Universitatea Tehnica Cluj-Napoca

Facultatea Automatica si Calculatoare

Specializare Calculatoare si Tehnologia Informatiei

Tehnici fundamentale de programare

*~Tema 3~*

Proiect Order management

Nume: Ștef David-Alexandru

Grupa: 30224

Cuprins:

1. Cerința Proiect …………………………………………………………. 2
2. Analiza Cerinței si Idei de Implementare.….…………………………… 2
3. Organizarea in pachete …………………………………………………. 4
4. Implementarea Cerinței ………………………………………………… 4
5. Diagrame ……………………………………………………………… 10
6. Concluzii ………………………………………………………………. 11
7. Bibliografie ……………………………………………………………. 11

**1. Cerința proiect**

Cerința acestui proiect se referă la analizarea si implementarea unei aplicații OrderManagement pentru procesarea comenzilor clienților pentru un depozit. Pentru a stoca produsele, clienții si comenzile se vor folosi baze de date relaționale. Aplicația va trebui sa permită procesarea comenzilor dintr-un fișier text dat ca argument din linia de comanda, mai apoi efectuați operațiunile solicitate, salvați datele in baza de date si generați rapoarte in format pdf. Pentru a implementa funcționalitatea completă a aplicației se pot utiliza sau adăuga alte pachete si clase decât cele specificate în indicațiile oferite.

Implementați un parser pentru a citi comenzile în stratul Prezentare (care va înlocui interfața grafică standard cu utilizatorul) și un generator de fisiere pdf pentru a genera rapoarte. De exemplu pentru informația citită, “Inser client: Ion Popescu, Bucuresti”, se va insera in baza de date un nou client cu numele Ion Popescu si adresa Bucuresti sau pentru informatia “Report client” se va genera un raport in format pdf un tabel ce va contine totii clientii si informatiile despre fiecare selectate in functie de campurile fiecarei entitati din baza de date utilizand comanda specifica sql: SELECT \* FROM table.

Se va implementa aplicatia folosindu-se Javadoc pentru documentarea claselor. Ca date de intrare vor se vor citi datele dintr-un fisier dat de catre utilizator care va contine pe fiecare linie cate o comanda care va fi preluata si transformata intr-o comanda sql. Prin comenzile introduse utilizatorul are posibilitatea de a crea un nou client, de a-l sterge si de a face update pe datele sale, de asemenea un produs poate fi adaugat, sters sau actualizat. Produsul va fi actualizat cand un client existent va face o comanda pentru un produs cu o anumita cantitate, cantitatea innitiala a produsului din deposit va fi modificata cu noua cantitate ramasa. De asemenea, la cererea utilizatorului se va crea cate un raport in format pdf pentru client, produs sau comanda care va contine datele despre fiecare din acestea trei.

Se vor utiliza tehnici de reflectie pentru a crea o clasa generica care va contine metodele de accesate a bazei de date: inserarea obiectului, actualizarea obiectului, stergerea unui obiect si gasirea sa. Pentru un anumit obiect care corespunde unui tabel din baza de date interogarile se vor genera dynamic prin reflectie.

**2. Analiza cerinței si Idei de Implementare**

In urma analizeri ceintei pentru aceasta aplicatie am proiectat o aplicatie care poate procesa urmatoarele comenzi de baza: Crearea, Gasirea, Stergerea si Actualizarea unui obiect pe o baza de date care este structurata in 4 tabele: produs, client, comanda si produsDorit. Proiectul dispune de urmatoarele functionalitati: gasirea tuturor clientilor produselor sau comenzilor din tabele, adaugarea unui produs sau client nou, stergerea unui produs sau client deja existent si actualizarea unui produs sau client cu noi date. De asemenea, se poate crea o comanda care va contine: clientul care face comanda, produsul care se comanda, cantitatea dorita de client si pretul pentru acea cantitate de produs; in urma acestei comenzi se va emite un bon.

a). Pentru **Găsirea** unuia sau mai multor **Clienti** in baza de date va trebui ca fiecare dintre clienti sa existe si acestia vor fi returnati. Un client poate fi cautat dupa id sau nume, iar baza de date va returna cu ajutorul comenzii SELECT continutul tabelului client. In cazul in care clientul/clientii cautati nu exista aplicatia va semnala acest fapt.

b). Pentru **Adăugarea** unui **Client** in baza de date, datele introduse de utilizator vor trebui sa contina toare campurile care se regasesc in clasa Clienti si anume: id-ul (cheie primara), numele si adresa clientului nou. De asemenea, aceste date trebuie introduse in mod corect deoarece aplicatia va semnala acest aspect. In cazul introducerii unui nou client care deja este regasit in baza de de date cu acelasi id se va face un “update” asupra datelor sale cu noile date introduse. In cazul in care datele introduse sunt eronate aplicatia va semnala acest fapt si Clientul nu va fi introdus in baza de date.

c). Pentru **Ștergerea** unui **Client** din baza de date este necesar ca acel client introdus de utilizator sa existe, el este cautat si sters in functie de id-ul sau. Daca un client este sters vor fi in acelasi timp sterse din baza de date toate comenzile plasate de acesta.

d). Pentru **Actualizarea** unui **Client** din baza de date este necesar ca acel client introdus de utilizator sa existe, el este cautat in functie de nume sau id si ii vor fi actulizate datele cu noile date introduse de catre utilizator.

e). Pentru **Vizualizarea** unuia sau mai multor **Produse** din baza de date este necesar ca acestea sa existe, iar in caz contrat aplicatia va semnala faptul ca nu a fost gasit produsul/ produsele cautat(e). Pentru a se afisa toate produsele se va utiliza comanda sql SELECT urmata de caracterul ‘\*’. Un anumit produs se va putea cauta in funtie de id-ul sau numele sau.

f). Pentru **Adăugarea** unui **Produs** in baza de date, datele introduse de utilizator vor trebui sa contina toare campurile care se regasesc in clasa Produs si anume: id-ul (cheie primara), numele, cantitatea si pretul noului produs. De asemenea, aceste date trebuie introduse in mod corect deoarece aplicatia va semnala acest aspect. In cazul in care datele introduse sunt eronate aplicatia va semnala acest fapt si Produsul nu va fi introdus in baza de date. In cazul introducerii unui nou produs care deja este regasit in baza de de date cu acelasi id se va face un “update” asupra datelor sale mai exact cantitatea existenta deja va fi insumata cu noua cantitate a produsului.

g). Pentru **Ștergerea** unui **Produs** din baza de date este necesar ca acel produs introdus de utilizator sa existe, el este cautat si sters in functie de id-ul sau numele sau.

h). Pentru **Actualizarea** unui **Produs** din baza de date este necesar ca acel produs introdus de utilizator sa existe, el este cautat in functie de nume sau id si ii vor fi actulizate datele cu noile date introduse de catre utilizator.

i). Pentru **Adăugarea** cu succes a unei noi **Comenzi** in baza de date trebuie sa fie indeplinite simultan mai multe conditii: sa existe clientul care plaseaza comanda, sa existe produsul in baza de date si mai mult de atat, cantitatea dorita de client si fie mai mica sau cel mult egala cu cantitatea existenta deja in depozit, iar in urma comenzii se va actualiza cantitatea acelui produs din depozit. In cazul in care cantitatea dorita este mai mare comanda nu se va efectua si se va afisa mesajul corespunzator. Totodata, in cazul in care una dintre datele mentionate sunt introduse intr-un mod eronat aplicatia va semnala acest lucru si comanda nu se va efectua. Daca produsul sau clientul nu exista in baza de date aplicatia sa afisa un mesaj de eroare. In urma comenzii effectuate aplicatia va genera o chitanta in format pdf. In clasa Comanda si totodata in tabela Comanda din baza de date se va retine id-ul clientului care plaseaza comanda, iar in clasa si tabela ProdusDorit se va retine id-ul comenzii, id-ul produsului, cantitatea si pretul total al produsului comandat.

**3. Organizarea in pachete**

In implementarea aplicatiei am organizat clasele pe pachete, mai exact 7 pachete in functie de functionalitatea fiecarei clase. Aceste pachete sunt: “connection”, “model”, “dataAccess”, “businessLayer”, “presentation”, “validator” si “start”.

3.1. In pachetul **connection** se afla o singura clasa care se ocupa cu crearea conexiunii cu baza de date relationala.

3.2. In pachetul **model** se afla clasele care creeaza fiecare dintre cele patru obiecte utilizate in implementarea aplicatiei.

3.3. In pachetul **dataAccess** se gasesc lasele care realizeaza instructiunile corespunzatoare pentru baza de date si implementeaza comenzile sql.

3.4. In pachetul **bussinesLayer** se afla clasele care aplica comenzile sql asupra bazei de date realizand operatiile necesare pentru fiecare Obiect in parte (Client, Comana, Produs si ProdusDorit) si mai apoi genereaza report-urile pentru fiecare obiect si bonurile.

3.5. In pachetul **presentation** se afla “unitatea de control si comanda” si anume pentru fiecare comanda data ca intrare de catre utilizator aplicatia raspunde in mod corespunzator comenzii.

3.6. In pachetul **validators** se afla clasele care valideaza datele de intrare pentru clienti si produse.

3.7.In pachetul **start** si afla main-ul care pune in functiune aplicatia.

**4. Implementare cerinței**

Dupa cum am mentionat mai sus am organizat clasele in pachete dupa functionalitatea pe care o indeplineste fiecare. In cele ce urmeaza voi prezenta detalii legate de implementarea fiecarei clase in parte.

**4.1. Clasa ConnectionFactory** este cea care realizeza conexiunile cu baza de date. Aceasta clasa are ca variabile de instanta un Logger care preia conexiunea, un Driver care utilizeaza jdbc-ul pentru baza de date MySql, un Url de tip localhost pe care se realizeaza conexiunea cu baza de date deja create, numele user-ului si parola acestuia. Metodele utilizate sunt createConnection() care dupa cum ii spune si numele creeaza conexiunea si trei metode care inchid Conexiunea, Statement-ul si ResultSet-ul.

**4.2. Clasa** **Clienti** corespunde tabelei Clienti din baza de date si are rolul de a “construi” un obiect de tipul Client care conform field-urilor din baza de date va avea un id, un nume si o adresa.

**4.3. Clasa** **Produs** corespunde tabelei Produs din baza de date si are rolul de a “construi” un obiect de tipul Produs care conform field-urilor din baza de date va avea ca si variabile de instanta un id, numele produsului, cantitatea stocata si pretul pentru fiecare produs.

**4.4. Clasa Comanda** corespunde tabelei Comanda din baza de date si are rolul de a “construi” un obiect de tipul Comanda care conform field-urilor din baza de date va detine id-ul comenzii si id-ul clientului care a plasat comanda respective.

**4.5. Clasa ProdusDorit** corespunde tabelei ProdusDorit din baza de date si are rolul de a “construi” un obiect de tipul ProdusDorit care conform field-urilor din baza de date va detine id-ul comenzii, id-ul produsului comandat, cantitatea de produs cumparata si pretul total pentru aceasta cantitate.

**4.6. Clasa ImplementQueriesDAO** este o clasa utilizata pentru “construirea” comenzilor sql care vor fi utilizate pentru a face operatii asupra tabelelor. Aceast clasa detine ca variabile de instanta un Logger care preia conexiunea si un type generic (T) care se “pliaza” pe fiecare Obiect utilizat. Aici se construiesc cele 4 query-uri care stau la baza implementarii celor patru operatii care se vor putea efectua asupra tabelelor si anume: Select, Insert, Update si Delete. Ca si model pentru o astfel de metoda ar fi:

*protected String createSelectQuery(String condition) {*

*StringBuilder sql = new StringBuilder();*

*sql.append("SELECT ");*

*sql.append(" \* ");*

*sql.append("FROM ");*

*sql.append(type.getSimpleName());*

*sql.append(" WHERE " + condition + " =?");*

*System.out.println(sql.toString());*

*return sql.toString();*

*}*

**4.7. Clasa CreateQueriesDAO** utilizeaza in principal metodele din Clasa ImplementQueriesDAO pentru a implementa cele patru comenzi asupra bazei de date. De asemenea, si in aceasta clasa avem un Logger care preia conexiunea. Metodele din aceasta clasa au similitudine sporita si anume fiecare metoda preia conexiunea intr-un obiect Connection, creeaza un statement pentru query-ul corespunzator comenzii sql care se doreste a fi implementata, mai apoi acest statement se foloseste pentru a executa query-ul creat, iar aceasta executie se va pune pe un obiect ResultSet care creaza obiectul care va fi returnat. Pentru aceast se utilizeaza metoda CreateObjects(ResultSet result). Un model al unei astfel de functii implementate este pentru comanda Select in functie de id:

*public T findById(int id) {*

*Connection conn = null;*

*PreparedStatement statement = null;*

*ResultSet result = null;*

*String query = createSelectQuery("id");*

*try {*

*conn = ConnectionFactory.getConnection();*

*statement = conn.prepareStatement(query);*

*statement.setInt(1, id);*

*result = statement.executeQuery();*

*return createObjects(result).get(0);*

*} catch (SQLException e) {*

*e.printStackTrace();*

*System.out.println("Error: the element can't be find!");*

*}*

*ConnectionFactory.close(result);*

*ConnectionFactory.close(statement);*

*ConnectionFactory.close(conn);*

*return null;*

*}*

**4.8. Clasa ClientDAO** este “implementata” utilizand tehnica reflectiei si tipul generic T, astfel toate metodele implementate in clasa CreateQueriesDAO vor putea fi utilizate pentru un obiect de tipul ClientDAO.

**4.9. Clasa ProdusDAO** este “implementata” utilizand tehnica reflectiei si tipul generic T, astfel toate metodele implementate in clasa CreateQueriesDAO vor putea fi utilizate pentru un obiect de tipul ProdusDAO.

**4.10. Clasa ComandaDAO** este “implementata” utilizand tehnica reflectiei si tipul generic T, astfel toate metodele implementate in clasa CreateQueriesDAO vor putea fi utilizate pentru un obiect de tipul ComandaDAO.

**4.11. Clasa ProdusDoritDAO** este “implementata” utilizand tehnica reflectiei si tipul generic T, astfel toate metodele implementate in clasa CreateQueriesDAO vor putea fi utilizate pentru un obiect de tipul ProdusDoritDAO.

**4.12. Clasa ClientBLL** are ca variabile de instanta un obiect de tip ClientDAO pe care il creeaza, indicele clientului curent, indicele pdf-ului creat pentru report-ul clientului si un obiect de tipul Validator care este utilizat pentru a valida datele clientului. De asemenea obiectul ClientDAO este folosit pentru a apela metodele din acea clasa(find, insert, delete, update) si in cazul in care o instructiune de tipul sql nu a avut succes se va prinde o exceptie cu mesajul corespunzator. Metoda de insertClient() are o functionalitate putin diferita fata de celelalte, ea nu doar apeleaza metoda de insert din clasa ClientDAO ci verifica daca clientul a fost inserat. In cazul in care nu a fost inserat se va insera, iar in caz contrar se va face un update asupra datelor sale.

*if (cl != null) {*

*try {*

*clientDAO.updateClient(fld2, fld1);*

*} catch (Exception e) {*

*e.printStackTrace();*

*}*

*} else*

*try {*

*String s = iClient + ", '" + fld1 + "', '" + fld2 + "'";*

*clientDAO.insert(s);*

*iClient++;*

*} catch (Exception e) {*

*System.out.println("The element have been inserted!");*

*}*

De asemenea, tot aici se afla si metoda care creeaza reportul pentru client, creeaza un document de tip pdf in folder-ul “reportClienti” parcurgand intreaga lista de client din DB se va crea un tabel cu 3 coloane(id, nume, adresa) si pentru fiecare client se va adauga un rand nou in tabel cu datele sale.

**4.13. Clasa ProdusBLL** are ca variabile de instanta un obiect de tip ProdusDAO pe care il creeaza, indexul produsului curent, indexul pdf-ului creat pentru report-ul produsului si un obiect de tipul Validator care este utilizat pentru a valida informatiile de spre produs. De asemenea, obiectul ProdusDAO este folosit pentru a apela metodele din acea clasa(find, insert, delete, update) si in cazul in care o instructiune de tipul sql nu a avut succes se va prinde o exceptie cu mesajul corespunzator. Metoda de insertProdus() are o functionalitate putin diferita fata de celelalte, ea nu doar apeleaza metoda de insert din clasa ProdusDAO ci verifica daca produsul a fost inserat. In cazul in care nu a fost inserat se va insera, iar in caz contrar se va face un update asupra cantitatii produsului; noua cantitate se va adauga la cea deja existenta.

*if (produs != null) {*

*String update = "";*

*update = update + (produs.getCantitate() + Integer.parseInt(fld2));*

*try {*

*updateProdus(update, fld1);*

De asemenea, tot aici se afla si metoda care creeaza reportul pentru produs, creeaza un document de tip pdf in folder-ul “reportProdus” parcurgand intreaga lista de produse din DB se va crea un tabel cu 4 coloane(id, numeProdus, cantitate, pret) si pentru fiecare produs se va adauga un rand nou in tabel cu informatiile sale.

**4.13. Clasa ComandaBLL** are ca variabile de instanta doua obiecte de tip DAO(Comanda si produsDorit) pe care le creeaza, indexul comenzii curente, indexul pdf-ului creat pentru report-ul comenzii, un ArrayList in care se vor pastra comenzile effectuate, si cate un obiect de tipul ProdusBLL si ClientBLL care se instantiaza. De asemenea, obiectul ComandaDAO este folosit pentru a apela metodele din acea clasa(find, insert, delete, update) si in cazul in care o instructiune de tipul sql nu a avut succes se va prinde o exceptie cu mesajul corespunzator. Metoda selectOrder este cea mai importanta din aceasta clasa pentru ca aceasta creeaza si efectueaza comanda: in conditiile in care exista in depozit cel cantitatea ceruta de client a produsului, insereaza in comandaDAO id-ul comenzii si al clientului, apoi insereaza in ProdusDoritDAO id,ul comenzii, id-ul clientului, cantitatea comandata si pretul total si creeaza o chitanta cu ajutorul metodei releaseReceipt(). In cazul in care nu s-a efectuat comanda din cauza faptului ca nu exista cantitatea ceruta se va elibera o chitanta care cuprinde un mesaj corespunzator.

*if (produs.getCantitate() - Integer.parseInt(fld3) >= 0) {*

*try {*

*comandaDAO.insert(c);*

*double suma = Integer.parseInt(fld3) \* produs.getPret();*

*pdDAO.insert(iComanda + ", " + produs.getId() + ", " + fld3 + "," + suma);*

*p = new ProdusDorit(iComanda, produs.getId(), Integer.parseInt(fld3), suma);*

*releaseReceipt(produs.getNumeProdus(), Integer.parseInt(fld3),*

*produs.getPret() \* Integer.parseInt(fld3), true);*

De asemenea, si in aceasta clasa exista metoda care creeaza reportul pentru comenzi.

**4.14. Clasa ProdusDoritBLL** are ca variabile de instanta un obiect de tip ProdusDoritDAO pe care il creeaza. Obiectul ProdusDoritDAO este folosit pentru a apela metodele din acea clasa(find, insert, delete, update) si in cazul in care o instructiune de tipul sql nu a avut succes se va prinde o exceptie cu mesajul corespunzator.

**4.15. Clasa SqlCommands** instanteaza toate obiectele de tipul BLL implementate in clase mentionate mai sus, are un ArrayList care contine lista de produse comandate si un string in care se va afla fisierul pentru parsare. Metoda parseInput() este cea care realizeaza parsarea fiecare linii a fisierului dat ca intrare.

*public void parseInput(String f) {*

*String[] fileRead = f.split("[:,]");*

*int i = 0;*

*while (fileRead != null && i < fileRead.length) {*

*if (fileRead[i].charAt(0) == ' ') {*

*fileRead[i] = fileRead[i].substring(1);*

*}*

*file.add(fileRead[i]);*

*i++;*

*}*

*}*

Cele trei metode ale clasei: CommandClient(), CommandProduct(), CommandOrder sunt cele care iau decizia pentru fiecare Obiect de tipul corespunzator, ce instructiune de tipul sql se va efectua in functie de cerintele utilizatorului aflate in fisier. De asemenea, functia principala command() va decide care dintre cele 3 metode vor fi apelate in functie de cuvintele cheie “client”, “product”, “order” care se regasesc in comanda data de catre utilizator.

*public void command() throws Exception {*

*if (file.get(0).contains("client")) {*

*commandClient();*

*} else if (file.get(0).contains("product")) {*

*commandProduct();*

*} else if (file.get(0).contains("Order") || file.get(0).contains("order")) {*

*commandOrder();*

*}*

*}*

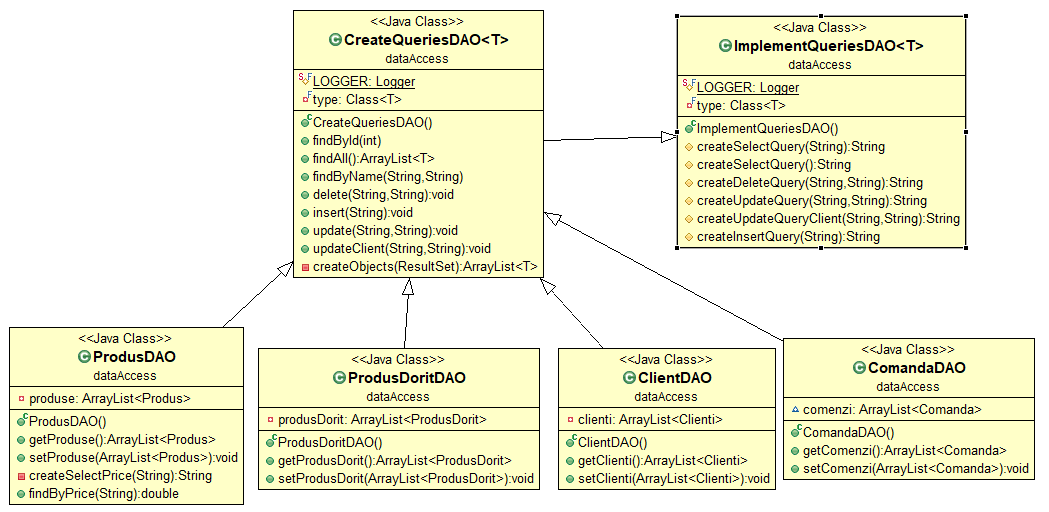
**4.16. Clasa ValidatorClient** este clasa in care se implementeaza cele doua metode validatorClientNume() si validatorClientAdresa(). Acestea sunt utilizate pentru a verifica daca datele introduse de catre utilizator pentru fiecare client sunt valide, mai precis numele sa contina doar litere si adresa litere si eventual cifre.

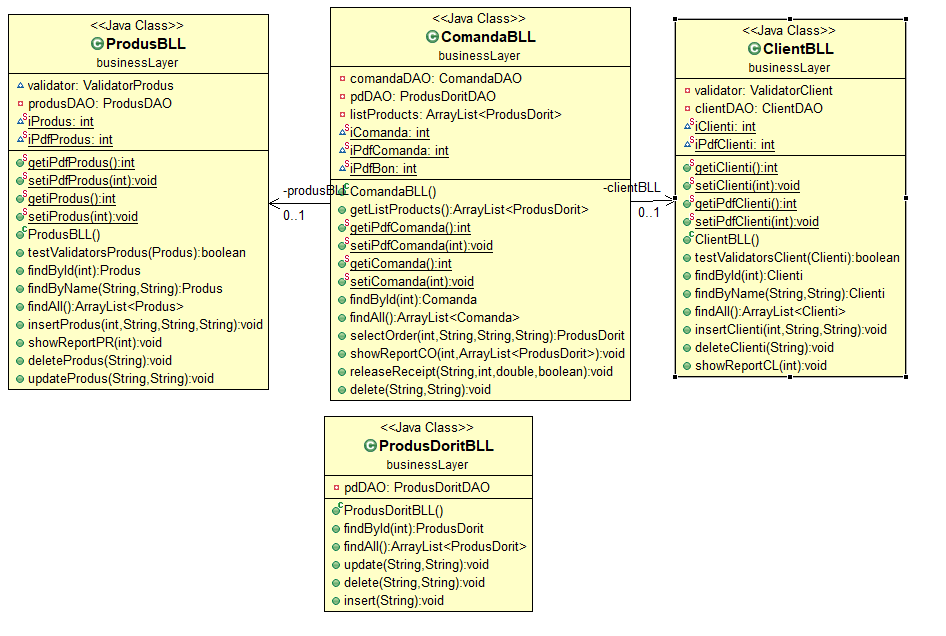
**4.17. Clasa ValidatorProdus** este clasa in care se implementeaza metoda de validare pentru numele produsului introdus, de asemenea, validarile pentru cantitatea produsului si pret se vor face in metoda CommandProduct() din clasa Sql Commands.

**4.18. Clasa Main** este cea in care se citeste fisierul introdus de catre utilizator, se instantiaza clasa SqlCommands si cele doua metode principale implementate in aceasta clasa: command() si parseInput() pentru a pune aplicatia in functiune.

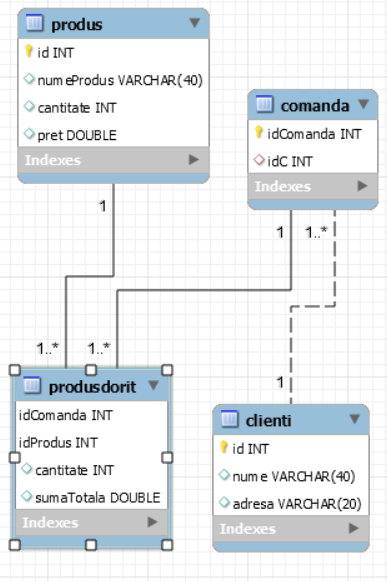
**5.Diagrame**

UML -> DAO and BLL:





REVERSE ENGINEER:



**6. Concluzii**

In concluzie aplicatia implementata pentru managementul unui depozit are aplicabilitate si in viata reala prin faptul ca poate fi utilizata pentru a tine evidenta clientilor care comanda, a produselor care se afla in stocul unui depozit si a comenzilor care s-au efectuat. De asemenea, se pot efectua operatii cum ar fi adaugarea unui nou client in evidenta in momentul in care acesta plaseaza o comanda sau stergerea acestuia, la fel si adaugarea produselor noi in depozit si stergerea lor automata in momentul in care nu se mai gaseste acel produs pe stoc. Toate acestea faciliteaza intretinerea si gestionarea unui astfel de depozit in viala reala.

**7. Bibliografie**

http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/ o <http://theopentutorials.com/tutorials/java/jdbc/jdbc-mysql-create-database-example/>

https://dzone.com/articles/layers-standard-enterprise

http://tutorials.jenkov.com/java-reflection/index.html • Creating PDF files in Java o https://www.baeldung.com/java-pdf-creation

https://www.baeldung.com/javadoc

https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html https://dev.mysql.com/doc/refman/5.7/en/using-mysqldump.html