**DOCUMENTAȚIE**

**ORDERS MANAGEMENT**

**TEMA 3**

NUME STUDENT: **ADELIN MOLDOVAN**

GRUPA:**302210**

**CUPRINS**

1. Obiectivul temei
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Proiectare
4. Imprelementare
5. Rezultate
6. Concluzii
7. Bibliografie

**1.Obiectivul temei**

In acest proiect se ia in considerare realizarea unei aplicatii “OrderManagement” cu scopul procesarii vanzarilor unui magazin alimentar. Pentru realizarea acestei cerinte este impusa folosirea unei baze de date. Scopul acestei baze de date este de a inmagazina date despre produsele dispuse, clientii magazinului si comenzile realizate de acestia.Cu mult mai mult, aplicatia trebuie sa fie structurata in mai multe pachete utilizand arhitectura impusa in cerinta proiectului si prezentata in suportul pentru aceasta tema, cu conditia de a se folosi minimum 4 pachete(Model classes, Bussiness logic classes, Presentation classes si Data access classes).

Obiectivele secundare ale acestui proiect sunt:

-stabilirea tabelelor ce vor fi prezente in baza de date;

-realizarea in workbench a bazei de date si legarea tabelelor prin intermediul foreign keyurilor si stabilirea cheilor primare, fara populare(aceasta se va face din java);

-crearea pachetelor necesare implementarii acestui proiect dupa cerintele prevazute anterior;

-implementarea claselor de tabele in care sa fie introduse gettere si settere pentru o mai usoara folosirea;

-producerea legaturilor cu baza de date create anterior;

-realizarea operatiilor de sinserare, de stergere, de update si de selectie din tabelele din baza de date;

**2.Analiza problemei, modelare, scenarii, cazuri de utilizare**

Pentru realizarea acestui proiect este necesara o cunoastere minima de comenzi mySQL invatate in cadrul materiei de baze de date realizare semestrul anterior si in baza caruia a fost nevoie de realizarea unei baze de date a unui aeroport, aspect ce a fost benefic pentru realizarea acestui proiect, Singurele cunostiinte necesare sunt cele de creare a unui tabele cu atributele de diverse tipuri si legarea lor si, totodata, metodele de stergere, inserare si selectie dintr-o tabela. In ceea ce priveste conectarea la baza de date din java, aceasta a fost realizata cu succes dupa prezentarea din bibliografia oferita pentru acest proiect.Trebuie tinut cont si de faptul ca se poate sa fie introduse date care sa nu fie valide: un client care doreste sa cumpere un produs inexistent sau o cantitate dintr-un produs care depaseste cantitatea detinuta de magazine, se introduce acelasi produs(caz in care se va aduna la cantitatea initial noua cantitate), se sterge un comparator sau un produs care nu exista in baza de date, introducea sub forme diverse ale datelor de intrare etc. Pentru a se usura lucrul pe tabele fiecare tabel principal va avea cate un id numar intreg.

**3.Proiectare**

Pentru realizarea acestei teme am folostit strucura de date *List* care este inclusa prin pachetul *java.util*. *List* implementeaza interfata *Collection.* Structura de date *List* contine liste de elemente indexate. Aceste liste pot contine elemente duplicate si astfel permit un control mai precip asupra pozitiei unui element prin intermediul indexului acelui element. Clasa *List* are metode implementate care ne usureaza treaba, de exemplut avem metoda *get(int index),* iar pentru a afla numarul de elemente din lista respectiva folosim metoda *size(),* iar pentru a putea sterge un obiect din lista folosim metoda *remove(Object a)*. In ceea ce priveste pachetele, vom avea 6 pachete pentru realizarea acestui proiect: bll, connection, dao, model, presentation, start. Fiecare pachet la randul lui contine clase specifice pentru interconectarea lor si pentru a putea ajunge la producerea datelor dorite si cerute de cerinta in sine.

**4.Imprelementare**

In cele ce urmeaza vor fi prezentate toate cele 6 pachete impreuna cu clasele corespondente acestora si metodele descrise in acestea.

Astfel avem pachetul **connection** care contine o clasa **ConnectionFactory** in care descriu metodele folosite la conectarea aplicatiei java la o baza de data creata de noi. La aceasta clasa m-am inspirat si am folosit codul primit in cadrul laboratorul in prezentarea aferenta acestui proiect.Asadar clasa **ConnectionFactory** contine atributele necesare pentru conectarea la o baza de date. Pentru a se putea face acest lucru acestea trebuie sa fie initializare inainte. Clasa in sine contine 4 metode statice prin care se face conectarea, initializarea si iesirea la o baza de date. Pentru conectarea in sine la baza de date se folosestea metoda **createConnection**  care returneaza conexiunea de tip *connection.* Mai mult de atat in aceasta clasa mai avem 3 metode folosite pentru inchiderea unei *connection, statement*(interogare) si *resultSet*(rezultate interogare).

Urmatorul pachet este pachetul **bll** contine pachetul **validators** unde sunt implementate 3 clase care pun conditii asupra datelor de intrare si care dupa cum le spune numele le valideaza. Pe langa acest pachet mai sunt inca 4 clase: *ClientBLL, OrdersBLL, ProductBLL* si pe langa acestea o exceptie denumita *ExceptieStock* pe care am folosita pentru a putea vedea daca lipsesc unele produse din stoc, adica daca nu sunt suficiente. Astfel celelalte 3 clase din acest pachet au urmatoarele roluri:

Clasa *ClientBLL* are atributul *validators* care este de tipul **List<Validators<Client>>,** astfel avem o lista de validatori de tipul client. Constructorul clasei creeaza o instata de tipul clase care au rolul de a valida datele de intrare. Metodele din aceasta clasa sunt **insertClient, findAll, updateClient, deleteClient si fintClient**(aceasta clasa am facut-o dupa exemplul primit in cadrul laboratorului si astfel ca am pus drepturile de autor pentru a fi totul ok). In aceste metode apeleze si metoda **validate** pentru a putea verifica datele. In cazul in care datele nu respecta conditiile de validare, aceasta metoda va arunca o exceptie, astfel metoda din clasa *ClientBLL,* arunca la randul ei exceptia primita, urmand sa ia expectia si sa o trateze, acestea intamplandu-se in interfata grafica afisand un mesaj text utilizatorului aplicatiei. In cazul clientilor verific doar varsta clientilor sa fie intre 10 si 60 de ani dupa cum a fost desrisa in prezentare.

Clasa *ProductBll*, are si ea atributul *validators,* de asemenea de tipul **List.** Astfel avem o lista de validatori pentru produse. In aceasta clasa am implementate urmatoarele metode findPrduct, insertProduct, findAll, updateProduct si deleteProduct. Aceste metode definite definesc o instanta noua de tipul **ProductDAO** si mai apoi creeaza interogarea.

Clasa *OrdersBll* are si ea de asemenea atributul *validators(*pe care l-am discutat si prezentate si in clase descrise anterior). In aceasta clasa avem constructorul respectiv care creeaza o instanta de tipul **List<Validators<Orders>>** si adauga un validator acestei liste. In aceasta clasa am implementat urmatoarele metoda *insertOrder*. Aceasta metoda poate arunca o exceptie de tipul ExceptieStock care este aruncate de metoda de validare pe care am apelat-o. Se apeleaza metoda de validare din *OrderStockValidator* unde verific daca stocul produsul este mai mic decat cantitatea ceruta de client si se va arunca o exceptie de tipul *ExceptieStock* , care mai apoi va fi tratata In interfata grafica. Tot aici am creat un fisier text cu detaliile clientului, produsul si detaliile despre comanda pe care a facut-o.

Urmatorul pachet este pachetul **model** care contine clasele principale care sunt prezente in tabelele initiale din baza de date. Astfel in acest pachet avem urmatoarele clase: *Clasa Client, Clasa Product, Clasa Orders.* **Clasa Client** contine urmatoarele atribute: id(de tipul int), name(de tipul String), address(de tipul String), email(de tipul String), age(de tipul int). Contructorul aceste clase are paramatri identici cu atributele descrise mai sus. In aceasta clasa am generat gettere si setter prin intermediul intellij-ului si de asemenea si metoda toString a fost generata prin aceasi metoda pentru a fi mai usor de implementat. **Clasa Product** contine urmatoarele atribute(specifice si baza de date generata in MYSQL Workbech): id(de tipul int), name(de tipul String), stock(de tipul int). Astfel constructorul acestei clase are si el 3 parametri acestia fiind identici cu atributele de mai sus. Avem si un construcot null pentru a creaa o instanta a clasei. De asemenea si in aceaasa clasa am generat gettere si settere prin intermediul intellij-ului si la fel si metoda toString(). **Clasa Orders** contine urmatoarele atribute: idOrder(de tipul inte), idClient(tot de tipul int), nameClient(de tipul String), idProduct(de tipul int), nameProduct(de tipul String) si ultimul este quantity(de tipul int). Constructorul clasei are parametrii identici cu atributele. Si la fel si in aceasta clasa am general gettere, settere prin intermediul intellij.

Urmatorul pachet este pachetul **dao**(data access object) este peachetul principal ce functioneaza pe baza de date. Acest pachet contine urmatoarele clase: *ClientDAO, ProductDAO, OrdersDAO* care mostenesc clasa *AbstractDAO.* Aceasa din urma este cea care transmite in baza de date comenzile. In aceasta clasa avem ca si argument de tip private Logger-ul pentru logarea in baza de date, respectiv type-ul, care stabileste tabela pe care se lucreaza metodele de tip query, au ca si scop crearea comenzii care trebuie trimisa in baza de date ca si fie exectutata si se realizeza printr-un StringBuilder. Prin intermediul acestora se realizeaza functiile de insert, update, delete si de selectie din tabele. Selectia se face atat in functie de id, cat si de nume si se returneaza un obiect de tipul cerut, dar se poate si genera o lista ce contine totalitatea obiectelor de acel tip din baza de date. In acest pachet se folosesc tipurile generice pentru realizarea lucrului pe un anumit tabel din baza de date, nefiind necesare alte metode in clasele ce mostenesc clasa principala. Clasa **AbstractDAO** este clasa prin care se genereaza interogarile prin tehnica numita reflexie. Tehnica reflectiei permite construirea interogariilor dinamic in functie de clasa primta ca argument. De exmplu prin tehnica reflectiei se pot obtine numele atributelor unei clase, astfel in cazul nostru avand clasele client, product si order(cu atribute numite la fel cu campurile tabelei) putem construi foarte usor o metoda(care genereaza si exectua o anumita interogarea) care poate fi mostenita si utlizata de orice clasa fara a modifica codul. In aceasta clasa sunt descrise metodele precum *findById(String setCol, String a)* primesc ca si argumente numele unei coloane si valoarea pentru a putea cauta elemente in tabele dupa aceste criterii. In interiorul metodei se creeaza conexiuniile cu baza de date, se apeleaza metoda *createSelectQuerey(String setCol)* unde se genereaza interogarea, *createObjects(resultSet)* care returneaza o lista de obiecte. Aceasta metoda functioneaza astfel, cat timpul mai multe rezoltate interogarii, se creeaza o instanta de tipul T, se itereaza fiecare camp al clasei T, si astfel pentru fiecare camp al clasei se obtine rezultatul interogaii si se actualizeaza atributele obiectului creeat. Toate celelalte clase DAO mostenesc metodele clasei AbstractDAO, si aceste metode genereaza interogariia asupra produselor, clientilor si comenzilor.

Urmatorul pachet este pachetul **start.** In acest pachet avem o clasa *Start* unde avem metoda

public static void main(String args[])

care incepe executia programului.

Ultimul pachet este pachetul **presentation**. In acest pacheta avem o singura clasa si anume View. In aceasta am definit partea de controller si partea de view. Mi s-a parut a fi o abordate mai eficienta si mai usor de inteles si realizat asa deoarece scriam totul intr-o singura clasa si astfel a fost mult mai usor de urmarit. Clasa extinde mai departe clasa Jframe si implementeaza ActionListener. Astfel constructorul clasei va adauga atributele definite in diferite panouri si astfel se va genera fereastra principala. Pentru panouri am folosit un layout de tipul *GridBagLayout(),* acesta fiind foarte usor de folosit deoarece este organizat sub forma de matrice. Astfel ca putem specifica linia si coloana. Pe fiecare buton se implementeaza ferestre noi, acestea ferestre le-am creat cu Jdialog. In metoda *ActionPerfomed,* se implementeaza daca un eveniment s-a intamplat asupra butoanelor. Pentru a construi o fereastra nou de dialog cu rezultatele noi se apeleaza metoda *getProprietati().* Aceasa metoda construieste un tabel, prin metoda reflexiei, din lista de obiecte primita ca si parametru.Tabelel au fost create prin JTable. In aceasta am foloist metodele *addColumn* si *addRow* care primesc vectori de siruri. Si mai avem metoda *getNumeColoana(Obj)* care returneaza un vector de siruri cu numele atributelor din obiectul primit ca si parametru.

**5.Rezultate**

Modul de a se verifica corectitudinea programului scris este prin a verifica pe parcurs prin intermediul selectului aplicat direct in workbench.

**6.Concluzii**

Acest proiect a avut ca si scop intelegerea mai bine a bazelor de date si de a genera rapoarte specific pentru anumite cerinte ale unui utilizator. Ca dezvoltari ulterioare se pot genera o multitudine de proiect din domeniul bazelor de date. Aceasta baze de date este una simpla, specifica unui mic magain de cartier sau a unui aprozar, dar poate fi dusa mai departe la un lanta de hypermarketuri ce au nevoie de o baza de date foarte bine pusa la punct pentru gestionarea clientilor, furnizorilor, produselor detinute pe stocuri, atat alimentare cat si nealimentare. Atributele pot fi mult mai multiple si de multe ori pot fi generate mesaje prin care sa se atentioneze neregulile cu privire la anumite produse sau comenzi. Prin aceasta modalitate se poate verifica daca a avut loc o eroare umana in realizarea unei facturi, in cazul in care cantitatea produselor din stoc nu corespunde cu cea a produselor din vaza de date a magazinului. Astfel cu acest proiect am invatat cum sa conectez o baza de date la o aplicatie java. Pentru a interactiona cu baza de date o librarie speciala .jar a trebuit adaugata in aplicatie(aceasta librarie este folosita pentru a comunica cu MYSQL server). Am creat un tabel utilizand Jtable. De asemenea in timpul realizarii acestui proiect am invata conceptul de reflexie in java. Am utilizat aceast concept pentru a genera automat interogarile in functie de clasa din care am apelat metoda.

**7.Bibliografie**

<https://gitlab.com/utcn_dsrl/pt-reflection-example/-/tree/master/>

<https://stackoverflow.com/questions/1051640/correct-way-to-add-external-jars-lib-jar-to-an-intellij-idea-project>

<https://www.geeksforgeeks.org/>

<https://mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/>

<https://www.baeldung.com/javadoc>

<http://tutorials.jenkov.com/java-reflection/index.html>