul va retine produsele cumparate pentru fiecare comanda plasata.

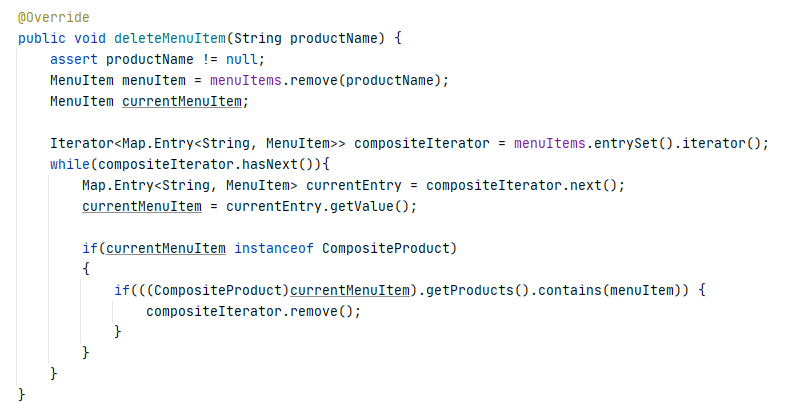




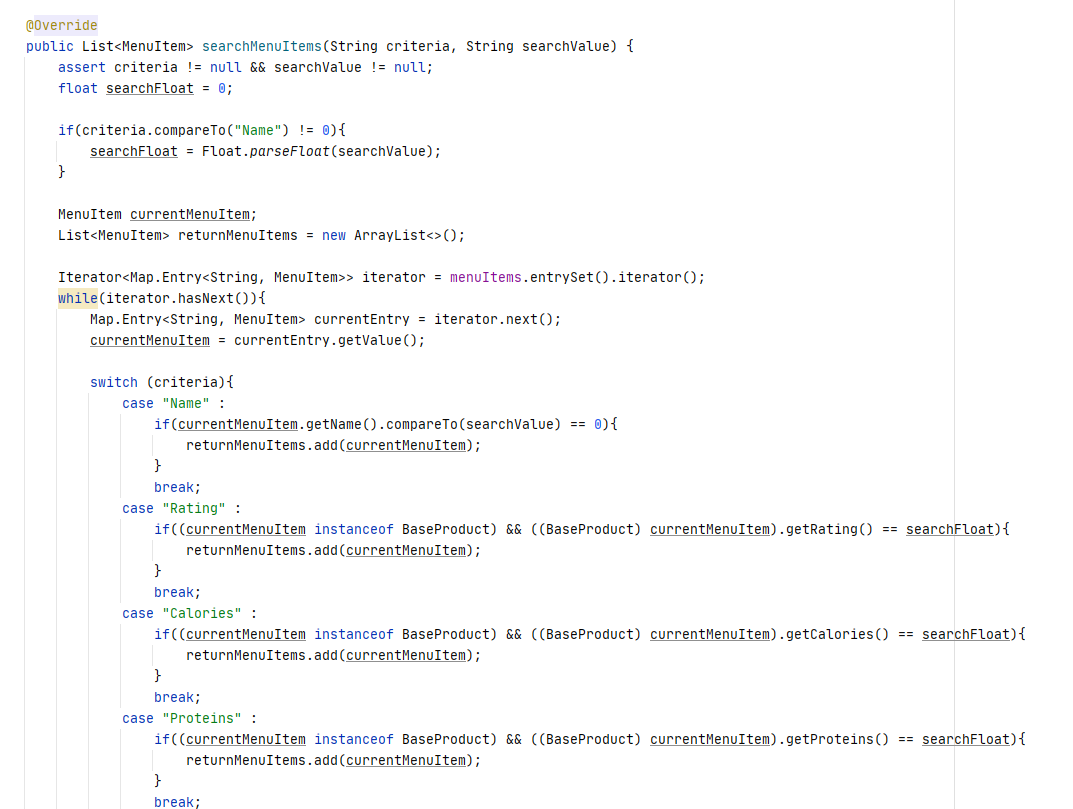
User-ii vor fi retinuti de catre private List<User> users, dar clasificarea lor va tine de clasa UserGUI, cand se va face logarea. Deoarece sunt accesati si aici, valorile din aceasta lista vor fi retinute de la o rulare la alta prin serializare.

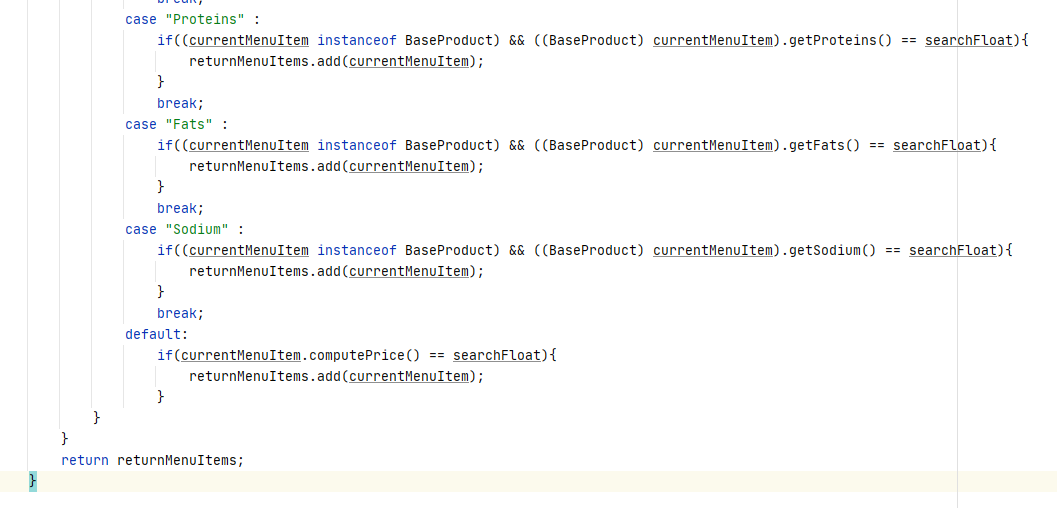
Celelalte metode care manipuleaza direct produsele din meniu se “ocupa” de editare in functie de criteriul cerut: 

Stergere:

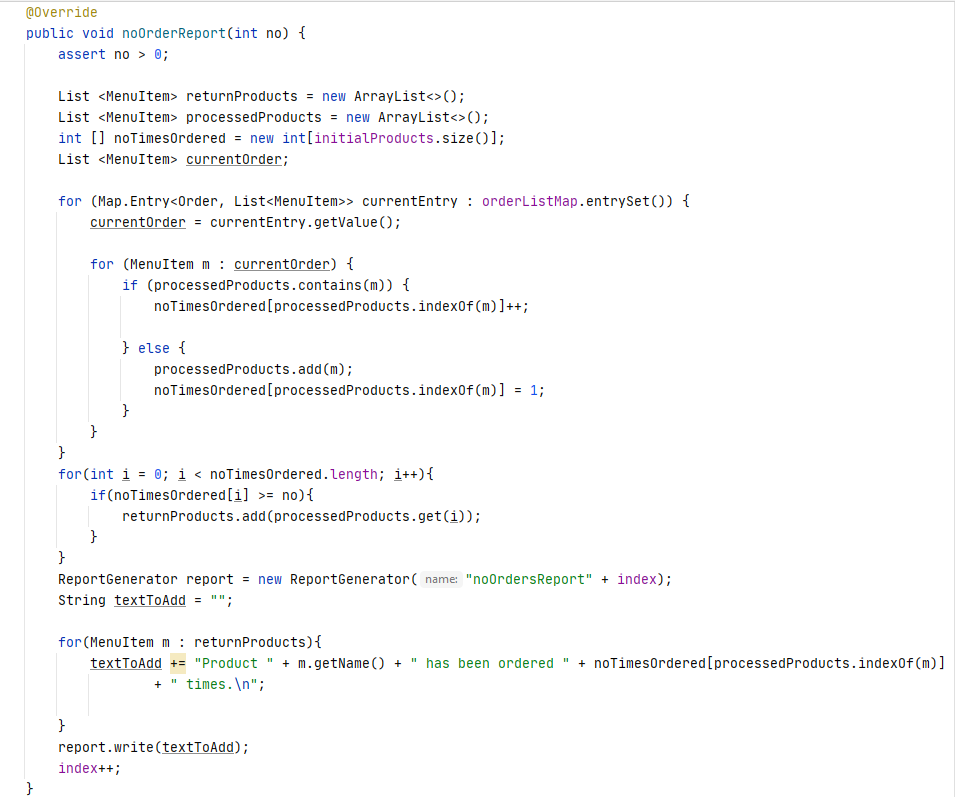


Si cautare in funtie de campurile specifice produselor:





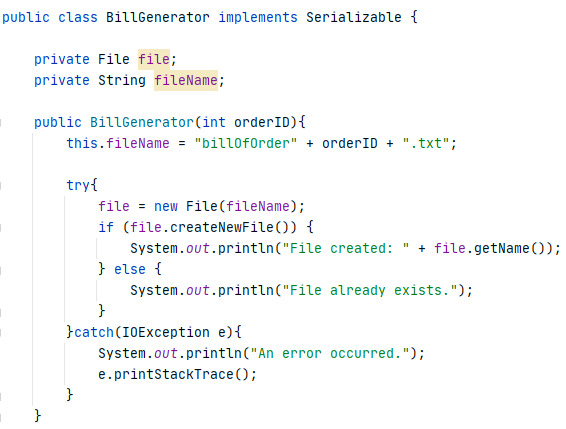
Restul metodelor tin de cele 4 rapoarte care pot fi generate de catre administrator. Vom folosi un obiect de tip ReportGenerator prin intermediul caruia vom scrie intr-un fisier \*.txt toate datele necesare:

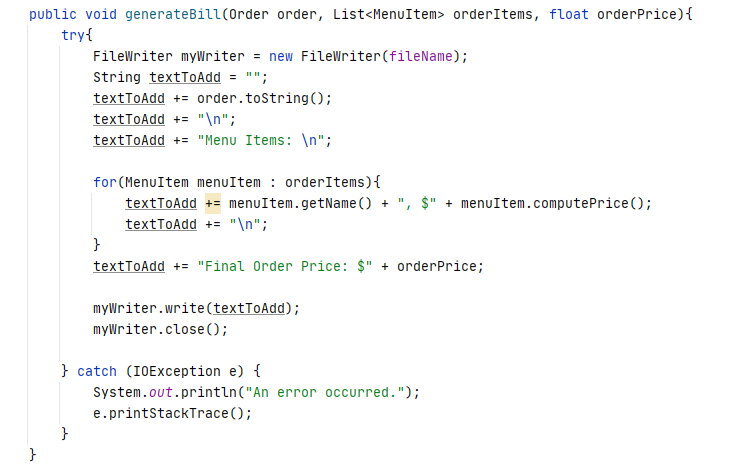
* Pentru interval: 
* Pentru numarul de ori de care au fost comandate: 
* Pentru valoare si numar de comenzi plasate: 
* Pentru ziua in care au fost comandate: 
* *Pachetul “data”*

***Clasa “BillGenerator”***

Va genera factura corespunzatoare comenzii plasate, cu conditia ca aceasta s-a efectuat cu succes.

Costructorul va initializa un fisier \*.txt in care vom scrie cu ajutorul metodei “generateBill” ce va primi ca parametru un obiect de tip “Order”, o lista de obiecte de tip “MenuItem” si valoarea totala a facturii, de ale caror date se va folosi:





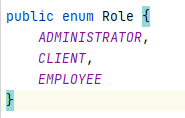
***Clasa “ReportGenerator”***

Asemenea clasei “BillGenerator”, cu diferenta ca datele ce se scriu in rapoarte sunt diferite:



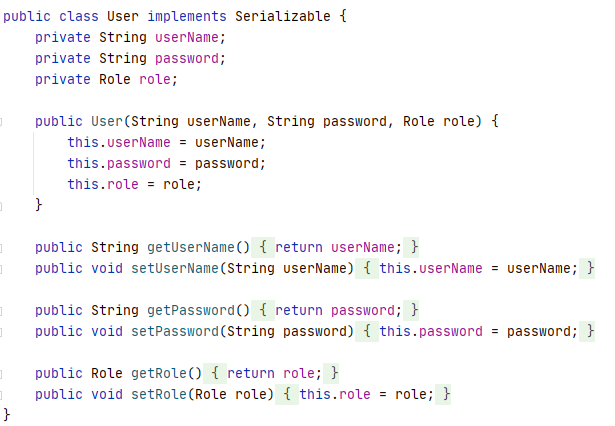
***Enumul “Role”***

Va fi folosit in iterarea prin rolurile pe care le au utilizatorii. Am utilizat aceasta implementare pentru o mai usoara intelegere a codului si o mai buna lizibilitate:



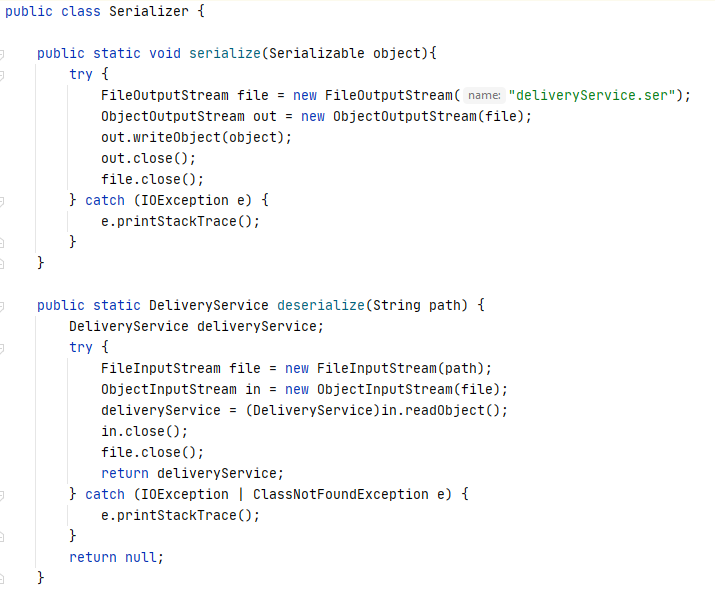
***Clasa “User”***

Va reprezenta utilizatorul ce se va putea loga in aplicatie. Astfel, campurile userName, password si role sunt specifice fiecaruia dintre acestia si vor fi folosite pentru identificarea si deschiderea ferestrei corespunzatoare fiecarui rol.



***Clasa “Serializer”***

Contine metodele specifice serializarii si deserializarii aplicatiei pentru fiecare rulare. Obiectele pentru care se apeleaza vor fi, astfel, tranformate intr-un fisier binar la fiecare inchidere a aplicatiei si transformat inapoi in instanta initiala atunci cand aplicatia este din nou pornita:

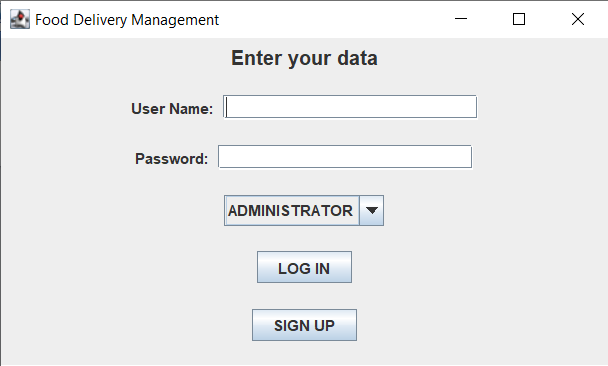


* *Pachetul “presentation”*

***Clasa “UserGUI”***

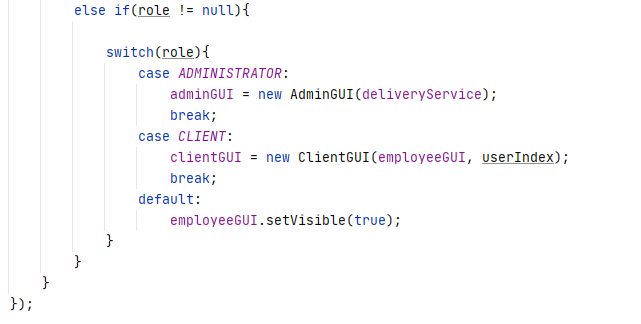
La fel ca si cele de mai jos, extinde clasa JFrame si mosteneste interfata Serializable.

Este o interfata grafica care se foloseste de 2 butoane pentru a seta si stabili / identifica utilizatorii ce se logheaza.



Astfel, prin butonul “LOG IN”, se verifica existenta unui utilizatorului care doreste sa intre in aplicatie, se testeaza corectitudinea parolei si i se identifica rolul pentru care se va deschide fereastra corespuzatoare (pentru administrator, client si angajat).





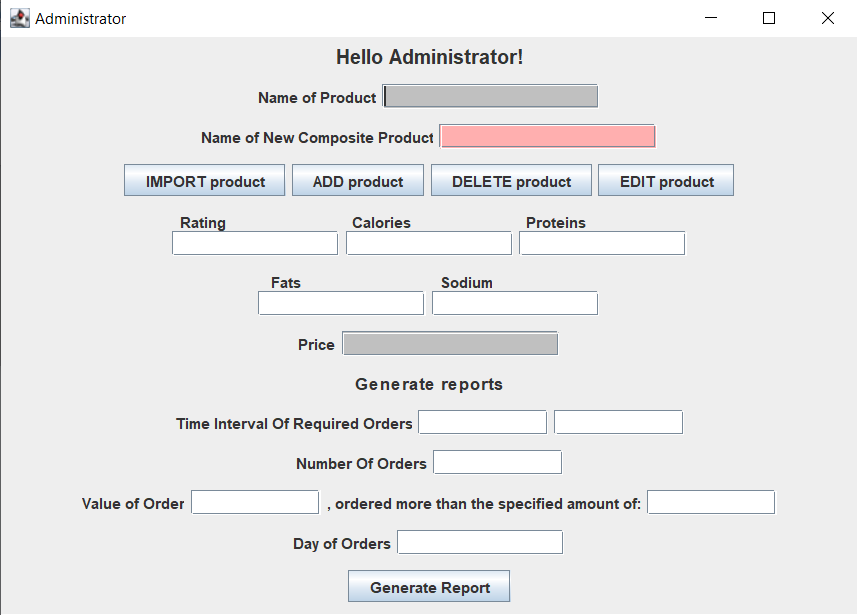
Prin celelalt, “SIGN UP”, un nou utilizator isi poate crea un nou cont, cu conditia ca un altul cu acelasi nume sa nu existe:



Lista de useri contine o referinta catre cea din obiectul deliveryService, astfel incat datele se vor salva de la o rulare la alta.

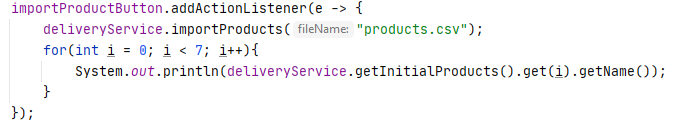
***Clasa “AdminGUI”***

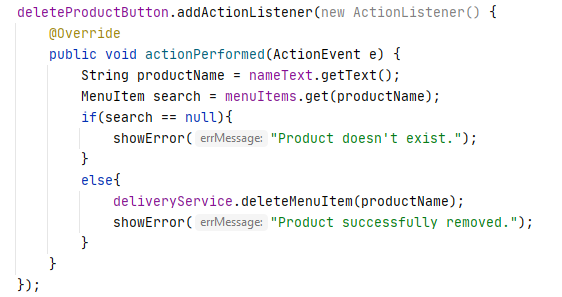
Este interfata grafica ce permite administratorului sa efectueze operatiile descrise la inceput.



Odata actionate, butoanele vor indeplini functiile descrise, nu inainte de a verifica existenta si corectitudinea datelor necesare, iar in caz de diferite erori, se vor afisa mesajele corespunzatoare de eroare.

Exemple:

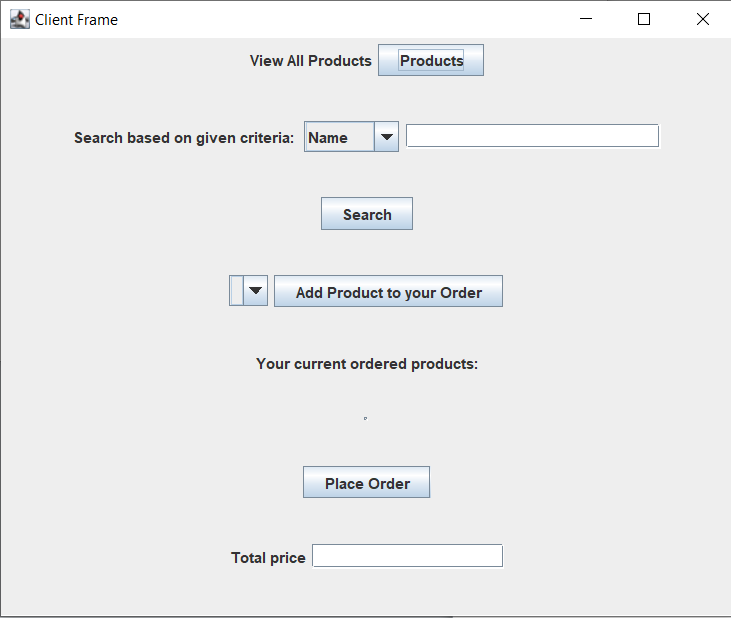




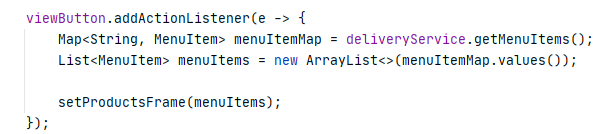


***Clasa “ClientGUI”***

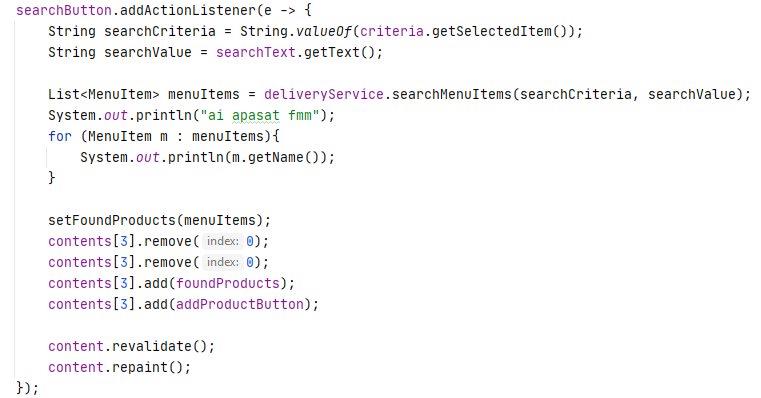
Este, de asemenea, interfata grafica care ii va permite clientului sa vizualizeze meniul si sa plaseze comenzi.



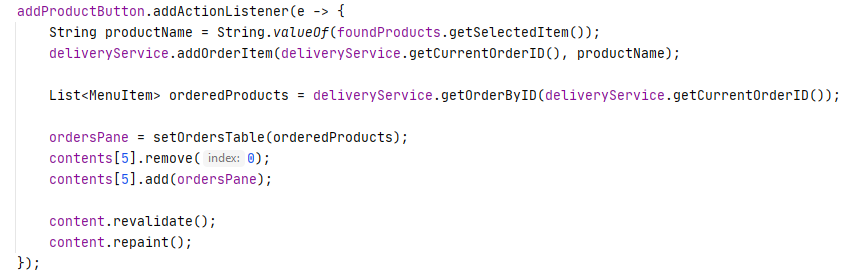
Butoanele sunt legate de actionListener-e care vor efectua operatiile cerute:



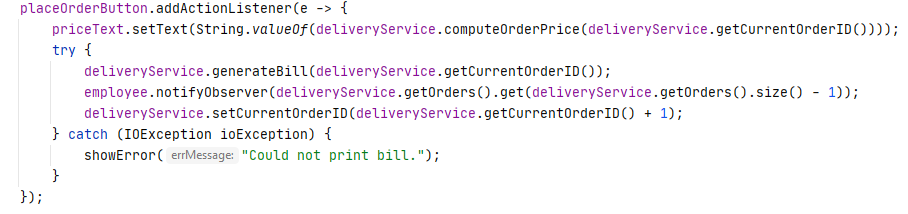
Rezulta in o noua fereastra care va contine un tabel cu produsele disponibile pentru achizitionat.



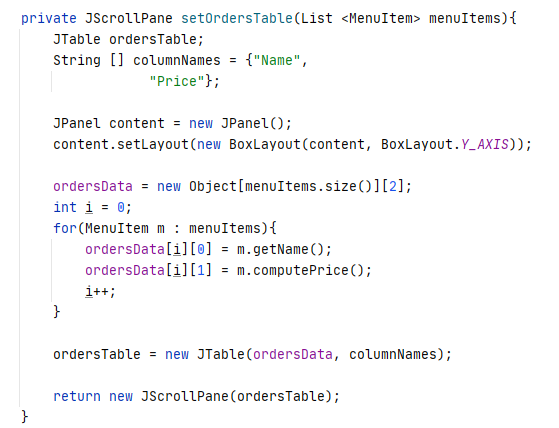
Rezulta in popularea ScrollPane-ului cu produsele cautate in functie de ctriteriile alese de client si valoarea ceruta.



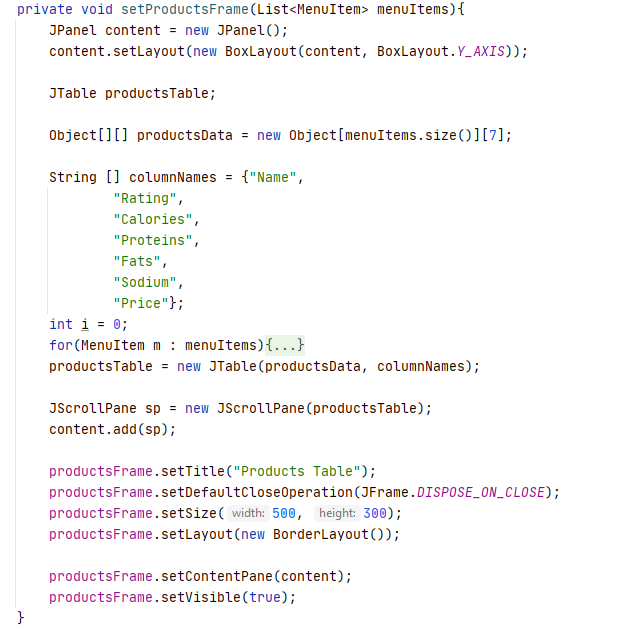
Va adauga produsul in cos si va afisa un tabel cu ce exista in comanda momentan.



Va calcula preturl comenzii, il va afisa si va genera factura cu toate datele necesare.

Mai avem functii de care ne ajutam , ca exemplu: 

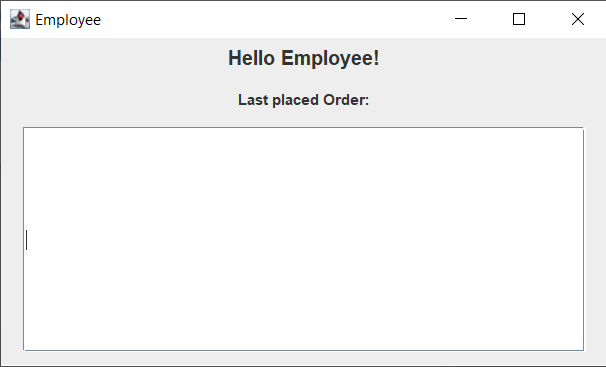
Si



Prin care vom afisa clientului datele cautate.

***Clasa “EmployeeGUI”***

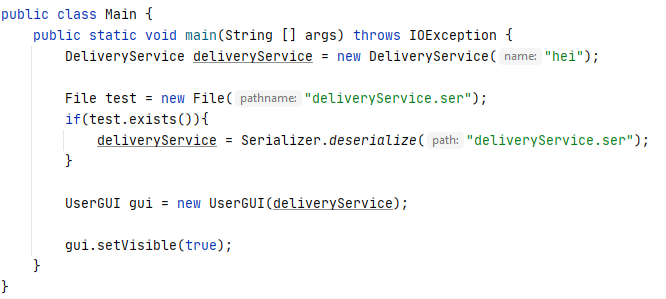
ultima interfata grafica disponibila pentru angajati, unde li se vor pune la dispozitie detaliile ultimei comenzi plasate, fiind notificati atunci cand un client va face o tranzactie de acest tip.



* *Pachetul “start”*

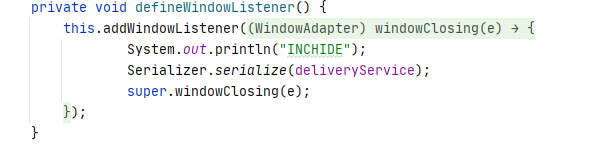
***Clasa “Main”***

Aceasta clasa porneste aplicatia si permite lucrul cu GUI, facand legatura intre toate celelalte clase si stabilind relatiile care vor duce la buna functionare a interfetei grafice si a programului din spatele acesteia.



Putem observa functionalitatea deserializarii, care va obtine un obiect din stream-ul de biti reprezentat de fisierul “deliveryService.ser” doar atunci cand acesta este creat (s-a mai efectuat o serializare anterior).

Serializarea se va face odata cu apasarea butonului de exit, cu care se iese din aplicatie, iar implementarea este facuta in clasa UserGUI, cea responsabila cu functionarea corecta a tututor celorlalte ferestre:



1. Rezultate

Aplicatia am testat-o luand-o de la 0 (stergand toate facturile deja existente si serializarea realizata cu celelalte teste), pentru o mai mare acuratete a rezultatelor. Am adaugat utilizatori cu nume reprezentative, astfel incat admin sa aiba rol de ADMINISTRATOR, client1, client2, ..., roluri de CLIENT si emp, rolul de angajat, EMPLOYEE.

Ca si administrator am importat, mai intai, produsele initiale din \*.csv, am adaugat cateva in meniu si am creat si adaugat si alte produse compuse.  
Prin introducerea de nume care nu existau nicaieri, ori nume de produse care nu au fost introduse in meniu, in functie de operatiile efectuate, am primit diferite mesaje de eroare.

Mai apoi, ca si client, am putut vizualiza meniul pus la dispozitie ca si admin, am putut face cautarea in functie de anumite criterii si, dupa ce am avut rezultatul acestei cautari la dispozitie, am putut adauga produse in “cos” pentru a putea, la final, plasa comanda -> s-a generat o factura, un fisier \*.txt in care am putut observa detaliile a ceea ce am achizitionat.

Nu in ultimul rand, ca si angajat, am putut observa detaliile ultimei comenzi – ID-ul clientului care a efectuat comanda, ID-ul comenzii respective, cat si data la care aceasta a fost plasata.

Rapoartele ce se pot genera din fereastra pentru administratori s-au generat dupa ce am introdus atat mai multi clienti, cat si mai multe comenzi si au afisat rezultatele cautate.

1. Concluzii

Prin implementarea acestui proiect s-a urmarit, in principiu, crearea unei aplicatii mai complexe, cu functionalitati diferite in functie de rolul utilizatorului ce o foloseste, si care are datele disponibile de la o rulare la alta.

Am reusit sa creez modele arhitecturale de Layers, sa folosesc tehnica serializarii.

As fi putut imbunatati modul in care se face editarea produselor, stergerea din produse compuse, cautarea in functie de mai multe criterii o data, anumite erori, etc..

1. Bibliografie

[*https://www.tutorialspoint.com/java/java\_serialization.htm*](https://www.tutorialspoint.com/java/java_serialization.htm)

[*https://www.baeldung.com/java-serialization*](https://www.baeldung.com/java-serialization)

[*https://www.geeksforgeeks.org/serialization-in-java/*](https://www.geeksforgeeks.org/serialization-in-java/)

[*https://www.youtube.com/watch?v=98DiwRp-KZk*](https://www.youtube.com/watch?v=98DiwRp-KZk)

[*https://www.baeldung.com/java-9-reactive-streams*](https://www.baeldung.com/java-9-reactive-streams)

[*https://www.baeldung.com/rxjava-vs-java-flow-api*](https://www.baeldung.com/rxjava-vs-java-flow-api)

[*https://www.w3schools.com/java/java\_files\_create.asp*](https://www.w3schools.com/java/java_files_create.asp)

[*https://docs.oracle.com/javase/8/docs/api/java/util/Date.html*](https://docs.oracle.com/javase/8/docs/api/java/util/Date.html)

[*https://stackoverflow.com/questions/907170/java-getminutes-and-gethours*](https://stackoverflow.com/questions/907170/java-getminutes-and-gethours)

[*https://knowm.org/get-day-of-week-from-date-object-in-java/*](https://knowm.org/get-day-of-week-from-date-object-in-java/)

[*https://www.java67.com/2015/08/how-to-load-data-from-csv-file-in-java.html*](https://www.java67.com/2015/08/how-to-load-data-from-csv-file-in-java.html)

[*https://stackoverflow.com/questions/16622462/windowclosing-method*](https://stackoverflow.com/questions/16622462/windowclosing-method)

*FUNDAMENTAL PROGRAMMING TECHNIQUES*

*ASSIGNMENT 4 – SUPPORT PRESENTATION -Teams-*