

Food delivery management system

Documentație

Nume: Moș Robert

Grupa: 30228

Cuprins

1. Obiectivul problemei
2. Analiza problemei
   1. Diagrama Use Case
3. Proiectarea problemei
   1. Diagrama de clase UML
4. Implementarea problemei
   1. Clase
   2. Metode
5. Rezultatele problemei
6. Concluzii
7. Bibliografie
8. Obiectivul problemei

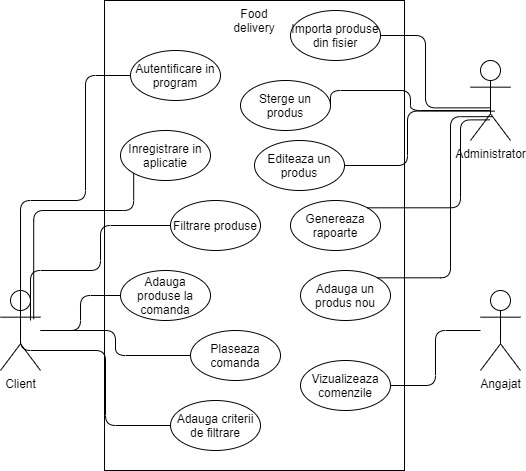
Obiectivul acestei teme este realizarea unui program în Java pentru food delivery. Cerinta problemei era să respecte organizarea pe layere, fiind dată o diagramă de clasă cu fiecare clasă corespunzătoare unui layer. Pentru această problemă trebuia implementată o interfață grafică cu un sistem de login unde clientul poate intra în contul său pentru a comanda anumite produse, si un sistem de inregistrare unde clientul iși poate face un cont in platforma respectivă. Clientul poate să vadă lista de produse care pot fi comandate, poate căuta un produs după un anumit cuvant cheie, dupa numarul de calorii, grasimi, sodiu sau proteine. Clientul poate face o comandă cu produsele selectate și acesta va primi o factură cu ora, data și pretul total al facturii. De asemenea administratorul se poate si el loga în cont si poate face următoarele operații: Poate importa produse in meniu dintr-un file .csv, poate adauga, sterge sau creea noi produse si poate să genereze un raport pe baza comenzilor în funcție de un anumit timp în care s-au efectuat comenzi sau in functie de un numar specificat de produse comandate sau produsele comandate intr-o zi specificată de cate ori au fost comandate. Pe de altă parte, angajatul va fi notificat de fiecare dată când se face o nouă comandă.

1. Analiza problemei

Problema a fost rezolvată în mare parte după modelul cu diagrama de clase din materialul suport. Programul este împărțit în mai multe pachete: DataLayer, Presentation, BusinessLayer pentru o vizualizare mai clară a programului. Cu ajutorul programului se pot creea conturi de client si administrator, fiecare dintre aceștia pot intra în conturile proprii unde pot să facă anumite operații în cazul clientului de comandare a produselor și de căutare a acestora, iar în cazul administratorului de adăugare de produse, editare, stergere și de generare a unor rapoarte după numarul de produse comandate sau după produsele comandate întru-un interval orar.

* 1. Diagrama Use Case

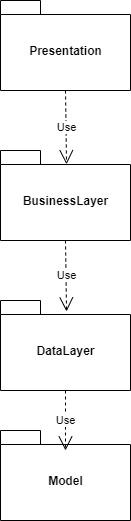
În această diagramă sunt prezentate toate operațiile care se pot face in programul implementat, de creare cont, intrare în cont, adăugare produse, ștergere produse, editare produse, creare de rapoarte, plasare de comandă, filtrare produse, importare de produse, vizualizare comanda.



1. Proiectarea problemei

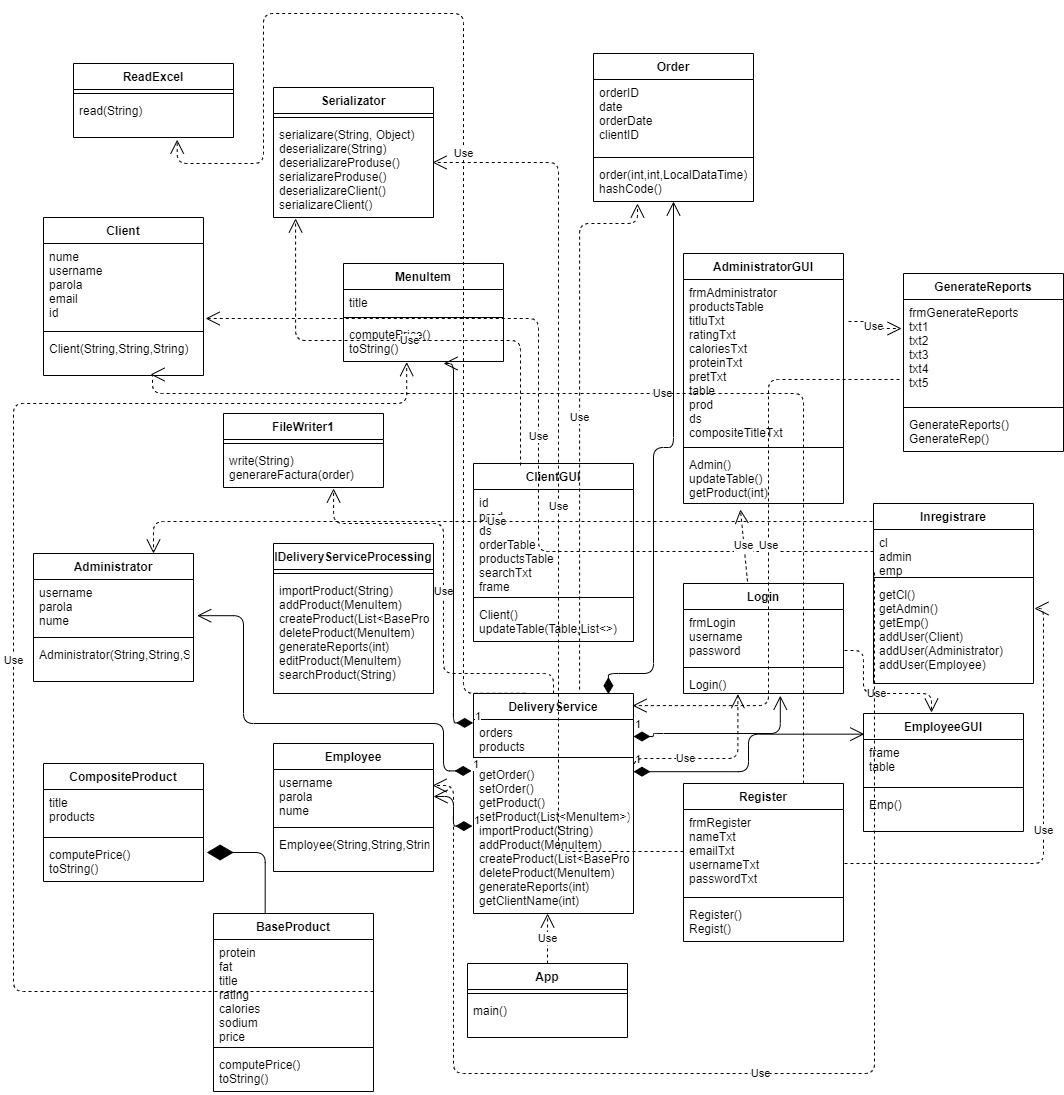
Pachetele Java ajută la organizarea mai multor module și grupează clase și interfețe. Eu pentru această problemă fiind una foarte complexă am respectat diagrama de clase pe care am primit-o în materialul suport și organizarea lor în pachete. Acestea fiind spuse, am organizat clasele în 3 pachete numite DataLayer care conține clasele, BusinessLayer care contine clasele, Presentation care conține clasele. Organizarea claselor în pachete este foarte utilă și practică pentru că oferă un aspect mai ordonat și ajută să înțelegem mai bine despre ce este vorba în programul respectiv.

Diagrama de pachete



* 1. Diagrama de clase UML

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificatii pentru software. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora ( numite și obiecte ). Diagrama UML ajută foarte mult la implementarea unui program pentru că iti poți face o idee mai clară la cum va arăta programul tău si va fi mai ușor de implementat.



1. Implementarea problemei
   1. Clase

Pentru rezolvarea acestei teme a fost nevoie de folosirea mai multor clase fiind o problema mai complexa care conține și o interfață.

Clasa BaseProduct: Contine toate caracteristicile unui produs cum ar fi numele produsului, ratingul, caloriile, proteinele acestuia, grasimile pe care le are acesta, cantitatea de sodiu și prețul. Această clasă mai conține gettere și settere si o metodă toString care ajută la afișarea produsului.

Clasa CompositeProduct: Această clasă extinde clasa MenuItem si implementează Serializable, în această clasă găsim un String care contine numele produsului compus și o listă de BaseProduct, gettere și settere și o metodă toString care afișează elementele compuse.

Clasa DeliveryService: Contine operatiile care se fac pe produse cum ar fi importarea de produse, adaugarea de produse, editarea de produse, crearea de produse, stergerea unor produse, generarea unor rapoarte specifice dupa anumite criterii, creare de comenzi si cautarea unui produs anume.

IdeliveryServiceProcessing: Contine operatiile care se fac pe produse cum ar fi importarea de produse, adaugarea de produse, editarea de produse, crearea de produse, stergerea unor produse, generarea unor rapoarte specifice dupa anumite criterii, creare de comenzi si cautarea unui produs anume.

Clasa MenuItem: Este o clasa abstracta care implementeaza Serializable.

Clasa Order: Această clasă conține caracteristicile unei comenzi cum ar fi orderid, id-ul unui client, data la care s-a efectuat comanda, mai conține și settere, gettere si o metoda hashCode.

Clasa FileWriter1: Contine 2 metode, una care scrie un mesaj intr-un fișier iar cealaltă generează o factură pentru a anumită comandă creată de către client.

Clasa ReadExcel: Această clasă citește dintr-un fișier de tip Excel produsele care se află în meniu, cu caracteristicile fiecăruia în parte.

Clasa Serializator: În această clasă se face serializarea și deserializarea pentru Produse, pentru conturile creeate de către clienți și pentru comenzile efectuate de către aceștia.

Clasa Administrator: În această clasă sunt date caracteristicile administratorului, de asemenea mai contine si gettere si setter.

Clasa Client: În această clasă se găsesc datele care sunt necesare pentru creearea contului de client in aplicatie, cum ar fi id, nume, email, username, parolă.

Clasa Înregistrare: Conține câte o listă de Clienti, Administratori si angajati. De asemenea mai contine si metode pentru adaugarea unui nou client, angajat sau administrator si metode pentru a verifica dacă la logare contul introdus este de client, administrator sau de angajat.

Clasa AdministratorGUI: Este interfata pentru administrator unde se pot vedea toate operatiile pe care acesta le poate face prin apasarea unor butoane cum ar fi de adaugare de produse, de importare, de editare, de stergere, de crea un produs compus, de a genera anumite rapoarte dupa anumite criterii.

Clasa ClientGUI: Este interfata pentru client unde se pot vedea toate operatiile pe care clientul le poate face prin apasarea butoanelor, acesta poate sa caute dupa un cuvant cheie un produs, poate vedea toate caracteristicile produsului si il poate adauga in cos iar apoi poate efectua de la buton o comanda pentru care se va efectua si o factura.

Clasa GenerateReports: Este clasa in care a fost implementata interfata pentru raporturi, care se fac in clasa administrator. Aceasta clasa contine niste text fielduri in care in functie de criteriu se introduc date si apoi tot in functie de criteriu se alege butonul specific criteriului ales pentru generarea raportului.

Clasa Login: Aceasta clasa este prima interfata care apare la pornirea programului, aceasta contine 2 campuri in care pot fi introduse datele unui cont, mai contine si 2 butoane, unul de login cu care se intra in cont dupa ce sunt introduse datele, iar unul de sign in care te va duce la o fereastra noua unde se poate creea un cont nou pentru client.

Clasa Register: Este interfata in care se poate face cont nou pentru client daca se doreste a face o comanda.

* 1. Metode

Metode din clasa CompositeProduct:

public int computePrice() {  
 return products.stream().mapToInt(BaseProduct::getPrice).sum();  
}

In aceasta metoda se calculeaza suma totala a produselor din composite product

Metode din clasa DeliveryService:

public void editProduct(String title, BaseProduct p) {  
 MenuItem m = *products*.stream().filter(p1 -> p1 instanceof BaseProduct)  
 .filter(p1 -> ((BaseProduct) p1).getTitle().equals(title)).collect(Collectors.*toList*()).get(0);  
 BaseProduct pr = (BaseProduct) m;  
 pr.setCalories(p.getCalories());  
 pr.setFat(p.getFat());  
 pr.setPrice(p.getPrice());  
 pr.setProtein(p.getProtein());  
 pr.setSodium(p.getSodium());  
  
}

Aceasta metoda editeaza caracteristicile unui produs dandu-se numele si produsul respectiv ca parametru.

public void deleteProduct(MenuItem p) {  
 String title;  
 if (p instanceof BaseProduct) {  
 title = ((BaseProduct) p).getTitle();  
 MenuItem m = *products*.stream().filter(p1 -> p1 instanceof BaseProduct)  
 .filter(p1 -> ((BaseProduct) p1).getTitle().equals(title)).collect(Collectors.*toList*()).get(0);  
 *products*.remove(m);  
 } else {  
 title = ((CompositeProduct) p).getTitle();  
 MenuItem m = *products*.stream().filter(p1 -> p1 instanceof CompositeProduct)  
 .filter(p1 -> ((CompositeProduct) p1).getTitle().equals(title)).collect(Collectors.*toList*()).get(0);  
 *products*.remove(m);  
 }  
}

Aceasta metoda sterge un produs primit ca parametru din tabelul cu produse folosindu-se de streamuri.

public String generateReports1(int startHour, int endHour) {  
 StringBuilder sb = new StringBuilder();  
 Set<Order> ord = *orders*.keySet().stream().filter(or -> startHour <= or.getOrderDate().getHour() && or.getOrderDate().getHour() <= endHour).collect(Collectors.*toSet*());  
 sb.append("Comenzile din intervalul orar " + startHour + " si " + endHour + ":");  
 int i = 1;  
 for (Order or : ord) {  
 sb.append("\n" + i + ": Order " + or.getOrderID() + "->\n");  
 *orders*.get(or).stream().forEach(p -> sb.append("\t" + p.toString()));  
 i++;  
 }  
 return sb.toString();  
}

Aceasta metoda genereaza un raport care este folosit in interfata de administrator pentru a vedea un raport de comenzi efectuate intre o ora de inceput si una de final. Ca parametii sunt date ora de inceput si ora de sfarsit si se va afisa raportul pentru produsele care au fost comandate in acest interval de timp.

public String generateReports2(int times) {  
 StringBuilder sb = new StringBuilder();  
 List<MenuItem> allprod = new ArrayList<>();  
 Set<Order> ord = *orders*.keySet();  
 Set<MenuItem> distinctProd = new HashSet<>();  
 List<MenuItem> finalProd = new ArrayList<>();  
 for (Order o : ord) {  
 List<MenuItem> list = *orders*.get(o);  
 allprod.addAll(list);  
 distinctProd.addAll(list);  
 }  
 sb.append("Produsele comandate de " + times + " ori:");  
 AtomicInteger i = new AtomicInteger(1);  
 distinctProd.forEach(p -> {  
 int count = (int) allprod.stream().filter(pr -> pr.getTitle().equals(p.getTitle())).count();  
 if (count >= times) finalProd.add(p);  
 i.getAndIncrement();  
 });  
 finalProd.forEach(p -> sb.append("\n" + i + ": " + p.toString()));  
 return sb.toString();  
}

Aceasta metoda genereaza un alt raport care va afisa toate produsele comandate de un numar mai mare de ori decat cel specificat. Aceasta metoda se va folosi tot in interfata administratorului pentru a genera raportul de produse care au fost comandate de un numar mai mare de ori decat cel specificat. Ca parametru al metodei este dat numarul de cate ori au fost comandate.

public List<MenuItem> searchProd(String criteriu, String val) {  
 if (criteriu.equals("Title"))  
 return *products*.stream()  
 .filter(p -> (p instanceof BaseProduct) ? ((BaseProduct) p).getTitle().contains(val) : ((CompositeProduct) p).getTitle().contains(val))  
 .collect(Collectors.*toList*());  
 else if (criteriu.equals("Rating")) {  
 String[] split = val.split("-");  
 double minVal = Double.*parseDouble*(split[0]);  
 double maxVal = Double.*parseDouble*(split[1]);  
 return *products*.stream().filter(p -> p instanceof BaseProduct)  
 .filter(p -> ((BaseProduct) p).getRating() >= minVal && ((BaseProduct) p).getRating() <= maxVal)  
 .collect(Collectors.*toList*());  
 } else {  
 String[] split = val.split("-");  
 int minVal = Integer.*parseInt*(split[0]);  
 int maxVal = Integer.*parseInt*(split[1]);  
 if (criteriu.equals("Calories")) {  
 return *products*.stream().filter(p -> p instanceof BaseProduct)  
 .filter(p -> ((BaseProduct) p).getCalories() >= minVal && ((BaseProduct) p).getCalories() <= maxVal)  
 .collect(Collectors.*toList*());  
 }  
 if (criteriu.equals("Protein")) return *products*.stream().filter(p -> p instanceof BaseProduct)  
 .filter(p -> ((BaseProduct) p).getProtein() >= minVal && ((BaseProduct) p).getProtein() <= maxVal)  
 .collect(Collectors.*toList*());  
 if (criteriu.equals("Fat")) {  
 return *products*.stream().filter(p -> p instanceof BaseProduct)  
 .filter(p -> ((BaseProduct) p).getFat() >= minVal && ((BaseProduct) p).getFat() <= maxVal)  
 .collect(Collectors.*toList*());  
 }  
 if (criteriu.equals("Sodium"))  
 return *products*.stream().filter(p -> p instanceof BaseProduct)  
 .filter(p -> ((BaseProduct) p).getSodium() >= minVal && ((BaseProduct) p).getSodium() <= maxVal)  
 .collect(Collectors.*toList*());  
 if (criteriu.equals("Price"))  
 return *products*.stream().filter(p -> p instanceof BaseProduct)  
 .filter(p -> ((BaseProduct) p).getPrice() >= minVal && ((BaseProduct) p).getPrice() <= maxVal)  
 .collect(Collectors.*toList*());  
 }  
 return null;  
}

Aceasta metoda cauta un produs in functie de caracteristicile lui. Este folosita in interfata client pentru a putea cauta un anumit produs cu ajutorul unui cuvant cheie si prin selectarea numelui, sau poate cauta un produs in functie de un interval care se da pentru rating, calorii, proteine, grasimi, sodiu si pret.

Metode din clasa FileWriter1:

public static void generareFactura(Order ord, List<MenuItem> prod) {  
 try {  
 File fileOutput = new File("factura\_" + ord.getOrderID() + ".txt");  
 FileWriter write = new FileWriter(fileOutput);  
 PrintWriter write1 = new PrintWriter(write);  
  
 String mesaj = "Comanda nr." + ord.getOrderID() + "\n" +  
 "Data: " + ord.getOrderDate() + "\n" +  
 "Produse:\n";  
 StringBuilder sb = new StringBuilder();  
 AtomicInteger i = new AtomicInteger(1);  
 prod.forEach(p -> sb.append((i.getAndIncrement()) + ": " + p.getTitle() +"\n"));  
 int price = prod.stream().mapToInt(MenuItem::computePrice).sum();  
 String msg = mesaj + sb.toString() + "\n" + "Pret total: " + price;  
 write1.println(msg);  
 write1.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}

Aceasta metoda genereaza o factura sub forma de fisier.txt pentru o comanda efectuata din interfata client. Aceasta metoda se apeleaza in interfata client la apasarea butonului plasare comanda si va creea fisierul unde se va afla comanda si datele despre aceasta.

Metode din Clasa Serializator:

static public void serializareClienti() {  
 try {  
 FileOutputStream fileOut = new FileOutputStream("clienti.txt");  
 ObjectOutputStream out = new ObjectOutputStream(fileOut);  
 Object obj;  
 out.writeInt(Inregistrare.*getCl*().size());  
 for (Client p : Inregistrare.*getCl*()) {  
 out.writeObject(p);  
 }  
 out.close();  
 fileOut.close();  
 } catch (IOException i) {  
 i.printStackTrace();  
 }  
}

Aceasta metoda este folosita pentru serializarea conturilor clientilor.

static public void deserializareClienti() {  
 try {  
 FileInputStream fileIn = new FileInputStream("clienti.txt");  
 ObjectInputStream in = new ObjectInputStream(fileIn);  
 Object obj;  
 int n = in.readInt();  
 for (int i = 0; i < n; i++) {  
 obj = in.readObject();  
 Client cl = (Client) obj;  
 Inregistrare.*addUser*(cl);  
 }  
 in.close();  
 fileIn.close();  
 } catch (EOFException ex) {  
 System.*out*.println(ex.getMessage());  
 } catch (IOException | ClassNotFoundException i) {  
 i.printStackTrace();  
 }  
}

Aceasta metoda este folosita pentru deserializarea conturilor clientilor.

1. Rezultatele problemei

În ceea ce priveste testarea problemei, programul meu funcționează fără nicio problemă. Pentru testarea acestuia se va creea un cont pentru client, iar apoi se va autentifica în cont pentru a putea plasa o comandă, după autentificarea în cont în cazul clientului, se va putea căuta un anumit produs în meniu și se va putea face comanda. În cazul administratorului, după autentificare se pot importa produse, adauga, sterge, edita sau se pot creea rapoarte pe anumite categorii.

1. Concluzii

Având în vedere tema propusă, a fost una destul de interesantă, foarte complexă având multe lucruri noi în cerință și care mi-a dat destul de mult de gândit. De asemenea a fost o temă foarte practică, un model de aplicație pe care poți comanda produse online prin intermediul aplicației la fel cum sunt și aplicațiile pe care le găsim în viața reală.

1. Bibliografie

Ca surse de informare am folosit Youtube, Wikipedia și Google căutând pe mai multe site-uri și uitându-mă la multe video-uri pentru implementarea acestei teme. De asemenea am folosit și link-urile din materialul suport pentru rezolvarea acestei probleme.