Aplicatie pentru procesarea comenzilor

Universitatea Tehnică Cluj-Napoca

Facultatea de Calculatoare și Automatică

Departamenul de Calculatoare și Tehnologia informației

Nume: Muresan George

Grupa: 30226

Materie: Tehnici de programare1.Obiectivul temei

Cerinta proiectului este: Procesarea comenzilor dintr-un depozit.

Obiectivul temei este de a implementa o aplicatie cu interfata grafica pentru a fi folosita ca aplicatie ce foloseste o baza de date si implementeaza opreatii cu datele.

O bază de date, uneori numită și bancă de date (abreviat BD), reprezintă o modalitate de stocare a unor [informații](https://ro.wikipedia.org/wiki/Informa%C8%9Bie) și [date](https://ro.wikipedia.org/wiki/Dat%C4%83) pe un suport extern (un dispozitiv de stocare), cu posibilitatea extinderii ușoare și a regăsirii rapide a acestora. La prima vedere sarcina poate părea banală. Totuși, în condițiile în care este vorba de a lucra cu milioane de elemente, fiecare putând consta din mari cantități de date care trebuie accesate simultan prin [Internet](https://ro.wikipedia.org/wiki/Internet) de către mii de utilizatori răspândiți pe întreg globul; și în condițiile când disponibilitatea aplicației și datelor trebuie să fie permanentă (de ex. pentru a nu pierde ocazia de a încheia afaceri), soluțiile bune nu sunt de loc simple.

De obicei o bază de date este memorată într-unul sau mai multe [fișiere](https://ro.wikipedia.org/wiki/Fi%C8%99ier). Bazele de date sunt manipulate cu ajutorul [sistemelor de gestiune a bazelor de date](https://ro.wikipedia.org/wiki/Sistem_de_gestiune_a_bazelor_de_date).

Cel mai răspândit tip de baze de date este cel relațional, în care datele sunt memorate în [tabele](https://ro.wikipedia.org/wiki/Tabel). Pe lânga tabele, o bază de date relațională mai poate conține: indecși, proceduri stocate, declanșatori, utilizatori și grupuri de utilizatori, tipuri de date, mecanisme de securitate și de gestiune a tranzacțiilor etc.

Alte tipuri de baze de date sunt modelul ierarhic, modelul orientat pe obiecte și, mai nou, modelul [XML](https://ro.wikipedia.org/wiki/XML).

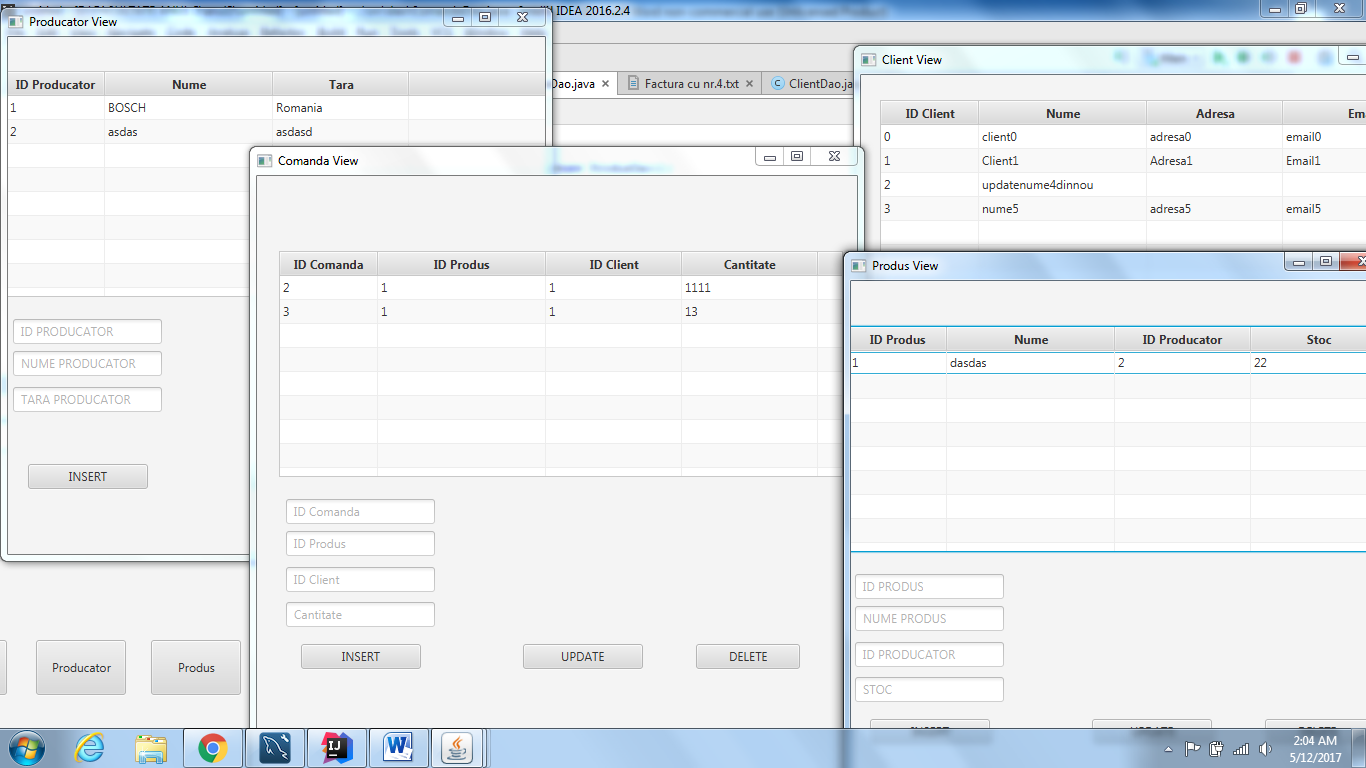
2.a) Analiza problemei

În partea de analiză a problemei trebuie depistate principalele clase, respectiv caracteristicile și funcționlitățile acestora și realizate legăturile dintre ele dar si implementarea opeartiilor. Programarea orientata pe obiecte oferă avantajul de a putea începe dezvoltarea unui proiect folosind doar informațiile de la suprafață, fără a fi nevoie de implementarea efectivă a funcționalităților. Această strategie este cunoscută sub numele “Top-Down”. Ea este foarte avantajoasă din punctul de vedere al găsirii componentelor constituente, deoarece pot fi găsite, relativ ușor, structuri cu o legătură directă în lumea reală( obiecte, acțiuni etc.). Din păcate această versatilitate vine cu prețul complexității, ea crescând spre măsură ce se avansează pe nivelele inferioare.In subiectul de fata trebuie sa analizam rezolvarea procesarii comenzilor dintr-un deposit.

2.b) Modelarea

In partea de modelare interfața cu utilizatorul este stratul cel mai de deasupra al unei aplicații, de aceea realizarea ei a reprezentat punctul de pornire al acestui proiect. În primul rând, ea trebuie să fie cât mai intuitivă și permisivă în ceea ce privește modul de introducere a datelor.

Dupa cum putem observa, datele de intrare si de iesire reies foarte bine din imaginea de mai jos.



2.c) Scenarii

Scenariile care pot sa apară și sa ducă la oprirea din funcționare a aplicatiei sau eventual la probelme de implementare pot fii:

* Stergerea anumitelor date care sunt chei straine in alte tabele
* Introducerea incorecta a datelor
* Supraincarcarea unor campuri

2.d) Cazuri de utilizare

Aplicația se poate folosi pentru:

* Lucrul cu bazele de date
* Simularea procesarii comenzilor dintr-un depozit

3.Proiectare

**Interfața grafică** este realizată cu JavaFX Scene Builder gasit in IntelliJ(una dintre ele), iar cealalta este realizata manual. Acesta generează un fișier .fxml care este introdus în proiect. ActionEvent-urile ale fiecărui buton sunt setate în JavaFX Scene Builder prin introducerea în câmpul “On Action” numele metodei care trebuie apelata și id-ul Text field-urilor, în câmpul “id”.

Dupa cum puteti observa, mai jos este reprezentata **diagrama UML** a claselor .

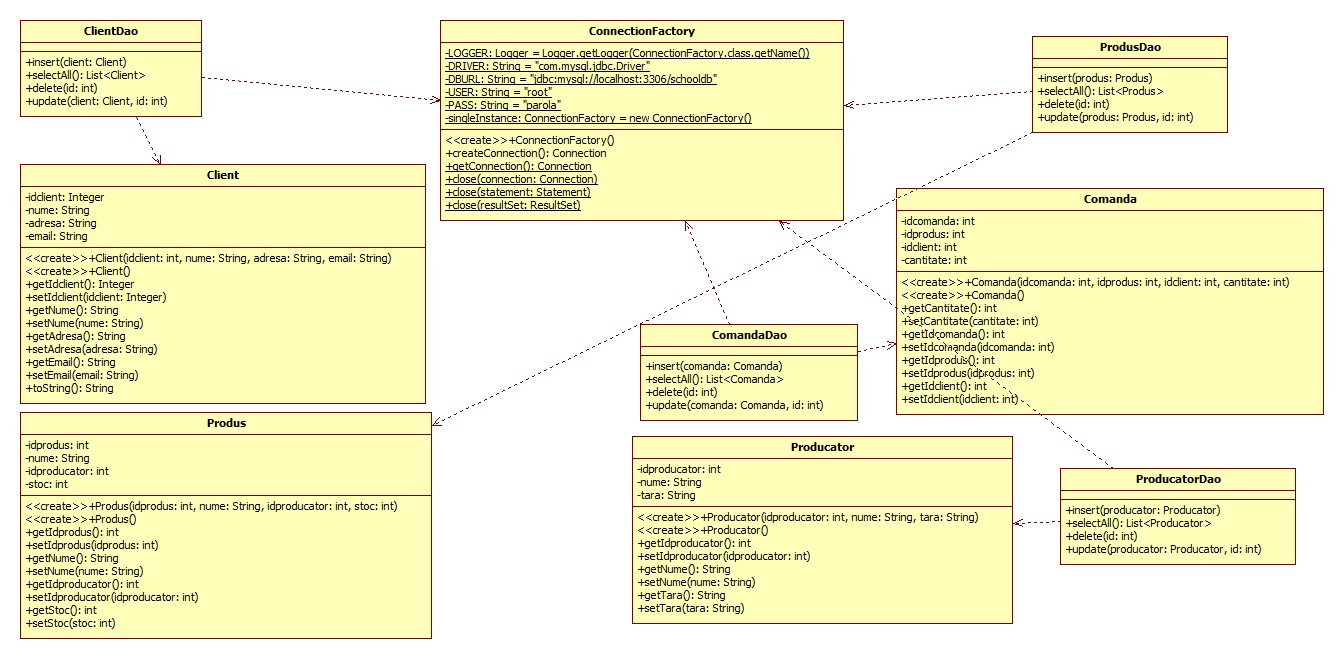
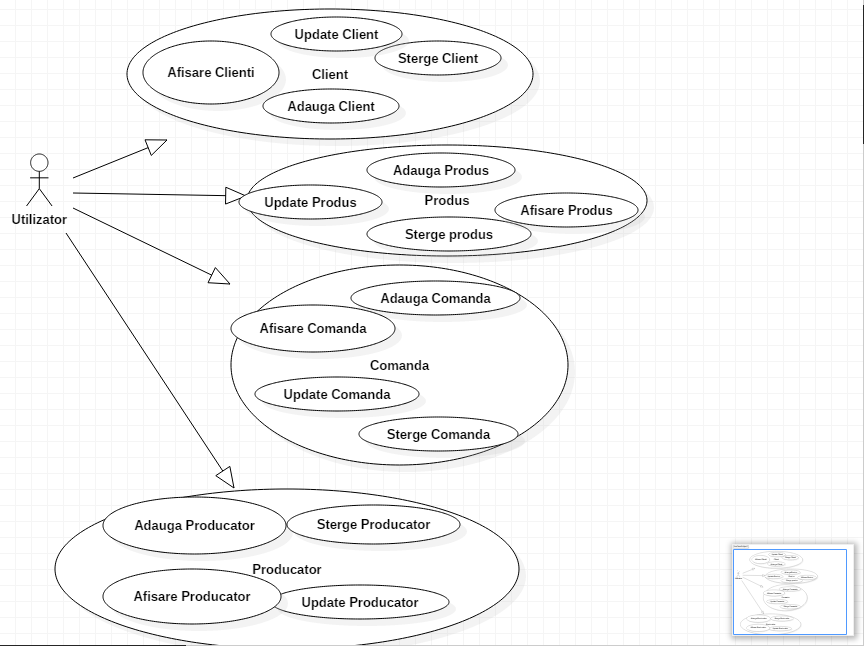


Diagrama **use case:**



Aplicatia mea foloseste 14 clase, fiecare fiindu-mi necesara pentru indeplinirea temei si mai exact “procesarea comezilor “.

**Clasa Client** este una dintre cele 14 clase, aceasta primeste ca attribute idclient,nume,adresa,email mai exact ceea ce imi definste mie un client.Aici am doar metode simple : gettere si settere si constructorul adecvat

**Clasa Comanda** este urmatoare clasa pe care am implementat-o si ea contine proprietatile unei comenzi si are atributele idcomanda ,idprodus,idclient, cantitate.Similar ca si clasa Client are doar metode simple .

**Clasa Producator** o alta clasa din cele 14, ea contine proprietatile unui producator si are atributele idporducator, nume, tara.Similar ca cele de mai sus implemnteaza doar metode simple .

**Clasa Produs** este similara cu cele de mai sus dar implemnteaza metodele adecvate si are ca attribute idprodus, nume, idproducator, stoc.

Toate acestea se gasesc in package-ul model.

Urmeaza package-ul dao care va implementa toate clasele din model numai ca in acestea vom avea doar metodele necesare pentru procesarea comenzilor si accesul la date.

* **ClientDao**
* **ProducatorDao**
* **ComandaDao**
* **ProdusDao**

Am mai creat un package, adica connection cel care se refera doar la o singura clasa unde se va face conexiunea cu baza de date.

* **ConnectionFactory**

Un alt package creat este bill unde am o clasa care imi va genera o facture.txt.

* **Factura**

Si ultimul package sample unde am mainul si tot ce tine de interfata grafica.

* **ControllerClientView**
* **ControllerComandaView**
* **ControllerMainWindow**
* **ControllerProducatorView**
* **ControllerProdusView**

4. Implementare si testare

Singurele metode implementate se gasesc in package-ul dao in fiecare clasa de aici .

**Clasa ClientDao**

**package** dao;  
**import** connection.ConnectionFactory;  
**import** model.Client;  
  
  
**import** java.sql.\*;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.logging.Level;  
  
**import static** com.sun.xml.internal.ws.spi.db.BindingContextFactory.***LOGGER***;  
  
*/\*\*  
 \* Created by George on 5/10/2017.  
 \* Clasa ClientDao implemeteaza metodele : insert, delete, update si selectall  
 \* Este folosita pentru accesarea si procesarea datelor din baza de date  
 \*/***public class** ClientDao {  
 */\*\*  
 \* Pentru a insera un client nou in baza de date  
 \** ***@param client*** *clientul nou ce va fi inserat  
 \*/* **public void** insert(Client client)  
 {  
 Connection connection=**null**;  
 PreparedStatement preparedStatement=**null**;  
 **try** {  
 connection=ConnectionFactory.*getConnection*();  
 preparedStatement=connection.prepareStatement(**"INSERT INTO client (idclient,nume,adresa,email)"** + **" VALUES (?,?,?,?)"**);  
 preparedStatement.setInt(1,client.getIdclient());  
 preparedStatement.setString(2,client.getNume());  
 preparedStatement.setString(3,client.getAdresa());  
 preparedStatement.setString(4,client.getEmail());  
 preparedStatement.executeUpdate();  
 System.***out***.println(**"INSERT INTO client(idclient,nume,adresa,email)"**+ **"VALUES (?,?,?,?)"**);  
 }**catch**(Exception e)  
 {  
 e.printStackTrace();  
  
 }**finally**{  
 **if**(preparedStatement!=**null**)  
 **try** {  
 preparedStatement.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 **if**(connection!=**null**)  
 **try** {  
 connection.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 }  
  
 }  
 */\*\*  
 \* Pentru afisarea datelor fiecarui client  
 \** ***@return*** *clienti  
 \*/* **public** List<Client> selectAll() **throws** SQLException {  
 List<Client> clienti=**new** ArrayList<Client>();  
 Connection connection=**null**;  
 Statement statement =**null**;  
 ResultSet rs=**null**;  
 **try** {  
 connection=ConnectionFactory.*getConnection*();  
 statement=connection.createStatement();  
 rs=statement.executeQuery(**"SELECT \* FROM client"**);  
 **while**(rs.next())  
 {  
 Client client =**new** Client();  
 client.setIdclient(rs.getInt(**"idclient"**));  
 client.setNume(rs.getString(**"nume"**));  
 client.setAdresa(rs.getString(**"adresa"**));  
 client.setEmail(rs.getString(**"email"**));  
 clienti.add(client);  
  
 }  
  
 }**catch**(Exception e){  
 e.printStackTrace();  
 }**finally** {  
 **if**(rs!=**null**)  
 {  
 **try**{  
 rs.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 }  
 **if**(statement!=**null**)  
 {  
 **try**{  
 statement.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 }  
 **if**(connection!=**null**)  
 {  
 **try**{  
 connection.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 }  
 }  
 **return** clienti;  
 }  
 */\*\*  
 \* Pentru stergerea unui client cu un id specificat  
 \** ***@param id*** *id-ul clientului care va fi sters  
 \*/* **public void** delete (**int** id)  
 {  
 Connection connection=**null**;  
 PreparedStatement preparedStatement=**null**;  
 **try** {  
 connection=ConnectionFactory.*getConnection*();  
 preparedStatement=connection.prepareStatement(**"DELETE FROM client WHERE idclient = ?"**);  
 preparedStatement.setInt(1,id);  
 preparedStatement.executeUpdate();  
 System.***out***.println(**"stergere client"**);  
  
 }**catch**(Exception e)  
 {  
 e.printStackTrace();  
 }**finally** {  
 **if**(preparedStatement!= **null**)  
 {  
 **try** {  
 preparedStatement.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 **if**(connection!=**null**)  
 {  
 **try** {  
 connection.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 }  
 */\*\*  
 \* Pentru reactualizarea datelor unui client  
 \** ***@param id*** *id-ul clientului care va fi updatat  
 \** ***@param client*** *clientul care va fi updatat  
 \*/* **public void** update (Client client,**int** id)  
 {  
 Connection connection=**null**;  
 PreparedStatement preparedStatement=**null**;  
 **try** {  
 connection=ConnectionFactory.*getConnection*();  
 preparedStatement=connection.prepareStatement(**"UPDATE client SET "**+ **"nume = ?,adresa = ?,email= ? WHERE idclient = ?"**);  
  
 preparedStatement.setInt(4,id);  
 preparedStatement.setString(1,client.getNume());  
 preparedStatement.setString(2,client.getAdresa());  
 preparedStatement.setString(3,client.getEmail());  
 preparedStatement.executeUpdate();  
  
 System.***out***.println(**"update client"**);  
  
 }**catch**(Exception e)  
 {  
 e.printStackTrace();  
 }**finally** {  
 **if**(preparedStatement!= **null**)  
 {  
 **try** {  
 preparedStatement.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 **if**(connection!=**null**)  
 {  
 **try** {  
 connection.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
  
  
}

Se poate observa ca am folosit javadoc pentru explicarea fiecarei metode si oferirea informatiilor utile pentru intelegerea si utilizarea claselor.

Ca prima metoda implmentata putem observa metoda insert, care imi va insera in baza de date un client nou ce il primeste ca parametru pe baza unui Query din Sql.

A doua metoda implementata este metoda selectall care imi va genera toti clienti din tabel, acesta metoda este de tipul unei liste de clienti si va returna tot o lista de clienti, si aceasta metoda e implementata tot pe baza unei interogari sql si pur si simplu merg prin fiecare client ii iau datele si le adaug listei mele de clienti.

A treia metoda implementata primeste ca parameter un id si se numeste delete, dupa cum putem deduce ca sterge clientul cu id-ul introdus.Si aceasta este la randul ei facuta printr-o interogare sql.

A patra si ultima metoda se numeste update si este folosita pentru reactualizarea datelor unui client.Aceasta este conceputa tot pe baza unei interogari SQL.

La fel si in celelalte clase din package-ul dao sunt aceleasi metode doar updatate corespunzator.

**package** dao;  
  
**import** connection.ConnectionFactory;  
**import** model.Client;  
**import** model.Comanda;  
**import** model.Producator;  
**import** model.Produs;  
**import** bill.Factura;  
  
**import** java.sql.\*;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.logging.Level;  
  
**import static** com.sun.xml.internal.ws.spi.db.BindingContextFactory.***LOGGER***;  
  
*/\*\*  
 \* Created by George on 5/11/2017.  
 \*/***public class** ComandaDao {  
 **public void** insert(Comanda comanda)  
 {  
 Connection connection=**null**;  
 PreparedStatement preparedStatement=**null**;  
 **try** {  
 *//insattiez un obicet de tipul produs  
 //cauta dupa id  
 //unde l-a gasit iau stocul d la linia aia si verific daca stocul e mai mare decat cantiattea  
 //daca stocul e mai mare inserez comanda si fac update la stoc* connection=ConnectionFactory.*getConnection*();  
 preparedStatement=connection.prepareStatement(**"INSERT INTO comanda (idcomanda,idprodus,idclient,cantitate)"** + **" VALUES (?,?,?,?)"**);  
 preparedStatement.setInt(1,comanda.getIdcomanda());  
 preparedStatement.setInt(2,comanda.getIdprodus());  
 preparedStatement.setInt(3,comanda.getIdclient());  
 preparedStatement.setInt(4,comanda.getCantitate());  
  
 preparedStatement.executeUpdate();  
 System.***out***.println(**"INSERT INTO comanda"**);  
 }**catch**(Exception e)  
 {  
 e.printStackTrace();  
  
 }  
  
 ProdusDao pdao=**new** ProdusDao();  
 **try** {  
 List<Produs> produse=pdao.selectAll();  
 **for** (Produs produs:produse) {  
 **if**(comanda.getIdprodus()==produs.getIdprodus()) {  
 **if** (comanda.getCantitate() < produs.getStoc()) {  
 **int** nval;  
 nval = produs.getStoc() - comanda.getCantitate();  
 produs.setStoc(nval);  
  
 **int** idpp=produs.getIdprodus();  
 produs.setIdprodus(idpp);  
  
 String numepp=produs.getNume();  
 produs.setNume(numepp);  
  
 **int** idprpr=produs.getIdproducator();  
 produs.setIdproducator(idprpr);  
  
 pdao.update(produs,produs.getIdprodus());  
 } **else** System.***out***.println(**"nu s-a putut efectua comanda"**);  
 }  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
  
 **finally**{  
 **if**(preparedStatement!=**null**)  
 **try** {  
 preparedStatement.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 **if**(connection!=**null**)  
 **try** {  
 connection.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 }  
  
 **try** {  
 Factura.*metoda*(comanda);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 *//List<Client> clienti=cld.selectAll();  
 // for (Client client:clienti) {  
 // System.out.println(client.getIdclient()+" "+client.getAdresa()+" "+client.getNume()+" ");  
  
 //}* }  
  
 **public** List<Comanda> selectAll() **throws** SQLException {  
 List<Comanda> comenzi=**new** ArrayList<Comanda>();  
 Connection connection=**null**;  
 Statement statement =**null**;  
 ResultSet rs=**null**;  
 **try** {  
 connection=ConnectionFactory.*getConnection*();  
 statement=connection.createStatement();  
 rs=statement.executeQuery(**"SELECT \* FROM comanda"**);  
 **while**(rs.next())  
 {  
 Comanda comanda =**new** Comanda();  
 comanda.setIdcomanda(rs.getInt(**"idcomanda"**));  
 comanda.setIdprodus(rs.getInt(**"idprodus"**));  
 comanda.setIdclient(rs.getInt(**"idclient"**));  
 comanda.setCantitate(rs.getInt(**"cantitate"**));  
 *// client.setEmail(rs.getString("email"));  
 //clienti.add(client);  
 //produse.add(produs);  
 //producatori.add(producator);* comenzi.add(comanda);  
  
 }  
  
 }**catch**(Exception e){  
 e.printStackTrace();  
 }**finally** {  
 **if**(rs!=**null**)  
 {  
 **try**{  
 rs.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 }  
 **if**(statement!=**null**)  
 {  
 **try**{  
 statement.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 }  
 **if**(connection!=**null**)  
 {  
 **try**{  
 connection.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 }  
 }  
 **return** comenzi;  
 }  
 **public void** delete (**int** id)  
 {  
 Connection connection=**null**;  
 PreparedStatement preparedStatement=**null**;  
 **try** {  
 connection=ConnectionFactory.*getConnection*();  
 preparedStatement=connection.prepareStatement(**"DELETE FROM comanda WHERE idcomanda = ?"**);  
 preparedStatement.setInt(1,id);  
 preparedStatement.executeUpdate();  
 System.***out***.println(**"stergere comanda"**);  
  
 }**catch**(Exception e)  
 {  
 e.printStackTrace();  
 }**finally** {  
 **if**(preparedStatement!= **null**)  
 {  
 **try** {  
 preparedStatement.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 **if**(connection!=**null**)  
 {  
 **try** {  
 connection.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 }  
 **public void** update (Comanda comanda,**int** id)  
 {  
 Connection connection=**null**;  
 PreparedStatement preparedStatement=**null**;  
 **try** {  
 connection=ConnectionFactory.*getConnection*();  
 preparedStatement=connection.prepareStatement(**"UPDATE comanda SET "**+ **"idprodus = ?,idclient = ?,cantitate = ?, WHERE idcomanda = ?"**);  
  
  
 preparedStatement.setInt(1,comanda.getIdprodus());  
 preparedStatement.setInt(2,comanda.getIdclient());  
 preparedStatement.setInt(3,comanda.getCantitate());  
 preparedStatement.setInt(4,id);  
  
 preparedStatement.executeUpdate();  
  
 System.***out***.println(**"update comanda"**);  
  
 }**catch**(Exception e)  
 {  
 e.printStackTrace();  
 }**finally** {  
 **if**(preparedStatement!= **null**)  
 {  
 **try** {  
 preparedStatement.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 **if**(connection!=**null**)  
 {  
 **try** {  
 connection.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
}

**package** dao;  
  
**import** connection.ConnectionFactory;  
**import** model.Client;  
**import** model.Producator;  
**import** model.Produs;  
  
**import** java.sql.\*;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.logging.Level;  
  
**import static** com.sun.xml.internal.ws.spi.db.BindingContextFactory.***LOGGER***;  
  
*/\*\*  
 \* Created by George on 5/11/2017.  
 \*/***public class** ProducatorDao {  
 **public void** insert(Producator producator)  
 {  
 Connection connection=**null**;  
 PreparedStatement preparedStatement=**null**;  
 **try** {  
 connection=ConnectionFactory.*getConnection*();  
 preparedStatement=connection.prepareStatement(**"INSERT INTO producator (idproducator,nume,tara)"** + **" VALUES (?,?,?)"**);  
 preparedStatement.setInt(1,producator.getIdproducator());  
 preparedStatement.setString(2,producator.getNume());  
 preparedStatement.setString(3,producator.getTara());  
  
 preparedStatement.executeUpdate();  
 System.***out***.println(**"INSERT INTO producator"**);  
 }**catch**(Exception e)  
 {  
 e.printStackTrace();  
  
 }**finally**{  
 **if**(preparedStatement!=**null**)  
 **try** {  
 preparedStatement.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 **if**(connection!=**null**)  
 **try** {  
 connection.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 }  
  
 }  
  
 **public** List<Producator> selectAll() **throws** SQLException {  
 List<Producator> producatori=**new** ArrayList<Producator>();  
 Connection connection=**null**;  
 Statement statement =**null**;  
 ResultSet rs=**null**;  
 **try** {  
 connection=ConnectionFactory.*getConnection*();  
 statement=connection.createStatement();  
 rs=statement.executeQuery(**"SELECT \* FROM producator"**);  
 **while**(rs.next())  
 {  
 Producator producator =**new** Producator();  
 producator.setIdproducator(rs.getInt(**"idproducator"**));  
 producator.setNume(rs.getString(**"nume"**));  
 producator.setTara(rs.getString(**"tara"**));  
 *// client.setEmail(rs.getString("email"));  
 //clienti.add(client);  
 //produse.add(produs);* producatori.add(producator);  
  
 }  
  
 }**catch**(Exception e){  
 e.printStackTrace();  
 }**finally** {  
 **if**(rs!=**null**)  
 {  
 **try**{  
 rs.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 }  
 **if**(statement!=**null**)  
 {  
 **try**{  
 statement.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 }  
 **if**(connection!=**null**)  
 {  
 **try**{  
 connection.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 }  
 }  
 **return** producatori;  
 }  
 **public void** delete (**int** id)  
 {  
 Connection connection=**null**;  
 PreparedStatement preparedStatement=**null**;  
 **try** {  
 connection=ConnectionFactory.*getConnection*();  
 preparedStatement=connection.prepareStatement(**"DELETE FROM producator WHERE idproducator = ?"**);  
 preparedStatement.setInt(1,id);  
 preparedStatement.executeUpdate();  
 System.***out***.println(**"stergere producator"**);  
  
 }**catch**(Exception e)  
 {  
 e.printStackTrace();  
 }**finally** {  
 **if**(preparedStatement!= **null**)  
 {  
 **try** {  
 preparedStatement.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 **if**(connection!=**null**)  
 {  
 **try** {  
 connection.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 }  
 **public void** update (Producator producator,**int** id)  
 {  
 Connection connection=**null**;  
 PreparedStatement preparedStatement=**null**;  
 **try** {  
 connection=ConnectionFactory.*getConnection*();  
 preparedStatement=connection.prepareStatement(**"UPDATE producator SET "**+ **"nume = ?,tara = ? WHERE idproducator = ?"**);  
  
  
 preparedStatement.setString(1,producator.getNume());  
 preparedStatement.setString(2,producator.getTara());  
 preparedStatement.setInt(3,id);  
  
 preparedStatement.executeUpdate();  
  
 System.***out***.println(**"update producator"**);  
  
 }**catch**(Exception e)  
 {  
 e.printStackTrace();  
 }**finally** {  
 **if**(preparedStatement!= **null**)  
 {  
 **try** {  
 preparedStatement.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 **if**(connection!=**null**)  
 {  
 **try** {  
 connection.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
}

**package** dao;  
  
**import** connection.ConnectionFactory;  
**import** model.Client;  
**import** model.Producator;  
**import** model.Produs;  
  
**import** java.sql.\*;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.logging.Level;  
  
**import static** com.sun.xml.internal.ws.spi.db.BindingContextFactory.***LOGGER***;  
  
*/\*\*  
 \* Created by George on 5/11/2017.  
 \*/***public class** ProdusDao {  
 **public void** insert(Produs produs)  
 {  
 Connection connection=**null**;  
 PreparedStatement preparedStatement=**null**;  
 **try** {  
 connection=ConnectionFactory.*getConnection*();  
 preparedStatement=connection.prepareStatement(**"INSERT INTO produs (idprodus,nume,idproducator,stoc)"** + **" VALUES (?,?,?,?)"**);  
 preparedStatement.setInt(1,produs.getIdprodus());  
 preparedStatement.setString(2,produs.getNume());  
 preparedStatement.setInt(3,produs.getIdproducator());  
 preparedStatement.setInt(4,produs.getStoc());  
  
 preparedStatement.executeUpdate();  
 System.***out***.println(**"INSERT INTO produs"**);  
 }**catch**(Exception e)  
 {  
 e.printStackTrace();  
  
 }**finally**{  
 **if**(preparedStatement!=**null**)  
 **try** {  
 preparedStatement.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 **if**(connection!=**null**)  
 **try** {  
 connection.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 }  
  
 }  
  
 **public** List<Produs> selectAll() **throws** SQLException {  
 List<Produs> produse=**new** ArrayList<Produs>();  
 Connection connection=**null**;  
 Statement statement =**null**;  
 ResultSet rs=**null**;  
 **try** {  
 connection=ConnectionFactory.*getConnection*();  
 statement=connection.createStatement();  
 rs=statement.executeQuery(**"SELECT \* FROM produs"**);  
 **while**(rs.next())  
 {  
 Produs produs =**new** Produs();  
 produs.setIdprodus(rs.getInt(**"idprodus"**));  
 produs.setNume(rs.getString(**"nume"**));  
 produs.setIdproducator(rs.getInt(**"idproducator"**));  
 produs.setStoc(rs.getInt(**"stoc"**));  
 *// client.setEmail(rs.getString("email"));  
 //clienti.add(client);* produse.add(produs);  
  
 }  
  
 }**catch**(Exception e){  
 e.printStackTrace();  
 }**finally** {  
 **if**(rs!=**null**)  
 {  
 **try**{  
 rs.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 }  
 **if**(statement!=**null**)  
 {  
 **try**{  
 statement.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 }  
 **if**(connection!=**null**)  
 {  
 **try**{  
 connection.close();  
 }**catch**(SQLException e)  
 {  
 e.printStackTrace();  
 }  
 }  
 }  
 **return** produse;  
 }  
 **public void** delete (**int** id)  
 {  
 Connection connection=**null**;  
 PreparedStatement preparedStatement=**null**;  
 **try** {  
 connection=ConnectionFactory.*getConnection*();  
 preparedStatement=connection.prepareStatement(**"DELETE FROM produs WHERE idprodus = ?"**);  
 preparedStatement.setInt(1,id);  
 preparedStatement.executeUpdate();  
 System.***out***.println(**"stergere produs"**);  
  
 }**catch**(Exception e)  
 {  
 e.printStackTrace();  
 }**finally** {  
 **if**(preparedStatement!= **null**)  
 {  
 **try** {  
 preparedStatement.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 **if**(connection!=**null**)  
 {  
 **try** {  
 connection.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 }  
 **public void** update (Produs produs,**int** id)  
 {  
 Connection connection=**null**;  
 PreparedStatement preparedStatement=**null**;  
 **try** {  
 connection=ConnectionFactory.*getConnection*();  
 preparedStatement=connection.prepareStatement(**"UPDATE produs SET "**+ **"nume = ?,idproducator = ?,stoc = ?, WHERE idprodus = ?"**);  
  
  
 preparedStatement.setString(1,produs.getNume());  
 preparedStatement.setInt(2,produs.getIdproducator());  
 preparedStatement.setInt(3,produs.getStoc());  
 preparedStatement.setInt(4,id);  
  
 preparedStatement.executeUpdate();  
  
 System.***out***.println(**"update produs"**);  
  
 }**catch**(Exception e)  
 {  
 e.printStackTrace();  
 }**finally** {  
 **if**(preparedStatement!= **null**)  
 {  
 **try** {  
 preparedStatement.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 **if**(connection!=**null**)  
 {  
 **try** {  
 connection.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
}

5.Concluzii

Ca o concluzie, acest proiect m-a ajutat sa imi consolidez cunostintele de programare orientata pe obiect dobandite in primul semestru si sa imi organizez munca in baza paradigmelor POO de asemenea acest proiect m-a ajutat sa inteleg mult mai bine utilizarea bazelor de date.

Sincer, singurul proiect care mi s-a parut atractiv.

In cadrul dezvoltarilor ulterioare se pot aminti urmatoarele: imbunatatirea interfetei grafice,adaugarea de teste pentru introducerea datelor, etc.