FOOD DELIVERY MANAGEMENT SYSTEM

DOCUMENTAȚIE

Studentă: Aurică Alina

Materie: Tehnici de programare

Profesor îndrumător: Mitrea Dan

CUPRINS:

1. Obiectiv
2. Studiul problemei
3. Implementare

* Diagrame UML
* Clase și metode
* GUI

1. Testare
2. Concluzii
3. Bibliografie
4. OBIECTIV:

Obiectivul proiectului este de a implementa și proiecta un order food delivery management system. Acesta trebuie să realizeze mai multe funcții depinzând de tipul de user. Userul admin realizează 4 rapoarte în funcție de anumite condiții, adaugă, șterge, modifică sau creează alte produse și importă lista de produse din fișierul .csv. Userul client poate crea o comandă și poate filtra meniul după termeni-cheie. Userul employee este notificat atunci când se realizează o comandă nouă.

Folosirea acestui food delivery management system se face printr-o interfață grafică pe mai multe nivele. Prima fereastră care se deschide este cea de login, iar în funcție de rolul avut de user se deschide fereastra specifică, care realizează anumite funcții.

1. STUDIUL PROBLEMEI:

Proiectul acesta prezintă multe noutăți, de la idea de interfață cu register și login, până la implementarea unor design patter-uri: Composite Design Pattern și Observer Design Pattern, alături de implementarea interfețelor Serializable și Observer și de Collection-ul HashMap.

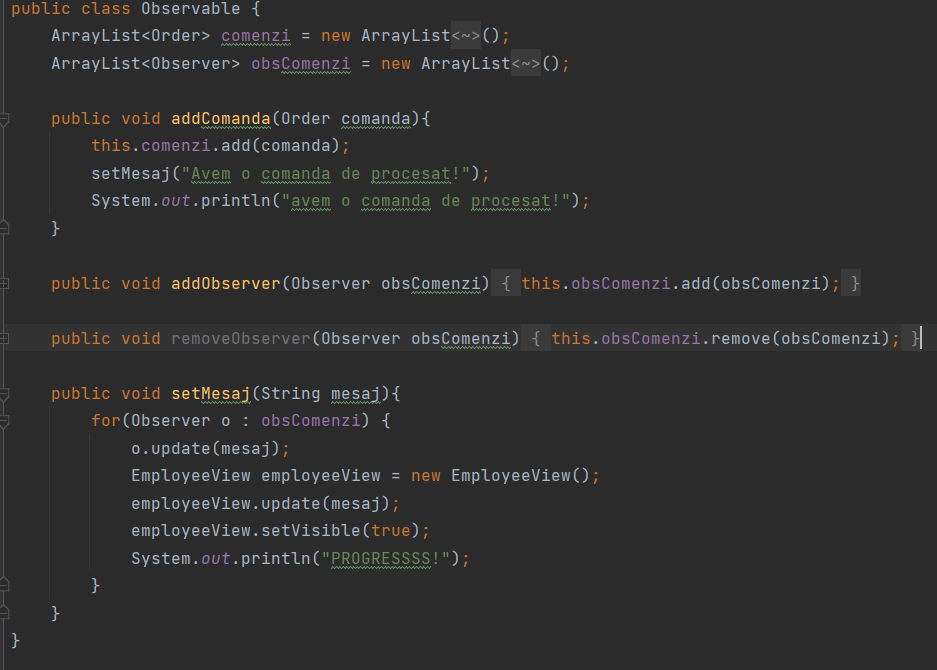
Composite Design Pattern funcționează pe principiul unui arbore binar, ce are noduri interne și frunze. Composite Design Pattern are următoarele componente:

* Componentă – Componentă declară interfața pentru obiectele din compoziție și pentru accesarea și gestionarea componentelor sale fii. De asemenea, implementează comportamentul implicit pentru interfața comună tuturor claselor, după caz.
* Leaf – Leaf definește comportamentul obiectelor primitive din compoziție. Reprezintă obiecte frunze în compoziție.
* Compozit – Compozit stochează componente copil și implementează operațiuni legate de copil în interfața componentelor.
* Client – Clientul manipulează obiectele din compoziție prin interfața componentelor.

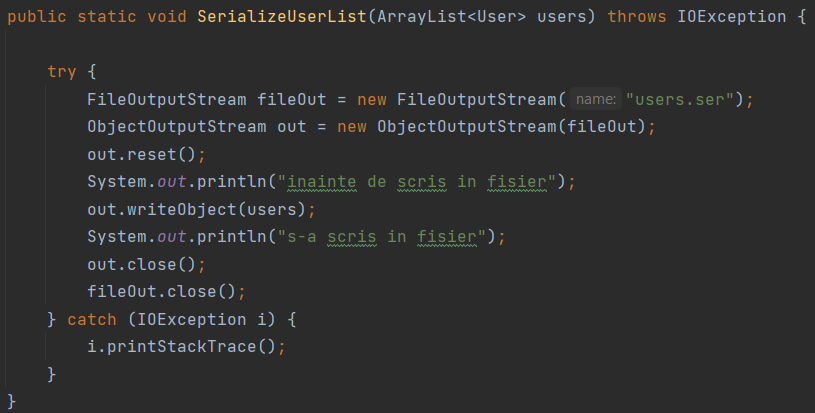
Am implementat acest design printr-o clasă abstractă MenuItem ce are ca și copii clasele BaseProduct și CompositeProduct.

Observer Design Pattern este un model de proiectare software în care un obiect, numit subiect, menține o listă a dependenților săi, numiți observatori și îi notifică automat despre orice schimbare de stare, de obicei apelând una dintre metodele lor.

Acest design pattern s-a construit în jurul interfeței grafice EmployeeView și a clasei Employee. La fiecare comandă efectuată de un client, un angajat este notificat. Interfața Observer este un element al Observer Design Pattern și este implementată de clasa EmployeeView. De asemenea, trebuie creată și clasa Observable, ce are următoarele metode, alături de implementarea lor:



Interfața Serializable ajută la serializarea obiectelor, adică la păstrarea datelor de la o rulare la alta. Pentru extragerea lor, vom folosi reversul numit deserializare. Toate clasele pe care dorim să le serializăm-deserializăm trebuie să implementeze această interfață.

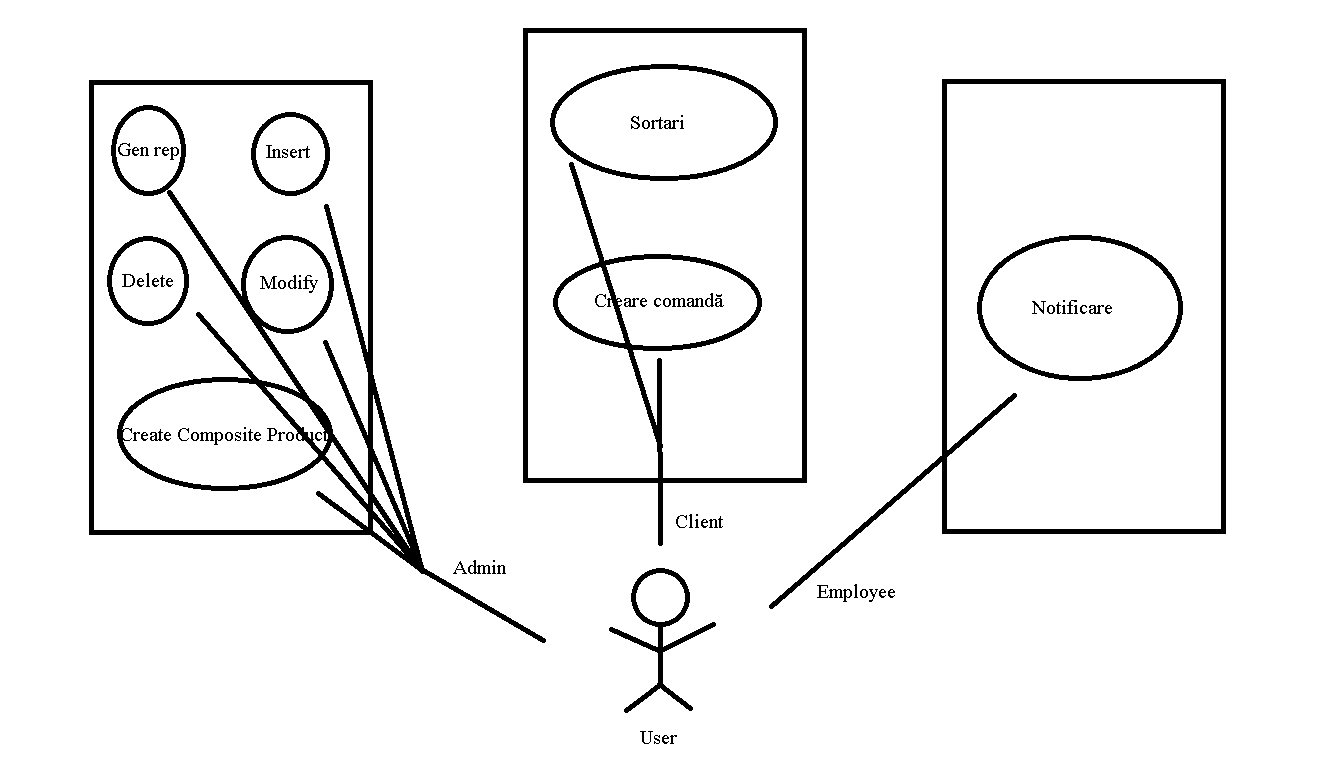


Pe lângă implementarea interfeței, aceasta mai trebuie să implementeze și o scriere in fișiere de tip .ser.

HashMap este o Java Collection care pe baza unei funcții de hashCode() realizează punerea într-un tabel de ArrayList-uri sau LinkedList-uri (practic, de List Collection). Funcția care realizează adăugarea într-un HashMap este put(). Noi am ales să o folosim atunci când ne dorim să stocăm o comandă.

Interfața de register și login foloseste la baza atributul role al clasei-mama User, deoarece putem să facem register doar la Client, iar Login-ul ne deschide automat pagina specifică tipului de utilizator.

Un mod simplu și usor de a vizualiza ceea ce e capabil să facă un order food delivery management system este printr-o diagramă Use Case:

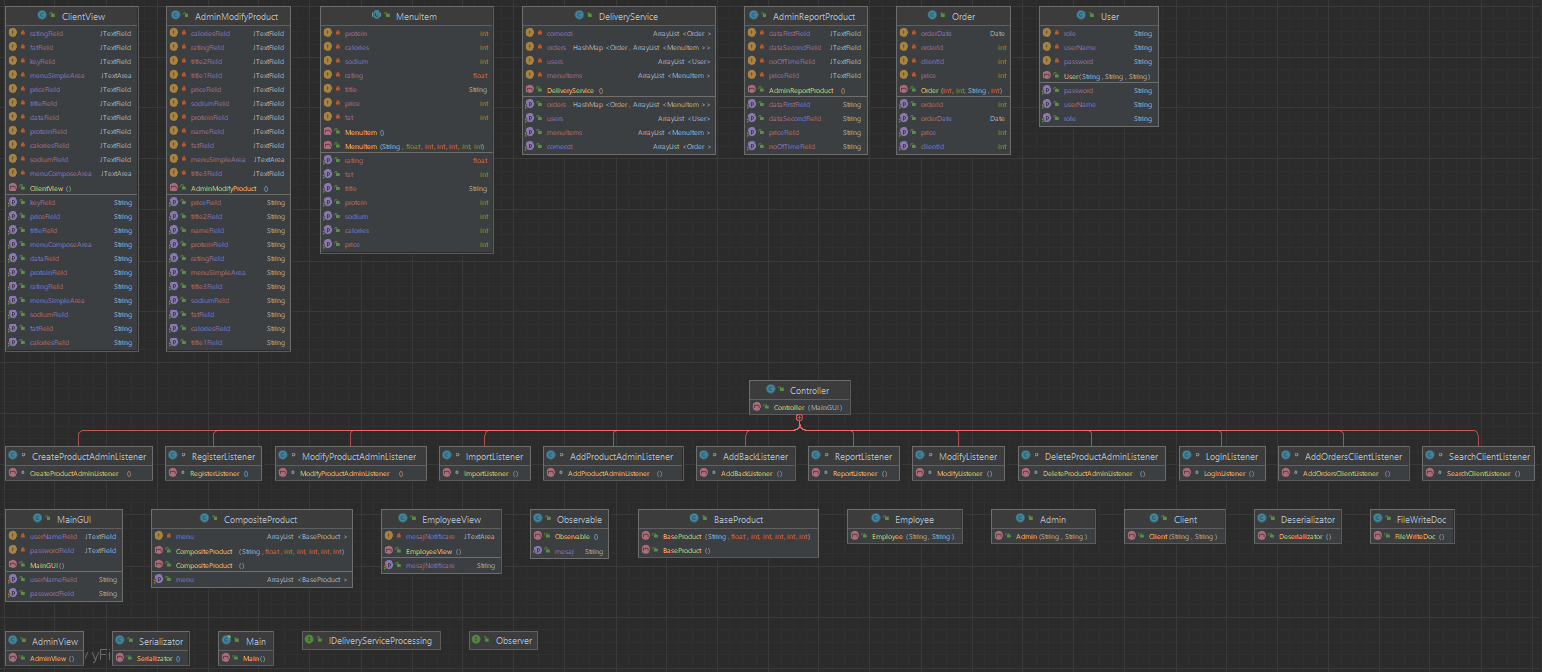


1. IMPLEMENTARE:

* DIAGRAMA UML:

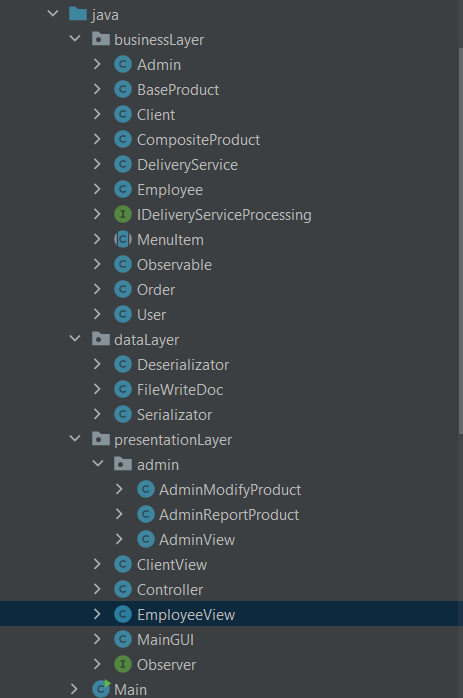
Unified Modeling Language (UML) este un limbaj standard pentru descrierea de modele si specificatii pentru software. UML a fost dezvoltatla început pentru reprezentarea complexității programelor care erau contruite utilizând programarea orientată pe obiect, al căror fundament este structurarea programelor pe clase și instanțele acestora (obiecte).

Am utilizat UML-ul pentru a vedea mult mai clar împărțirea pe clase a codului, precum și variabilele instanță și metodele pe care le-am utilizat în dezvoltarea order food delivery management system.



* CLASE ȘI METODE:

Proiectul este împărțit în 3 mari pachete: Presentation Layer – unde are loc scrierea claselor specifice interfeței grafice cu utilizatorul, Business Layer unde se desfășoară intreaga logică a proiectului, iar în Data Layer sunt părțile de scriere în fișier.



Clasa Main controlează tot programul, deoarece în ea instranțiem un obiect de tip MainGUI și un obiect de tip Controller, care are ca și parametru în constructor, un obiect de tip MainGUI.

public class Main {  
 public static void main(String[] argv){  
 MainGUI gui = new MainGUI();  
  
 Controller controller = new Controller(gui);  
 gui.setVisible(true);  
 }  
}

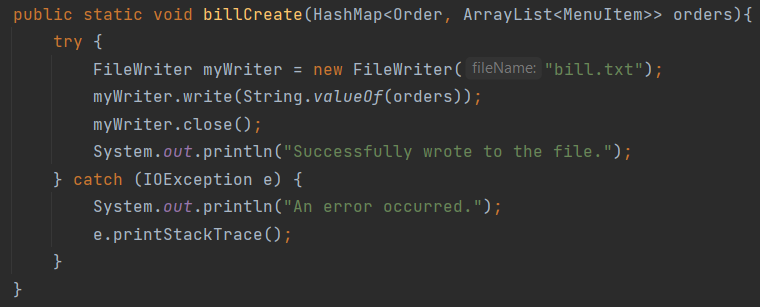
Pachetul DataLayer:

Acest pachet conține clasele Serializator, Deserializator și FileWriteDoc.

Clasa Serializator conține metodele SerializeUserList(), SerializeMenuItem(), SerializeOrders() și SerializeComenzi(). Toate realizează aproximativ același lucru, diferența fiind de tipul de ArrayList pe care îl serializează.

Clasa Deserializator conține metodele DeserializeUserList(), DeserializeMenuItem(), DeserializeOrders() și DeserializeComenzi(). Toate realizează aproximativ același lucru, diferența fiind de tipul de ArrayList pe care îl citește din fișierul .ser și pe care îl deserializează.

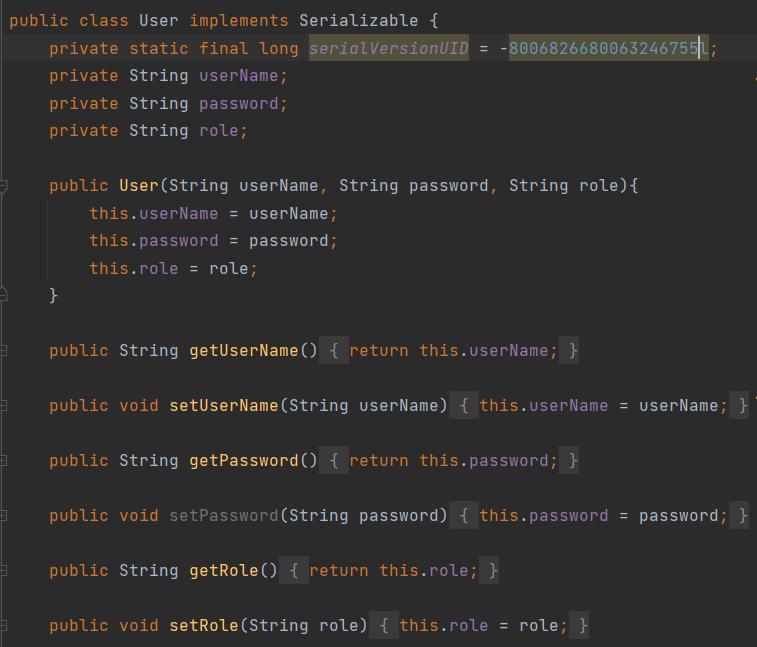
Clasa FileWriteDoc este folosită pentru scrierea bill.txt și a celor 4 rapoarte. Implementează următoarele 5 metode billCreate(), report1Create(), report2Create(), report3Create() și report4Create().

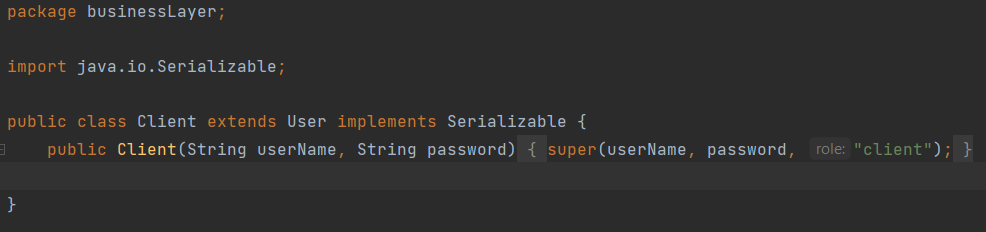


Pachetul BusinessLayer:

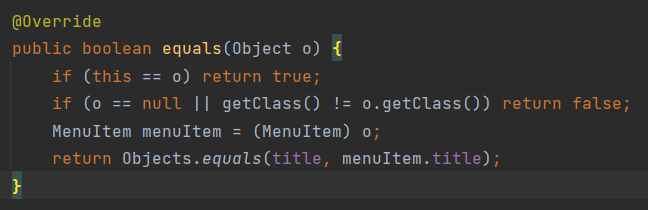
Acest pachet este cel mai amplu și, așa cum menționam mai sus, conține mare parte din logica și împărțirea pe clase a proiectului. În acest mod am ajuns să avem în acest minunat pachet 11 clase și interfețe.

Vom începe discuția prin clasa părinte User. Aceasta are ca și mari atribute username, password și role. Este moștenită de clasele Admin, Employee și Client. Atunci când are loc apelul constructorului super() pentru fiecare clasă în parte, are loc și setarea automată a atributului role. Astfel, în clasa Admin va prelua stringul ”admin”, în clasa Employee, stringul ”employee” și în clasa Client, stringul ”client”. A nu se uita de menționat faptul ca toate aceste clase moștenesc getterele și setterele din clasa User.

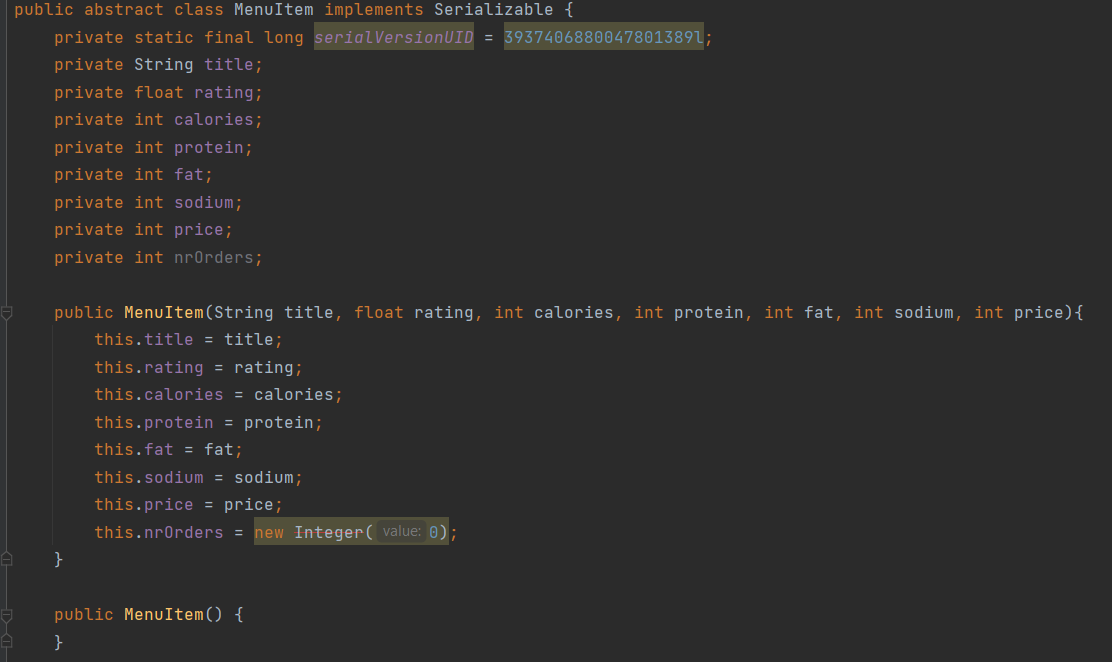


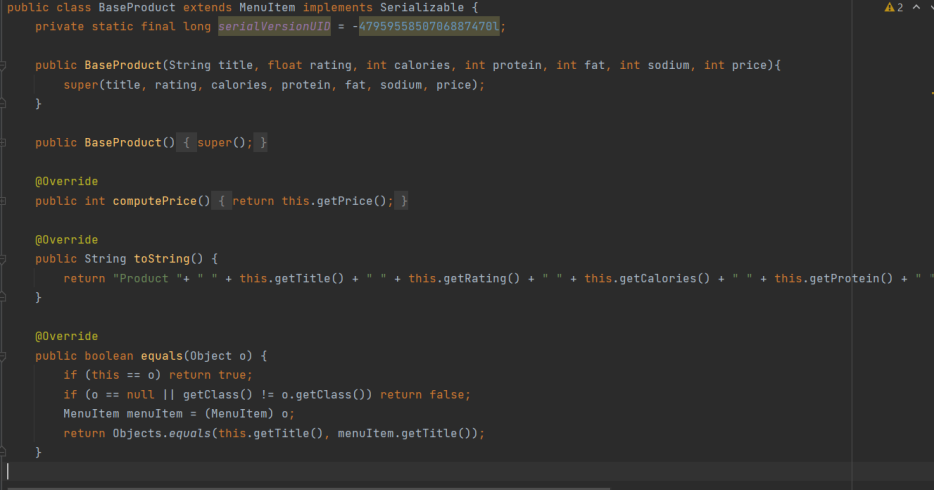


Clasa MenuItem, împreună cu clasele BaseProduct și CompositeProduct ilustrează implementarea design patternului Composite. Clasa MenuItem este o clasă abstract, ce conține toate metodele abstracte, dar și toate atributele unui MenuItem. Dintre acestea voi enumera numai price, fat, calories și title. BaseProduct moștenește MenuItem, cu toate metodele înafară de equals() și toString() pe care le suprascrie.



Mai jos se află o poză a modului în care arată clasa MenuItem:



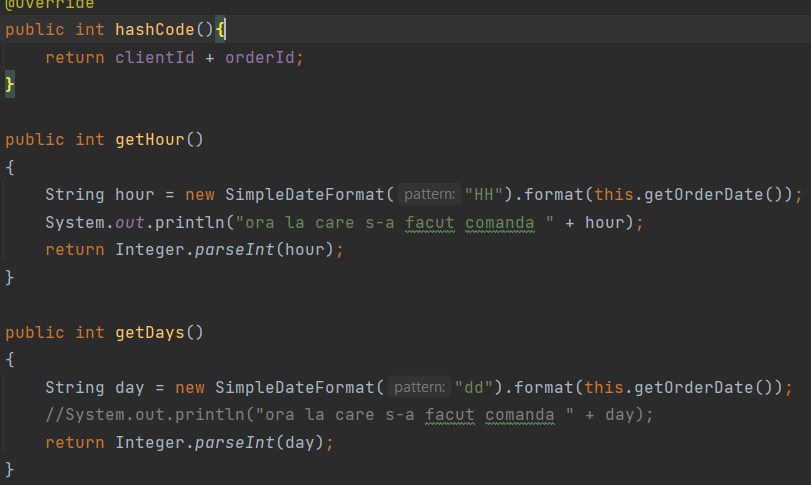


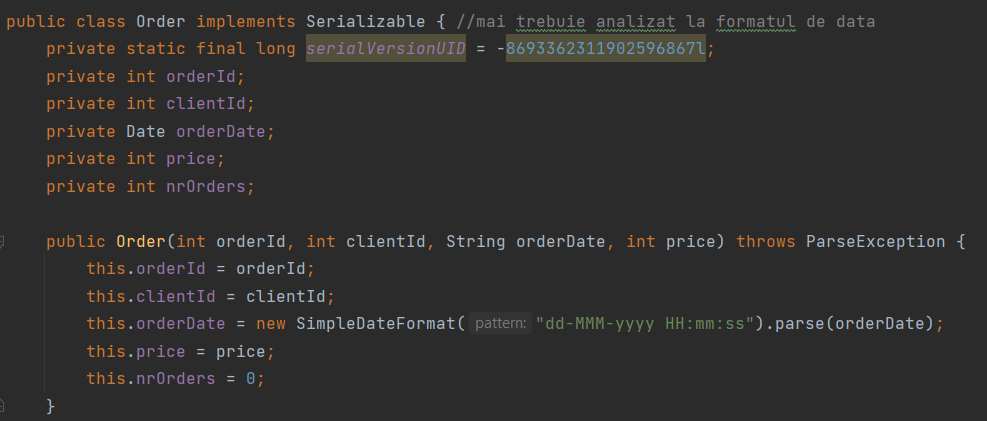
Clasa CompositeProduct conține un ArrayList de BaseProduct și multe metode prin care compun diferiții parametri.

@Override  
public int computePrice() {  
 int result = 0;  
 for(BaseProduct p: menu)  
 {  
 result += p.getPrice();  
 }  
 return result;  
}

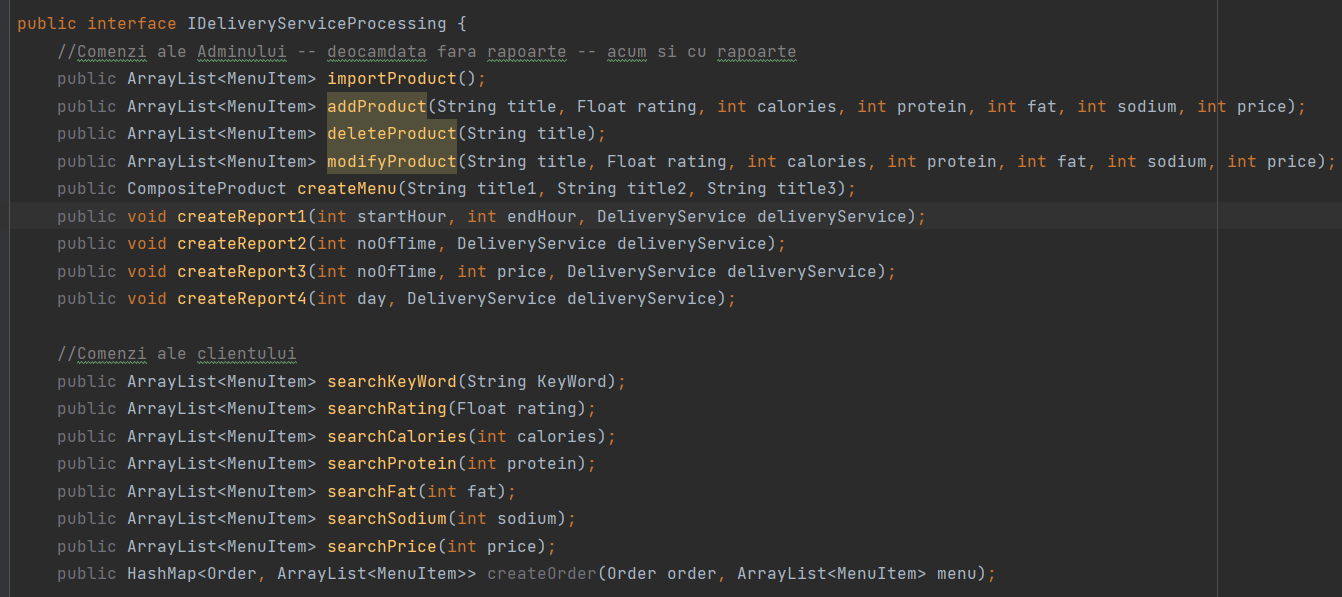
A se specifica că principala metodă la care se face @Override este computePrice(), deoarece prețul este suma fiecărui produs din acel ArrayList de BaseProduct.

Clasa Order conține atributele orderId, clientId si orderDate. Aceasta din urmă este de tip Date, având un format specific, din care noi, prin funcțiile getDays() și getHour(), putem obține ziua și ora. Am suprascris funcția hashCode(), pentru a fi siguri că se calculează corect hashCode-ul. orderId si clientId se calculează în Controller atunci când se realizează o comandă nouă.



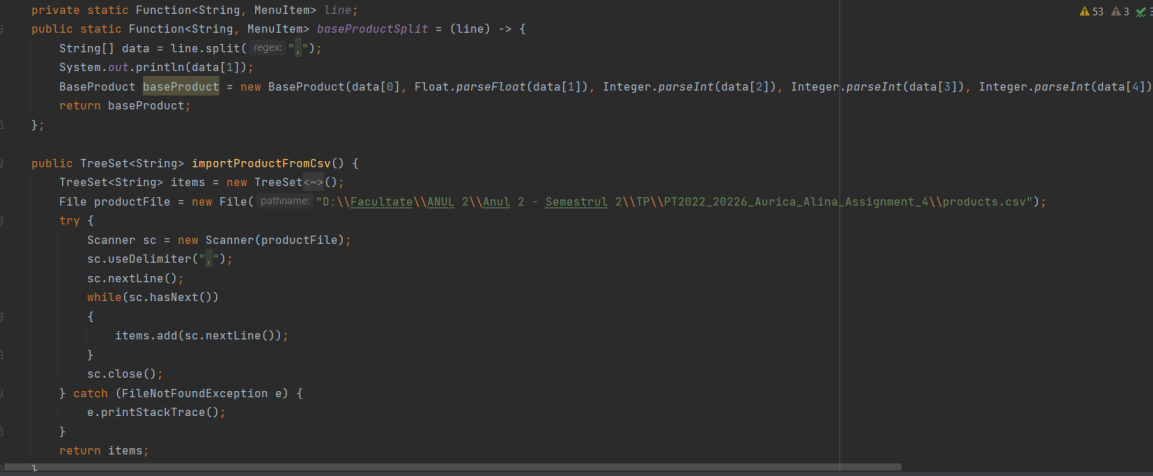


Clasa DeliveryService și interfața IdeliveryServiceProcessing sunt în strânsă legătură, deoacere prima o implementează pe a doua. În interfață avem anunțate metodele pe care trebuie să le realizeze admin-ul și client-ul. Implementarea acestora se găsește în DeliveryService.

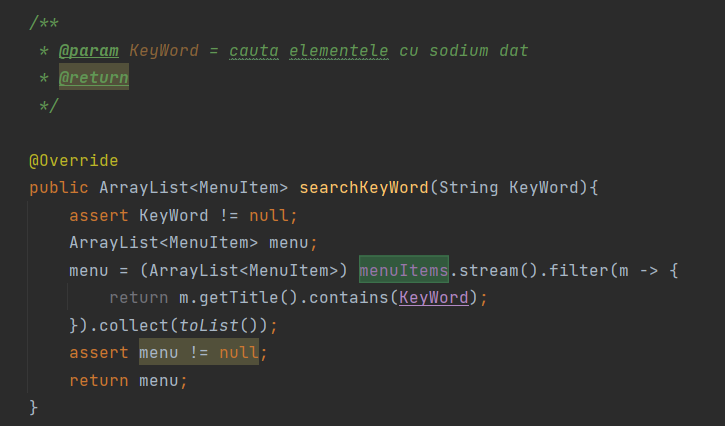


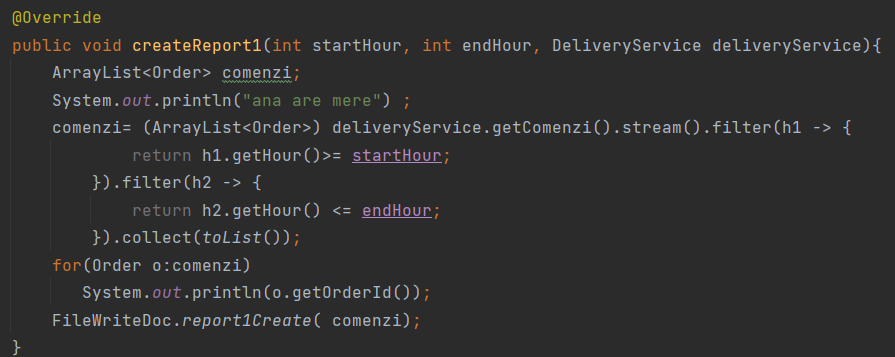
Printre cele mai interesante metode ale proiectului se află importCSV(), unde, cu ajutorul lambda expression-urilor reușim să extragem conținutul unui fișier de tip .csv, să eliminăm duplicatele și de-abia după aceea să facem split în funcție de ”,” pentru a crea elemente de tip BaseProduct.

@Override  
public ArrayList<MenuItem> importProduct() {  
 invariant();  
 DeliveryService delivery = new DeliveryService();  
 menuItems = (ArrayList<MenuItem>) delivery.importProductFromCsv().stream().map(*baseProductSplit*).collect(*toList*());  
 return menuItems;  
}



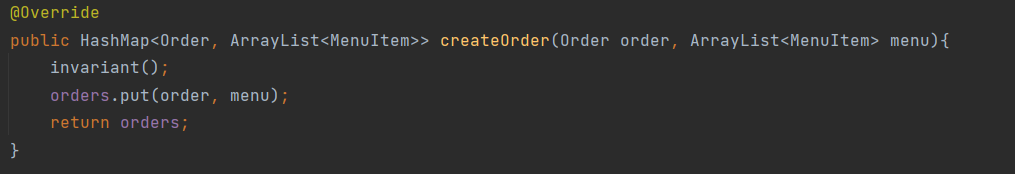
Există numeroase metode care folosesc lambda expression în această clasă, precum cele 4 metode de createReport() și cele de filtrare după un atribut.



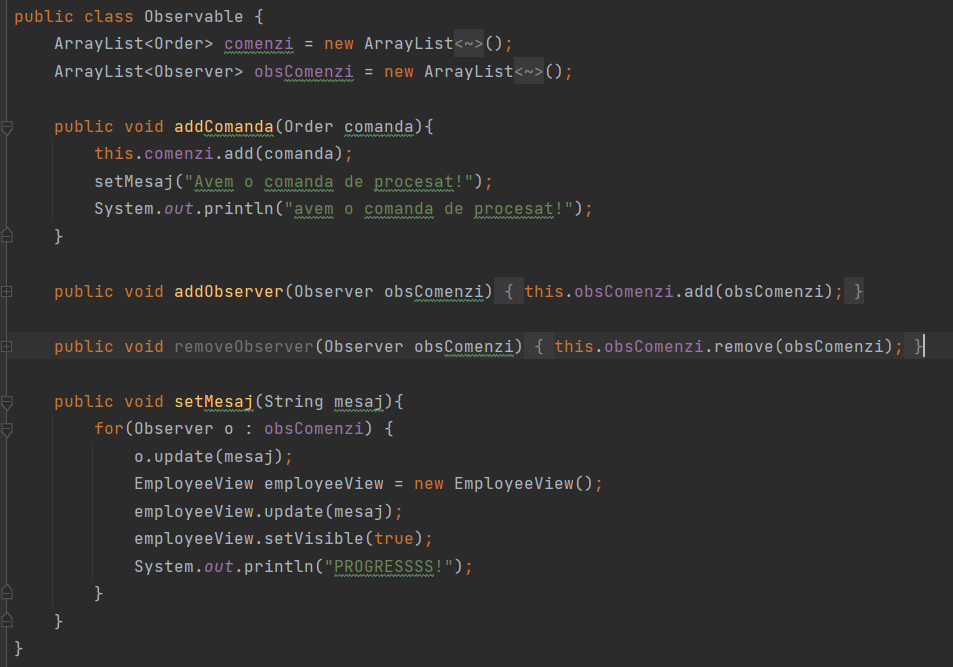


Lambda expression au fost adăugate în Java 8. O expresie lambda este un bloc scurt de cod care preia parametri și returnează o valoare. Expresiile Lambda sunt similare cu metodele, dar nu au nevoie de un nume și pot fi implementate chiar în corpul unei metode. Principalele metode pe care le-am folosit noi sunt map(), filter() și stream().

Tot în clasa DeliveryService avem implementată și metoda care creează un Order, numită createOrder():



Clasa Observable este clasa cu ajutorul căreia încercăm să notificăm un angajat. Face parte din Observer Desing Pattern.



Metoda addComanda() are următoarea logică: De fiecare dată când introducem în comenzi o comandă, activăm și setMesaj() care să-mi seteze un mesaj ce ar trebui să apară în interiorul ferestrei de la Employee.

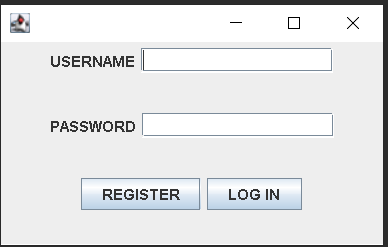
Utilizăm și metodele addObserver() și removeObserver() pentru a adăuga/șterge un Obiect de tip Observer.

* GUI:

Interfața grafică cu utilizatorul este creată din clasele din pachetul presentationLayer.

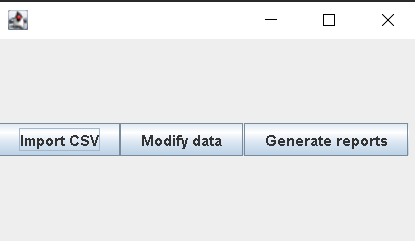
Pachetul conține 7 clase și o interfață, dintre care clasele clasele care formează întreaga interfață grafică a adminului sunt plasate în pachetul admin. Legătura dintre Main și MainGUI se realizează prin intermediul clasei Controller.

Clasa principală, MainGUI este, de fapt, fereastra de Register și Login.

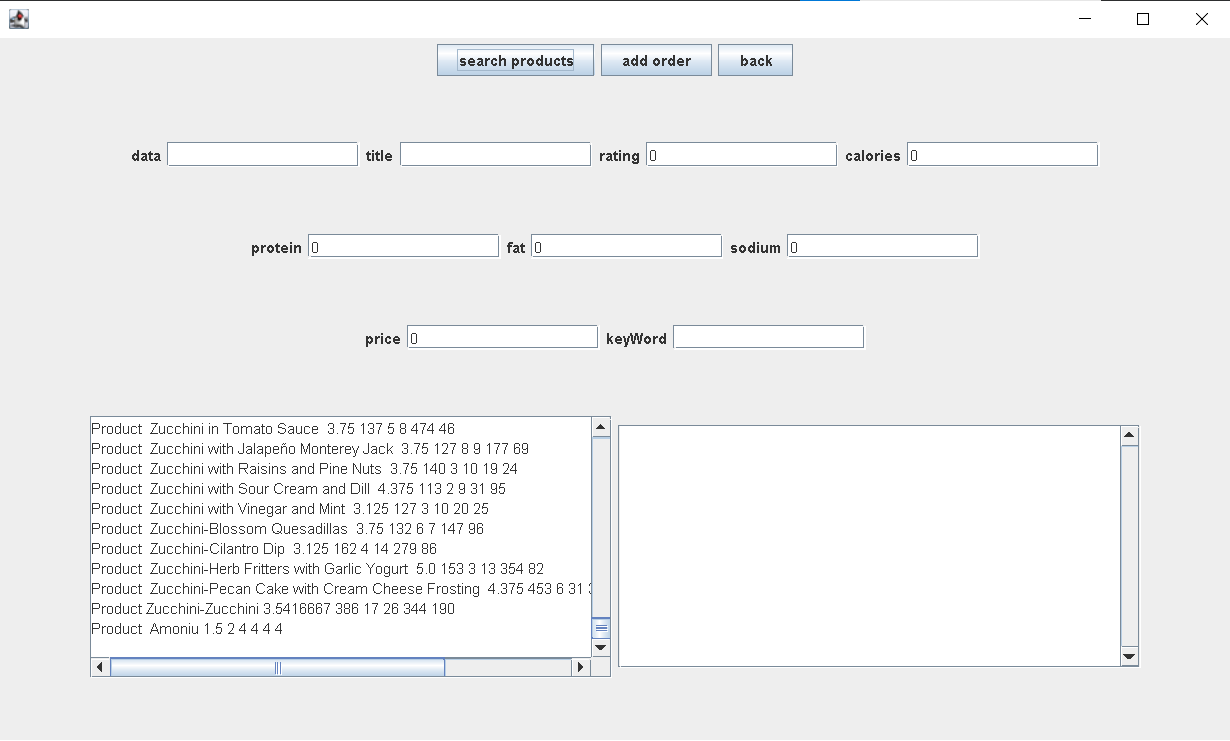


Prin introducerea unui UserName și a unei parole, utilizatorul poate realiza un register sau un log in. În funcție de tipul de utilizator se vor deschide ferestre diferite. Funcționalitatea celor două butoane este realizată prin ActionListeneri (prin apăsarea lor).

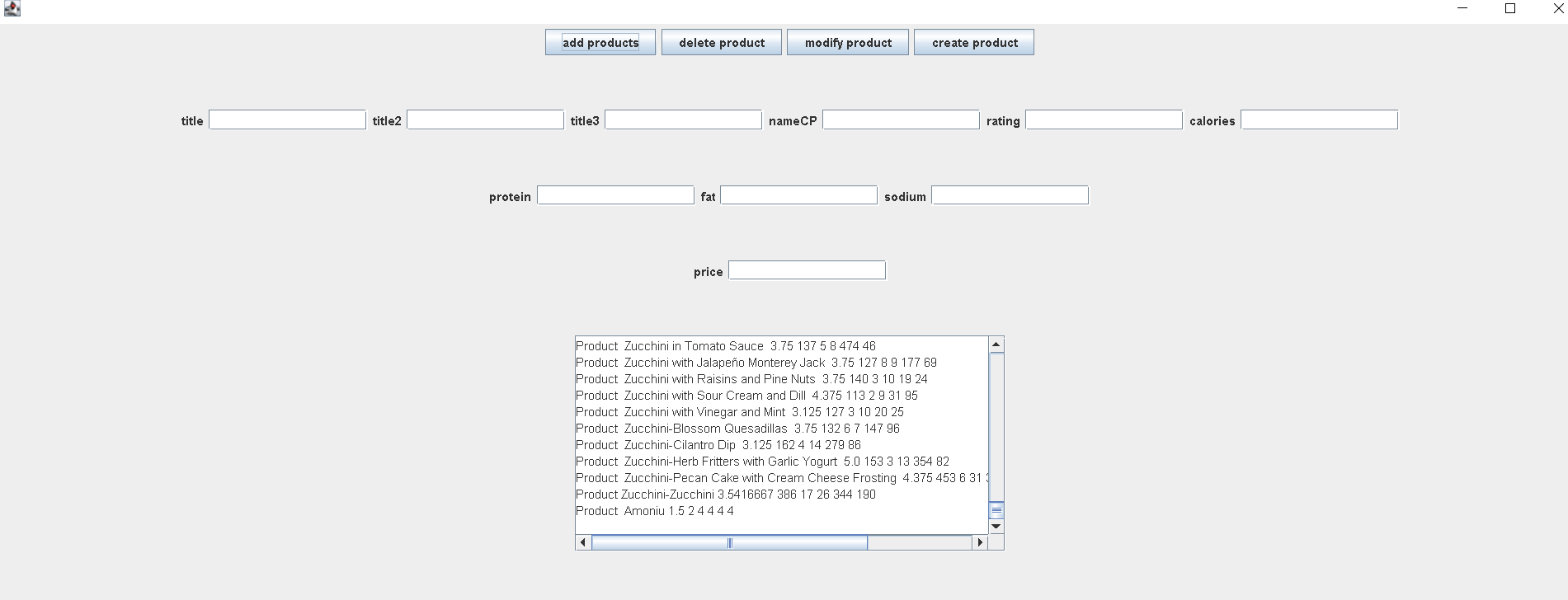
Aceasta este principala fereastră care se deschide pentru admin:



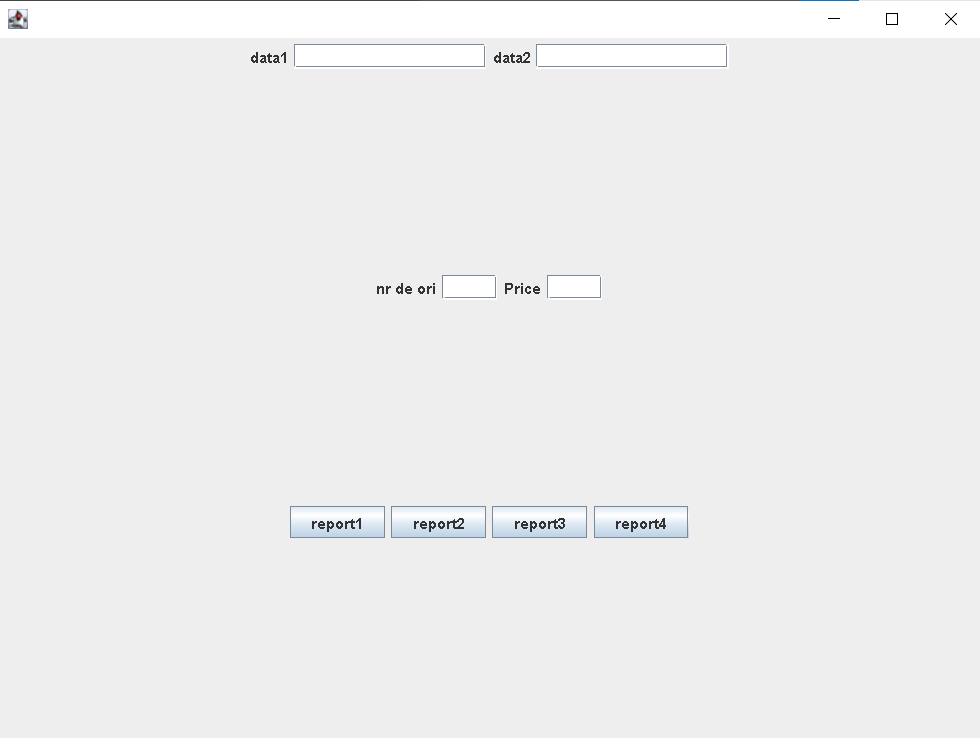
Aceasta este fereastra pentru un Client:



În momentul în care se deschide fereastra Admin-ului, acesta poate face import .csv prin apăsarea butonului respectiv, poate să modifice date, intrând, prin apăsarea butonului Modify data, în interfața de modificare de date sau poate să genereze rapoarte, prin apăsarea butonului Generate Raports, care îl duce în fereastra specifică acțiunii întreprinse.



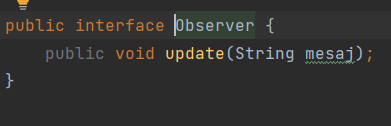
Adminul poate adăuga produse nou, poate șterge, poate modifica și poate crea dailyMenu, doar printr-o simplă apăsare de buton. De asemenea, el poate vizualiza lista de produse printr-un JtextArea.



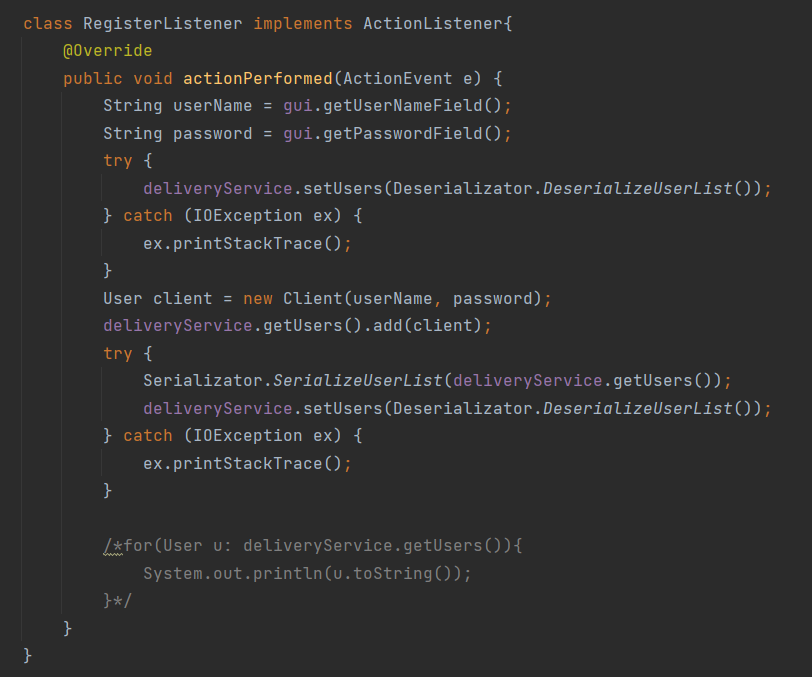
Cea de mai sus este fereastra pentru generarea de rapoarte. Prin introducerea datelor corespunzătoare și prin apăsarea butonului corespunzător, se generează un raport.

ClientView-ul are aproximativ aceeași structura ca și si cel de la admin pentru modificarea datelor, numai că cel dintâi poate să obțină o listă a produselor cu un anumit criteriu sau poate realiza o comandă.

Fereastra de Employee conține doar un JtextArea unde primește mesajul de procesare a comenzii. Clasa EmployeeView implementează Observer, interfață prin a cărei metoda notificăm angajul.

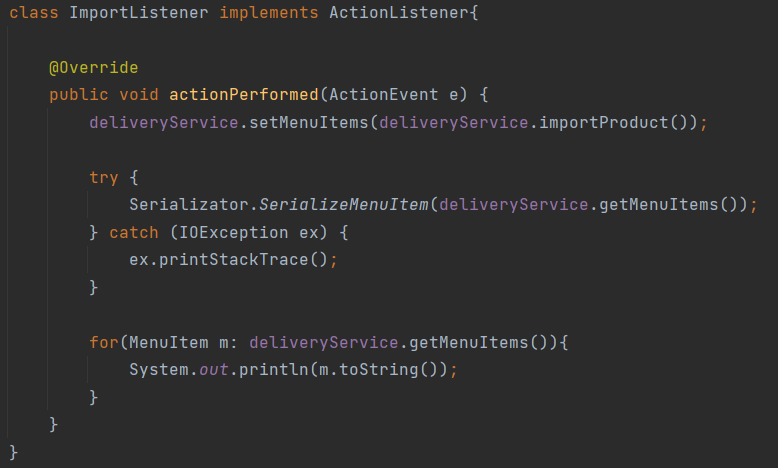


Clasa Controller este de departe cea mai complexă clasă a întregului proiect, cuprinzând toate clasele interne necesare butoanelor. Toate clasele interne implementează ActionListener, cu metoda aferentă acesteia, actionPerformed().



Clasa RegisterListener: Din interfața grafică se preiau userName-ul și parola, cu acestea se creează un Client nou, pe care îl adăugăm în lista de users. Vom folosi serializarea și deserializarea pentru păstrarea datelor.

Clasa LogIn: Din interfață preluam datele, parcurgem ArrayList-ul de users și atunci cand credențialele corespund, verificăm rolul pentru a ști ce fereastră trebuie să deschidem.

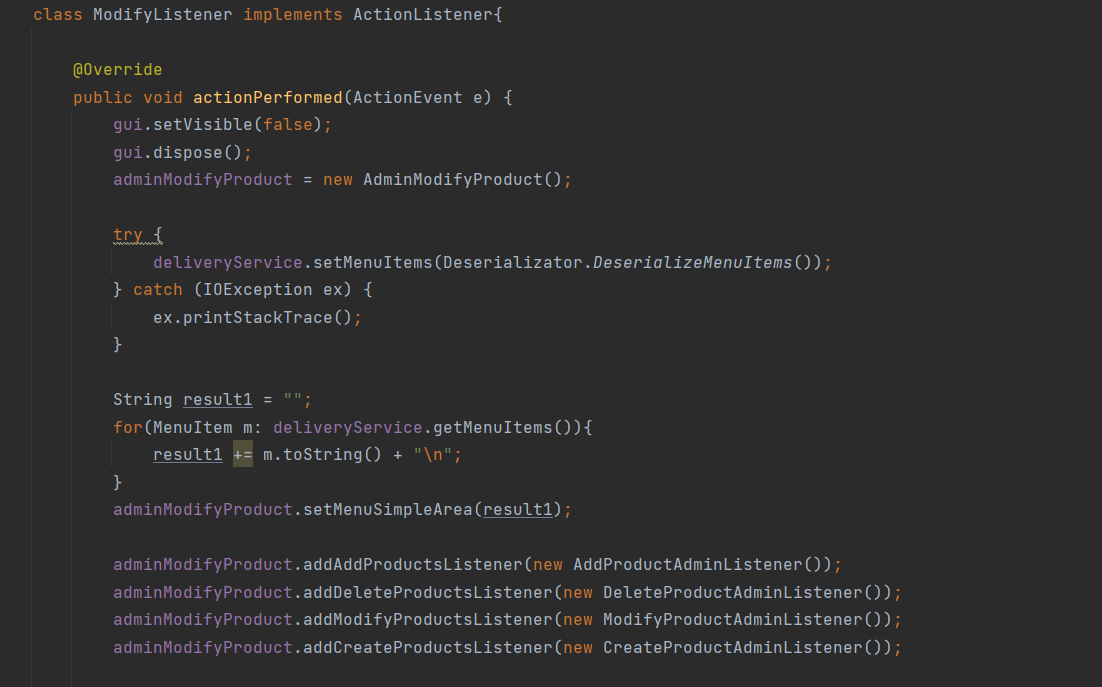


Clasa ImportListener: Pur și simplu se apelează metoda importProduct din clasa DeliveryService.



Clasele AddProductAdminListener, DeleteProductAdminListener, ModifyProductAdminListener au implementări similare, adică implementează funcționalitățile de modificare de date specifice adminului.

Clasa pentru care cele 3 clase antemenționate sunt folosite, este clasa ModifyListener.



Ea doar preia datele din menuItem ce a fost deserializat înainte și le adaugă în JtextArea. De asemenea, instanțiază butoanele necesare fiecărei comenzi.

Mai sunt multe alte clase pe lângă acestea a căror implementare se găsește în cod. Toate mergând pe exact aceeași idee.

1. TESTARE:

Testarea se realizează prin intermediul interfeței grafice cu utilizatorul (GUI). Practic introducem datele, apăsăm butoanele și vedem ce se întâmplă.

1. CONCLUZII:

Proiectul necesită multe îmbunătățiri, de la crearea posibilității de a comanda mai mult de 4 produse, până la implementarea mai bună a serializării și deserializării, alături de faptul că rapoartele ar putea funcționa mult mai bine.

Concluzia acestui proiect pe care tocmai l-am realizat este că un astfel de dispozitiv ușurează foarte mult viața oamenilor care utilizează magazinele online.

În ceea ce mă privește, proiectul mi-a oferit oportunitatea de a-mi dezvolta lucrul cu limbajul Java, cu paradigma programării orientate pe obiect. Mi-a oferit oportunitatea de a crea un proiect complex, ce utilizează numeroase librării, precum: java.lang.observer, javax.swing și java.awt. De asemenea, m-a ajutat să-mi reamintesc tehnicile de programare învățate semestrul trecut.

Nu în ultimul rând, aș vrea să menționez că proiectul este realmente unul foarte complex, care conține numeroase noi tehnice de programare, printre care: Composite Design Pattern, Observer Design Pattern, lambda expression, serializare, deserializare și lucrul cu HashMap-uri.

1. BIBLIOGRAFIE:

<https://stackoverflow.com/>

<https://www.geeksforgeeks.org/>

<https://dsrl.eu/courses/pt/materials/A4_Support_Presentation.pdf>