|  |
| --- |
| Universitatea Tehnică din Cluj-Napoca |
| Sistem de procesare a comenzilor |
| Tehnici de programare – Tema 3 |

|  |
| --- |
| Bolba Raluca Maria, grupa 30225  4/19/2015 |

Cuprins

1. Obiectivul temei ................................................................................................................2
2. Analiza problemei, modelare, scenarii, cazuri de utilizare ................................................2
3. Proiectare (diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfață utilizator) ...........................................................................7
4. Implementare și testare...................................................................................................13
5. Rezultate ..........................................................................................................................16
6. Concluzii, dezvoltări ulterioare ........................................................................................17
7. Bibliografie .......................................................................................................................17

1.Obiectivul temei

*Enunțul problemei :* Sa se implementeze un Sistem de gestiune a comenzilor pentru procesarea comenzilor unor clienti. Aplicatia utilizeaza cel putin clasele: Comanda, Client, DPComenzi (Departamentul de procesare al comenzilor), Produs si Depozit. Clasele DPComenzi si Depozit utilizeaza un ArboreBinarCautare pentru stocarea datelor.

* Analiza domeniului aplicatiei, determinarea structurii si comportamentului claselor. Identificarea diagramelor de utilizare, diagramelor de clase UML si a doua diagrame de secventa.
* Implementarea claselor aplicatiei. Utilizarea comentariilor javadoc pentru documentarea claselor.
* Implementarea sistemului de utilitati a programului pentru determinarea substocului, suprastocului, filtre...
* Scrierea metodelor de testare aferente.

Sistemul de procesare ar comenzilor va fi implementat sub forma unui magazin online de cosmetice.

În cadrul acestuia ar trebui să fie posibilă efectuarea unor operații precum : înregistrarea unui client în baza de date a acelui magazine, logarea acestuia în contul său, vizualizarea produselor, plasarea unei comenzi, vizualizarea comenzilor efectuate și confirmarea primirii produselor comandate. De asemenea, sistemul ar trebui să ofere efectuarea unor operații doar de către un administrator al acelui magazin. Aceste operații pot fi, de exemplu : adăugarea unui produs, ștergerea unui produs, vizualizarea acestora, vizualizarea clienților, vizualizarea comenzilor și alte date utile pentru acesta.

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

Pentru o bună implementare a sistemului de procesare a comenzilor, trebuie mai întâi să definim conceptele folosite, să stabilim datele de intrare și cum sunt ele furnizate și procesate pentru a fi folosite în cadrul sistemului pentru a rezulta date de ieșire corecte.

Având în vedere faptul că stocul de produse și chiar de comenzi poate fi destul de mare în cadrul unui magazine, aceste date trebuiesc stocate în așa fel încât accesul la ele să se facă rapid, ușor și eficient. Pentru a realiza acest lucru am ales folosirea unei baze de date, care pe lângă informațiile legate de produse și comenzi mai conține și informații legate și despre clienții acelui magazin. În principiu, o clasă din system va consta într-un tabel în baza de date, iar un obiect din sistemul implementat in Java va fi asociat cu o linie din tabelele din cadrul bazei de date. Deoarece un client poate alege mai multe produse în cadrul unei comenzi și pentru că limbajul SQL de implementare a bazei de date nu oferă structura de tip vector de elemente, avem nevoie de un tabel suplimentar, în care se va reține un identificator al produsului ce a fost comandat, câte exemplare au fost alese și de către ce comandă aparțin acestea.

Datele de intrare pentru acest sistem sunt : datele personale ale unui client dacă acesta dorește să își facă un cont pentru a putea realiza o comandă, intrarea acestuia în cont prin specificarea unui nume de utilizator și o parolă, adăugarea unor produse în coș, confirmarea anumitor comenzi, adăugarea informațiilor despre un produs nou (lucru realizat de administrator), specificarea produsului ce dorește să se șteargă, specificarea comenzii care trebuie procesată etc.

Pentru a putea face deosebire între clienții obișnuiți ai magazinului și administratorul acesteia, am stabilit ca atunci când se face logarea utilizatorului, dacă username-ul și parola introdusă sunt ambele „*admin”* atunci se face intrarea în contul administratorului.

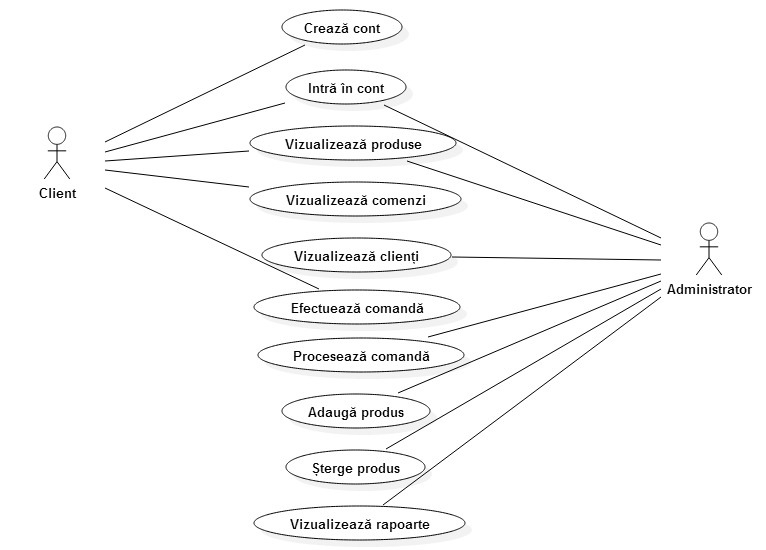
Datele de ieșire sunt și ele variate dar cele mai importante sunt : afișarea tuturor produselor, afișarea comenzilor (speficice pentru client, generale pentru administrator), afișarea clienților, afișarea coșului de cumpărături și de asemenea returnarea unor date precum produsele care au rămas puține în stoc, produse care sunt în număr mai mare decât capacitatea stocului, numărul total de produse din depozit etc.

Interfața grafică cu utilizatorul trebuie să fie una simplă, intuitivă și care să permită utilizatorului să efectueze operațiile specificate, atât cele specifice clienților magazinului, cât și administratorului. În esență, aceasta ar trebui să fie asemănătoare cu formatele întâlnite în ziua de astăzi în cadrul oricărui magazine online.

Diagrama cazurilor

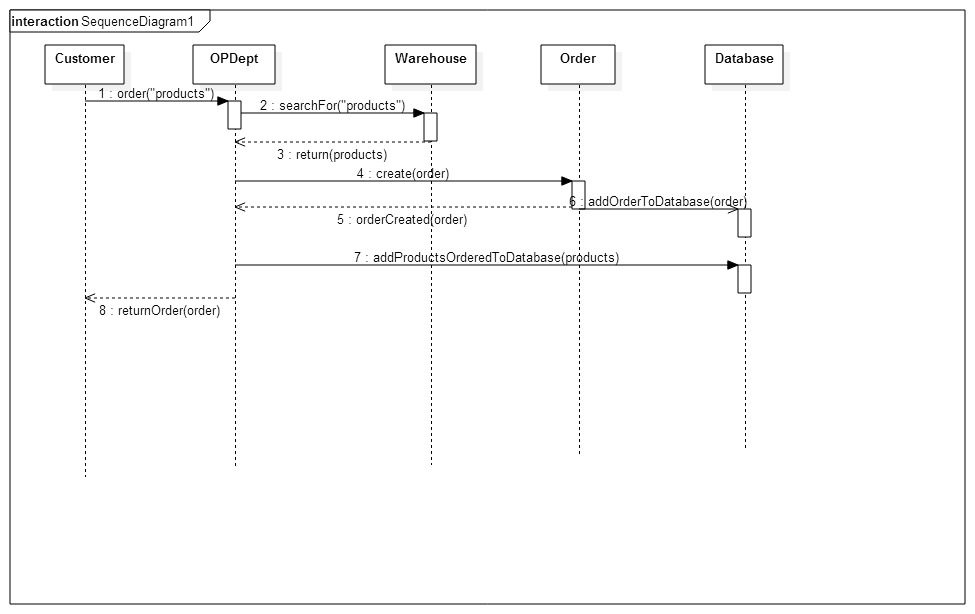
*Actorii* reprezintă o categorie de utilizatori, în cazul nostru este vorba despre client și despre administrator.

*Funcționalitățile* sau cazurile de utilizare sunt acțiunile pe care le pot face actorii. Cum am mai specificat, acțiunile pe care le poate face un client sunt : creare unui cont, intrarea în cont, vizualizarea produselor, adăugarea produselor în coș, plasarea comenzii, vizualizarea comenzilor, confirmarea primirii comenzilor, iar acțiunile care pot fi făcute de administrator sunt : intrarea în cont, vizualizarea produselor, adăugarea unui produs, ștergerea unui produs, vizualizarea clienților, vizualizarea comenzilor, procesarea unor comenzi și vizualizarea anumitor filtre. Cele specificate în diagrama de utilizare sunt cele mai generale, de exemplu pentru efectuarea unei comenzi de către client nu specificăm toți pașii (adăugarea produselor în coș, finalizarea comenzii, confirmarea primirii).



Diagrame de secvență

*Plasarea unei comenzi de către un client :*



*Crearea unui cont nou:*

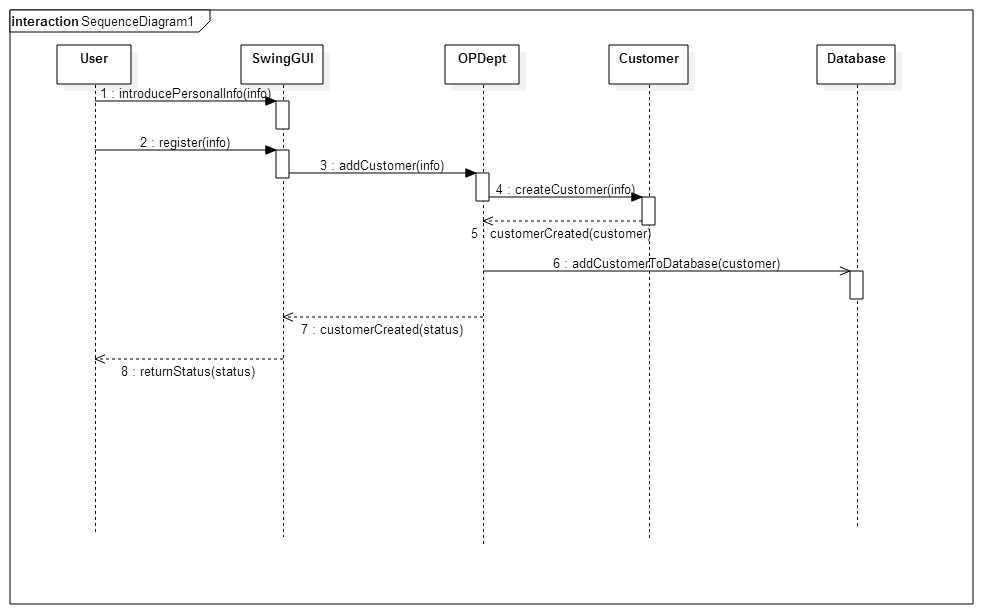
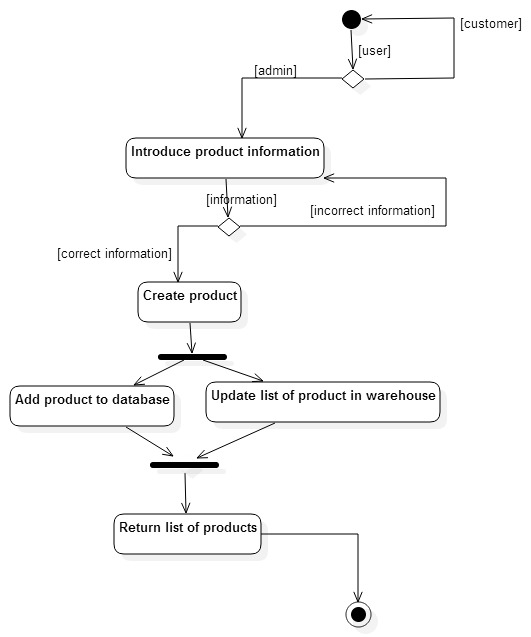


Diagrama de activitate

*Introducerea unui nou produs de către administrator :*



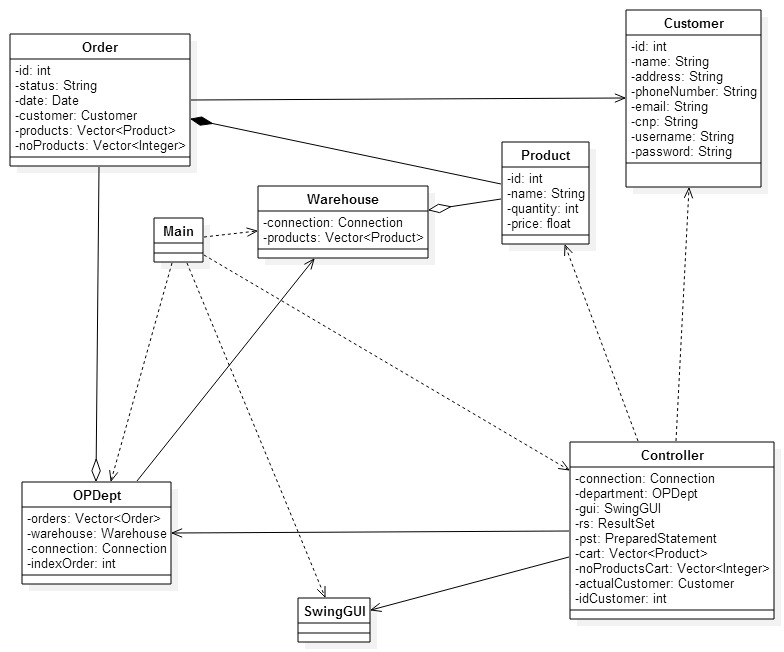
3. Proiectare (diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfață utilizator)

3.1 Proiectare clase

Pentru implementarea acestei probleme, am considerat necesare următoarele clase : *Customer, Product, Order, Warehouse, OPDept, SwingGUI, Controller, Main.*

* Clasa *Customer*, după cum sugerează și numele, modelează conceptul de client al magazinului pentru care am realizat sistemul de procesare a comenzilor. Atributele acestei clase sunt : id-ul clientului, util pentru lucrul cu baza de date, numele, adresa, numărul de telefon, adresa de email, codul numeric personal, numele de utilizator și parola. Obiectele acestei clase sunt instanțiate prin constructorul cu același nume, care primește ca parametrii id-ul și restul datelor personale ale clientului.
* Clasa *Product* modelează conceptul de produs din cadrul magazinului online, în cazul nostru un produs cosmetic. Clasa are ca atribute un id al produsului, numele produsului, numărul de exemplare al acestui produs în stoc și prețul produsului. Obiectele acestei clase sunt instanțiate în constructorul cu același nume care primește ca parametrii atributele produsului menționate mai sus. Clasa implementează clasa *Comparable* , utiliă pentru a compara produsele în funcție de id-ul acestora.
* Clasa *Order* modelează conceptul de comandă pe care o realizează un client al magazinului. Atributele acestei clase sunt : un identificator al comenzii, util pentru lucrul cu baza de date, un status al comenzii care poate fi *„Created”, „InProgress”* sau *„Completed”*, data la care a fost plasată comanda, un obiect de tip *Customer* care reprezintă clientul care plasează comanda, o listă de produse reprezentând produsele care s-au comandat și o listă de cantități reprezentând numărul de produse comandate pentru fiecare produs. Obiectele acestei clase sunt instanțiate în constructorul cu același nume care primește ca parametrii toate atributele comenzii, în afară de status-ul acesteia. Acesta va fi setat în funcție de alți factori.
* Clasa *Warehouse* modelează conceptul de depozit care conține produsele care se vând în cadrul magazinului. Acesta are ca atribut o listă de produse care se află în depozit. Obiectele acestei clase sunt instanțiate în constructorul cu același nume care primește ca parametru un obiect de tip *Connection*, prin care se poate face conexiunea cu baza de date. În cadrul constructorului se aduc produsele existente în baza de date și se plasează în lista de produse.
* Clasa *OPDept* modelează conceptul de departamentul de procesare a comenzilor. Această clasă are ca atribute o listă de comenzi și un obiect de tip *Warehouse*, pe baza cărora se vor efectua anumite operații. Obiectele acestei clase sunt instanțiate în cadrul constructorului cu același nume care primește ca parametru un obiect de tip *Connection*, prin care se poate face conexiunea cu baza de date. În cadrul constructorului se aduc comenzile existente în baza de date și se plasează în lista de comenzi.
* Clasa *SwingGUI* modelează conceptul de interfață grafică cu utilizatorul prin care un utilizator poate interacționa cu aplicația propriu-zisă. Ca bibliotecă de dezvoltare a interfeței grafice am folosit *Swing*.
* Clasa *Controller* modelează conceptual de controller care face legătură între partea grafică a sistemului și modelul acestuia. Printre atributele acestei clase se numără un obiect de tip *SwingGUI, OPDept* între care se fac principalele legături, dar și alte atribute necesare precum un obiect de tip client în care păstrăm clientul care este logat în contul său, o listă de produse și o alta de cantități care reprezintă împreună coșul de cumpărături al clientului etc.
* Clasa *Main* este clasa în care se pornește aplicația, prin instanțierea unor obiecte de tip *Warehouse, OPDept, SwingGUI, Controller* și în care se realizează conexiunea cu baza de date. Acest lucru se realizează prin înregistrarea unui Driver JDBC și realizarea conexiunii cu metoda *DriverManager.getConnection().*

3.1 Diagrama UML



3.3 Relațiile între clase

Pe baza diagramei UML se observă relațiile între cele opt clase.

Între clasa *Order și Product* există relația de compoziție, care reprezintă o relație specială de asociere, mai exact o formă de agregare cu posesiune puternică conform căreia părțile nu pot supraviețui agregatului. În cazul nostru, clasa *Order* reprezintă agregatul, ale cărei părți, respectiv instanțele clasei *Product* nu pot exista independent.

Între clasa *Warehouse și Product, OPDept și Order* relația de agregare , care reprezintă o relație specială de asociere ce modelează relația întreg – parte între un agregat(întreg) și părțile sale. Agregatul reprezintă în cazul nostru clasa *Warehouse,* respectiv *OPDept*, iar partea clasa *Product,* respectiv *Order.*

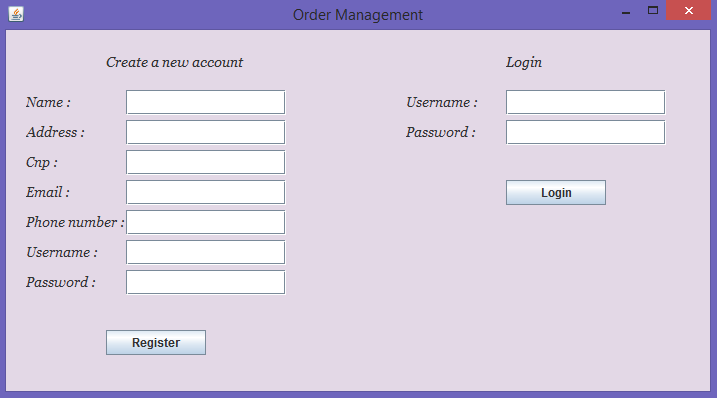
Între clasa *Controller și OPDept și SwingGUI,* între *OPDept și Warehouse,* între *Order și Customer* există relația de asociere care reprezintă cazul în care un obiect dintr-o clasăfolosește serviciile unui obiect al celeilalte clase.

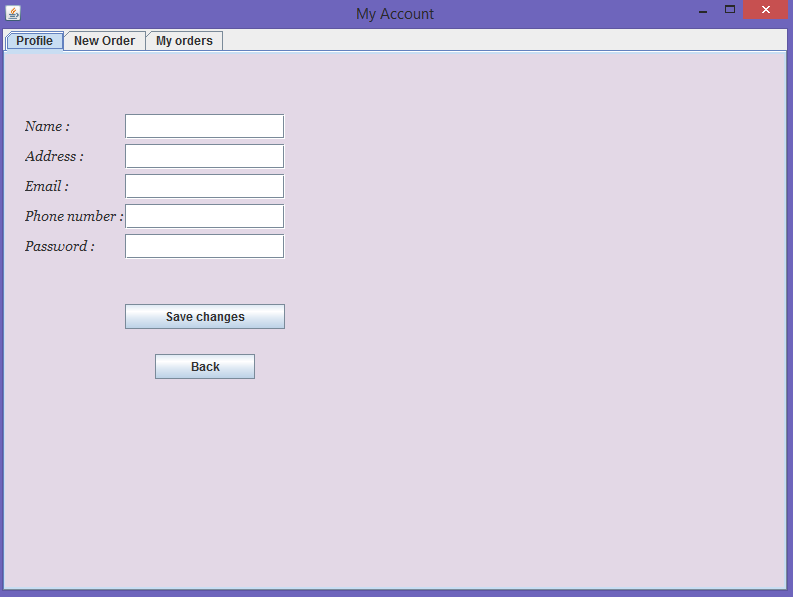
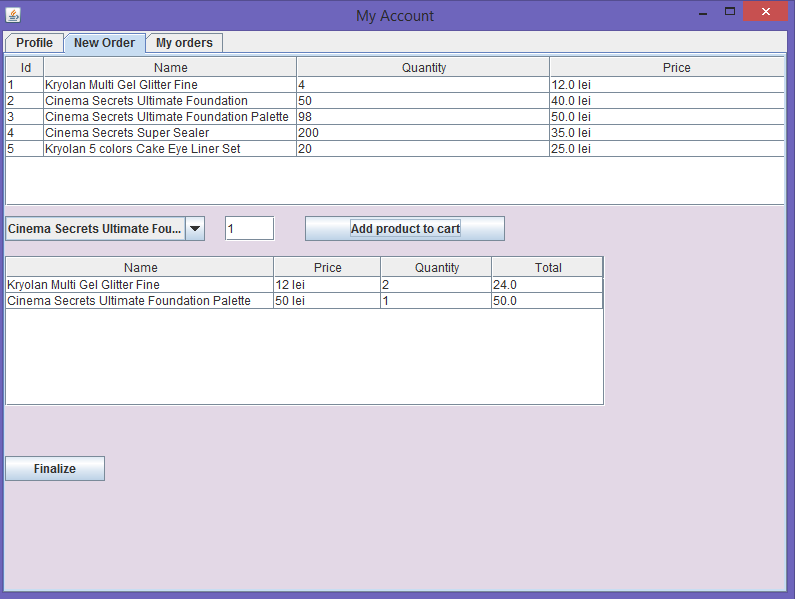
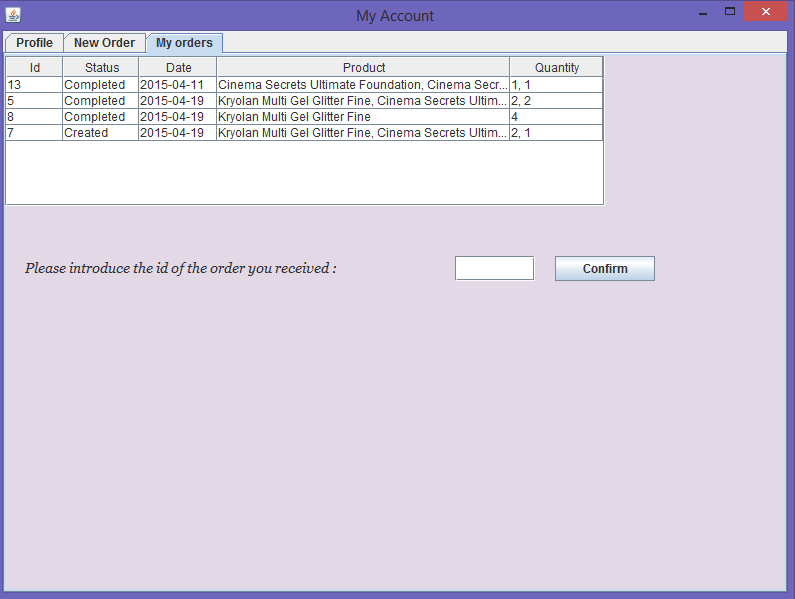
Restul relațiile care se pot vedea și din diagrama UML sunt de tip dependență, prin care o clasă foloseșe obiecte ale altei clasei.

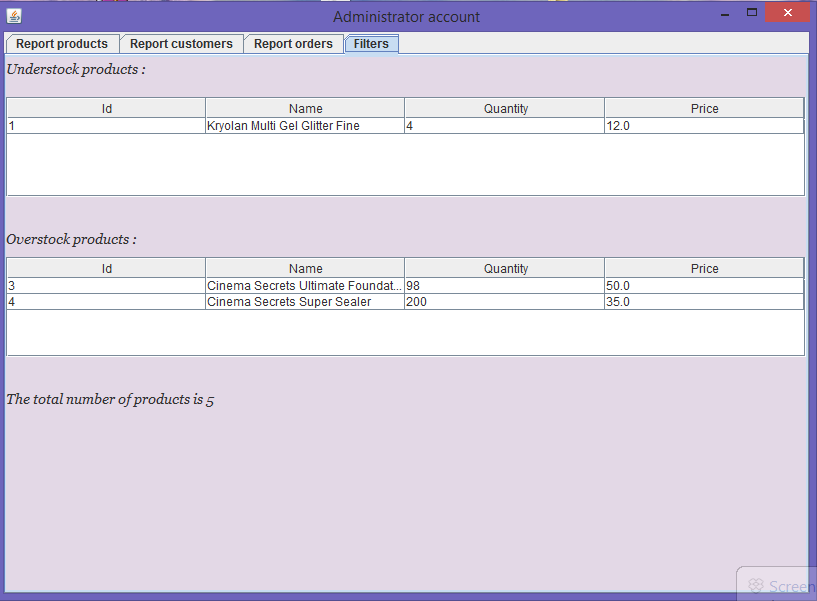
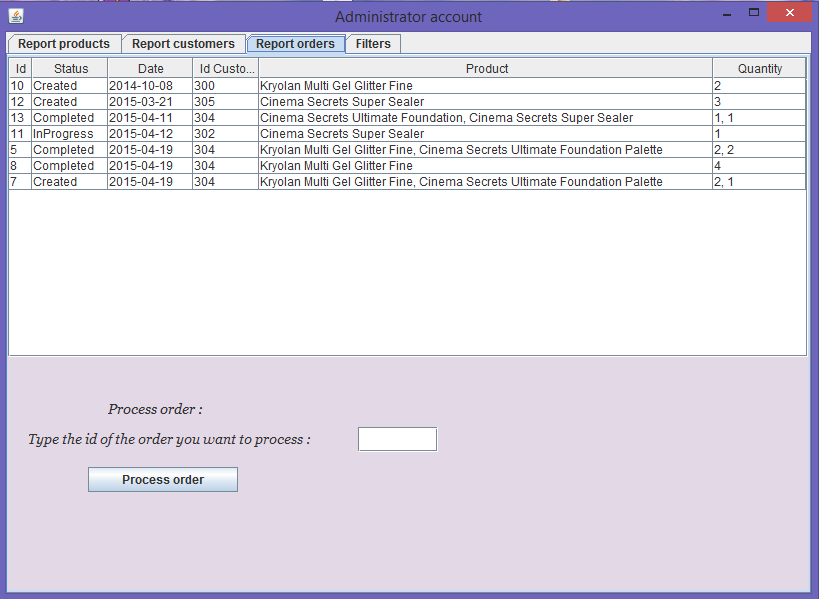
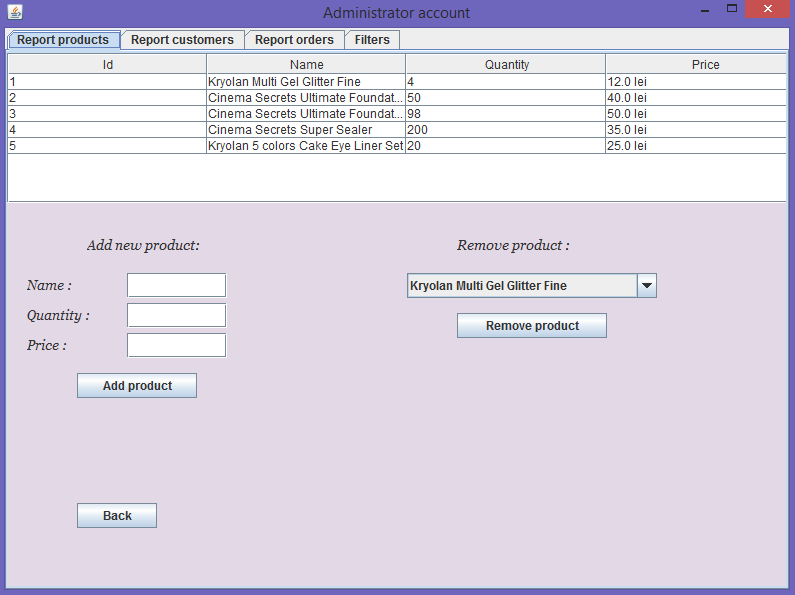
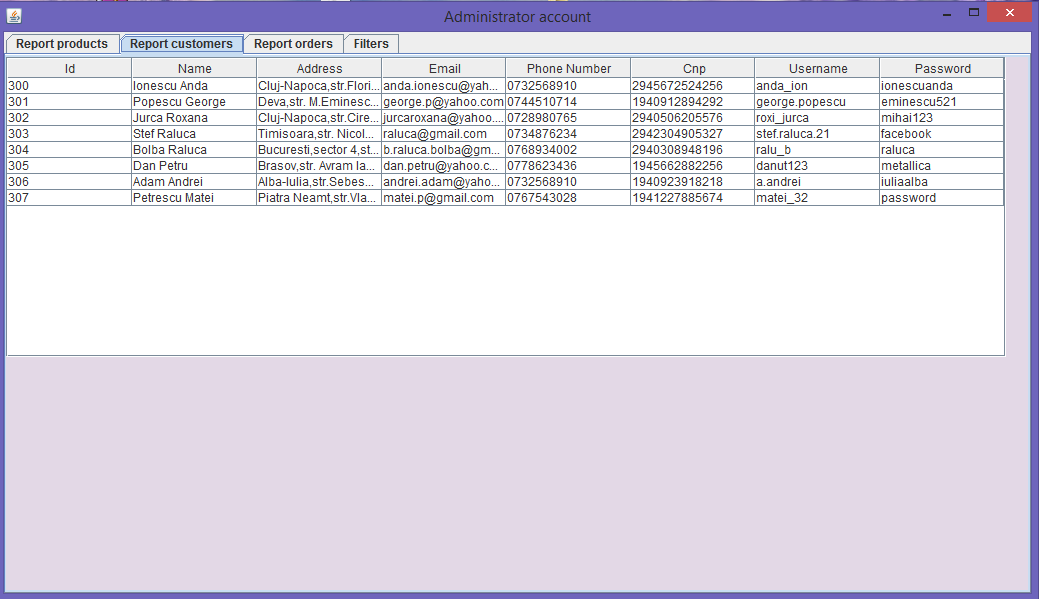
3.4 Interfața utilizator

Așa cum am mai menționat, interfața grafică cu utilizatorul am implementat-o cu ajutorul bibliotecii Swing. Principalele clase folosite în cadrul realizării acestei interfețe sunt : J*Button, JComboBox, JLabel, JTextField, JFrame , JdesktopPane, JPasswordField etc.*

Fereastra de început a aplicației este împărțită în două părți : în partea stângă observăm formularul clasic de creare a unui cont de către client prin introducerea datelor personale, a unui nume de utilizator și a unei parole, iar în partea dreaptă se observă câmpuri în care utilizatorul își introduce numele de utilizator și parola pentru a intra în cont. Dacă numele de utilizator sau parola, acesta va fi atentionat printr-un mesaj. Un mesaj de confirmare a crearii contului este de asemenea primit de client la apăsarea butonul *Register.*



 După ce clientul intră în cont, fereastra acestuia are mai multe tab-uri, acestea fiind *Profile, New Order și My Orders*. În cadrul primului, clientul poate schimba anumite date legate de contul său, precum numele, adresa, parola, email-ul sau numărul de telefon. În cadrul celui de-al doilea tab, clientul poate efectua o nouă comandă. În partea de sus observăm un tabel cu toate produsele magazinului cât și cantitatea acestora în stoc, sub acesta un câmp de tip *JcomboBox* în care clientul poate alege numele produsului pe care dorește să îl adauge în coș, un câmp în care introduce cantitatea dorită și un buton *Add product to cart* în urma apăsării căruia produsul respectiv se adaugă în coșul reprezentat tot sub formă de tabel mai în josul paginii. Atunci când clientul apasă butonul *Finalize order* aceasta va fi plasată și va apărea în cea de-al treilea tab, cel cu comenzile efectuate de client.

 Dacă în pagina principală utilizatorul se înregistrează cu username-ul și parola *admin* atunci el va fi redirecționat către fereastra corespunzând contului administratorului. Și această fereastră este împărțită pe patru tab-uri : *Report Products, Report Customers, Report Orders și Filters*. În cadrul primui tab, administratorul vede produsele sub forma tabelului din partea de sus, iar in partea de mijloc al ferestrei el poate alege adăugarea unui nou produs sau ștergerea unui existent. În cel de-al doilea tab se observă un tabel conținând toate datele despre clienții magazinului, iar în cel de-al treilea tab administratorul poate vizualiza toate comenzile efectuate, precum și starea în care se află acestea. Acesta poate alege procesarea unei comenzi prin introducerea id-ului acesteia și apăsarea butonului *Process order*.

4. Implementare și testare

4.1 Implementare

* Clasa *Customer* modelează conceptul de client al magazinului. Metodele acestei clase sunt :
* **public** Customer(**int** id, String name, String address, String phoneNumber, String email, String cnp, String username, String password) – constructorul clasei
* **public** **int** getId()– metodă care returnează id-ul clientului
* **public** String getName()– metodă care returnează numele clientului
* **public** String getAddress()– metodă care returnează adresa clientului
* **public** String getPhoneNumber()– metodă care returnează numărul de telefon al clientului
* **public** String getEmail()– metodă care returnează email-ul clientului
* **public** String getCnp()– metodă care returnează cnp-ul clientului
* **public** String getUsername()– metodă care returnează numele de utilizator al clientului
* **public** String getPassword()– metodă care returnează parola clientului
* **public** **void** setPhoneNumber(String newPhoneNumber) – metodă care setează numărul de telefon cu unul nou
* **public** **void** setEmail(String newEmail) - metodă care setează email-ul cu unul nou
* **public** **void** setPassword(String newPassword) – metodă care setează parola cu una nouă
* **public** **void** placeOrder(Vector<Product> products, Vector<Integer> noProducts) – metodă prin care clientul plasează o comandă, mai exact apelează o metodă din OPDept de creare a unei comenzi
* **public** **void** receiveOrder(Order order) – metodă prin care clientul afirmă confirmarea unei comenzi prin setarea statusului acesteia în *Completed*
* Clasa *Product* modelează conceptual de produs din cadrul magazinului. Metodele acestei clase sunt :
* **public** Product(**int** id, String name, **int** quantity, **float** price) ) – constructorul clasei
* **public** **int** getId()) – metodă care returnează id-ul produsului
* **public** String getName()– metodă care returnează numele produsului
* **public** **int** getQuantity()– metodă care returnează cantitatea în stoc a produsului
* **public** **float** getPrice()– metodă care returnează prețul produsului
* **public** **void** setName(String newName) – metodă care setează numele produsul cu unul nou
* **public** **void** setQuantity(**int** newQuantity) ) – metodă care setează cantitatea produsului cu una noua
* **public** **void** setPrice(**float** newPrice) ) – metodă care setează prețul produsului cu unul nou
* **public** String toString()) – metodă care returnează produsul sub formă de șir de caractere
* **public** **int** compareTo(Object arg0) – metodă suprascrisă datorită implementării interfeței *Comparable*, aceasta compară două produse pe baza id-ului
* **public** **boolean** equals(Object arg0) – metodă suprascrisă, care verifică dacă două produse sunt egale pe baza id-ului acestora
* Clasa *Order* modelează conceptual de comandă. Metodele acestei clase sunt :
* **public** Order(**int** id, Date date, Customer customer, Vector<Product> products, Vector<Integer> noProducts) – constructorul clasei
* **public** **int** getId()– metodă care returnează id-ul comenzii
* **public** Date getDate()– metodă care returnează data comenzii
* **public** Customer getCustomer()– metodă care returnează clientul care a realizat comanda
* **public** Vector<Product> getProducts()– metodă care returnează lista de produse care au fost comandate
* **public** Vector<Integer> getNoProducts()– metodă care returnează lista de cantități ale produselor comandate
* **public** String getStatus()– metodă care returnează statusul comenzii
* **public** **void** setStatus(String newStatus) – metodă care setează statusul comenzii cu unul nou.
* Clasa *Warehouse* modelează conceptual de depozit. Metodele acestei clase sunt :
* **public** Warehouse(Connection connection) – constructorul clasei
* **public** **void** getProductsFromDatabase()– metodă care preia din cadrul bazei de date produsele existente și le adaugă la lista de produse ale depozitului
* **public** Vector<Product> getProducts()– metodă care returnează produsele existente în depozit
* **public** **void** addProduct(Product product) – metodă care adaugă un produs în depozit, atât la lista de de produse ale acestuia, cât și în baza de date.
* **public** **boolean** removeProduct(Product product) – metodă care elimină un produs din lista de produse a depozitului, cât și din baza de date, identificarea lui făcându-se pe baza id-ului, care este unic pentru fiecare produs.
* **public** **int** numberProducts()– metodă care returnează numărul total de produse din depozit.
* Clasa *OPDept* modelează conceptual de departament de procesare a comenzilor. Metodele acestei clase sunt :â
* **public** OPDept(Connection connection) – constructorul clasei
* **public** **void** getOrdersFromDatabase()– metodă care preia comenzile existente din baza de date și le adaugă la lista de comenzi
* **public** Warehouse getWarehouse()– metodă care returnează depozitul de produse
* **public** Vector<Order> getOrders()– metodă care returneaza lista de comenzi din cadrul departamentului
* **public** **static** **void** addCustomer(Customer customer) – metodă care adaugă un nou client în baza de date. Întâi se verifică dacă există deja în baza de date un client cu username-ul introdus, iar în caz afirmativ, clientul este atenționat asupra acestui lucru și rugat să aleagă alt username. În caz negativ, acesta este adăugat la baza de date.
* **public** **static** **void** addOrder(Customer customer, Vector<Product> products, Vector<Integer> noProducts) – metodă care adaugă o nouă comandă, atât la lista de comenzi a departamentului, cât și în cadrul bazei de date. Pentru a realiza acest lucru se generează un id pentru această comandă, se determină data comenzii care reprezintă data sistemului la care a fost apelată metoda și se inserează aceste date împreună cu id-ul clientului în tabelul *orders* din baza de date. Pentru a insera și produsele și cantitățile alese, se parcurg cele două liste și se inserează în tabelul din baza de date *product\_order*, în care se reține și id-ul din cadrul cărei comenzi fac parte.
* **public** **boolean** processOrder(Order order) – metodă care procesează o comandă. În cadrul acestei metode se verifică dacă comanda există în lista de comezi și dacă statusul acesteia este *Created*. În caz afirmativ, se parcurg toate produsele comandate, și se verifică dacă acestea există în depozit și dacă cantitățile alese sunt mai mici sau egale cu cantitățile aflate pe stoc. În caz afirmativ, aceste cantități sunt scăzute din valorile stocului fiecărui produs, iar comanda este setată ca fiind *InProgress.*
* Clasa *Controller* modelează conceptual de controller, care face legătura între interfața grafică și modelul programului. Metodele aceste clase sunt :
* **public** **void** addOrderToModel(DefaultTableModel model, **int** idCustomer) – metodă care preia comenzile din baza de date și le adaugă la *model*, pentru ca mai apoi acest *model* să fie transmis interfeței grafice, pentru ca aceste comenzi să fie afișate utilizatorului sub formă de tabel. Comenzile se preiau în funcție de *idCustomer,* respectiv : dacă acesta este -1, atunci comenzile trebuie vizualizate de către administrator, deci trebuie adăugate toate, iar dacă *idCustomer* este diferit atunci trebuie adăugate comenzile care sunt realizate de clientul cu acel id.
* **public** **void** addProductModel(DefaultTableModel modelProduct, JComboBox listProducts) – metodă care preia produsele din baza de date și le adaugă la *modelProduct*, pentru ca mai apoi acest *modelProduct* să fie transmis interfeței grafice, pentru ca produsele să fie afișate utilizatorului sub formă de tabel. În același timp, produsele sunt adăugate la *listProducts*, care va fi și el trimis interfeței grafice, prin care clientul poate alege ce produs să introducă în coș, iar administratorul ce produs să șteargă.
* **public** **void** addCustomerModel(DefaultTableModel model) – metodă care preia clientii din baza de date și le adaugă la *model*, pentru ca mai apoi acest *model* să fie transmis interfeței grafice, pentru ca acești clienți să fie afișați administratorului sub formă de tabel.

4.2 Testare

Testarea acestei aplicații am realizat-o prin așa numitele *teste unitare*, care se referă la testarea individuală a unor unități separate dintr-un sistem software. În cazul nostru se testează comportamentul particular al unei clase și al metodelor. Testarea am făcut-o asupra claselor *Customer, Order, Product.* Testarea am implementat-o cu ajutorul framework-ului *TestNG*.

5. Rezultate

În urma rulării aplicației se observă că aceasta rulează fără problem, efectuarea comenzilor se face în mod corect , iar modificările ce apar asupra stocului de produse, a procesării comenzilor sau a adăugării de clienți noi au loc în timp real. Se mai observă că atunci când utilizatorul introduce date de intrare invalide acesta este atenționat cu mesaje de eroare corespunzătoare.

6. Concluzii

În concluzie, se poate spune că am reușit implementarea unui sistem de procesare a comenzilor. Problemele ridicate de acest proiect au fost în cazul meu lucrul cu baza de date, nevoie de a o actualiza în permanență și de asemenea, actualizarea interfeței astfel încât să nu existe redundanțe.

Deși aplicația respectă în mare cerințele problemei, ea poate fi îmbunătățită prin :

* Modificarea interfeței grafice, astfel încât fiecare produs să aibă asociată și o poză, iar adăugarea produselor în coș să nu se facă prin selectarea unui nume dintr-o listă, ci prin apăsarea unui buton pentru fiecare produs.
* Adăugarea unor operații precum ștergere din coș, anulare comandă, actualizare produs.

7. Concluzii

<https://docs.oracle.com/javase/tutorial/jdbc/basics/connecting.html>

<https://docs.oracle.com/javase/tutorial/uiswing/components/table.html>