Universitatea Tehnică Cluj-Napoca

Documentaţie

Tema 3

Order Management

Mureşan Ioana Diana

CTI-Romana

Gr.30227

Cuprins

1.Cerinte Functionale................................................... 3

2.Obiective ...................................................................3

2.1.Obiectiv Principal.................................................3

2.2.Obiective Secundare............................................3

3.Analiza Problemei ......................................................3

4.Proiectare ..................................................................5

4.1.Structura proiectului...............................................5

4.2.Diagrama de clase................................................6

5.Implementare.............................................................8

6.Rezultate ..................................................................16

7.Concluzii si Dezvoltari Ulterioare..............................22

8.Bibliografie ..............................................................22

**1.Cerinte Functionale**

Proiectarea unei aplicatii de tipul Order Management care proceseaza comenzile unui client pentru un depozit.

**2.Obiective**

**2.1. Obiectiv principal**

Obtinerea unei aplicatii care proceseaza comenzile unui client pentru un depozit, folosindu-se baze de date pentu a stoca produsele, clientii si comenzile.

Obiectivele obligatorii pentru acceptarea temei sunt: respectarea paradigmelor OOP, aplicatia organizata pe pachete, clase de maxim 300 de linii(exceptie clasele de UI), metode de maxim 30 de linii(exceptie case cu switch-uri multe) si Java naming conventions.

Cerintelele pentru nota 5 sunt: crearea unei interfete grafice din care se pot vizualiza clientii si produsele din tabele, optiunea de a insera, a edita sau a sterge un produs sau un client din baza de date, crearea unei comenzi prin selectarea unui produs si a unui client cu afisarea unui mesaj in cazul in care cantitatea dorita depaseste stocul depozitului, iar in caz contrar se decrementeaza stocul, folosirea a cel putin 3 tabele: Clienti, Produse si Comenzi, folosirea tehnicilor de reflection si documentatia.

**2.2. Obiective secundare**

Pentru atingerea obiectivului final se parcurg urmatoarele etape:

-alegerea structurilor de date

-impartirea pe clase

-dezvoltarea algoritmilor

-implementarea solutiei

-testare

**3.Analiza Problemei**

In prima faza, de analiza a cerintei ne vom raporta la conceptele programarii orientate pe obiecte care au la baza ideea unificarii datelor cu modalitatile de prelucrarea a acestora si manevreaza enititati reprezentate sub forma de obiecte.Programarea Obiectuala ofera posibilitati de modelarea a obiectelor, a proprietatilor si a realtiilor dintre ele, dar si posibilitatea de a descompune o problema in componentele sale.

Vom studia atent enuntul problemei pentru a putea stabili care sunt obiectele cu care vom lucra mai departe, cum vor interactiona unele cu altele si ce metode pot fi implementate .Astfel putem spune deja ca vor exista clasele Client, Product si Order.

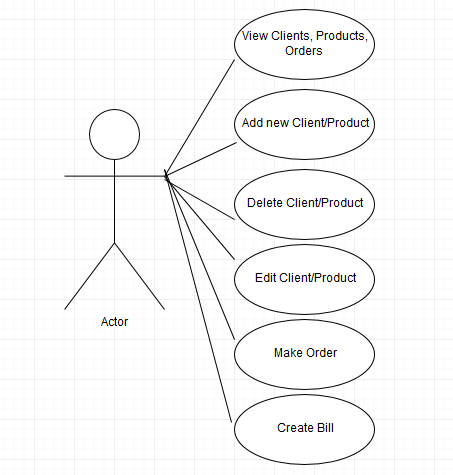
De asemenea, vom avea nevoie de o conexiune la baze de date pe care am realizat-o cu ajutorul MySQL Workbench, unde am creat un server si baza de date cu care voi lucra mai departe si care contine cele 3 tabele: Client, Product si Comenzi.

Cerinta problemei presupune si folosirea tehnicilor de Reflection.Reflection ne ofera informatii despre clasa careia apartine obiectul si despre metodele clasei care pot fi executate folosind acel obiect.

Acest mecanism ne este oferit de catre pachetul java.lang.reflect. Reflectia inseamna autoexaminare, adica permite determinarea structurii clasei. In limitele managerului de securitate putem afla metodele clasei, constructorii clasei si restul membrilor clasei. Cateodata putem sa modificam starea obiectlui prin apelul metodelor specifice sau putem construi obiecte noi . Mecanismul de reflectie este utilizat de componentele Java (Java beans) pentru determinarea  capabilitatilor obiectelor pe timpul executiei. Membrii clasei sunt atributele si metodele. Metodele se impart la randul lor in doua categorii, constructori si metode obisnuite. Principalele clase care definite in acest pachet sunt:

* java.lang.reflect.Field
* java.lang.reflect.Method
* java.lang.reflect.Constructor

**Diagrama use case**

****

Utilizatorul are posibilitatea de a vizualiza tabelul de produse sau tabelul de client, poate edita un client sau produs, poate insera un nou client sau un nou produs, poate sterge un produs sau client dupa id, poate face o comanda si poate crea o facture pentru o anumita comanda.

**4.Proiectare**

**4.1.Structura proiectului**

Aplicatia foloseste Layered Architecture, adica este structurata pe mai multe nivele:

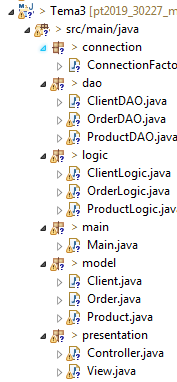
-Presentation Layer: continne clasele care defines interfata grafica

-Business Layer: contine clasele care incapsuleaza logica aplicatiei

-Data access Layer: contine clasele care definesc intergorarile si conexiunea cu baza de date

Si mai exista un Layer, si anume Model, care contine clasele ce defines tabelele din baza de date.

Aceasta tehnica de programare este folosita in special in aplicatiile de tip client-server.Avantajul acestei tehnici este reprezentat de flexibilitatea acordata echipelor de dezvoltarea prin faptul ca acestea pot sa actualizeze si sa imbunatateasca anumite parti ale aplicatiei independent de celelalte.Spre exemplu, interfata grafica poate fi modificata fara a fi afectat nivelul de business sau de data access.



Aplicatia mea este structurata in 6 pachete:

* Pachetul connection-contine o clasa ConnectionFactory care realizeaza conexiunea la baza de date.
* Pachetul dao-se refera la layer-ul de data access si contine clasele care defines interogarile
* Pachetul logic:-contine clasele cu logica aplicatiei
* Pachetul model:-clasele care defines tabelele din baza de date
* Pachetul presentation:-se ocupa de interfata grafica prin intermediul claselor View si Controller
* Pachetul main:contine clasa Main

**4.2.Diagrama de Clasa**

Diagramele de clase sunt folosite pentru a specifica structura statica a sistemului, adica ce clase exista in sistem si care este legatura dintre ele.

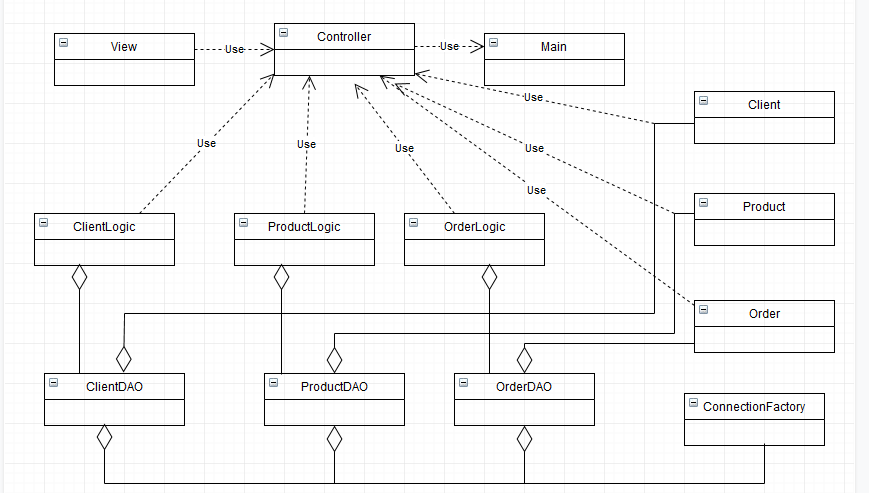
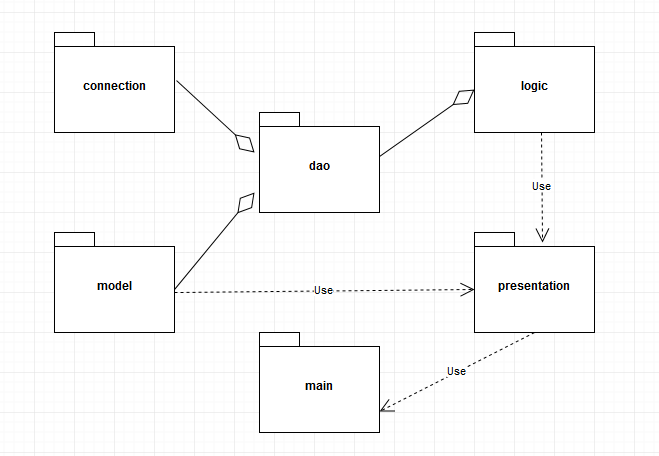


Diagrama de pachete



**5.Implementare**

**Model**

In pachetul model se afla cele 3 clase: Client, Product si Order care definesc cele 3 tabele din baza de date.Atributele acestor clase coincid cu coloanele tabelelor si sunt definite doar metode de tip getter si setter.

Atribute:

Client:

**private** **int** id;

**private** String name;

**private** String email;

**private** String address;

Product:

**private** **int** id;

**private** String name;

**private** **float** price;

**private** **int** cant;

Order:

**private** **int** id;

**private** **int** productId;

**private** **int** cant;

**private** **int** clientId;

**private** **float** sum;

**Data Access**

In pachetul dao se gasesc clasele ClientDAO, ProductDAO, OrderDAO care definesc interogarile ce se realizeaza asupra tabelelor si conexiunea la baza de date.

Am realizat interogari de tip: Insert, Select, Update si Delete.

Un exmplu de interogare este prezentat mai jos, pentru actualizarea unei inregistrari dupa id:

**private** **final** **static** String ***updateStatementString***="UPDATE client set name=?, address=?, email=? WHERE id=? ";

Acesta este query-ul ce va fi executat

**public** **static** **int** update(**int** clientId,String name,String address, String email) {

Connection dbConnection = ConnectionFactory.*getConnection*();

**int** updated=-1;

PreparedStatement updateStatement = **null**;

ResultSet rs = **null**;

**try** {

updateStatement = dbConnection.prepareStatement(***updateStatementString***);

updateStatement.setString(1, name);

updateStatement.setString(2, address);

updateStatement.setString(3, email);

updateStatement.setInt(4,clientId);

updated=updateStatement.executeUpdate();

} **catch** (SQLException e) {

***LOGGER***.log(Level.***WARNING***,"ClientDAO:update " + e.getMessage());

} **finally** {

ConnectionFactory.*close*(rs);

ConnectionFactory.*close*(updateStatement);

ConnectionFactory.*close*(dbConnection);

}

**return** updated;

}

Interogare pentru selectarea tuturor intregistrarilor:

Metoda va returna un ArrayList<Object> care va fi folosit in metoda createTable din View.

In ResultSet se vor afla toate inregistrarile din tabel, iar pentru fiecare inregistrare se va crea un obiect de tip Product care va fi apoi convertit in obiect Object pentru a putea fi adaugat in ArrayList.

**public** **static** ArrayList<Object> select() {

ArrayList<Object> toReturn = **new** ArrayList<Object>();

Connection dbConnection = ConnectionFactory.*getConnection*();

PreparedStatement selectStatement = **null**;

ResultSet rs = **null**;

**try** {

selectStatement = dbConnection.prepareStatement(***selectStatementString***);

rs = selectStatement.executeQuery();

**while**(rs.next())

{

**int** productId=rs.getInt("id");

String name = rs.getString("name");

String price = rs.getString("price");

**int** cant = rs.getInt("cant");

**float** pr=Float.*parseFloat*(price);

Product p = **new** Product(productId, name, pr, cant);

Object o=(Object)p;

toReturn.add(o);

}

} **catch** (SQLException e) {

***LOGGER***.log(Level.***WARNING***,"ProductDAO:select " + e.getMessage());

} **finally** {

ConnectionFactory.*close*(rs);

ConnectionFactory.*close*(selectStatement);

ConnectionFactory.*close*(dbConnection);

}

**return** toReturn;

}

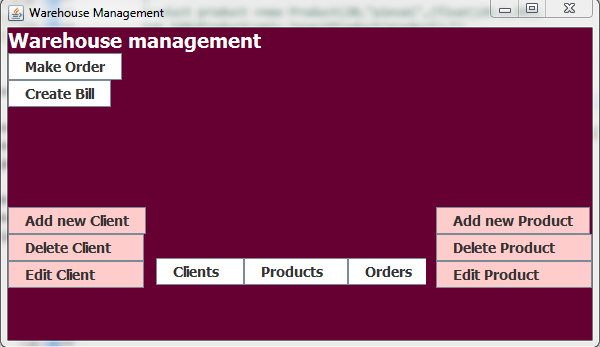
**Business Logic**

In pachetul logic clasele ClientLogic, ProductLogic si OrderLogic incapsuleaza logica aplicatiei si face legatura intre Data Access si Presentation.

**Presentation**

Pachetul presentation contine clasele View si Controller care realizeaza interfata grafica.

Frame-ul principal este:



La apasarea butoanelor Clients, Products sau Orders se va deschide un frame in care va fi afisat tabelul corespunzator.

Pentru asta am folosit tehnica Reflection in metoda createTable pentru a crea un obiect de tipul JTable.Metoda primeste ca si parametru un array de Object si genereaza header-ul tabelului extragand prin reflection proprietatile obiectului si apoi populeaza tabelul cu valorile elementelor din lista.

**public** JTable createTable(ArrayList<Object> objects)

{

JTable table;

Object first=objects.get(0);

ArrayList<String> columnNames=**new** ArrayList<String>();

**for** (Field field: first.getClass().getDeclaredFields())

{

field.setAccessible(**true**);

**try** {

String name=field.getName();

columnNames.add(name);

}**catch** (IllegalArgumentException e) {

e.printStackTrace();

}

}

Object[] columns=columnNames.toArray();

Object [][] data=**new** Object[20][20];

**int** i=0;

**for** (Object obj:objects)

{

**int** k=0;

**for** (Field field :obj.getClass().getDeclaredFields())

{

field.setAccessible(**true**);

**try**

{

Object value=field.get(obj);

data[i][k]=value;

}**catch** (IllegalArgumentException e) {

e.printStackTrace();

}**catch** (IllegalAccessException e) {

e.printStackTrace();

}

k++;

}

i++;

}

table=**new** JTable(data,columns);

**return** table;

}

Aceasta metoda este folosita mai departe in metoda createFrameTable care are ca parametru tot un array de Object, si care va crea tabelul, il va pune intr-un JScrollPane si va suprascrie JFrame-ul this de fiecare data cand se doreste vizualizarea unui anumit tabel prin metoda setFrame.

**public** **void** createFrameTable(ArrayList<Object> objects)

{

JTable table=createTable(objects);

table.setBackground(**new** Color(255, 204, 204));

table.setFont(**new** Font("Tahoma", Font.***BOLD***, 12));

JTableHeader h = table.getTableHeader();

h.setBackground(**new** Color(102, 0, 51));

h.setForeground(**new** Color(255,255,255));

h.setFont(**new** Font("Tahoma",Font.***BOLD***,14));

JScrollPane p=**new** JScrollPane(table);

**this**.table=table;

setFrame(p);

}

**public** **void** setFrame(JScrollPane p)

{

**this**.frame.setSize(**new** Dimension(600,350));

**this**.frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

**this**.frame.setLocation(600, 100);

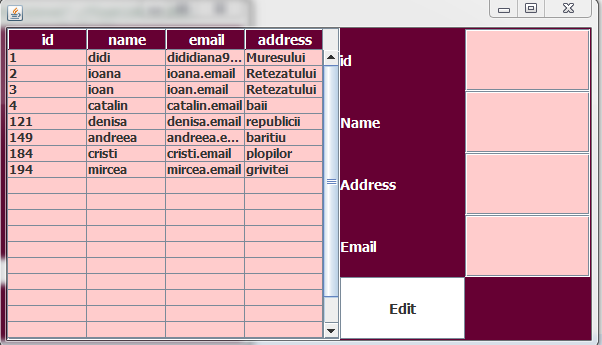
**this**.frame.setContentPane(p);

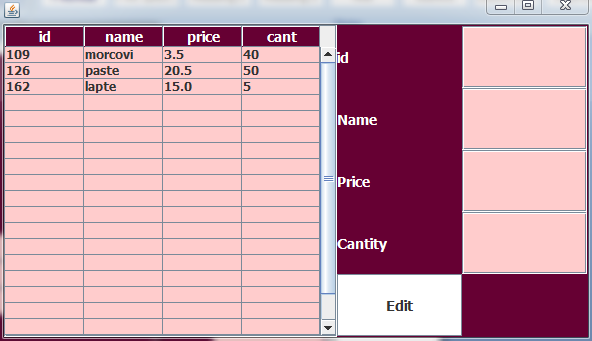
**this**.frame.setVisible(**true**);

}

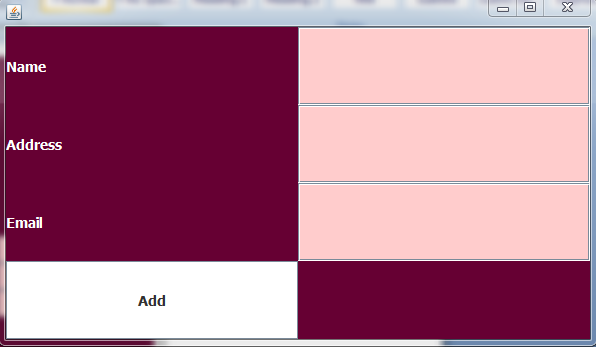
Mai departe, pentru fiecare optiune de Edit Client/Product sau Insert Client/Product se va suprascrie frame-ul corespunzator pentru introducerea datelor.Pentru optiunea de Delete Client/Product se va deschide un JOptionPane in care utilizatorul trebuia sa introduca id-ul produsului sau clientului, iar acesta va fi sters.

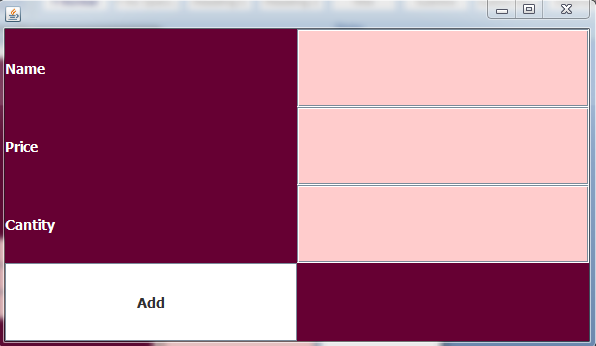
Edit Client- Edit Product



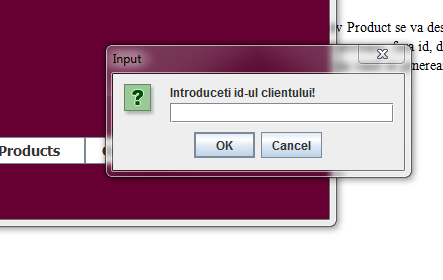
****

In fereastra am decis sa afisez si tabelul de clienti sau de produse pentru a putea alege elementul care se doreste a fi modificat.Utilizatorul trebuie sa introduca neaparat id-ul deoarece cautarea elementului care se doreste a fi editat se face dupa id.Restul campurilor pot sa ramana goale.

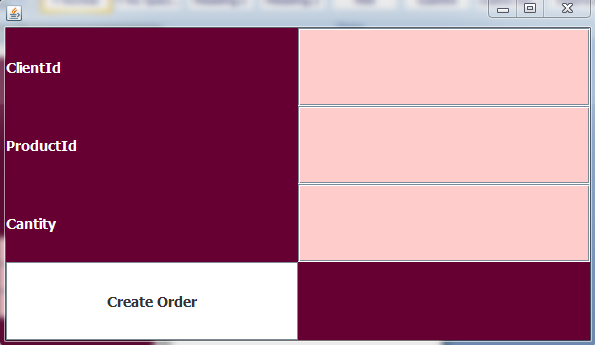
Insert Client-Insert Product

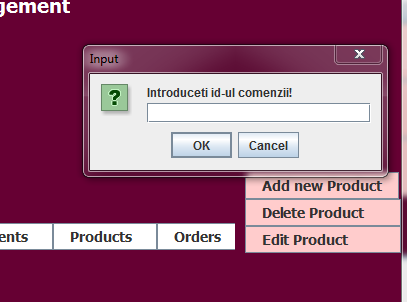


La apasarea unuia din butoanele Add new Client sau Add new Product se va deschide una din ferestrele de mai sus, iar utilizatorul trebuie sa introduca datele necesare, fara id, deoarece am ales sa il introduc automat prin generarea de numere random pana cand se genereaza un id care nu exista in tabel.

 Pentru stergerea unui client sau a unui produs se va introduce id-ul, elementul va fi cautat dupa id si va fi sters.

Pentru crearea unei comenzi se apasa butonul Make Order care va deschide o fereastra in care utilizatorul trebuie sa introduca id-ul clientului, id-ul produsului si cantitatea, iar daca stocul permite efectuarea comenzii aceasta va putea fi vizualizata apoi in tabelul de comenzi, iar daca nu se va afisa un mesaj cu „Not enough products!”.





Butonul Create Bill va crea o factura pentru id-ul comenzii introdus de utlizator in JoptionPane.

Se va crea un fisier text bill.txt prin metoda din clasa Controller:

**public** **static** **void** writeBill(String fileContent) **throws** IOException

{

String[] words=fileContent.split(" ");

BufferedWriter writer = **new** BufferedWriter(**new** FileWriter("bill.txt"));

**for** (String w:words)

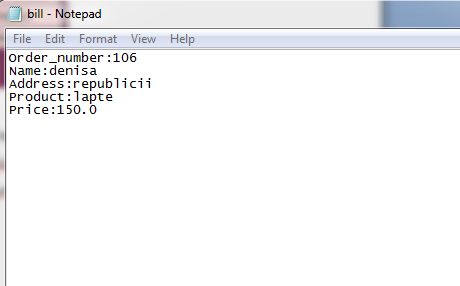
{

writer.write(w);

writer.newLine();

}

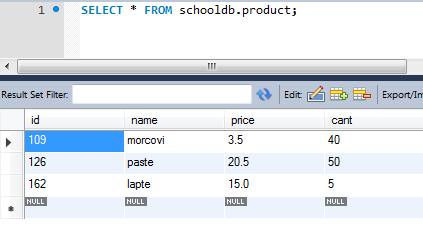
writer.close();

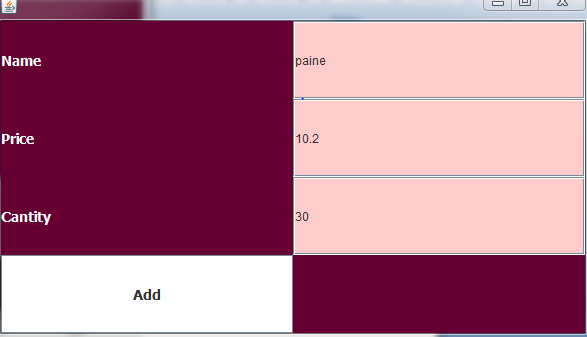
 }

Clasa Controller este cea care controleaza toate aceste operatii, si in functie de butonul apasat va apela diferite metode din nivelul business logic.

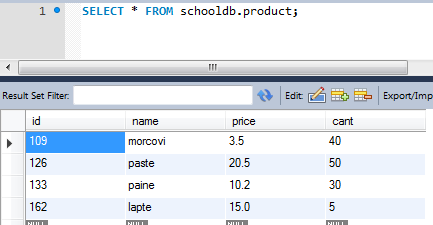
6.Rezultate

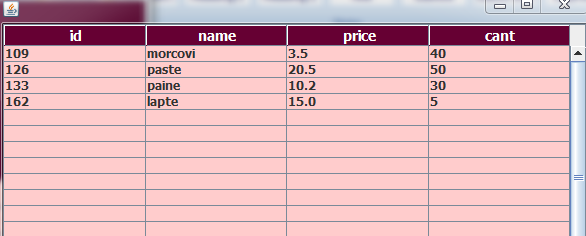
Pentru verificarea rezultatelor voi testa optiunile de insert, update, delete si creare comanda.

Initial in baza de date in tabelul produse se afla urmatoarele inregistrari:

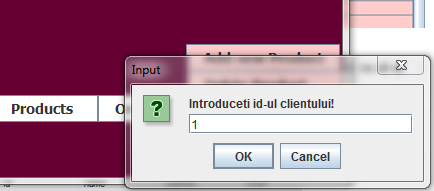
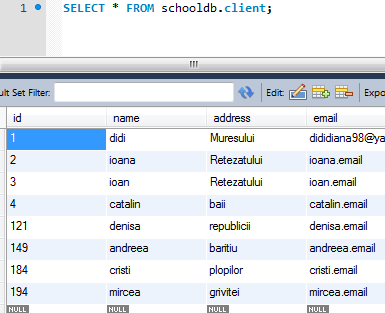
 Vom introduce urmatorul produs:

Dupa apasarea butonului Add tabelul product din baza de date va fi actualizat, la fel si tabelul afisat de aplicatie.

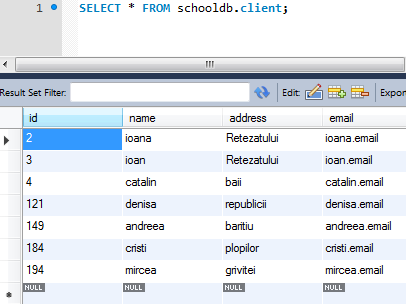
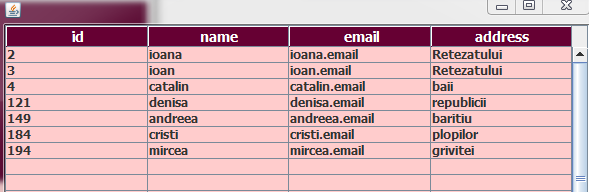


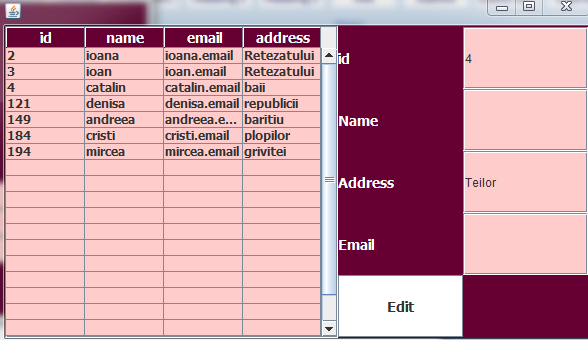


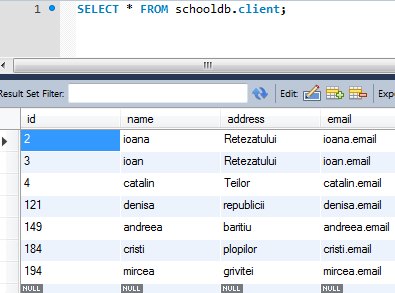
Daca se doreste stergerea unui client, spre exemplu clientul cu id-ul 1 :

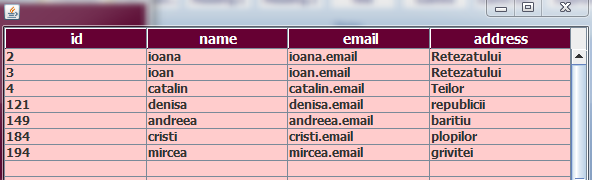


Clientul a fost sters din baza de date.



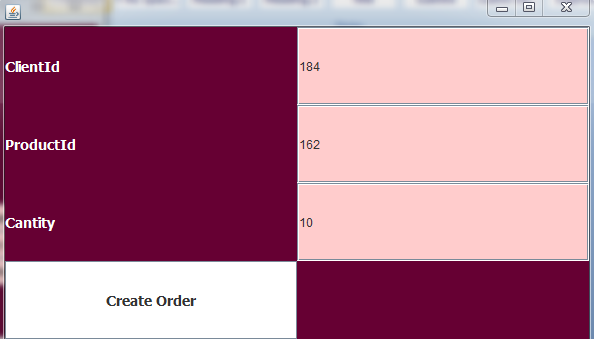
Pentru editarea unui client, vom lua clientul cu id-ul 4 si ii vom modifica adresa cu „Teilor”.

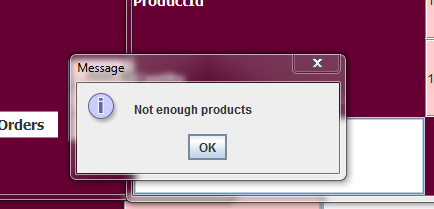
 Dupa apasarea butonului Edit baza de date va fi actualizata:



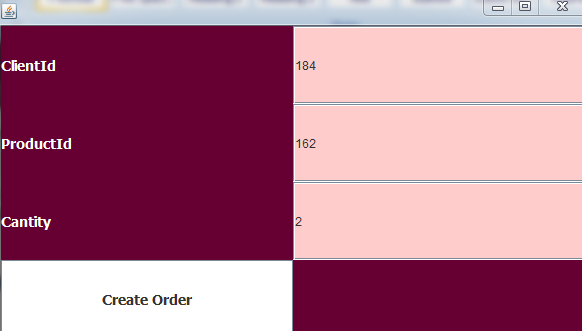
Daca se doreste efectuarea unei comenzi vom alege un client, un produs si cantitatea dorita.Pentru prima rulare vom alege o cantitate mai mare decat cea existenta in stoc pentru a

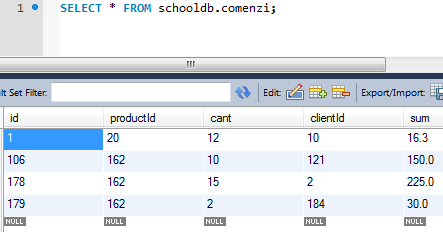
vedea ca aplicatia afiseaza un mesaj de eroare; produsul cu id-ul 162 exista in cantitatea=5

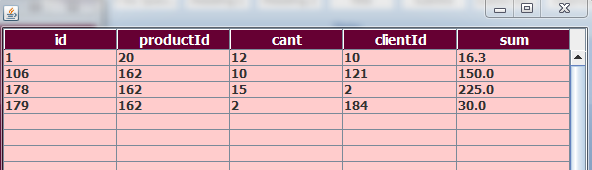




Pentru o cantitatea mai mica, se va crea comanda si se va actualiza stocul.Se va calcula de asemena si suma ca produs intre pretul produsului si cantitate.

Pret produs=15 x cantitate=2 =30







7.Concluzii si Dezvoltari Ulterioare

Consider ca aceasta aplicatie este foarte usor de folosit deoarece m-am straduit sa creez o interfata grafica destul de atragatoare si in acelasi timp simpla.Dezvolatarea aplicatiei m-a ajutat sa ma familiarizez cu conceptul de Three-Tier-Architecture si sa inteleg avantajele aceste tehnici de programare.In acelasi timp am inteles cum functioneaza si tehnica Reflection, care consider ca este de asemenea foarte folositoare.

In ceea ce priveste dezvoltarea ulterioara, aceasta aplicatie de Order Management poate fi extinsa cu usurinta tocmai pentru ca am folosit organizarea pe cele 3 layere si astfel se pot aduce imbunatatiri interfetei grafice fara a modifica nivelele de business logic sau data access.Pentru o aplicatie mai ampla se mai pot implementa metode la nivelul data access si business logic pentru a realiza mai multe interogari, spre exemplu se pot aplica mai multe filtre, cum ar fi cautarea dupa nume,adresa etc.De asemenea,metodele care acceseaza baza de date puteau fi implementate de o clasa generica, iar interogarile pentru accesarea bazei de date pentru un obiect specific care corespunde unui anumit table sa fie generate dinamic prin Reflection.

8.Bibliografie

* <https://ms.sapientia.ro/~manyi/teaching/oop/oop_romanian/curs16/curs16.html>
* <https://www.techopedia.com/definition/24649/three-tier-architecture>
* <https://bitbucket.org/utcn_dsrl/pt-layered-architecture/src/master/>
* <http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW3_Tema3/Tema3_HW3_Indications.pdf>
* <https://stackoverflow.com/questions/8491687/write-newline-into-a-file>
* <https://alvinalexander.com/java/java-mysql-update-query-example>
* <https://alvinalexander.com/java/java-mysql-delete-query-example>
* <https://alvinalexander.com/java/java-mysql-select-query-example>