ORDERS MANAGEMENT

Nume:Nita Iosif-Gabriel

Grupa: 30223

# CUPRINS

[1. Obiectivul temei 3](file:///C:\Users\Gabriel\Downloads\PT2021-2022_Documentation_Template%20(3).doc#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](file:///C:\Users\Gabriel\Downloads\PT2021-2022_Documentation_Template%20(3).doc#_Toc95297886)

[3. Proiectare 3](file:///C:\Users\Gabriel\Downloads\PT2021-2022_Documentation_Template%20(3).doc#_Toc95297887)

[4. Implementare 3](file:///C:\Users\Gabriel\Downloads\PT2021-2022_Documentation_Template%20(3).doc#_Toc95297888)

[5. Rezultate 3](file:///C:\Users\Gabriel\Downloads\PT2021-2022_Documentation_Template%20(3).doc#_Toc95297889)

[6. Concluzii 3](file:///C:\Users\Gabriel\Downloads\PT2021-2022_Documentation_Template%20(3).doc#_Toc95297890)

[7. Bibliografie 3](file:///C:\Users\Gabriel\Downloads\PT2021-2022_Documentation_Template%20(3).doc#_Toc95297891)

Obiectivul temei

Obiectivul principal

Obiectivul principal al acestei teme este proiectarea si implementarea unei aplicateii responsabile cu procesarea comenzilor unor clienti intr-un depozit, utilizant baze de date relationale pentru stocarea clientilor, a produselor si a comenzilor corespunzatoare. In plus, acesata aplicatie trebuie structurata dupa o anumita arhitectura, si anume layered architecture.

Pasii urmati pentru indeplinirea obiectivului sunt:

* Dezvoltarea unor scenario in care aplicatia ve fi dezvoltata
* Alegerea structuriilor de date, structura proiectului trebuie sa fie una eficienta
* Impartirea corespunzatoare a claselor si a rolurilor acestora
* Implementarea si dezvoltarea algoritmilor corespunzatori pentru toate cazurile
* Implementarea solutiilor, combinarea algoritmilor
* Testarea si gasirea cazurilor care pot cauza probleme aplicatie, remedierea pe cate se poate a acestora.

Analiza problemei, modelare, scenarii sau cazuri de utilizare

Procesare comenzilor unor client este un proces foarte intalnit in viata de zi cu zi. Pentru implementarea acestei aplicatii am folosit o baza de date cu trei tabele corespunzator clientilor, produselor si comenzilor. Am creat 3 interfete grafice intuitive si destul de usor de folosit si pentru o persoana care nu este specializata in domeniul acesta. Utilizatorul poate introduce un nou client in tabelul corespunzator din baza de date, poate sterge un client existent introducand id-ul acestuia, ii poate modifica numele cat si adresa. Poate cauta un client anume dupa id-ul acestuia si poate afisa toti clientii din tabel.

De asemena, utilizatorul poate introduce un produs nou in tabela corespunzatoare din baza de date poate modifica numele unui produs deja existent din tabel, poate modifica pretul si cantitatea produselor existente, poate sterge un produs care exista deja in tabel, poate gasi un produs dupa id si poate afisa toate produsele din tabelul acestora.

Utilizatorul poate selecta clienti cat si produse si poate introduce o anumita cantitate pentru a plasa o comanda. Un client poate plasa o comanda doar pentru un singur produs, fiecare comanda in parte este afisata intr-un fisier text asociat celei comenzi.

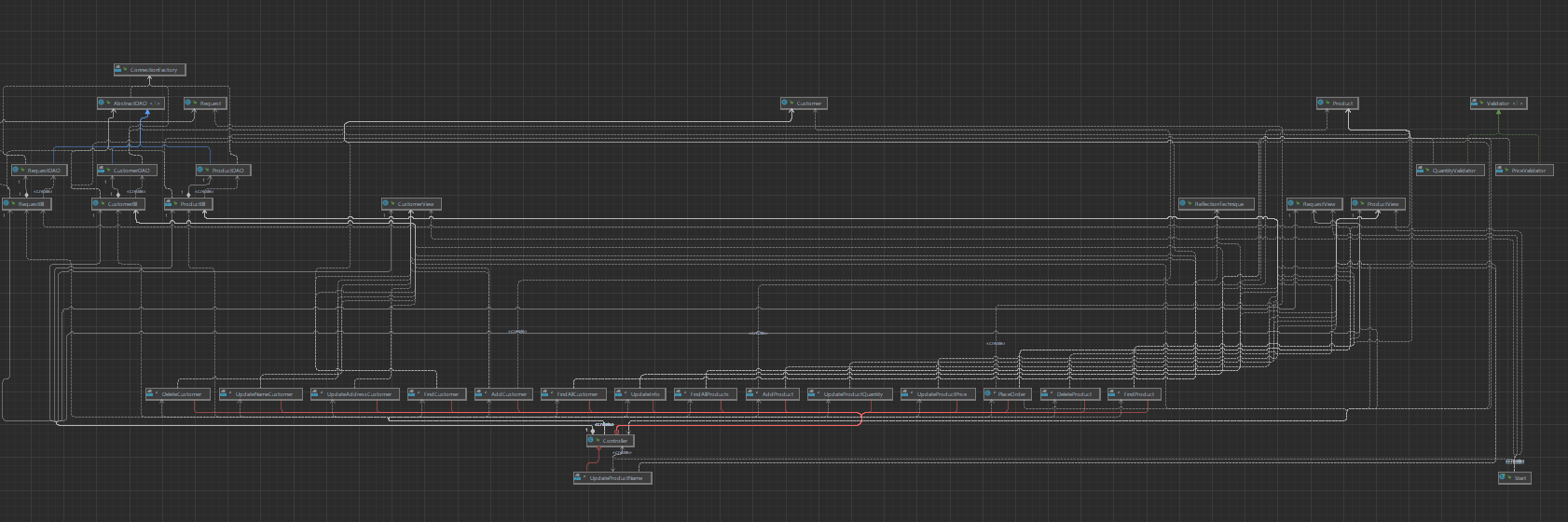
Fiecare tabela in parte are o interfata grafica usor de utilizat.

# Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

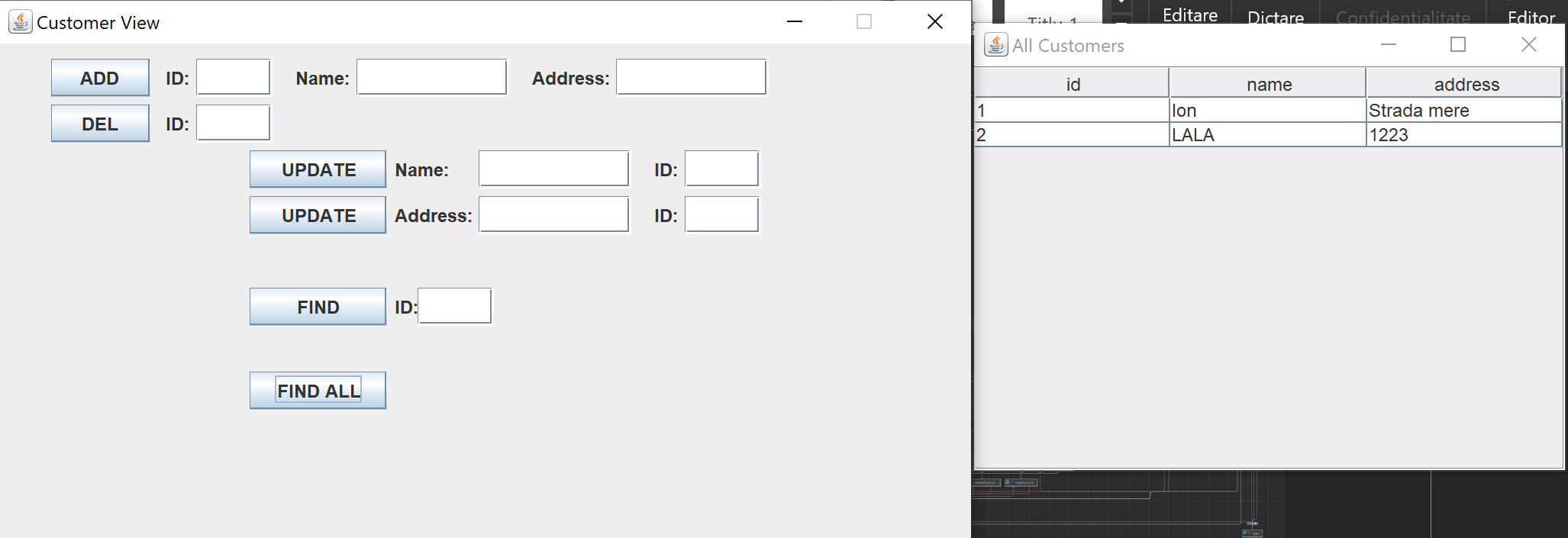
|  |  |
| --- | --- |
| **Package** | **Class** |
| main -> java -> **connection** | ConnectionFactory |
| main -> java -> **model** | Customer  Product  Request |
| main -> java -> **dao** | AbstractDAO  CustomerDAO  ProductDAO  RequestDAO |
| main -> java -> **bll** | CustomerBll  ProductBll  RequestBll |
| main -> java -> bll -> **validators** | Validator (**Interface**)  ProductPriceValidator  ProductQuantityValidator |
| main -> java -> **presentation** | CustomerView  ProductView  RequestView  Controller |
| main -> java -> **start** | ReflectionTechnique  Start |

|  |  |
| --- | --- |
| **Class** | **Descriere** |
| ConnectionFactory | Clasa are atributele necesare conectarii la baza de date: numele driver-ului, locatia bazei de date, numele de utilizator și parola pentru accesarea server-ului MySQL. Ca și metode avem: metoda care creeaza conexiunea cu baza de date, metoda care returneaza conexiunea creata, metoda care inchide o conexiune, metoda care inchide un query și metoda care inchide rezultatul returnat de un query. |
| Customer | Clasa are atributele care descriu un client: id, nume și adresa. Pe langa metodele prestabilite de set și get și de constructori, mai avem metoda suprascrisa toString pentru o afișare mai decenta. |
| Product | Clasa are atributele care descriu un produs: id, nume, cantiate și pret. Pe langa metodele prestabilite de set si get si de constructori, mai avem metoda suprascrisa toString pentru o afisare mai decenta. |
| Request | Clasa are atributele care descriu o comanda: id, numele produsului comandat, numele clientului care a facut comanda, cantiate și pret. Pe langa metodele prestabilite de set și get si de constructori, mai avem metoda suprascrisa toString pentru o afisare mai decenta. |
| AbstractDAO | Clasa generica care contine metodele de accesare specifice tuturor tabelelor din baza de date. Avem un parametru important care ne spune tipul clasei obiectului cu care a fost apelata o anumita metoda. Ca si metode avem: metode care formeaza query-urile de care avem nevoie, metoda care returneaza toate elementele aflate la un moment dat intr-un anumit tabel, metoda care gaseste un anumit element dupa id, metoda care adauga un element intr-un anumit tabel, metoda care sterge un element dupa id dintr-un anumit tabel, metoda care modifica numele unui client sau a unui produs din tabel si o metoda care creeaza o lista de obiecte de tipul clasei din model asociate tabelului corespunzator. |
| CustomerDAO | Clasa care conține metodele de accesare a bazei de date specifice tabelului client. Avem o singura metoda specifica tabelului client: modificarea adresei unui client deja existent în tabelul aferent în funcție de id-ul dat corespunzator. |
| ProductDAO | Clasa care contine metodele de accesare a bazei de date specifice tabelului produs. Avem doua metode specifice tabelului produs: modificarea cantitații unui produs deja existent in tabel în funcție de id-ul corespunzator si modificarea pretului unui produs deja existent in tabelul aferent în funcție de id-ul corespunzator. |
| RequestDAO | Clasa care conține metodele de accesare a bazei de date specifice tabelului comanda. Avem patru metode specifice tabelului comanda: modificarea numelui unui client care a plasat o comanda deja existenta în funcție de id-ul corespunzator, modificarea numelui unui produs deja existent într-o comanda în funcție de id-ul corespunzator, modificarea cantitatii unui produs dintr-o comanda deja existenta în funcție de id-ul corespunzator și modificarea prețului deja existent într-o comanda în funcție de id-ul corespunzator. |
| CustomerBll | Clasa care se ocupa cu logica aplicației legata de tabelul client. Clasa are un singur atribut principal și anume un obiect de tipul ClientDAO care asigura conectarea logicii cu metodele care manipuleaza baza de date. Toate metodele care se afla aici apeleaza metodele care acceseaza baza de date prin prisma atributului. |
| ProductBll | Clasa care se ocupa cu logica aplicației legata de tabelul produs. Clasa are un singur atribut principal și anume un obiect de tipul ProdusDAO care asigura conectarea logicii cu metodele care manipuleaza baza de date. Toate metodele care se afla aici apeleaza metodele care acceseaza baza de date prin prisma atributului. Clasa mai are un atribut de tipul lista de validatori care ne ajuta sa validam un produs care se vrea introdus la un moment dat. |
| RequestBll | Clasa care se ocupa cu logica aplicației legata de tabelul comanda. Clasa are un singur atribut principal și anume un obiect de tipul RequestDAO care asigura conectarea logicii cu metodele care manipuleaza baza de date. Toate metodele care se afla aici apeleaza metodele care acceseaza baza de date prin prisma atributului. |
| Validator (**Interface**) | Interfata care contine metoda de validare care trebuie suprascrisa de toate clasele care implementeaza interfata. |
| ProductPriceValidator | Clasa care implementeaza interfața descrisa mai sus și care suprascrie metoda de validare astfel incat sa valideze produsul doar daca pretul se afla in intervalul declarat prin atributele finale declarate. |
| ProductQuantityValidator | Clasa care implementeaza interfața descrisa mai sus și care suprascrie metoda de validare astfel încat sa valideze produsul doar daca cantitatea se afla între capetele intervalului declarat prin atributele finale declarate. |
| CustomerView | Clasa are ca atribute toate componentele de la nivel de interfata care necesita manipularea ulterioara pentru fereastra tabelului client. Conține constructorul interfeței, metode care returneaza textul din TextField-uri, metode care adauga ascultatori pentru fiecare buton aferent si metoda care seteaza textul unei etichete cu clientul cautat sau cu mesajul excepției în caz ca nu se gasește niciun client. |
| ProductView | Clasa are ca atribute toate componentele de la nivel de interfata care necesita manipularea ulterioara pentru fereastra aferenta tabelului produs. Conține constructorul interfeței, metode care returneaza textul din TextField-uri, metode care adauga ascultatori pentru fiecare buton aferent și metoda care seteaza textul unei etichete cu produsul cautat sau cu mesajul exceptiei în caz ca nu se gaseste niciun produs. |
| RequestView | Clasa are ca atribute toate componentele de la nivel de interfata care necesita manipularea ulterioara pentru fereastra aferenta tabelului comanda. Conține constructorul interfetei, metoda care returneaza textul din TextField și metode care adauga ascultatori pentru fiecare buton aferent. |
| Controller | Clasa care leaga clasele de logica a aplicatiei cu clasele ce tin de interfata grafica. Are sase atribute principale, trei atribute care reprezinta obiecte de tipul fiecarei ferestre grafice implementate și trei atribute care reprezinta obiectele care se ocupa cu logica bazei de date. In cadrul constructorului initializez atributele de tip fereastra grafica cu cele primite ca si parametrii, initializez atributele care se ocupa cu logica bazei de date si asociez metoda ascultatoare pentru fiecare buton din fiecare fereastra grafica în parte. |
| ReflectionTechnique | Clasa care foloseste tehnica reflexiei pentru a extrage proprietatile si valorile proprietatilor unui obiect. Are doua metode: una care extrage proprietatile obiectului primit ca parametru și le returneaza sub forma de vector și a doua care extrage valorile proprietatilor obiectului primit ca parametru si le returneaza sub forma de vector. |
| Start | Clasa care conține metoda main. Aici instantiez fiecare fereastra grafica pe care o primește controller-ul ca parametru în constructorul aferent și setez fiecare fereastra grafica ca fiind visibila. |

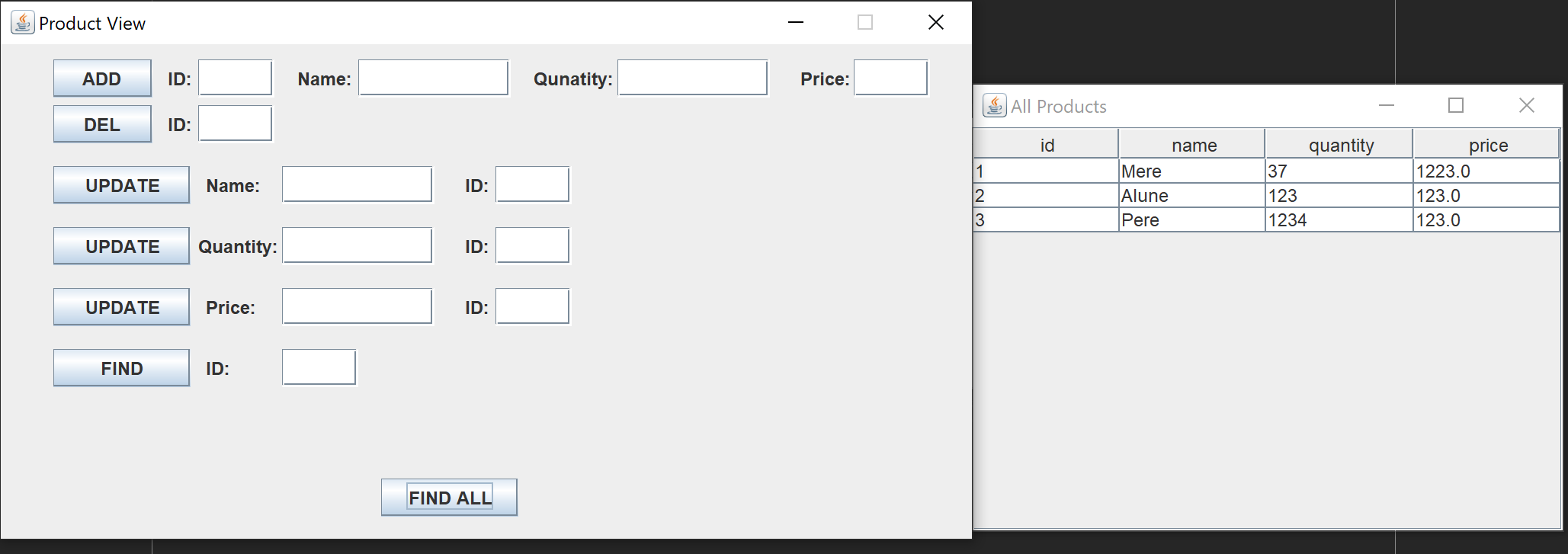
Diagrama UML



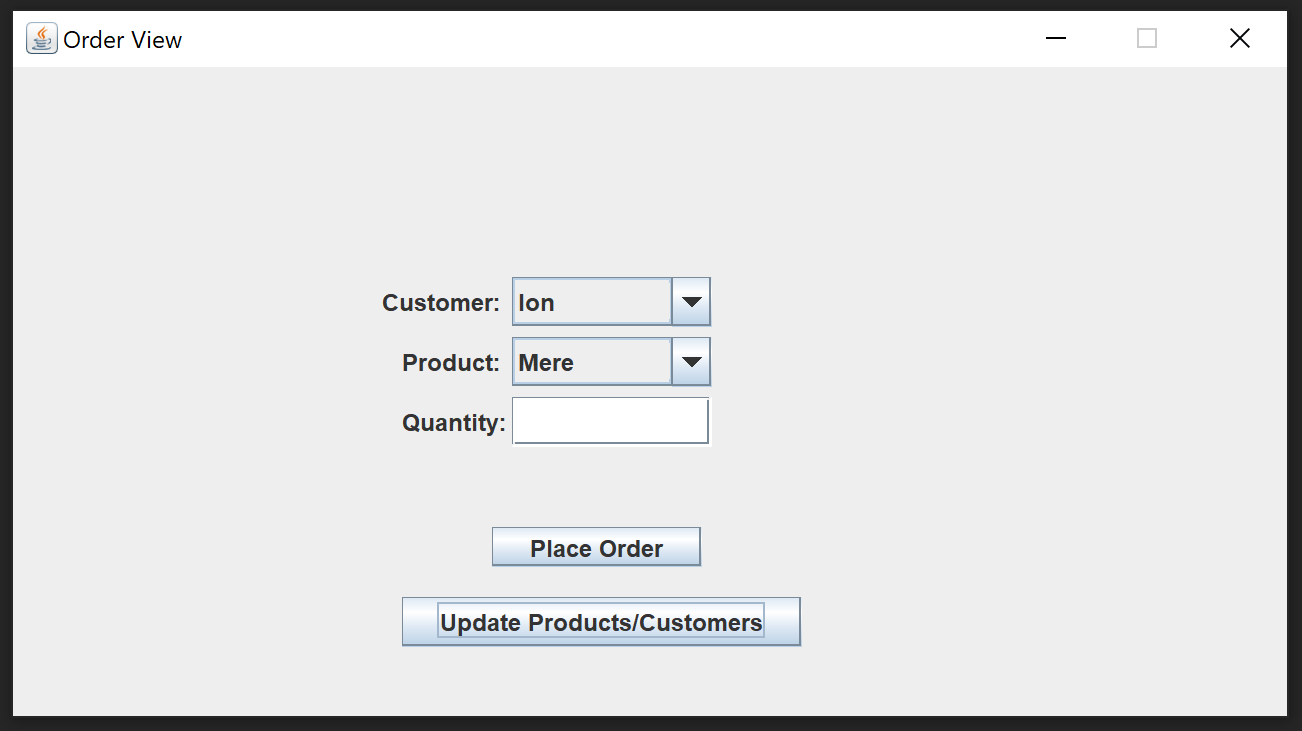
Interfata pentru Client:



Interfata pentru Produs:



Interfata pentru Comanda:



Implementare

Avand in vedere faptul ca sunt destul de multe clase o sa le descriu pe cele mai importante dupa parerea mea.

Clasa ConnectionFactory

Clasa ConnectionFactory este clasa care se ocupa cu conexiunea dintre aplicatie si server-ul MySQL care contine baza de date cu tabelele corespunzatoare. Atributele sunt necesare conectarii la baza de date: numele driver-ului, locatia bazei de date, numele de utilizatorm si parola pentru accesarea server-ului.

In cadrul metodei care creaza conexiunea , se declara o conexiune locala asupara careia se incearca egalarea cu conexiunea rezultata din apelare metodei care returneaza o conexiune pe bazata pe locatia bazaei de date, a numelui de utilizator si a parolei.

O imagine care conține text

Descriere generată automat

Clasa AbstractDAO

Clasa care contine metodele asociate celor 3 tabele din baza de date. Am adaugat mai jos metoda care creeaza un query de adaugare a unui element intr-o tabela. Am luat pe rand fiecare camp din obiectul cu care se apeleaza metoda si am salvat numele intr-un String, dupa sintaza MySQL si apoi am finista sintaza cu cate valori trebuie inserate in tabela

O imagine care conține text

Descriere generată automat

Metodele urmatoare sunte metodele de stergere din tabel si de actualizare a datelor, pentru fiecare metoda am folosit un StringBuilder pentru a crea sintaxa MySQL crespunzatoare operatiei implementate.

O imagine care conține text

Descriere generată automat

Metoda de inserare intr-un tabel este implementata pe baza unei conexiun cu baza de date si unui query. Pentru fiecare ? din query am luat pe rand fiecare camp din obiectul cu care s-a apelat metoda si am extras metodele de set si get asociate campului respective. Dupa editarea query-ului acestea se executa si apoi inchidem toate variabilele folosite.

O imagine care conține text

Descriere generată automat

Metoda de actualizare a numelui din aceasta clasa este folosita doar pentru 2 dintre tabele si anume Product si Customer, pentru tabela Request am facut o alta metoda in RequestDAO.

Aici am deschis o conexiune cu baza de date si am editat query-ul astfel incat in locul valorii care trebuie modificate am adaugat valoarea primita ca parametru si pe locul valorii cu care se cata in tabela respective am adaugat valoarea id-ului primit din nou ca parametru dupa am executa query-ul si am inchis variabilele.

O imagine care conține text

Descriere generată automat

Clasa ProductBll

Clasa care se ocupa cu logica aplicatiei legate de tabela product. In toate metodele din acesata clasa este apelata metoda corespunzatoare care acceseaza tabela din baza de date cu parametrii necesari si am retrunat valorile corespunzatoare.

O imagine care conține text

Descriere generată automat

Clasa Controller

Clasa Controller se ocupa cu legaturile dintre interfetele grafice si aplicatia implementata. Acesta clasa contine un obiectele de tip View si cele de tip Bll care impementeaza metodele necesare aplicatie.

O imagine care conține text

Descriere generată automat

Clasa secudara care implementeaza asculatatorul din interfata grafica si adauga clientul in baza de date.

O imagine care conține text

Descriere generată automat

Clasa secundara care implemeteaza ascultatorul butonului FIND ALL, acesta metoda creaza un tabel in care se afiseaza toti clientii care sunt in tabelul din baza de date

Aici m-am folosit de metoda reflexiei pentru a extrage campurile si valoarea unui obiect si pentru a crea tabelul de afisare. Avand in vedere ca un JTabel are nevoie de un vector de nume de coloane si o matrice de elemente care vor reprezenta elementele de pe fiecare linie din tabel, mi-am declarat o listă de clienți pe care o creez cu ajutorul metodelor care extrag campurile si valorile câmpurilor. Vectorul care cotine numele de coloane îl inițializez cu vectorul cu nume de câmpuri returnat de metoda respectivă. Matricea am declarat-o de dimensiunea listei de elemente in care se afla toti clientii din tabel si de dimensiunea vectorului de coloane. Apoi, fiecare linie din matrice a fost asociată cu vectorul de valori corespunzător campurilor returnat. Apoi am creat o noua fereastra care contine tabelul cu toti clientii din baza de date.

O imagine care conține text

Descriere generată automat

Rezultate

Toate rezultatele acestei aplicatii sunt date de ferestele interfetei grafice.

Concluzii

In urma implementarii acestui proiect pot declara ca mi-am adus aminte multe lucruri legate de Java si MySQL lucruri foarte importante in dezolvoltarea aplicatiilor care lucreaza cu aceste doua limbaje de programare foarte utilizare. Dezvoltarile ulterioare ar fi adaugare mai multor indicatii pentru utilizatorul aplicatiei si poate as gandi altfel unele metode pentru a avea o performanta mai mare.

Bibliografie

Nu am avut nici o sursa din care sa ma inspir la redactarea acestei documentatii. Acestea sunt ideile mele care au venit pe parcursul proiectarii si implemetarii proiectului si anumite ideei care mi-au venit pe moment.