**ORDER MANAGEMENT**

***Materie: Tehinici de Programare***

***Elev: Apostol Ovidiu-Catalin***

***Grupa: 30223,semigrupa 1***

***Specializare: CTI RO, an 2***

*Cuprins*

[*Obiectivul temei* 2](#_Toc35361509)

[*Analiza problemei* 2](#_Toc35361510)

[*Proiectare* 4](#_Toc35361511)

[*Implementare* 5](#_Toc35361512)

[*Rezultate* 8](#_Toc35361513)

[Concluzii 11](#_Toc35361514)

[*Bibliografie* 11](#_Toc35361515)

# *Obiectivul temei*

Principalul obiectiv al acestei teme este realizarea unei simulari a unui Warehouse prin combinarea unui program java cu o baza de date.Pentru realizarea acestui lucru trebuie sa dobandim cunostiinte in domeniul bazelor de date , a conexiunii dintre o baza de date si programul nostru Java , scrierea de pdf-uri , formatarea de text intr-un pdf utilizand o librarie externa (“ITEXTPDF”).

Pentru a indeplini aceste obiective trebuie mai intai sa ne creem urmatoare clase de baza dupa modelul pus la dispozitie de profesorul de la laborator:

* Trebuie respectata arhitectura layered in care sa avem atat componente de DAO- care fac conexiunea dintre Baza De Date si Program , Presenter – care se ocupa de afisare si control ,Model – clasele care reprezinta tabelele dn baza de date si Bussiness logic – care se ocupa de parsarea argumentelor si apelarea metodelor respective fiecarei clase .
* Folosirea “Reflection” , pentru a creea clase Generice (<T>) cu metode care vor putea fi apelate de orice clasa noua Model care se afla ca si Tabela in baza de date si are aceleasi argumente .

# *Analiza problemei*

Pentru inceput trebuie sa realizam conexiunea dintr baza de date si programul nostru java . Acesta se va realiza prin introducerea unei librarii externe, JDBC , apoi vom creea clasa pentru conexiune conform exemplului pus la dispozitie de profesor , modificand in cazul nostru daca este nevoie : numele bazei de date , userul , parola ( aici eu am catalin99- parola default este root si pentru a fi rulat pe un alt dispozitiv trebuie **modificate la userul si parola de pe dispozitivul care o ruleaza.**

O a doua problema care se pune este abstractizarea tabelelor in limbaj java . Mai apoi realizarea conexiunilor prin clasa DAO intre java si baza de date . Pentru a generaliza problema vom crea o clasa generica AbstractDAO care va putea fi extinsa si va avea metode generale ce vor putea fi apelate de orice clasa Model

***Modelare***

Un prim pas in modelarea acestei probleme il reprezinta parcurgerea template-urilor puse la dispozitie de profesor pentru o mai usoara intelegere a abstractizarii acestei probleme (transpunerea ei in cod ) . Dupa care v-a trebui sa completam campurile pe care le avem de completat , si sa adaugam diferite metode , argumente proprii in functie de metoda de implementare aleasa de fiecare .

Pentru a implementa tabelele in Cod vom creea clase de model care vor avea exact acelasi nume ca si Tabela din baza de date si exact aceleasi argumente.

Apoi trebuie sa gasim o metoda sa creem o clasa Generica pentru Database connection.Astfel trebuie sa creem niste metode prin Reflection care sa mearga pentru orice clasa noua Model corespunzatoare unei tabele din baza noastra de date. Asta o vom face prin copierea clasei AbstractDAO si adaugarea metodelor de care avem nevoie pentru programul nostru si creeare claselor specifice fiecarei tabela(clasa Model) prin extindere clasaei AbstractDAO<T> unde in locului lui T se va afla clasa noastra.

***Utilizare (use-case)***

Pentru a folosi aplicatia , o vom apela din linia de comanda dupa generarea fisierului . Jar. Vom avea de dat 1 argument care va fi numele fisierului .

Astfel dupa apel se vor citi comenzile din fisier linie cu linie dupa care se vor executa si se vor genera pdf-urile pentru fiecare caz ( pentru reporturi sau erori).

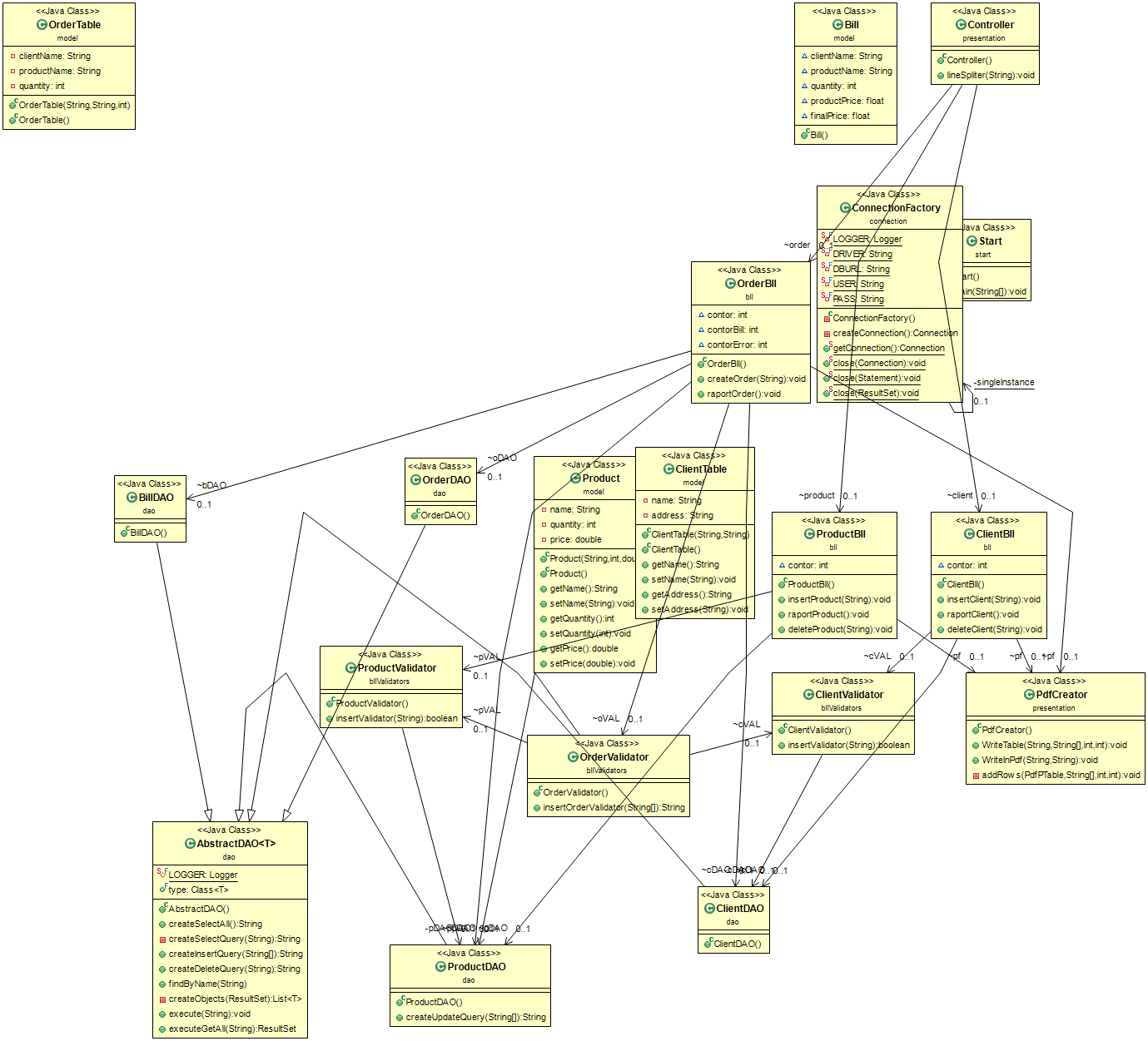
**PS:Jar-ul nu va functia daca utilizatorul nu are acelasi user si parola ca mine , pentru a functia va trebuii sa se modifice in clasa de conexiune user-ul si parola de la mysql database si sa se genereze jar-ul din nou ( sau sa se ruleze din GUI-ul respectiv(eclipse la mine).**

# *Proiectare*

***Decizii de proiectare***

Pentru proiectarea acestei teme vom urma urmatorii pasi :

* Mai intai vom transcrie template-urile de cod puse la dispozitie in slideurile despre reflection si layered architecture puse la dispozitie de catre profesorul de la curs in Assignment3 .
* Apoi vom modifica acel template dupa bunul nsotru plac pentru a rezolva problemele noastre si vom adauga metode pentru a putea executa operatiile necesare gestionarii unui Warehouse respect(inser/ delete/ report/ update ).
* Dupa vom adauga clasele de care avem nevoie dupa bunul plac si viziunea proprie a fiecaruia asupra problemei. Eu am construit clasele de Buisnesslogic care se ocupa de parsarea argumentelor. Cele de validator care se ocupa de validarea query-ului. Verifica daca se poate executa , iar daca nu afiseaza un mesaj de eroare intr-un pdf.
* ***Diagrama UML***



# *Implementare*

Cea mai mare parte din implementare se afla in JAVADOC.

***Pachetul bll-Bussiness Logic***

In pachetul de bussiness logic avem clasele care primesc argumentele parsate, creeaza obiectele si apeleaza metodele pentru generarea si executare query-urilor.

***Clasa ClientBll***

Clasa de Bussiness logic pentru Client care face primeste argumentele si le parseaza mai departe catre componenta de DAO .

Ca si metode , pe langa constructorul care intializeaza variabilele si gettere /settere avem :

***Public void insertClient(String line)***

Metoda de inserarea client , primeste argumentele clientului si apeleaza metoda de creeare si executare a query-ului de inserare client.

\* @param line- linia care trebuie impartita in argumente

***public void raportClient()***

* Metoda de raport client care apeleaza query-ul de formare al report-ului si de generare a pdf-ului .
* @param line

***public void deleteClient(String line)***

* Metoda de delete client , primeste argumentele clientului si apeleaza metoda de creeare si executare a query-ului de delete client.
* @param line

***Clasa ProductBll***

Clasa de Bussiness logic pentru Produs care face primeste argumentele si le parseaza mai departe catre componenta de DAO .

Ca si metode pe langa constructor avem :

***public void insertProduct(String line)***

* \* Metoda de inserarea produs , primeste argumentele clientului si apeleaz metoda de creeare si executare a query-ului de inserare produs sau updatare a stockului daca este cazul.
* \* @param line

***public void raportClient()***

* Metoda de raport product care apeleaza query-ul de formare al report-ului si de generare a pdf-ului .
* @param line

***public void deleteClient(String line)***

* Metoda de delete client , primeste argumentele produsul si apeleaza metoda de creeare si executare a query-ului de delete produs.
* @param line

***Clasa OrderBll***

Clasa de Bussiness logic pentru Client care face primeste argumentele si le parseaza mai departe catre componenta de DAO .

Ca si metode avem :

***public void createOrder(String line)***

* Metoda de inserarea order , primeste argumentele order-ului si apeleaza metoda de creeare si executare a query-ului de inserare order, care in caz de succes va updata stock-ul produsului din order si va genera Bill-ul ( din clasa Bill) , altfel va generea in pdf cu un mesaj de eroare specific errorii care a aparut
* \* @param line

***public void raportOrder()***

* \* Metoda de raport order care apeleaza query-ul de formare al report-ului si de generare a pdf-ului
* \* @param line

***Pachetul bllValidator***

Pachetul bllValidator contine 3 clase de Validators : pentru Client ,Product si respectiv Order . Aceste clase contin fiecare o metoda de insertValidate pentru ca este singura comanda pe care o executam si pot aparea errori la incercarea de inserare .Aceste metode primesc argumentele comenzii si vor returna true in caz ca se poate executa comanda , altfel , un mesaj de erroare corespunzator care va fi mai apoi pusa intr-un pdf de eroare.

***Pachetul connection***

Clasa de connection este clasa care creeaza conexiunea dintre programul nostru java si baza de date . Ca si metode are metoda de creeare a conexiunii intre program si baza de date , getConnection care returneaza conexiunea create anterior , de executare a unui statement sau de inchidere .

***Pachetul dao***

Acesta este este pachetul care contine clasele care construiesc query-urile si le executa , astfel ele translateaza codul java in comenzi pentru baza de date si le executa si returneaza rezultatul.

***Clasa AbstractDao***

* Clasa generica AbstractDAO care are metodele generice si va fi "specializata prin intermediul parametrului <T>
* @param <T> - parametrul <T> este clasa care va determina type-ul metodelor si va specializa metodele generice pentru acea clasa.

Ca si metode principale avem :

***public AbstractDAO()***

* Constructorul clasei AbstractDAO care seteaza type-ul la type-ul clasei \* "apelante".

***public String createSelectAll()***

* Creeaza un String care va fi un query( pentru un Select \* ) si il returneaza
* @return String query.

***private String createSelectQuery(String field)***

* \* Functia creeaza un Query de tipul (Select \* where field=?;)
* \* @param field - field-ul care trebuie cauta ( de exemplu "ClientName" , "address").
* \* @return un QUERY

***public String createInsertQuery(String[] args)***

* Creeaza un query de insert in tabela specificata de <T>
* @param args - atributele care trebuie adaugate in tabla
* @return UN QUERY

***public String createDeleteQuery(String argument)***

* Creeaza un query de delete sub forma Delete from table where argument=?;
* @param argument - argumentul din where clause
* @return Un QUERY

***public T findByName(String name)***

* Functia apeleaza un query de tipul Select \* from table where name=name;
* @param name - numele dupa care se cauta
* @return Functia returneaza primul element gasit sub forma unei clase de tipul <T>

***private List<T> createObjects(ResultSet resultSet)***

* MEtoda apelata de findByName care creeaza obiecte de tipul <T> din ResultSetul selectului
* @param resultSet - ResultSet-ul obtinut in urma executiei functiei de query.
* @return o Lista de elemente de tipul <T> .

***public void execute(String query) throws SQLException***

* \* Functia execute un updateQuery pentru insert/update etc.
* @param query
* @throws SQLException
* @return o Lista de elemente de tipul <T> .

***public ResultSet executeGetAll(String query) throws SQLException***

* Functia executa un query de tipul select si returneaza ResultSet-ul obtinut
* @param query
* @return ResultSet-ul obtinut in urma cautarii.
* @throws SQLException

***Clasele ClientDao/OrderDao/BillDao/ProductDAO***

Aceste clase sunt specializari ale clasei AbstractDAO . Ele extind AbstractDAO cu tipul specific al fiecareia dintre ele .

***Pachetul MODEL***

Acest pachet contine clasele Model pentru cele 4 tabele pe care le avem in baza de date impreuna cu atributele specifice :

Clasele sunt :

***Clasa ClientTable***

* Ca atribute avem : - **String** name

- **String** address

***Clasa Product***

* Ca atribute avem : - **String** name

- **int** quantity

- **float** price

***Clasa OrderTable***

* Ca atribute avem : - **String** clientName

- **String** productName

***-*int** quantity

***Clasa Bill***

* Ca atribute avem : - **String** clientName

- **String** productName

***-*int** quantity

- **float** productPrice

-**float** finalPrice

***Pachetul presentation***

Acest pachet contine doua clase :

***Clasa controller***

**Aceasta clasa contine o singura metoda de splitLine care:**

Aceasta metoda primeste linia de la functia main , afla ce comanda trebuie executata si apeleaza metoda specifica , parsand atributele mai departe.

***Clasa PDFCreator***

Aceasta clasa se ocupa cu creeare pdf-urilor si scrierea in ele sub formatul dorit .

Contine 2 metode principale :

***public void WriteTable(String name, String[] str, int n, int nr)***

* Functia va creea un tabel in care se va afla un tabel dupa urmatoarii parametrii:
* @param name - numele pdf-ului ce urmeaza a fi creat
* @param str - Array-ul de stringuri de unde se vor lua argumentele de pus in celula
* @param n - numarul de coloane pe care le va avea Tabelul
* @param nr - numarul de linii pe care le va avea tabelul.

***public void WriteTable(String name, String[] str, int n, int nr)***

* This method creates a pdf with a name and a text int it.
* @param name - The name of the pdf
* @param text - The text in the pdf

Si o metoda secundara :

***private void addRows(PdfPTable table, String[] str, int n, int i)***

* Aceasta metoda este apelata de metoda writeTable si adauga o linie dupa parametrii:
* @param table -tabelul in care se va adauga o linie
* @param str - Array-ul de stringuri de unde o sa luam parametrii
* @param n - numarul de coloane
* @param i - indicele i ne ajuta sa vedem la ce linie suntem .

***Pachetul start***

Acest pachet contine o singura clasa ,clasa start cu metoda main . In metoda main vom deschide fisiererul pentru a citit comenzile , citeste cate o linie si o trimite spre controller pentru a putea fi procesata.

# *Rezultate*

Am reusit sa citesc orice tip de instructiune dintre cele precizate si sa le execut. In caz de succes se vor genera rapoartele in pdf-uri cu numele RaportClient1,RaportProduct1,RaportClient2,… etc .

In caz de eroare aceasta va fi trata ca atare .Spre exemplu daca se face un order cu un produs care nu se afla in stock se genereaza un pdf OrderError1 in care se va specifica numele persoanei care a facut comanda si faptul ca produsul : nume\_produs nu se afla in stoc . La fel si pentru stoc insuficient sau daca clientul nu se afla in Baza de date .

# Concluzii

# 

Consider ca in aceasta tema am invatat sa lucreaz mult mai bine cu clase generice , ceea ce e foarte bine la creeare proiectelor cat mai generale . Am reusit si sa imi imbunatasesc cunostiintele de mysql si connection prin JDBC invatand sa creez o conexiune intre o baza de date si un program java si sa apelez query-uri si comenzi mysql din interiorul programului java.

Ca si posibilitati de dezvoltare , am putea modifica modul de Delete pentru client si produs astfel ca foreign key-ul din order sa nu genreze o eroare . Eu am folosit metoda Delete on cascade , dar ar fi mai optim sa nu stergem toate order-urile vechi deoarece acestea ar fi bune la creearea de raporte statistice pentru a sti ce produse au fost vandute mai mult . De altfel am putea adauga si un mod de HOLD pentru orderurile ce nu pot fi gestionate din cauza stoclui insuficient pana la refacerea stocului.

# *Bibliografie*

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

<https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html>

<http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_3/Assignment_3_Indications.pdf>

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

<https://www.baeldung.com/java-pdf-creation>

<https://dzone.com/articles/layers-standard-enterprise>