**Documentație Tema 3**

Realizator:

Stroia Lucian Dorin

**Cuprins**

1. *Obiectivul lucrării*
2. *Scopul realizării aplicație*
3. *Abordarea problemei*
4. *Diagrame UML*
5. *Implementarea programului*
6. *Rezultatele obținute*
7. *Concluzii*
8. *Bibliografie*

**Obiectivul lucrării**

*Obiectivul acestei lucrări este reprezentat de simularea unor achiziții a unul sau mai multe produse, de către un anumit client, acest client fiind o persoană care a fost stocată într-o bază de date, având anumite proprietăți prin intermediul cărora se diferă unul față de celălalt.*

*Este posibilă efectuarea unor produse dintr-un anumit stoc, fiind caracterizate printr-o anumită cantitate.*

*Proiectul are la temelie 4 tabele in care se stochează anumite date despre clienți, produse, respectiv comenzile pe care aceștia le procesează.*

*.*

**Scopul realizării acestei aplicații**

*Această aplicație a fost creată cu scopul de a simula cererea unor produse dintr-o magazie, fiind necesar inițial sa se realizeze stocarea persoanei care dorește să efectueze comanda respectiva, fiind salvate într-o bază de date anumite informații despre client, informații care urmează să fie necesare pentru generarea unei facturi.*

*Această factură are scopul de a informa clientul în legătură cu toate cumpărăturile pe care acesta le-a făcut însă, factura este necesară și pentru furnizor, pentru a știi informații în legătură cu persoana care a efectuat o anumită comandă dar și în legătură cu produsele pe care ei urmează să le livreze în limita stocului disponibil.*

*Toate aceste acțiuni sunt posibil realizabile prin intermediul unei interfețe care face legătura dintre utilizator și baza de date, astfel, clientul având acces la vizualizarea unor informații despre produsele disponibile, poate să își facă o idee în legătură cu ce comandă să facă.*

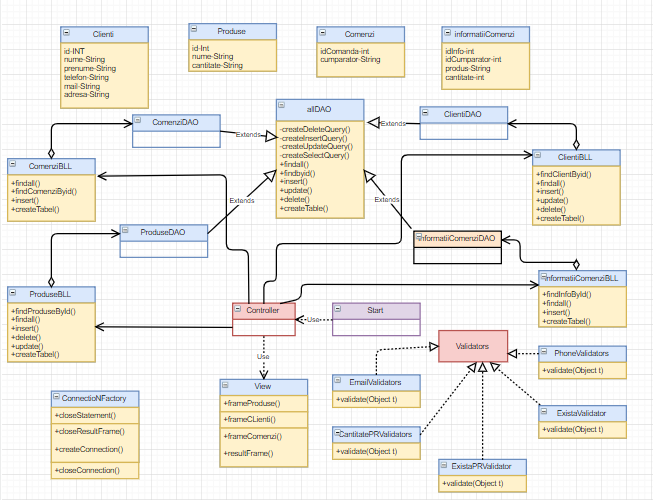
**Abordarea problemei**

*Pentru realizarea acestui program, a fost adăugată o nouă dependență, în fișierul ”pom.xml” prin intermediul căreia se face o legătură cu programul ” MySQLWorkBench ”, program în care au fost inițiate cele 4 tabele necesare pentru păstrarea informațiilor în legătură cu clienții, produsele dar și comenzile efectuate.*

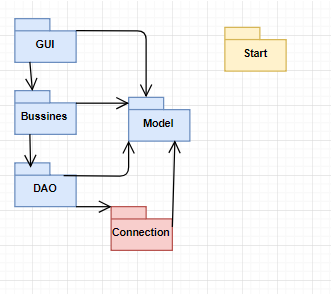
*Structura acestui program are o parte care face referire la interfață, una care realizează logica programului, și încă o parte care face legătura cu baza de date, astfel, accesul utilizatorului la datele stocate în baza de date fiind succesivă, sunt protejate anumite proprietăți ale produselor, clienților sau ale comenzilor, dar sunt evitate și anumite ilegalități pe care un utilizator ar putea să le facă.*

**Diagrame UML**

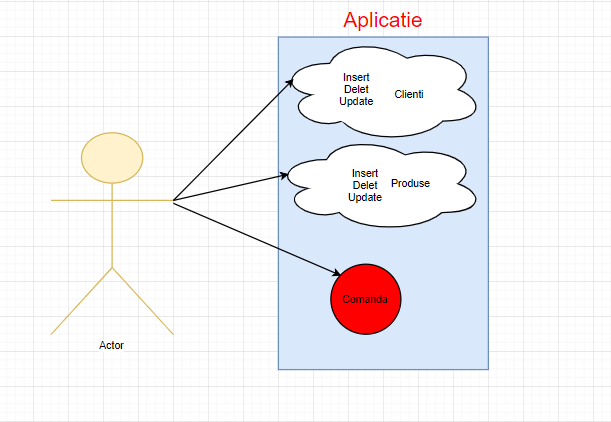
*1.DIAGRAMA DE CLASE*

**

*2.DIAGRAMA DE PACHET*



*3.DIAGRAMA USE-CASE*

**

**Implementarea programului**

*Implementarea propriu-zisă a programului este organizată în 6 pachete, unul dintre ele conținând încă un pachet, fiind o ramură a pachetului principal.*

***1.Pachetul GUI***

*În acest pachet, sunt implementate 2 clase, care controlează majoritatea acțiunilor care sunt efectuate în aplicație, una dintre acestea fiind ”Controller”, care gestionează toate acțiunile utiliztorului.*

*Clasa View*

*Această clasa extinde JFrame, aici sunt ințializate toate variabilele din clasa JFrame, utile pentru informarea si interacțiunea dintre utilizator și program. Constructorul acestei clase inițiază fereastra principală, care reprezintă mai mult un meniu din care utilizatorul poate să aleagă una dintre opțiunile:*

*1.Clienți*

*2.Produse*

*3.Comenzi*

*Ulterior, sunt realizate anumite metode care inițiază alte frame-uri, prin intermediul cărora, utilizatorul alege anumite acțiuni care aduc modificări la baza de date la care este conectată aplicația sau afișează conținutul tabelelor din această bază de date.*

*Clasa Controller*

*În constructorul acestei clase, sunt inițializate listenerele care indică ce buton a apăsat utilizatorul, butoanele fiind numite cu o anumită semnificație, indicând acțiunea dorită de către utizilator.*

*În funcție de butonul ales de către utilizator în primă fază, atunci când trebuie să aleagă secțiunea asupra căreia dorește să se facă anumite acțiuni, se execută o anumită porțiune de cod, care aduce modificări în baza de date sau afișează conținutul tabelului corespunzător secțiunii în care utilizatorul se află*

*În clasa în care este definit listener-ul pentru butonul care realizează generarea chitanței, sunt utilizate câteva funcții din librăria specifică fișierelor de tip PDF, astfel, în urma operațiilor indicate de către utilizator, se generează un fișier PDF care indică date despre utilizator dar si informații legate de produsele pe care acesta dorește să le comande.*

*Pentru generarea tabelelor într-un format care îi permite utilizatorului să afle informațiile pe care le dorește, a fost creată o funcție care primește ca și argument, în funcție de secțiunea în care utilizatorul se află, o listă de obiecte astfel, este generat un tabel vizibil în cadrul interfeței grafice.*

***2.Pachetul BussinesLogic***

*În acest pachet, sunt definite 4 clase care sunt corespunzătoare fiecare cu câte un tabel din baza noastră de date, având rolul de a gestiona partea logică a programului, anumite acțiuni propuse de către utilizator prin acționarea asupra unui buton reprezentativ, având nevoie de anumite permisiuni, sunt verificate caracteristicile esențiale pentru realizarea acțiunilor. De exemplu, pentru introducerea unui nou client, este necesară introducerea unui email în care este obligatoriu să se găsească grupul de litere ”@yahoo.com”. Există un pachet interior acestui pachet în care sunt declarate mai multe clase care au un nume sugestiv, ex: ”EmailValidators”.*

*Aceste clase implementează o interfață definită de către mine, interfață care obligă clasele care o implementează să definească metoda validate, prin care se primește ca și argument un obiect, și sunt aplicate câteva verificări care permit trecerea utilizatorului la următoare etapă și actualizarea bazei de date.*

*În pachetul general, BussinelLogic, în care sunt definite cele 4 clase, care seamănă din punctul de vedere al implementării, sunt descrise si metode precum inserare, findById, fiindAll, delete, s.a.m.d*

*Obiectul prin intermediul căruia fiecare clasă interacționează cu baza de date este diferit în fiecare clasă, fiind definit fiecare in pachetul DatabaseAcces, este construit in momentul în care se apelează constructorul obiectului care aparține de clasa care se ocupa cu logica (BussinesLogic).*

*Fiecare clasă are proprii validators, fiind inițiată o listă de validatori la începutul clasei, validatorii care sunt folosiți fiind specifici fiecărei clase și au rolul de a verifica caracteristica specifică clasei respective.*

***3.Pachetul DatabaseAcces***

*În acest pachet, sunt definite 5 clase, 4 dintre ele fac referire la cele 4 tabele care au fost create in baza de date, aceste 4 clase extinzând ultima clasă creată, ”allDAO” primește un obiect, numit T. Acest obiect este transmis in momentul în care una dintre cele 4 clase extind clasa ”allDAO” cu parametrul specific tipului la care face referire. De exemplu, clasa ClientDAO extinde clasa allDAO cu parametrul <Clienti>.*

*Cele 4 clase extinzând clasa allDAO, practic clasa principală devine allDAO care indiferent de obiectul pe care îl primește, trebuie să definească metode generice care prin intermediu Reflexiei, îndeplinesc acțiunile generate de către utilizator, și fac referire către tabela specifică.*

*Deci, toate metodele definite în clasa allDAO sunt realizate astfel încât indiferent de tipul clasei care extinde această clasă principală. De exemplu, clasa ClientiDAO extinde allDAO cu parametrul ”Clienti”, în momentul în care sunt generate acțiuni asupra Clienților, sunt accesate metodele din allDAO, metode care identifică tipul clasei si a obiectului care apelează metodele respectiva.*

*Sunt generate câteva metode care au rolul de a returna un String care reprezintă sintaxa SQL care aduce modificări la tabela accesată sau o afișează.*

*Exemplu:*

**private** String createDeleteQuery(String id)

{

StringBuilder sb = **new** StringBuilder();

sb.append("DELETE ");

sb.append("FROM ");

sb.append(type.getSimpleName());

sb.append(" WHERE ");

**if**(type.getSimpleName().equalsIgnoreCase("Clienti"))

sb.append("idClienti= ");

**else** **if**(type.getSimpleName().equalsIgnoreCase("Produse"))

sb.append("idproduse= ");

**else** **if**(type.getSimpleName().equalsIgnoreCase("Comenzi"))

sb.append("idComenzi= ");

sb.append(id);

**return** sb.toString();

}

*Sunt definite metode principale care au rolul de a realiza acțiunile inițiate de către utilizator prin intermediul interfeței grafice, precum findall, findbyid, insert, delete sau update, aceste metode returnând ori o listă de obiecte, cum este în cazul metodei findall, un obiect anume, in cazul metodei findbyid sau aduc anumite modificări asupra tabelei returnând un string care indică dacă acțiunea respectivă a fost efectuată cu succes.*

*Metodele amintite mai sus, realizează în primă faza conexiunea cu baza de date, prin intermediul clasei ConnectFactory, pregătește statement-ul prin metodele de createInsertQuery de exemplu, si se execută Query-ul respectiv, rezultatul fiind salvat într-un ResultSet.*

*Trecerea de la rezultatul de tip ResultSet la o listă de obiecte este realizată prin intermediul funcției CreateObjects, care, inițializează în primă fază o listă de obiecte, sunt parcurse elementele din ResultSet, este salvată instanța obiectului identificat din ResultSet, iar prin intermediul unui for care parcurge field-urile obiectului respectiv, sunt salvate si introduse într-un obiect, valorile extrase din ResultSet, la final, acest obiect fiind adăugat în lista care urmează să fie returnată.*

*La finalul clasei allDAO, este pusă în evidență metoda care creează tabelul care urmează să fie afișat pentru utilizator, pentru a vizualiza rezultatele dorite.*

**public** DefaultTableModel createTabel(List<T> afisare) {

Field[] a=type.getDeclaredFields();

String[] primaColoana = **new** String[a.length];

**int** i=0;

**while**(i!=a.length)

{

a[i].setAccessible(**true**);

primaColoana[i]=a[i].getName();

i++;

}

// in sirul de stringuri primaColoana am salvat numele coloanelor, adica prima linie din tabel

DefaultTableModel tabelModel= **new** DefaultTableModel();

tabelModel.setColumnIdentifiers(primaColoana);

**if** (afisare == **null**) {

**return** tabelModel;

}// daca lista de clienti este goala, atunci se creeaza un tabel care contine doar capul de tabel

i=0;

**int** j=0;

Object[] obiecteleClientului = **new** Object[afisare.size()+1];

**while**(i!=afisare.size())

{

**for**(Field as:a)

{

as.setAccessible(**true**);

T obiectulExtras =afisare.get(i);

**try** {

obiecteleClientului[j]=as.get(obiectulExtras);

} **catch** (IllegalArgumentException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IllegalAccessException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}j++;

}

tabelModel.addRow(obiecteleClientului);

j=0;

i++;

}

**return** tabelModel;

}

***4. Pachetul Model***

*În acest pachet, sunt definite cele 5 clase care conțin variabilele de instanță specifice pentru fiecare obiect, spre exemplu numele, prenumele, mail-ul telefonul, adresa unui client. Aici sunt definite si get-erele si set-erele specifice dar si metodele toString care au fost folosite pentru vizualizarea în consolă rezultatului muncii mele.*

***5. Pachetul Connection***

*În acest pachet, este definită clasa ConnectionFactory, care are rolul de a realiza conexiunea cu baza de date, prin intermediul unui nume de utilizator si a unei parole indicate de către mine.*

*Sunt definite câteva metode de inițializare a unei conexiuni, de inchidere a unui statement, a unui resultSet, s.a.m.d.*

***6. Pachetul Start***

*Conține doar clasa principală main, care definește metoda public static void main, în care este inițializat frame-ul principal la pornirea programului si construit obiectul de tip controller care este practic creierul întregului program*

*.*

**REZULTATELE OBTINUTE**

*În urma rulării programului creat, se pot observa cum sunt posibile anumite acțiuni precum inserarea unui noi client în baza de date, ștergerea, modificarea si afișarea tuturor clienților din baza de date, similar fiind și în cazul elementelor de tip Produs.*

*Practic, este posibil aducerea unor modificări la o bază de date, ulterior, după înregistrare( inserarea clientului cu caracteristicile personale prin care se diferențiază de ceilalți clienți), este posibil generarea unor comenzi care aduc diminuarea stocului produsul pe care dorim să îl comandăm, la final, generarea unei chitanțe sau a unei facturi, fiind de ajutor pentru ambele părți, atât pentru consumator cât și pentru producător.*

**CONCLUZII**

*Concluziile pe care le putem sustrage în urma realizării acestui program, pun în evidență modul în care sunt gestionate acțiunile asupra unei baze de date, care conține mai multe tabele unde sunt stocate anumite informații necesare pentru ținerea unei evidențe în ceea ce privește stocul produselor sau existența unor clienți, solicitarea produselor fiind onorată prin intermediul datelor salvate în baza de date care se ocupă cu Clienții.*

*Practic, atunci când se dorește comandarea unui anume produs, trebuie verificat inițial stocul produsului respectiv pentru a indica dacă este sau nu posibilă efectuarea comenzii, mai apoi, după procesarea comenzii, diminuarea stocului fiind esențială pentru a păstra o informație necesară pentru clienții următori care doresc să comande același produs.*

*Modul în care s-a ușurat scriere de cod, a fost realizat prin intermediul procedeului de Reflexie, un procedeu foarte interesant si vital pentru mai multe cazuri, în care număr este mai mare, fiind mai multe tabele. Accesul la tabelele respective este indicat de tipul care este instanțiat odată primirea parametrului care indică tipul obiectului. Exemplu Client care vine la pachet cu anumite caracteristici care indica field-urile pe care obiectul respectiv le are.*

*Astfel, este pusă în evidență relația dintre tabele, la o modificare într-un anumit tabel, fiind posibilă modificarea automată în tabelul în care există o anumită corespondență.*

*Lucru cu bazele de date permite dezvoltarea unei viziuni mai ample în ceea ce privește lanțul care se formează de exemplu într-o industrie comercială, în care o modificare asupra unei anumite date, propagă modificări ulterioare care permit o bună gestionare a evidenței resurselor.*

***BIBLIOGRAFIE***

<http://users.utcluj.ro/~igiosan/teaching_poo.html>

Materialul pus la dispoziție ca model.