**DOCUMENTATIE TEMA 3**

**Nume prenume Tigarean Lucian**

**Grupa 302210**

**Profesor Laborator Assist Antal Marcel**

Contents

[1. Cerinte Functionale 3](#_Toc476131445)

[2. Obiective 3](#_Toc476131446)

[2.1. Obiectiv Principal: 3](#_Toc476131447)

[2.2. Obective Secundare: 3](#_Toc476131448)

[3. Analiza Problemei 3](#_Toc476131449)

[4. Proiectare 3](#_Toc476131450)

[4.1. Structuri de date 3](#_Toc476131451)

[4.2. Diagrama de clase 3](#_Toc476131452)

[4.3. Algoritmi 3](#_Toc476131453)

[5. Implementare 4](#_Toc476131454)

[6. Testare 4](#_Toc476131455)

[7. Concluzii si Dezvoltari Ulterioare 4](#_Toc476131456)

[8. Bibliografie 4](#_Toc476131457)

# Cerinte Functionale

Implementarea unei aplicatii care sa permita lucrul cu bazele de date aflate pe un server. Aceasta aplicatie trebuie sa perimita si sa ofere o interfata prin care un utilizatorul obisnuit sa poata introduca, modifica si salva date fara a fi nevoit sa stie un limbaj de programare specific pentru manipularea datelor în sistemele de manipulare a bazelor de date relaționale.

# Obiective

## Obiectiv Principal:

Implementarea unei aplicatii care sa ofere o interfata usor de folosit, pentru operatii efectuate asupra unei baze de date( de exemplu introducere, modificare, stergere, afisare date).

Interfata trebuie sa fie usor de folosit chiar si de persoanele care nu au cunostinte legate de un limbaj de programare specific pentru manipularea datelor în sistemele de manipulare a bazelor de date relaționale.

## Obective Secundare:

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Dezvoltarea de use case-uri si scenarii | Sunt descrise use case-uri pentru efectuarea operatiilor de inserare, stergere | 3 |
| Alegerea structurilor de date | Alegerea unei structuri de date convenabile atat din punct de vedere al eficientei cat si a memoriei ocupate | 4 |
| Impartirea pe clase | Impartirea pe pachete si clase | 4 |
| Dezvoltarea algoritmilor | Dezvoltarea algortimilor pentru validatori | 4 |
| Implementarea solutiei | Implementarea solutiei utilizand bazele de date si proprietatea de reflectie | 5 |
| Testare | Testarea realizata pe parcursul proiectarii | 6 |

# Analiza Problemei

Diagrama use case pentru inserarea unui client:

1. Se alege tabela Client din lista cu tabele(aflata in partea stanga)
2. Se apasa pe butonul Add new object.
3. In fereastra noua deschisa se introduc datele
4. Dupa introducerea datelor se apasa pe butonul insert.
5. Se inchid ferestrele de dialog

Diagrama de use case pentru stergerea unui produs

1. Se alege tabela Product din lista cu tabele(aflata in partea stanga)
2. Se apasa pe butonul Delete object.
3. In fereastra noua deschisa se alege conditia de sterge(prin alegerea unei coloane si specificarea valorii)
4. Se apasa pe butonul Delete Product.
5. Se inchid ferestrele de dialog.

# Proiectare

## Structuri de date

Structura de date folosita este List din pachetul java.util. List implementeaza interfata Collection. Structura de date List contine liste(secvente) de elemente indexate. Listele pot contine duplicate si permit un control precis asupra pozitiei unui element prin intermediul indexului acelui element. Clasa List are metode implementate care ne permite sa lucram mai usor, de exemplu pentru a adauga un element in lista folosim metoda add(Object a), pentru accesare indexata folosim metoda get(int index), pentru a afla numarul de elemente din lista folosim metoda size(), pentru stergere putem folosi atat metoda remove(Object a) cat si remove(int index) care sterge obiectul dat, respectiv sterge obiectul de pe pozitia data.

## Diagrama de clase

View

presentation

-pane3: JPanel

...

+retrieveNameColumn(Object): String[]

+retrieveProperties(List<T>): void

AbstractDAO<T>

dao

-LOGGER:Logger

-type: Class<T>

+AbstractDAO()

...

ConnectionFactory

connection

-LOGGER: Logger

-Driver: String

...

-ConnectionFactory()

+getConnection():Connection

...

ClientDAO<T>

dao

+ClientDAO()

+insert(Object):void

+findAll():List<Client >

...

ProductDAO<T>

dao

+ProductDAO()

+insert(Object):void

+findAll():List<Product >

...

OrdersDAO<T>

dao

+OrdersDAO()

+insert(Object):void

+findAll():List<Orders>

...

Client

model

-id: int

-name: String

-address: String

-email: String

-age: String

+Client()

+Client(int,String,String,String,int)

...

Product

model

-id: int

-name: String

-stock: int

+Product()

+Product(int, String, int)

...

Orders

model

-idOrder: int

-idClient: int

-nameClient: String

-idProduct: int

-nameProduct: String

-quantity: int

+Orders(int, int,String,String, int)

...

OrderStockValidator

Bll.validators

ClientAgeValidator

Bll.validators

-MIN\_AGE:int

-MAX\_AGE: int

+ClientAgeValidator()

+validate(Client):void

ClientBll

bll

+ClientBll()

...

OrderBll

bll

+OrdersBll()

...

ProductBll

bll

+ProductBll()

...

+OrderStockValidator()

+validate(Orders):void

0..\*

0..\*

0..\*

-validators

-validators

-validators

<Java Interface>

Validator<T>

bll.validators

+validate(T): void

## Algoritmi

In acest proiect nu s-au folosit algoritmi complicati, algoritmi simpli de interactionare cu baza de date si calcularea unor conditii de validare a datelor de intrare.

# Implementare

Pentru realizarea acestui proict s-au realizat sase pachete fiecare avand cateva clase. S-a ales distribuirea claselor in pachete pentru o mai buna organizare si portabilitate a proiectului.

Pachetul **model** contine trei clase, fiecare clasa corspunde unei tabele din baza noastra de date.

* Clasa **Client** contine atributele id de tipul int, name de tipul String, address de tipul String, email de tipul String si age de tipul int. Constructorul clasei are cinci parametri prin care se initializeaza atributele, pe langa acest constructor este definit si un consctructor null(in care nu se initializeaza nici un atribut) pentru a putea crea o instanta a clasei. Metodele din aceasta clasa sunt de tipul getter-lor si setter-lor prin care se obtine si se modifica atributele(atributele clasei au modificatorul de vizibiliate private, adica doar in interiorul clasei se pote vedea si seta valoarea atributelor).
* Clasa **Product** contine atributele id de tipul int, name de tipul String, stock de tipul int. Constructorul clasei are trei parametri prin care se initializeaza atributele, pe langa acest constructor este definit si un consctructor null(in care nu se initializeaza nici un atribut) pentru a putea crea o instanta a clasei. Metodele din aceasta clasa sunt de tipul getter-lor si setter-lor prin care se obtine si se modifica atributele(atributele clasei au modificatorul de vizibiliate private, adica doar in interiorul clasei se pote vedea si seta valoarea atributelor).
* Clasa **Orders** contine atributele idOrder de tipul int, idClient de tipul int, nameClient de tipul String, idProduct de tipul int, nameProduct de tipul String si quantity de tipul int. Constructorul clasei are sase parametri prin care se initializeaza atributele. Metodele din aceasta clasa sunt de tipul getter-lor si setter-lor prin care se obtine si se modifica atributele(atributele clasei au modificatorul de vizibiliate private, adica doar in interiorul clasei se pote vedea si seta valoarea atributelor).

Pachetul **connection** contine o clasa in care se descriu metodele folosite pentru conectarea la o baza de date.

* Clasa ConnectionFactory contine atributele necesare pentru conectarea la o baza de date( acestea sunt initializate), constructorul clase are modificartorul de viziblitate de tipul private. Clasa contine patru metode statice prin care se face conectarea si inchiderea la baza de date. Pentru conectarea la baza de date se apeleaza metoda statica createConnection care returneaza conexiunea de tip connection. Alte trei metode sunt folosite pentru inchiderea unei connection(conexiunea), statement(interogare) si resultSet(rezultatele interogarii.

Pachetul **dao** contine 4 clasa care definesc interogariile, pentru a putea interactiona cu baza de date.

* Clasa **AbstractDAO** este clasa prin care se genereaza interogariile prin tehnica numita reflectie. 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 pentru construirea interogarilor de tipul select, insert, update, delete. Metoda findById(String setCol, String a) primeste 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 createSelectQuery(String setCol) unde se genereaza interogarea(pe baza colanei primita ca argument), se executa interogarea si se apeleaza metoda createObjects(resultSet) care returneaza o lista de obiect(rezultatele interogarii sunt transforamte in obiecte si stocate intr-o lista).

Metoda **createObjects**(ResultSet) returneaza o lista de obiect(ca rezultat al interogarii). Algoritmul functioneaza in felul urmator: cat timp mai sunt rezultate ale interogarii, se creeaza o instanta de tipu T(clasa specificata in metodele care o extind aceasta clasa, de exemplu Client, Product, Order), se itereaza fiecare camp al clasei T, pentru fiecare camp al clasei se obtine rezultatul interogarii si se actualizeaza atributele obiectului creat. La final se obtine si se returneaza o lista de obiect ca rezultat al interogarii.

Toate metodele din aceasta clasa functioneaza dupa principiile prezentate mai sus.

* Clasele ClientDAO, OrdersDAO, ProductDAO extinde clasa AbstractDAO mostenind astfel metodele implmentate in AbstracDAO care se concretizeaza prin specificarea clasei asupra careia se generaza interogariile. Astfel aceste clase mostenesc metode care implementeaza interogari de tipul insert, update, select si delete. In aceste metode s-ar mai putea implemeta metode care genereaza interogarii specifice doar unei anumite tabele.

Pachetul **bll** contine 4 clase, 3 clase care valideaza si conditioneaza datele de intrare inainte de a se crea interogarea si o clasa care defineste o exceptie(ExceptieStock folosita pentru a semnala lipsa produselor in stock).

* Clasa ClientBll are atributul validators care este de tipul List<Validators<Client>>, o lista de validatori de tipul client. Constructorul clasei creeaza o instanta de tipul ArrayList<Validators<Client>> iar apoi sunt adaugati in aceasta lista instante de noi clase validatoare care verfica datele de intrare. Metodele din aceasta clasa sunt insertClient, findAll, updateClient, deleteClient, findClien. In aceste metode, unde este cazul, se apeleaza metoda validate din clasa validatoare pentru a verifica datele, In caz in care datele nu respecta conditiile de validare atunci metoda arunca o exceptie, metoda din ClientBll arunca la randul ei exceptia primita, urmanda ca prinderea si tratarea exceptiei sa se faca in clasa care implementeaza interfata grafica afisand un mesaj utilizatorului. In aceasta clasa se utilizeaza validatori doar pentru inserare, unde se verifica daca varsta este intre 10 si 60.
* Clasa ProductBll, la fel ca si clasa ClientBll, are atributul validators, de tipul List<Validators<Product>>, o lista de validatori de tipul Product. In constructorul clasei ar trebuie sa se adauge validatori, dar din cauza faptului ca nu am creat unul a ramas doar scheletul pentru dezvoltari ulterioare. Metodele din aceasta clasa sunt findProduct, insertProduct, findAll, updateProduct si deleteProduct, aceste metode creeaza o noua instanta de tipul ProductDAO si apeleaza metoda corspunzatoare pentru a se crea interogarea.
* Clasa OrdersBll are atributul validators(prezentat si in celelate clase din pachet). Constructorul clasei creeaza o noua instanta de tipul List<Validators<Orders>> si adauga un validator acestei liste. Metoda insertOrder poate arunca o exceptie de tipul ExceptieStock aruncata de metoda validatoare apelata. Se apeleaza metoda validate din OrderStockValidator unde se verifica daca stockul produsuli corespunde cu cantitatea ceruta in momentul creari, in caz in care stockul este mai mic decat cantitatea ceruta se arunca o exceptie(ExceptieStock), urmata a fi tratata in clasa care implementeaza interfata grafica. In metoda insertOrder daca validarea a fost cu succes, se apeleaza metoda insertOrders pe o noua instanta OrdersDAO creata, se actualizeaza stockul Produsului si se creeaza un fisier text extern cu detaliile clientului, detaliile produsuli si detalii despre comanda efectuata.

Pachetul **bll.validators** contine o interfata si doua clase care implemeteaza aceasta interfata.

* Clasa ClientAgeValidator are doua atribute care MIN\_AGE si MAX\_AGE initializate, metoda validate() verifica daca obiectul de tipul Client are atributul age cuprins intre cele doua variabile statice si finale intializate la inceput. Daca atributul age nu este in acest interval atunci se arunca o exceptie verificata de tipul IllegalArgumentException.
* Clasa OrderStockValidator implementeaza interfata Validator si suprascrie metoda validate. Metoda validate primeste ca argument un obiect de tipul Orders, cauta produsul primit in alta tabela si verifica daca produsul are stockul mai mare decat cantitatea ceruta in comanda. In cazul in care nu se indeplineste aceste conditii se arunca o exceptie de tipul ExceptieStock.
* Inetrfata Validator defineste metoda validate care poate sa arunce o exceptie.

Pachetul **presentation** contien clasa view unde este implementata interfata grafica a proiectului.

* Clasa **View** defineste atat partea de controller cat si partea de view. Aceasta clasa extinde clasa Jframe si implementeaza interfata ActionListener. Constructorul clasei adauga atributele definite in diferite panouri pentru a forma fereastra principala. Panourile au un layout de tipul GridBagLayout(), un layout usor de folosit, organziat in forma de matrice unde putem specifica locul(linia si coloana din matrice) elementului care urmeaza sa il inseram in panou. Pe fiecare buton am adaugat ascultatori, prin suprascrierea metodei actionPerformed(), unde pentru fiecare buton tratam si implementam noi ferestre dupa caz. Noile ferestre(ferestre de dialog) sunt construire cu JDialog unde se adauga panouri, ascultaori si functionalitati dupa caz.

Pentru a intelege implementarea asculatatorilor explicam modul de functionare pe un caz concret. In metoda actionPerformed (metoda care trebuie implementata din interfata ActionListener) daca un eveniment s-a intamplat asupra butoanelor si sursa este butonulC4(butonul pentru afisare tuturor inregistrariilor dintr-o tabela), se creeaza o lista unde se retine rezultatul apelarii metodei findAll asupra unei instante noi de Client sau Product. Pentru a construi o noua fereastra dialog cu rezultatele obtinute si stocate in lista se apeleaza metoda retrieveProperties prezentata mai jos.

Metoda retrieveProperties(List<T>) este o metoda void care construieste un tabel, prin metoda reflectiei, din lista de obiecte primita ca si parametru. Construirea tabelului s-a realizat cu JTable. Prin folosirea metodelor addColumn si addRow care primesc vectori de string-uri s-a format tabelul. Metoda flectiei consta in iterarea campurilor(adica atributelor clasei) si aflarea numelui prin metoda getName(), astfel s-a putu realiza construirea dinamica a tabelelor printr-o singura metoda.

O alta metoda din acesta clasa este retrieveNameColumn(Object) care returneaza un vector de string-uri cu numele atributelor din obiectul primit ca si parametru.

# Testare

Testarea s-a realizat pe parcursul dezvoltarii aplicatiei, urmarind comportamentul si rezultatele( comparate cu cele asteptate de noi) astfel s-au corectat unele erori si s-a optimizat programul.

# Concluzii si Dezvoltari Ulterioare

Realizand acest proiect am invata 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( aplicatia java utilizeaza aceasta librarie pentru a comunica cu MYSQL server). Am invatat sa creez un tabel utilizand JTable si sa creez ferestre de dialog utlizand JDialog.

Realizand acest proiect am mai invata despre conceptul de reflectie in java, si am utlizat acest concept pentru a genera automat interogariile in functie de clasa din care apelam metoda. Utlizand acesta proprietate de reflectie am generat tabelele in functie de atibutele unei clase.

Dezvoltarie ulterioare. Proiectul mai are nevoie de cateva imbunatatiri precum ar fi:

* O interfata mult mai interactiva cu elemente noi, cu functionalitati noi, mult mai aranjata si ordonata care poate oferi utlizatorului o experienta cat mai usoara de utilizare
* Adaugarea de validatori noi, care sa verifice si sa restrictioneza mai mult datele de intrare.
* Definirea de noi exceptii si tratarea lor in interfata grafica prin afisarea unui mesaj pentru a informa utlizatorul ca introduce date invalida.
* Operatiile realizate cu succes, precum ar fi inserarea, sa fie confiramte printr-un mesaj afisat in interfata grafica.

# Bibliografie

http://baseprogramming.com/blog1/2017/08/24/automating-jdbc-crud-operations-with-reflection/

https://stackoverflow.com/questions/3549206/how-to-add-row-in-jtable