MINISTRY OF EDUCATION AND RESEARCH
TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

TEHNICI DE PROGRAMARE

TEMA_3

ORDER MANAGEMENT

Student: Tóth Szilveszter Zsolt

Gr.: 30224

1) Obiectivul temei

La aceasta tema am avut de implementat o aplicatie care gestioneaza comenzile intr-o magazie/magazine etc., informatiile fiind stocate intr-un server de baze de date. Instructiunile vor fi intr-un fisier extern, care vor fi citite si prelucrate corespunzator. Operatiile posibile sunt: adaugare si/sau stergerea unui client si/sau produs din tabel, efectuarea unei comenzi si crearea unor rapoarte bazate pe informatiile actuale din baza de date. Deci de exemplu daca avem o instructiune de inserare a clientului X, pe baza informatiilor din fisierul de intrare cream un query care va fi trimis bazei de date si se va insera clientul X.

2) Analiza problemei

Vom apela la o solutie cat mai orientata pe obiect, de accea este necesar (pentru a avea o lizibilitate mai mare asupra codului) sa grupam toate clasele folosite, de aceea vom folosi package-uri. Ca si o prima analiza, din start avem nevoie de 4 clase pentru efectuarea operatiilor, o clasa main in care vom citi si apela aceste clase si o clasa care realizeaza legatura dintre aplicatia java si baza de date in care sunt stocate informatiile despre magazie. Cele 4 clase se pot numi si clasele DAO (Database Acces classes), fiindca prin intermediul acestora este transmisa informatia intre cele doua platforme.

Deci, asa cum am mentionat si la punctul anterior, in aplicatia noastra vom citi instructiunile, care ulterior vor fi procesate. Este nevoie sa ne folosim de fiecare informatie din lista de instructiuni(adica este esential sa extragem fiecare informatie, fiindca nu este tot una daca inseram un nou product, sau stergem un client). Vom apela la functia split din regex, ca sa putem sparge stringurile de intrare pe cuvinte, dupa care o sa avem o decizie de luat: care dintre cele 4 operatii trebuie efectuata la acest pas. Pentru asta avem nevoie de un switch care, pe baza primului cuvant din instructiune, apeleaza clasa necesara pentru a procesa informatiile.

La partea de stocare a informatiile trebuie se ne definim 3 tabele:CLIENT, care are ca si atribut un id, un nume si un oras in care isi are domiciliul, PRODUCT, in care vor fi stocate produsele care au un id, denumire, pret si cantitate, iar in final un tabel numit ORDER, in care vor fi stocate la cate un id numele clientului care a efectuat cumanda, numele produsului, si cantitatea dorita. (Nota: pentru a putea lega tabelele intre ele cat mai usor posibil, in tabelul ORDER vom aveam doar id-urile clientilor is a produselor, ca si valori numerice)

3) Proiectare si imlementare

Pentru a putea realiza o baza de date, vom folosi MySQLServer si MySQLWorkbench. Prin intermediul acestui program se pot crea cu usurinta tabele corespunzatoare pentru aplicatia noastra. Relatiile dintre tabele se pot vedea mai jos, nimic extra, tabelul client si produs sunt legate prin intermediul tabelului order (in tabela order sunt cate un foreign key la primary key-urile celorlalte tabele). Cum se poate vedea si din diagrama, am ales ca si primary key-urile id-urile, cu acestea fiind cel mai usor de jonglat. Codul pentru acesta este:

```
create database if not exists store;  //creare bazei de date
create table clientul (id int, nume varchar(45), oras varchar(45), primary key(id));
create table produs (id int, nume varchar(45), cantitate int, pret float, primary key(id));  //tabele
create table cumpara (id int, idClient int, idProdus int, cantitate int, primary key(id));
alter table cumpara add foreign key(idClient) references clientul(id);  //legarea lor intre ele
alter table cumpara add foreign key(idProdus) references produs(id);
```

Si rezultatul arata asa:

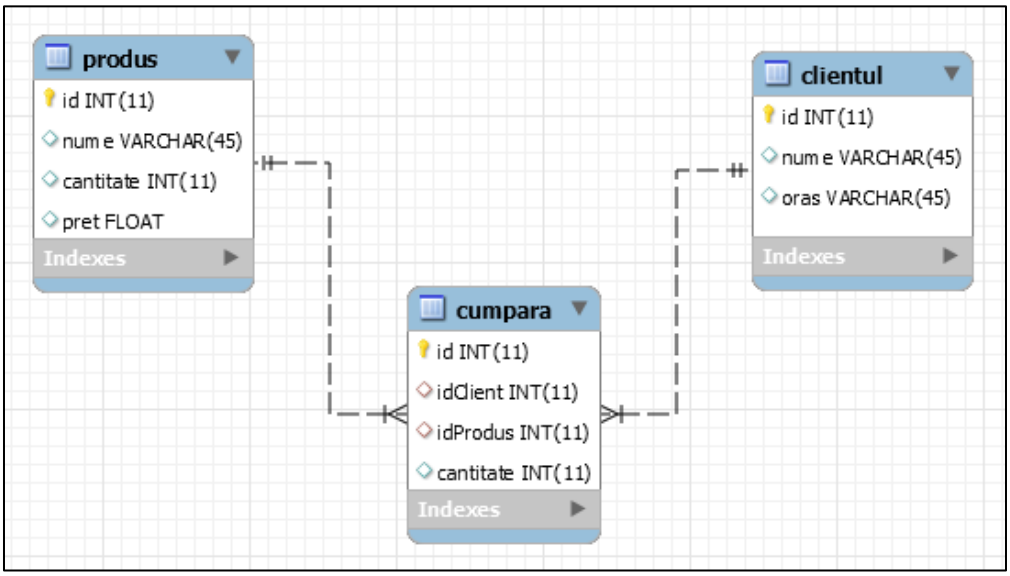
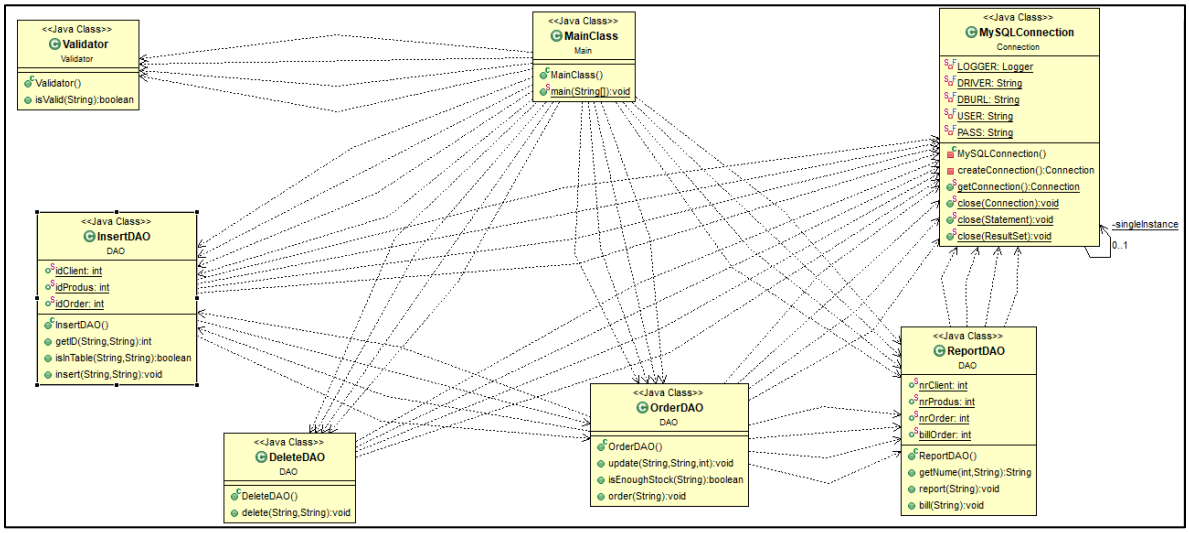


Diagrama claselor arata in felul urmator:



La partea ce tine de baza de date nu mai este ce discuta, fiindca populatea sa va face direct din aplicatia java, care incepe prin citirea secventiala a instructiunilor din fisierul de intrare dat ca si argument din consola. Acest lucru se intampla in clasa main.

```
File inFile= new File( args[0]);
Scanner myRead= new Scanner( inFile);
Validator v= new Validator();
while( myRead.hasNextLine()) {
String data=myRead.nextLine();
if (v.isValid( data)== true)
{
        String op=data.substring(0, 6);
        switch (op) {
        case "Insert":
        String unde= data.substring(7, 14);
        if (unde.equals("client:")) {
                data = data.substring(15,data.length());
        }
        else if (unde.equals( "product")) {
                data = data.substring(16,data.length());
        }
        InsertDAO i=new InsertDAO();
        i.insert(data, unde);
        break;
}
```

In acest cod este descris practic functionalitatea clasei main. In primul rand ne declaram fisierul pe care o vom folosi la citire, mai apoi il parcurgem pana la final, si la fiecare iteratie vom avea pasii: se verifica daca inputul este valid, ceea ce consta in verificarea daca sunt si alte caractere inafara de litere, numere si caractere de separare. Daca conditiile nu se indeplinesc, programul va afisa un mesaj de eroare in consola. In caz afirmativ, programul va intra in switch si va verifica pe baza primului cuvant din instructiune ce operatie este cerut. In cod am pus doar ramura pentru insert, fiindca cele trei ramase sunt foarte asemanatoare cu aceasta si se executa la fel. Se creaza un obiect de tip insert iar se apeleaza functia din aceasta clasa.

Clasa InsertDAO

```
public static int idClient=1;
public static int idProdus=1;
public static int idOrder=1;

public int getID(String data, String unde) {
int id=0;
String query="select id from " + unde + " where nume='"+data+"'" ;
Connection dbConnection=MySQLConnection.getConnection();
PreparedStatement statement=null;
ResultSet rs=null;
try {
        statement = dbConnection.prepareStatement( query.toString());
} catch ( SQLException e) {
        e.printStackTrace();
}
try {
        rs= statement.executeQuery( query);
```

```
            rs.next();
            id = rs.getInt("id");
} catch ( SQLException e) {
e.printStackTrace();
}

MySQLConnection.close (statement);
MySQLConnection.close (dbConnection);
return id;
}

public boolean isInTable( String nume, String unde) throws SQLException {
String query=null;
if (unde.equals("product")==true) {
query="select nume from produs";
}
else if (unde.equals("client:") == true) {
query="select nume from clientul";
}
else
return false;
Connection dbConnection=MySQLConnection.getConnection();
PreparedStatement statement = null;
statement = dbConnection.prepareStatement (query.toString());
ResultSet rs=statement.executeQuery (query);

while (rs.next())
{
String aux=rs.getString ("nume");
if (aux.equals(nume) == true)
return true;
}
return false;
}

public void insert (String data, String unde) throws SQLException {
String[] arr=data.split(", ",-2);
if (isInTable(arr[0],unde) == false) {
String query="insert into ";
if (unde.equals("client:")) {
query += "clientul values (";
query += Integer.toString (idClient);
query = query + ", '" + arr[0] + "', '" + arr[1] +"')";
idClient++;
}

else if (unde.equals("product")){
query += "produs values (";
query += Integer.toString (idProdus);
query = query + ", '" + arr[0] + "', " + arr[1] + ", " + arr[2] +")";
```

```
idProdus++;
}
else
{
query += "cumpara values (";
query += Integer.toString(idOrder);
int idClient=getID(arr[0],"clientul");
int idProdus=getID(arr[1],"produs");
query = query + ", " + idClient + ", " + idProdus + ", " + arr[2] +")";
idOrder++;
}
Connection dbConnection=MySQLConnection.getConnection();
PreparedStatement statement=dbConnection.prepareStatement(query.toString());
statement.executeUpdate(query);

MySQLConnection.close(statement);
MySQLConnection.close(dbConnection);
}
else {
if (unde.equals("product")) {
OrderDAO u=new OrderDAO();
u.update("produs", data,1);
        }
        }
}
```

Am presupus ca este necesar ca codul sa fie pus si aici, pentru a putea explica mai bine tot ceea ce se intampla. Restul codului poate fi vizionat in folderul cu surse. Aceasta clasa are in ea implementata 2 metode suplimentare, care sunt necesare pentru inserare: getID si isInTable. Functie isInTable verifica daca ceea ce dorim sa inseram se afla deja in tabel sau nu. Si exista doua cazuri, prima in care nu este gasit acel produs si este se continua cu inserarea propriu-zisa, sau a doua in care este gasit. Aici sunt din nou doua cazuri: daca dorim sa inseram un client, se lasa balta, adica nu va fi inserat de doua ori, sau daca este vorba despre un produs, in acest caz vom face update la cantitatea produsului din tabel. Pentru acest update ne vom folosi de functia ,,update,, din clasa OrderDAO.

Dar sa ramanem la cazul in care elementul nu este gasit in tabel. In acest caz se incepe si se creaza query-ul necesar pentru a realiza o inserare. In general aceasta arata asa: insert into <numele_tabelei> values(val1, val2,…). Vom sparge stringul in care se afla exact numele obiectului si restul informatiilor necesare. La inceputul clasei sunt 3 variabile statice care preactic tin contorul id-urilor pentru cele 3 tabele folosite. Acest lucru este necesar pentru care fiecare tupla sa aiba un id unic, acesta fiind primary key-ul sau. Dupa ce query-ul este pregatit corespunzator, acesta este trimis la baza de date. Este important sa nu uitam sa inchidem aceste conexiuni pentru a evita erorile. Instructiunile care realizeaza aceasta legatura este scris cu rosu in fragmentul de cod scris mai sus.

A treia functie din aceasta clasa este getID, care cauta in tabelul precizat ca si parametru id-ul numelui precizat tot ca si parametru si returneaza valoarea numerica. Avem nevoie de aceasta functie fiindca in tabela ORDER sunt stocate doar id-urile, dar noi la intrare primim efectiv numele clientului si produsului.

In continuare sunt descrise pe scurt celelalte clase prin intermediul carora se fac operatiile in tabele:

DeleteDAO

Aceasta clasa este responsabila pentru stergerea tuplelor din tabele. Functioneaza ca si operatia descrisa anterior, doar ca aici nu sunt nevoie de functii suplimentare. Doar se creaza un query pe numelui dat la intrare si se

trimite la baza de date, tot asa, se foloseste acea secventa care este colorata cu rosu. La modul general, un query de stergere are forma: delete from <numele_tabelei> where <conditie>.

OrderDAO

Realizarea unei comenzi se rezuma defapt la doi pasi: o inserare pe baza id-ului si actualizarea cantitatii produsului din tabel. Mai intai este verificat printr-o functie auxiliara daca pe stoc sunt suficiente produse pentru a satisface comanda. Daca da, se realizeaza pasii descrisi anterior, sau daca nu, este afisat un mesaj de eroare pe consola.

RaportDAO

Aplicatia trebuie sa fie in stare sa genereze chitanta pentru fiecare vanzare/cumparare. In aceasta clasa este implementata functia ,,bill,, in care se afla id-ul si numele clientului care a cumpara, ce si cate a cumparat, iar la final cat a costat. De asemenea trebuie sa se genereze raporturi pentru tabelele din baza de data atunci cand se cere de la intrare.

MySQLConnection

Aceasta este esentiala pentru a putea realiza aplicatia. Ea face legatura dintre baza de date si mediul java, si daca analizam problema, pe aceasta relatie sa bazeaza toata solutia temei. Din aceasta cauza trebuie sa avem in vedere sa inchidem mereu conexiunea, pentru a evita erorile care pot duce la functiuni eronate a aplicatiei.

4) Rezultate

Corectitudinea se poate verifica cel mai usor folosind rapoartele generate. Aici se pot vedea daca s-au inserat corespunzator, daca s-au efectuat operatiile de de update, de stergere. Un exemplu de raport ar arata un urmatorul fel:

```
Raportul pentru tabelul cerut:
ID: 1 | Nume: Ion Popescu | Oras: Bucuresti
ID: 2 | Nume: Luca George | Oras: Bucuresti
```

(pentru tabelul client)

```
Raportul pentru tabelul cerut:
IdOrder: 1 | (ID)Nume: (2)Luca George | (ID)Produs: (1)apple | Cantitate: 5
IdOrder: 2 | (ID)Nume: (2)Luca George | (ID)Produs: (4)lemon | Cantitate: 5
```

(pentru order)

5) Concluzii

Aceasta tema a fost interesanta din punct de vedere a faptului ca a trebuit sa folosim, pe langa eclipse, si un alt mediu de dezvoltare, sau mai bine sa ,,refolosim,, cunostintele din semestrele anterioare. Mie personal mi-a placut sa exploatam si aceasta parte a programarii in java, mai ales ca invatat, pe langa multe alte chestii, ca se pot crea fisiere pdf direct din codul sursa.

Daca vorbim de posibile dezvoltari in viitor, as putea numi 3 lucruri care ar fi utile si pentru utilizator, dar si pentru programator. Cand se afiseaza mesajul de eroare atunci cand nu sunt suficiente produse pe stoc, pe langa afisarea comenzii care nu s-a putut efectua, programul sa mai afiseze si cate produs sunt in momentul de fata si cate ar mai trebui ca sa se poata finaliza comanda. O alta inbunatatire ar fi ca programul sa nu fie atat de ,,strict,, cand vine vorba de instructiunile de intrare. Aici ma refer sa ignore spatiile, sau virgulele in plus, si situatii de astea care pot apara la orice utilizator. Si ultime chestie la care m-am gandit ar fi o inbunatatire majora este ca aplicatia sa permita cumpararea a mai multor produse in aceeasi instructiune. In momentul de fata acest aspect este limitat, si se poate cumpara un singur produs pe linie.

6) Bibliografie

https://www.baeldung.com/java-pdf-creation

https://www.youtube.com/watch?v=5yIoA6JdS18

http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_3/Assignment_3_Indications.pdf