**TEMA 3 – DOCUMENTATIE**

Ghitun Patricia Roxana

Grupa 30227

Cuprins :

1. Obiectivul temei
2. Analiza problemei
3. Proiectare
4. Implementare
5. Testare
6. Rezultate
7. Concluzii
8. Bibliografie
9. Obiectivul temei

Obiectivul acestei teme este de a dezvolta o aplicatie care se ocupa cu managementul unui magazine . Aceasta aplicatie interactioneaza cu baza de date si utilizatorul care doreste sa vada prima data ,dupa care sa cumpere produsele existente in magazine . Utilizatorul poate atat sa vada produsele, dar si istoricul comenzilor care s-au facut pana in acel timp. Acesta poate sa vada , sa adauge, sa stearga si sa actualizele produse. La finalizarea unei comenzi se vor afisa datele despre comanda respectiva pe o factura .

1. Analiza problemei

Pentru rezolvarea acestui proiect am avut nevoie de cunostinte in domeniul bazelor de date mysql . Comenzile executate in acest proiect au fost folosite cu aceste cunostinte . Pentru fiecare operatie a fost implementata o metoda ce foloseste o interogare sql .

1. Proiectare

Pentru a realiza acest proiect , am creat o baza de date “bd” folosind mySQL Workbench 6.0 . Am folosit aceasta baza de date pentru a crea tabele ce vor retine datele pe care le voi folosi ulterior in proiect . Pe baza acestei baze de date am creat interogari pentru diferite operatii precum : afisare , adaugare , stergere si actualizare .

Ca structuri de date folosite , am ales folosirea ArrayList<>-urilor pentru a retine datele despre fiecare element dintr-o tabela din baza de date .

In baza de date au fost create trei tabele :

Tabela Customer : Aceasta tabela retine clientii care vin la magazine si fac o comada de produse . S-a retinut pentru fiecare client un identificator de tip INT , numele de tip VARCHAR , varsta de tip INT si adresa sa de tip VARCHAR de asemenea .

Tabela Product : Aceasta retine produsele existente in magazin . Pentru fiecare produs am retinut urmatoarele date de identificare : id-ul sau , numele produsului , pretul unui singur produs de acest tip si stocul sau disponibil in magazin .

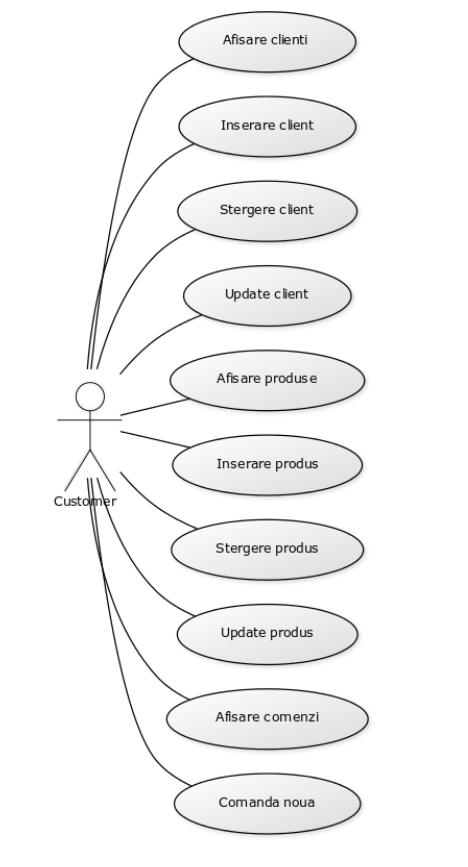
Tabela Order : Aceasta tabela retine comenzile efectuate de catre clienti . Pentru fiecare comanda se inregistreaza : id-ul comenzii , id-ul clientului care face comanda respectiva , id-ul produsului cumparat si de asemenea totalul de plata al comenzii care va fi calculat ulterior din inmultirea pretului produsului cu cantitatea cumparata .

Proiectul este structurat pe o serie de pachete , fiecare avand un rol bine determinat .

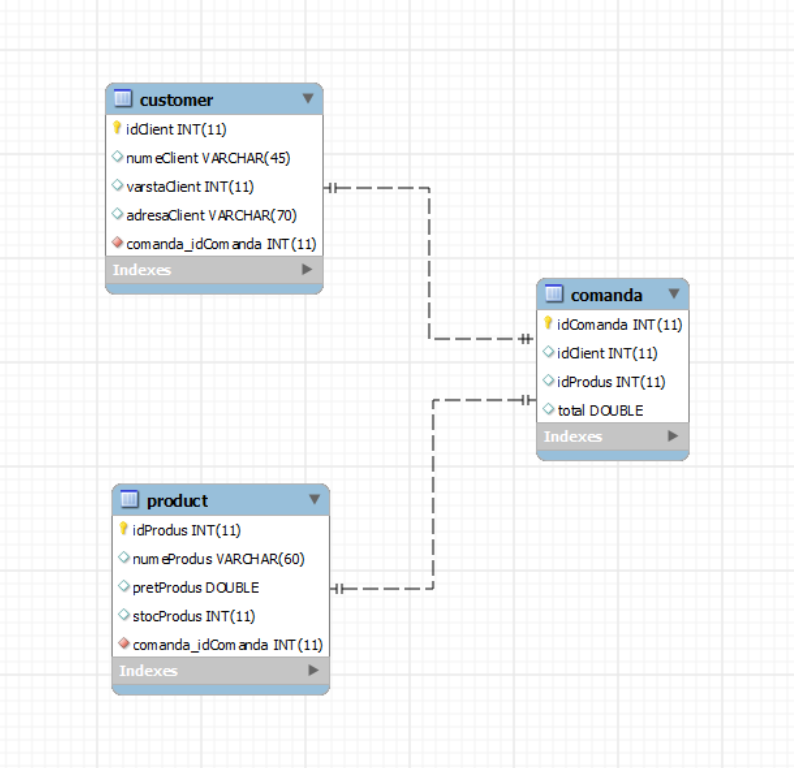
Pachetele sunt :

* MODEL : Acest pachet contine clasele corespunzatoare tabelelor din baza de date .
* PRESENTATION : Acest pachet contine clasele ce implementeaza interfata grafica cu utilizatorul .
* BLL : Business Layer : Acest pachet reprezinta clase ce incapsuleaza logica aplicatiei .
* DAO : Data Access Layer : Acest pachet contine clasele ce realizeaza interogarile ce comunica cu baza de date . Aici sunt metodele ce implementeaza query-ul cu care se acceseaza datele din baza de date .
* CONNECTION : Acest pachet contine clasa ConnectionFactory ce face legatura aplicatiei cu baza de date .

**USER CASE :**



**DIAGRAMA BAZEI DE DATE :**

 In aceasta diagrama se pot observa tabelele : Customer retine informatiile despre un client , Product despre un produs iar comanda despre o comanda .

Fiecare dintre acestea are o cheie primara (idClient/Produs/Comanda) . Tabelul Comanda are in plus doua chei straine : idClient si idProdus;

**DIAGRAMA DE PACHETE :**

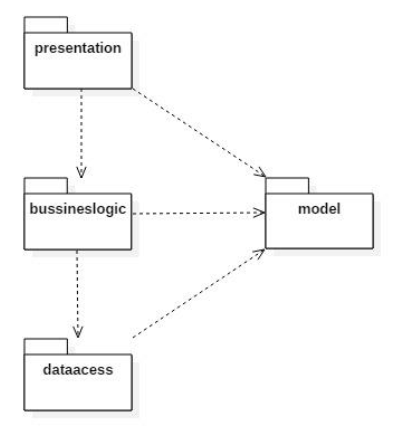
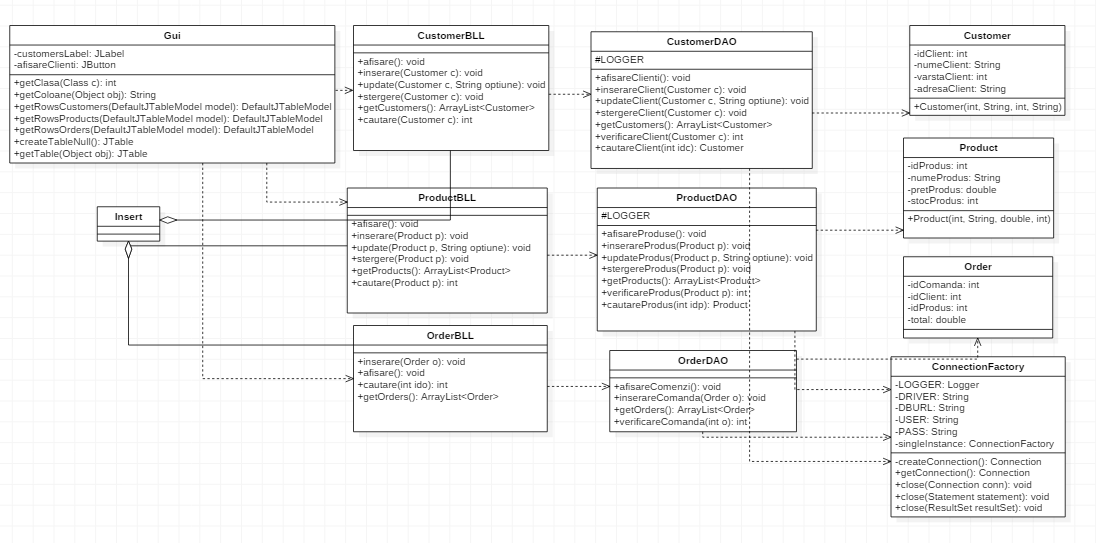
Aplicatia a fost realizata cu ajutorul Layered Architecture , fiind impartita pe pachete ce comunica intre ele doar intr-o singura directie.

DIAGRAMA UML DE CLASE :

Cu ajutorul acestei diagrame UML a avut loc proiectarea intregului proiect, fiind evidentiate pachetele , clasele , metodele dar si legaturile dintre ele , respectand formatul OOP.



4.Implementare

Pachetul **MODEL** :

* Clasa **Customer** , clasa **Product** si clasa **Order** :Fiecare defineste pe rand atributele unui client, sau produs sau a unei comenzi . Toate aceste trei clase corespund tabelelor Customer , Product si Order din baza de date a magazinului . Fiecare dintre ele implementeaza interfata Comparable<Object> pentru a sorta ulterior clientii , produsele si comenzile dupa id-ul lor . S-a implementat acest lucru pentru o vizualizare cat mai ordonata a datelor .

Pachetul **DAO** : Acest pachet contine clasele care se ocupa de legatura cu baza de date .

* Clasa **CustomerDAO** : in aceasta clasa sunt implementate metodele ce folosesc interogari
* pentru a accesa date din baza de date .

Sunt implementate cinci metode , fiecare pentru afisarea , inserarea , actualizarea, stergerea si cautarea clientilor . In plus mai sunt implementate si doua metode , una de afisare a datelor in consola si una de verificare daca sa nu existe doi client cu acelasi id . Aceasta ultima metoda a fost implementata pentru a fi folosita de exemplu la adaugarea unui nou client , sa nu adaugi unul care are deja acel id , sau la stergere , sa verifice inainte de a sterge daca exista clientul pe care se doreste sa fie sters , la fel si la actualizare .

Se apeleaza Collections.sort in metoda de getCustomers ( metoda ce returneaza un ArrayList de customeri ) pentru a sorta clientii dupa id .

* Clasa **ProductDAO** : este similara cu clasa CustomerDAO ,doar ca adaptata pentru produsele din magazin . La fiecare metoda implementata se apeleaza ConnectionFactory.getConnection() pentru a realiza initial conexiunea cu baza de date .
* Clasa **OrderDAO** : Aceasta clasa este putin diferite de celalalte doua deoarece nu implementeaza si metodele de actualizare si stergere comanda, ci doar de afisare si inserare .

Pachetul **BLL** : Acest pachet mapeaza clasele din dao si are un subpachet numit validators unde sunt create 2 clase de validare a varstei si numelui unui customer . Cele doua clase implementeaza interfata Validator ce contine metoda validate .

In acest pachet fiecare clasa din DAO are un corespunzator : CustomerBLL , ProductBLL , OrderBLL . In fiecare dintre aceste clase s-a creat cate o metoda pentru fiecare metoda din dao care in momentul in care este apelata , este apelata si metoda din dao .

Acest lucru a fost implentat pentru a nu folosi in Presentation clasele din DAO .

Pachetul **CONNECTION** : In acest pachet este create clasa **ConnectionFactory** care realizeaza conexiunea bazei de date cu aplicatia .

Pachetul **DEFAULT** : in acest pachet este clasa main din care se porneste aplicatia .

Pachetul **PRESENTATION** : acest pachet incapsuleaza clasele care se ocupa cu interfata grafica a aplicatiei . El contine clasele : Gui , InsertCustomer , InsertProduct , NewOrder , DeleteCustomer , DeleteProduct , UpdateCustomer , UpdateProduct , ViewCustomers , ViewProducts si ViewOrders.

Interfata grafica a fost implementata cu ajutorul mai multor clase , cea de baza fiind clasa Gui . In aceasta clasa am creat fereastra de baza a aplicatiei ce este reprezentata de trei coloane, cate una pentru fiecare tabel din baza de date .

Butoanele din interfata creeaza un obiect de tipul operatiei dorite si prin constructorul acestuia se deschide un o noua fereastra . Fiecare operatie descrisa pe buton este implementata intr-o clasa noua , specifica pentru fiecare tabel , adica : o clasa pentru inserare de client , o clasa pentru inserare produs , etc . Fiecare dintre aceste clase folosesc obiecte din pachetul BLL pentru a avea acces la metodele din baza de date corespunzatoare operatiei alese. Fiecare fereastra se va inchide datorita setarii frame.setDefaultCloseOperation (JFrame.EXIT\_ON\_CLOSE) pe butonul x.

Pentru a afisa in tabele datele din baza de date s-a folosit metoda reflection .

In clasa Gui am creat :

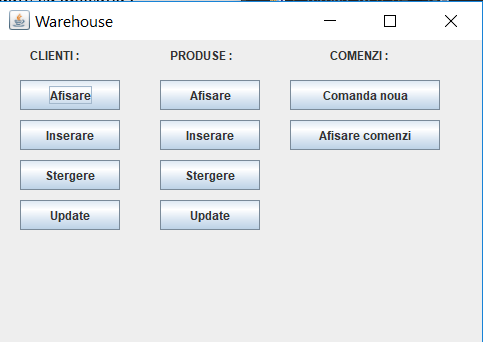
* o metoda “getClasa” ce returneaza un intreg ce reprezinta : 0 , pentru clasa Customer , 1 pentru clasa Product si 2 pentru clasa Order . Aceasta metoda verifica daca parametrul dat , o clasa , este una dintre cele trei clase .
* o metoda “getColoane” ce returneaza un vector de string-uri ce vor reprezenta coloanele tabelului . Metoda are ca parametru un obiect si cu ajutorul unui vector de field-uri retin atributele clasei din care face parte obiectul respectiv . Dupa aceea , parcurg sirul de atribute si ii adaug numele in vectorul de string-uri.
* Pentru fiecare tabel am creat cate o metoda “getRowsCustomers/Products/Orders” . In fiecare dintre aceste metode am un sir de obiecte “rows” si cate un arraylist de obiecte(de tipul tabelului) si un sir de metode preluate cu ajutorul metodei .getDeclaredMethods();

Dup ace parcurg metodele , invoc metoda cu numele de get ce returneaza un arrayLIst pe care il salvez in arraylist-ul declarat. Dupa ce am acest arraylist , il parcurg cu fiecare obiect si il adaug in vectorul de obiecte rows . In final , aceasta metoda returneaza un model pentru tabel , model in care este adaugat acel vector creat rows.

* O metoda de createTableNull ce creeaza un tabel gol si ii seteaza proprietatea de a nu fi editat .
* O metoda getTable . Aceasta este principala metoda care va fi ulterior apelata de fiecare data cand vreau sa afisez un tabel pe ecran . In interiorul acestei metode declar vectorul de coloane ce primest date de la metoda getColoane si un model . Daca obiectul dat ca parametru este 0 , 1 , 2 (client produs sau comanda – apelez functia getClasa pentru a afla asta) atunci modelul va primi rezultatul metodei respective . In final metoda getTable returneaza un model cu coloanele preluate intr-un vector de string-uri si datele preluate cu ajutorul invocarii metodei din clasa respectiva.

1. Testare

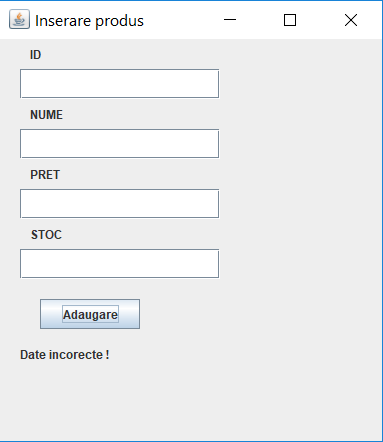
La momentul rularii programului se va deschide o fereastra cu toate optiunile utilizatorului : optiuni legate de client , de produse sau de comenzi , cum ar fi :

Utilizatorul are 3 coloane , fiecare cu cate un buton ce reprezinta operatia implementata .

De exemplu , daca utilizatorul doreste sa vada toti clientii sau toate produsele sau chiar toate comenzile , acesta poate apasa butonul de afisare corespunzator coloanei pe care o doreste .

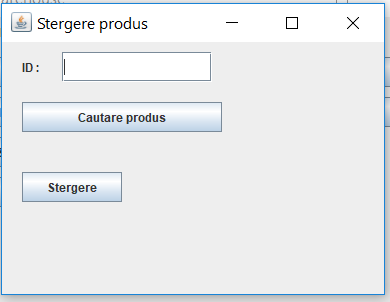
Coloanele de client si produse au aceleasi denumiri pe butoane deoarece fac lucruri similare .

De exemplu , daca se doreste sa se insereze un nou produs , va aparea urmatoarea fereastra :

Datele pe care trebuie sa le introducem in momentul inserarii unui nou produs , similar unui nou client sunt : id-ul , numele , pretul si stocul . Aceste doua elemente din urma sunt inlocuite cu varsta si adresa in cazul inserarii unui nou client in baza de date .

In cazul in care ramane vreun camp gol , se va afisa un mesaj “Date incorecte!”.

In cazul in care , id-ul introdus este la fel cu al unui produs sau client existent deja in baza de date , se va afisa in acelasi loc un mesaj “Produs existent !” sau in cazul inserarii clientului : “Client existent !”.

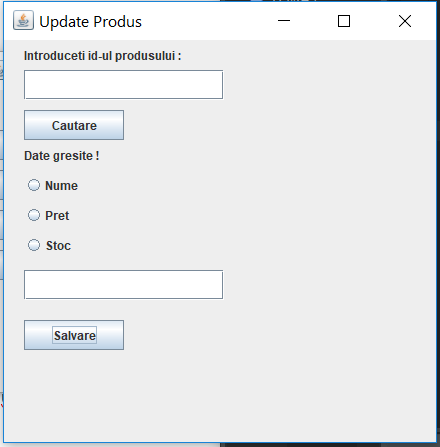
Pentru stergerea unui client/produs avem urmatoarea fereastra :

Daca campul id-ului ramane gol , va fi afisat un mesaj sub buton “Id-ul introdus nu este corect ”.

Dupa ce a fost introdus id-ul , se cauta In baza de date pentru a vedea daca apartine sau nu (se foloseste metoda de cautare din BLL) iar daca da , se va afisa prodsului sau clientul respective iar in caz contrat un mesaj ca nu a fost gasit .

La final , la apasarea butonului de stergere , produsul sau clientul va fi sters din baza de date utilizand metoda stergere din BLL.

Pentru actualizarea unui client/produs existent va fi afisata urmatoarea fereastra :

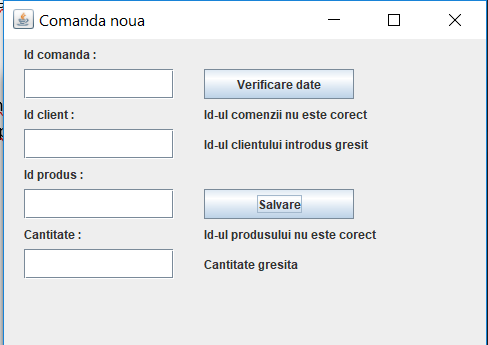
Daca in campul de id nu se introduce nimic , va fi afisat mesajul de “Date gresite” .

Dupa ce se introduce id-ul , se apasa butonul cautare si din nou se apeleaza metoda de cautare in baza de date . Daca se gaseste clientul/produsul , acesta va fi afisat iar in caz contrar va fi afisat un mesaj .

Dupa ce a fost gasit produsul sau clientul , utilizatorul trebuie sa aleaga una dintre cele trei optiuni prezentate : Nume, Pret sau Stoc in functie de ce doreste sa modifice .

Dupa ce selecteaza o optiune, trebuie sa introduca noile date corespunzatoare in al doilea camp dupa care apasa “Salvare” . In acest moment , modificarile vor fi salvate in baza de date in tabelul respectiv .

Pentru a crea o noua comanda, se va apasa pe butonul de “Comanda noua” de pe coloana “Comenzi” . La apasarea acestui buton va aparea fereastra :

Pentru inceput, utilizatorul va introduce un id al comenzii si un id al clientului care face comanda respectiva. Dupa introducerea acestor date , se va apasa butonul de verificare date pentru validarea lor . In cazul in care campurile sunt goale vor fi afisare mesajele corespunzatoare (“Id-ul comenzii nu este correct ” si “Id-ul clientului introdus gresit”).

In cazul in care id-ul comenzii este deja introdus in baza de date inseamna a fost create deja o comanda cu acest identificator si atunci va fi afisat un mesaj de eroare .

Daca id-ul clientului nu apartine bazei de date , adica acel client nu este inregistrat , va aparea un mesaj de eroare sub butonul de “Verificare date”.

Dupa ce ambele id-uri au fost validate , se pot introduce datele legate de produsul dorit , cum ar fi : id-ul produsului si cantitate pe care dorim sa o cumparam . De asemenea , in cazul in care id-ul produsului pe care l-am introdus nu apartine bazei de date , se va afisa un mesaj corespunzator .

In cazul in care , id-ul apartine bazei de date , dar cantitatea introdusa este mai mare decat stocul disponibil , va fi afisat un mesaj de “Stoc insuficient” , iar astfel clientul nu poate comanda mai mult decat stocul disponibil in magazin .

1. **Concluzii**

Din aceasta tema am imbunatatit cunostintele legate de gestionarea bazelor de date si folosirea acestora in Java .

Dezvoltari ulterioare :

Aplicatia reuseste cu succes sa creeze comenzi bazate pe datele din baza de date , dar , pe o comanda poate fi adaugat un singur produs . Ca dezolvare ulterioara se va implementa un algoritm prin care sa fie posibila adaugarea mai multor produse intr-o comanda , ca in viata de zi cu zi . Pentru aceasta implementare trebuie creat un nou tabel ProductOrdered in care sa avem ca si atribute : id-ul produsului , id-ul comenzii din care face parte , cantitatea cumparata pe comanda respectiva si eventual pretul total al produsului (cantitate \* pret pentru bucata )

In acest fel va fi posibila efectuarea mai multor comenzi .

Bibliografie

<http://www.coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW3_Tema3/Tema3_HW3_Indications.pdf>

<https://docs.oracle.com/javase/tutorial/jdbc/basics/processingsqlstatements.html>

<https://stackoverflow.com/questions/13400075/reflection-generic-get-field-value>

<http://www.java2s.com/Tutorial/Java/0240__Swing/CreatingaJTable.htm>

<https://alvinalexander.com/java/edu/pj/jdbc/jdbc0002>

<https://www.geeksforgeeks.org/reflection-in-java/>

<http://www.baeldung.com/java-reflection>

<https://www.javatpoint.com/steps-to-connect-to-the-database-in-java>

<https://alvinalexander.com/java/edu/pj/jdbc/jdbc0003>