Order Management aplication for processing customer

**Dondas Bianca - Elena**

**Grupa 30228**

**Profesor Dr. Ing. Cristina Pop**

**Cuprins**

# Obiectivul temei

1. Analiza problemei
   1. Analiza
   2. Diagrama use case
2. Proiectare
   1. Diagrama UML
   2. Pachete
      1. Business Logic
      2. Model
      3. Presentation
      4. Data access
   3. Clase
3. Implementare
   1. Clasele Client, Order, Product
   2. Clasele ClientOperations, OrderOperations, ProductOperations
   3. Clasa Connection
   4. Clasa Client, comenzi, produs, interfata (Pentru GUI)
4. Rezultate
5. Concluzii
6. Bibliografie

# Obiectivul temei

Luați în considerare o aplicație de OrderManagement pentru procesarea comenzilor clienților pentru un depozit. Bazele de date relaționale sunt utilizate pentru a stoca produse, clienți și comenzi. În plus, aplicația utilizează (în mod minim) următoarele clase:

• Clasele de model - reprezintă modelele de date ale aplicației (de exemplu, comandă, client, produs)

• Clase de Business Logic - conțin logica aplicațiilor (de exemplu, OrderProcessing, WarehouseAdmin, ClientAdmin)

• Clase de prezentare - clase care conțin interfața grafică a utilizatorului

• Clase de acces la date - clase care conțin accesul la baza de date

Alte clase și pachete pot fi adăugate pentru a implementa funcționalitatea completă a aplicației.

1. **Analiza problemei**

a. Analizați domeniul aplicației, determinați structura și comportamentul claselor sale și desenați o extensie

Diagrama clasei UML.

b. Implementați clasele de aplicații. Utilizați javadoc pentru a documenta clasele.

c. Utilizați tehnici de reflecție pentru a crea o metodă care primește o listă de obiecte și generează antetul unui tabel prin extragerea prin reflecție a proprietăților obiectului și apoi popularea tabelului cu valorile ale elementelor din listă.

JTable createTable (obiecte listă <Object>);

d. Implementați un sistem de programe de utilitate pentru raportare, cum ar fi: stocuri, totaluri, filtre etc.

Aceasta aplicatie are ca si obiectiv interactionarea in timp real cu baza de date a depozitului si cu utilizatorul, proiectarea si implementarea unui program ce va fi folosit in procesarea unor comenzi. Aceste comenzi vor veni de la anumiti clienti, care vor dori sa cumpere anumite produse in anumite cantitati de la un magazin, practic sa plaseze o comanda. Produsele se pot gasi in stocul magazinului in cantitati suficiente sau insuficiente. Comenzile se definesc printr-un anumit client care le plaseaza, un id de identificare, numele produsului comandat, cantitatea dorita. Clientii se definesc printr-un ID unic, ce ne va ajuta sa-i diferentiem. Dupa efectuarea unei comenzi si realizarea acesteiea, produsele cumparate vor fi scazute din stocul magazinului. De asemenea, trebuie sa avem si posibilitatea de a improspata stocul sau de a actualiza datele unui client sau a unei comenzi in cazul in care se doreste aceasta operatie. Utilizatorul poate sa vizualizeze, sa adauge, sa caute, sau sa actualizeze produsele.

In primul rand pentru a putea rezolva aceasta tema avem nevoie de o buna intelegere a bazelor de date. Asadar, o **bază de date** (abreviat **BD**), reprezintă o modalitate de stocare a unor informații și date 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 cantități de date care trebuie accesate simultan prin 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. Bazele de date sunt manipulate cu ajutorul sistemelor de gestiune a bazelor de date.

Cel mai răspândit tip de baze de date este cel relațional, în care datele sunt memorate în tabele. 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.

Aplicatia foloseste o baza de date externa formata in MySql WorkBench unde sunt retinute toate datele despre clientii, produse si comenzi. In cadrul aplicatiei se mai pot efectua si alte operatii, operatii de a adauga un/o client/ produs/ comanda in baza de date, de a sterge un/o client/ produs/ comanda, de a face edita informatiile despre un/o client/ produs/ comanda. Daca utilizatorul introduce o cantitate care depaseste cantitatea stocului atunci cand se face o comanda, pe ecran va aparea un mesaj de eroare si se va astepta reintroducerea cantitaii dorite pentru comanda.

Pentru o mai buna intelegere a codului si operatiile in sql trebuie sa avem cunostinte despre procedurile stocate din sql, ele au fost folosite la operatiile de adaugare, stergere si editare.

O procedură stocată este o mulţime ordonată de instrucţiuni SQL stocată permanent pe server şi compilată la utilizare. Procedurile stocate reprezintă o modalitate de a crea rutine şi proceduri care să fie rulate pe server de către procesele serverului. Aceste proceduri pot fi lansate în execuţie de o aplicaţie apelantă sau de declanşatoare sau de regulile de integritate. Deoarece SQL-server permite administrarea bazelor de date din sistem este logic ca serverul este locul cel mai bun pentru rularea procedurii stocate. Procedurile stocate pot să returneze valori şi să modifice, să execute operaţii de comparare cu valorile precomparate folosite de sistem. Procedurile stocate pot să primească şi să returneze valori care nu provin neaparat dintr-o tabelă ci sunt calculate prin execuţia procedurii. Procedurile stocate rulează pe servere care sunt calculatoare puternice.

Cateva avantaje a procedurilor stocate:

1. Procedurile stocate rulează pe servere care sunt calculatoare puternice;

2. Deoarece ele sunt stocate pe server nu mai necesită transfer de informaţii din baza de date, ele fiind pe acelaşi calculator;

3. Procedura beneficiază de un acces direct, rapid şi imediat la baza de date ce permite o prelucrare rapidă a informaţiilor;

4. Permite partajarea logicii aplicaţiei între aplicaţii diferite;

5. Simplifică integrările parametrizate facilitând rularea repetată aceleiaşi interogări cu seturi diferite;

6. Oferă facilităţi pentru dezvoltarea C/S prin modularizarea aplicaţei separând modulele pentru client şi server, ce permite scurtarea timpului de execuţie şi uşurarea gestiunii aplicaţei;

7. Componentele de pe server pot fi dezvoltate separat de componentele client fapt ce face posibilă refolosirea lor între aplicaţile client;

8. Ele pot furniza mecanisme de securitate: gestiunea vederilor şi a procedurilor stocate pot fi folosite ca instrumente de asigurare a securitătii . Pot fi create proceduri stocate pentru operatiile de adăugare/ modificare/ ştergere/ citire controlând programat fiecare din aceste tipuri de acces la informaţii;

9. Impune pe partea de server reguli orientate spre date, permit impunerea de reguli şi relaţii logice care sporesc controlul informaţiilor în sistem;

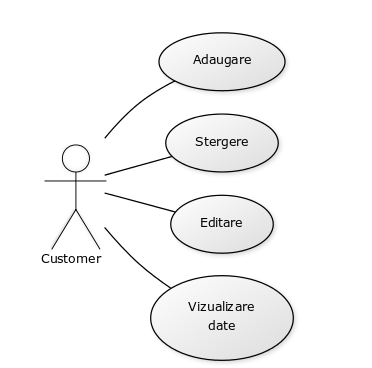
Aplicatia este foarte usor de folosit, deoarece are o interfata “user friendly”;

Pasul 1: se ruleaza aplicatia

Pasul 2: utilizatorul va aleage din ecranul principal una dintre optiunile: Client/ Produs/ Comanda

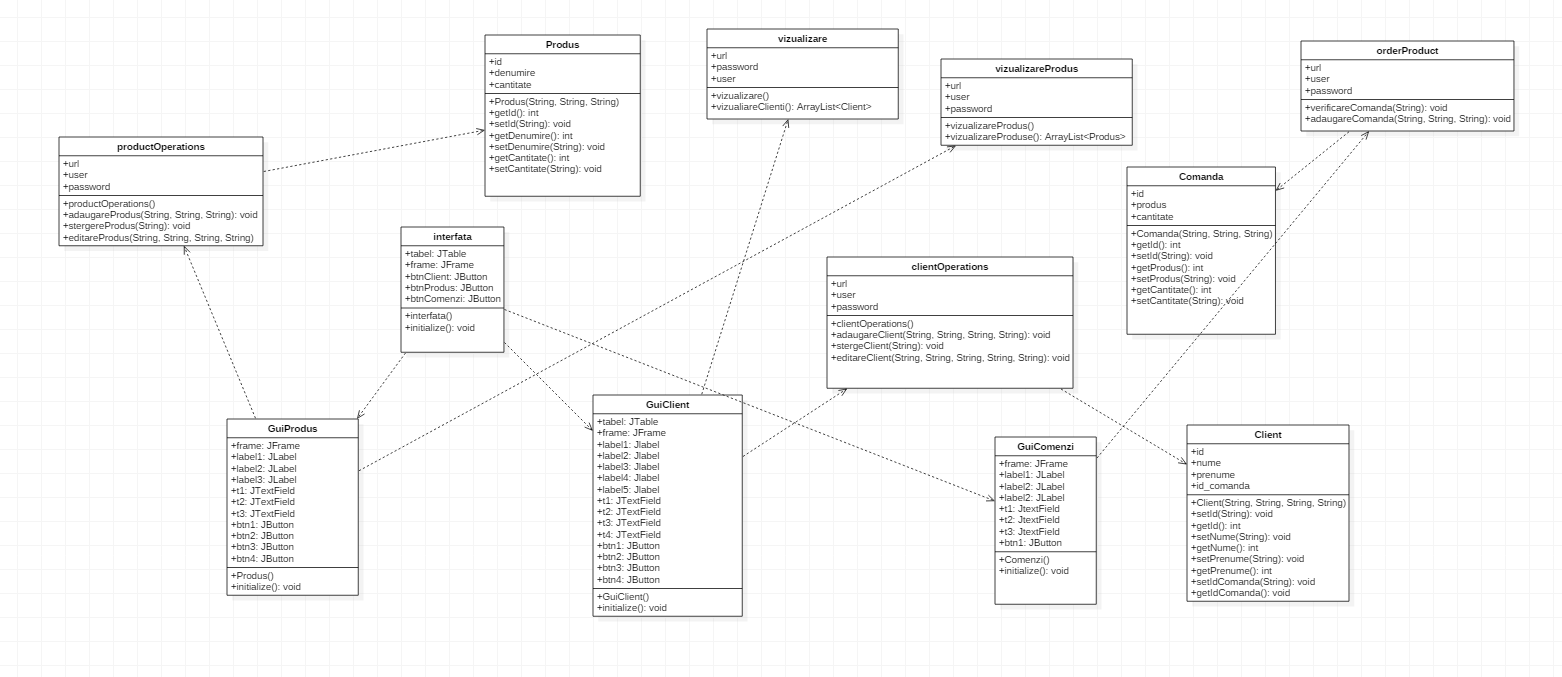
Pasul 3: la oricare optiune aleasa se va intra intr- un meniu unde se poate adauga un client (respectiv un produs sau o comanda), sterge un client (respectiv un produs sau o comanda), edita un client (respectiv un produs sau o comanda) sau vizualiza tabelul cu date corespunzator fiecarei optiuni ( client, produs sau furnizor).

**Diagrama use case a proiectului:**



1. **Proiectare**

**Diagrama UML de clase**



Aceasta aplicatie foloseste o arhitectura de tip Layerd Architecture, adica avem mai multe straturi fiecare avand responsabilitati diferite. Se folosesc mai multe pachete, fiecare repezentant cate un strat. Pachetele folosite sunt: Business Logic, Data Access, Model si Prezentation.

*Pachetul “Model”*

Acest pachet contine cele 3 clase de baza ale proiectului: clasa Client, clasa Comanda si clasa Produs.

Aceste 4 clase contin construtorul clasei respective precut si settere si gettere (de exemplu clasa Client are constructorul + metodele setId(), setNume(), setPrenume, setIdComanda() + metodele getId(), getNume(), getPrenume, getIdComanda()) ele sunt implementate in exemplul de cod de mai jos:

**public** String id, nume, prenume, id\_comanda;

**public** Client(String id, String nume, String prenume, String id\_comanda) {

**this**.id = id;

**this**.nume = nume;

**this**.prenume = prenume;

**this**.id\_comanda = id\_comanda;}

**public** String getId() {

**return** id;}

**public** **void** setId(String id) {

**this**.id = id;

}

**public** String getNume() {

**return** nume;

}

**public** **void** setNume(String nume) {

**this**.nume = nume;

}

**public** String getPrenume() {

**return** prenume;

}

**public** **void** setPrenume(String prenume) {

**this**.prenume = prenume;}

**public** String getId\_comanda() {

**return** id\_comanda;

}

**public** **void** setId\_comanda(String id\_comanda) {

**this**.id\_comanda = id\_comanda;}

*Pachetul “Data accesss”*

Acest pachet contine clasele: clientOperations, productOperation, orderProduct si conection.

In clasa conection se face conexiunea cu baza de date astfel:

**this**. url= "jdbc:mysql://localhost:3306/tema3" + "?useSSL=false"; //acesta reprezinda url-ul bazei de date

**this**.user= "Userul bazei de date";

**this**.password= "Parola bazei de date";

**try** {

*connection*=DriverManager.*getConnection*(url, user, password);

}

**catch**(Exception e)

{

e.printStackTrace();

}

In clasa clientOperations se efectueaza operatiile cerute in cerinta, adica: operatia de adaugare a unui client, de stergere a unui client, de ediatare a unui client. Va fi prezentat mai jos codul de adaugare a unui client in baza de date.

**public** **void** adaugareClient(String id\_c, String nume, String prenume, String id\_com )

{ **try** {

CallableStatement myStmt=**this**.*connection*.prepareCall("{call adaugare\_client(?,?,?,?)}");

myStmt.setString(1, id\_c);

myStmt.setString(2, nume);

myStmt.setString(3,prenume);

myStmt.setString(4,id\_com);

myStmt.execute();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

Ca argumente ale functiei avem patru stringuri ele reprezentand datele unui rand din tabelul clienti. Parte de cod „call adaugare\_client(?,?,?,?)” apeleaza procedura stocata, aceasta procedura retine codul in sql pentru adaugare unui rand in tabel. Ea are patru parametrii, unul pentru fiecare coloana a unui rand. Adaugarea parametrilor se face cu „myStmt.setString(1, id\_c);”, deci in loc de primul „?” se va lua „id\_c” adica primul paramtru al functei adaugareClient. Procedura stocata a operatiei de adaugare client este urmatoarea:

CREATE PROCEDURE adaugare\_client(in a varchar(20),in b varchar(45),in c varchar(45))

insert into produse(id, nume, prenume, id\_comanda) values (a,b,c,); , asemenea se fac si operatiile de stergere si editare.

Clasa productOperations este la fel cu clasa client doar cu metodele implementate pe tabela „produse” din baza de date.

Clasa orderProduct contine doar metoda de adaugare a unei comenzi.

*Pachetul “Business Logic”*

Acest pachet contine clasele vizualizare si vizualizareProdus, care implementeaza metodele pentru vizualizarea datelor din tabela client, respectiv produs.

Se executa query-ul respectiv in sql si se adauga intr-o lista de clienti datele din tabel.

**public** ArrayList<Client> vizualizareClienti()

{

ArrayList<Client> list = **new** ArrayList<Client>();

String query = "SELECT \* FROM clienti ";

PreparedStatement st = **null**;

ResultSet rs = **null**;

**try** {

st = *connection*.prepareStatement(query);

rs = st.executeQuery(query);

rs.next();

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

Client client = **new** Client(rs.getString("id"), rs.getString("nume"), rs.getString("prenume"),rs.getString("id\_comanda"));

list.add(client);

}

} **catch** (Exception ex) {

ex.printStackTrace();

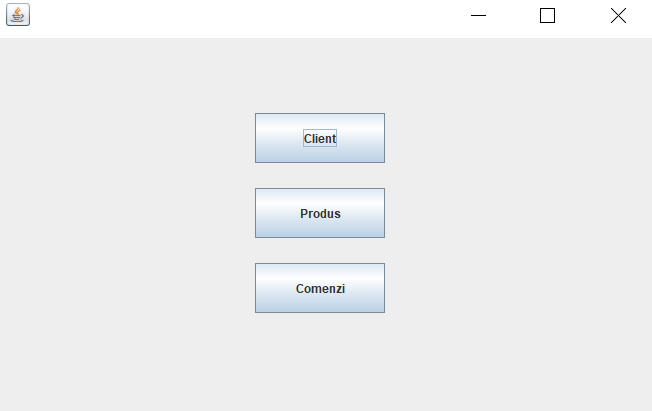
}

**return** list;

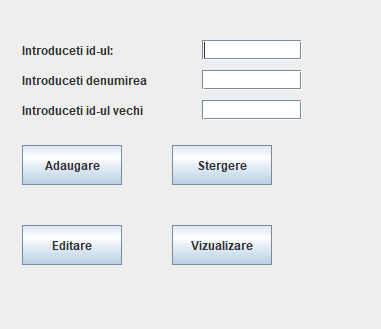
}

1. **Rezultate**

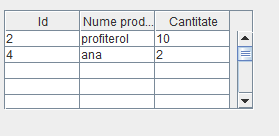
Dupa rularea aplicatiei se deschide meniul principal:



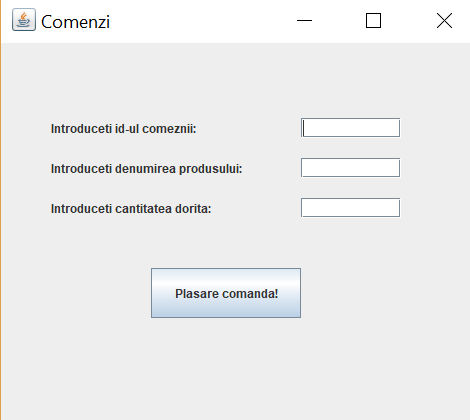
Se alege meniul in care vrea sa se intre, de exemplu produs:



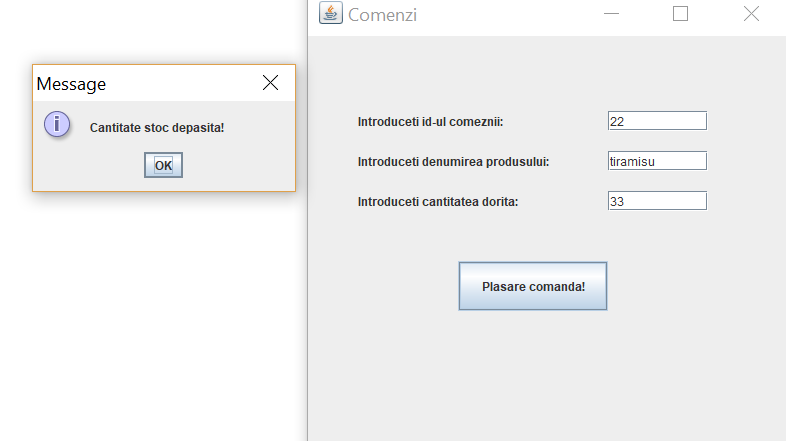
Se introduce datele si se allege operatia dorita, toate cele 4 operatii se executa in timp real. Daca se alege operatia de vizualizare, pe ecran va aprea un tabel cu datele din tabelul bazei de date:



Daca se intra in meniul comanda se deschide un frame care arata asa:



Se introduc datele, iar in timp real acestea se introduc in tabelul pentru comenzi. In cazul in care se comanda o cantitate mai mare decat cea din stoc va aparea un mesaj de eroare.



1. **Concluzii**

In concluzie aceasta nu a fost o tema greu de realizat, dar a fost una foarte buna pentru a exersa si dezvolta principiile oop. De asemenea am invatat sa lucram si sa ne conectam in java cu o baza de date, sa invatam despre proceduri stocate si sa ne amintim comenzi in sql.

1. **Bibliografie**

* <https://www.youtube.com/watch?v=P0opfx23Czw>
* <https://stackoverflow.com/questions/4468669/how-to-generate-javadoc-html-in-eclipse>
* <https://ro.wikipedia.org/wiki/Baz%C4%83_de_date>
* <http://www.tutorialspoint.com/uml/uml_basic_notations.htm>
* <https://stackoverflow.com/questions/37628/what-is-reflection-and-why-is-it-useful>
* <http://www.oracle.com/technetwork/articles/java/javareflection-1536171.html>