**DOCUMENTATIE TEMA NR.3**

ORDER MANAGEMENT

Student: Bumbu Iulia-Diana

An: II Semestrul: 2

Grupa: 30228

CUPRINS:

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

1.1.Obiectiv principal:

Principalul scop al proiectului este de a realiza o aplicatie care sa gestioneze comenzile clientilor si sa stocheze informatii despre clienti, produsele existente si comenzile facute intr-o baza de date definita in MySql. Astfel, fiecare client inregistrat poate adauga in comanda sa produse disponibile si dupa finalizarea comenzii sa se genereze o factura. De asemenea, exista posibilitatea de a genera rapoarte cu privire la componentele tabelelor din baza de date in orice moment.

1.2.Obiective secundare:

Pentru o dezvoltare optima a unei aplicatii de acest tip trebuie realizati anumiti pasi pentru indeplinirea obiectivului principal.

Principalele **obiective secundare** sunt:

* Realizarea USE CASE-urilor si a scenariilor:

O diagrama a cazurilor de utilizare ofera o descriere generala a modului in care va fi utilizat sistemul, furnizand o privire de ansamblu asupra functionalitatilor care doresc a fi oferite de sistem. (capitolul 2)

* Identificarea structurilor de date:

Structura de date este o metoda sistematica de stocare a informatiilor intr-un calculator astfel incat sa poata fi folosite in mod eficient. (capitolul 3)

* Impartirea claselor:

Pentru a gestiona mult mai usor logica bazei de date si logica realizarii functionalitatilor dorite proiectul se imparte in mai multe layere cu functionalitati diferite, programul avand astfel o arhitectura 3-tier (capitolul 3)

* Dezvoltarea algoritmilor:

Se va determina strategia necesara pentru a gestiona prelucrarea comenzilor(capitolul 3)

* Implementarea solutiei:

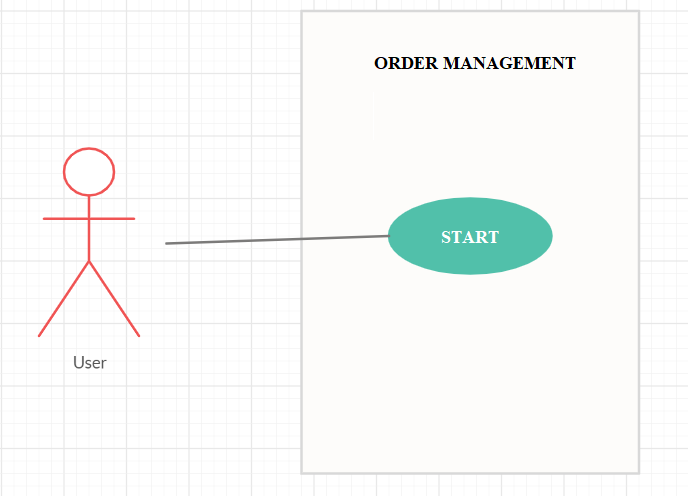
Este reprezentata de descrierea in cod a tuturor claselor anterior explicate, a metodelor necesare gestionarii tabelelor precum si procesarea de fisiere pentru a testa functionalitatea programului creat. (capitolul 4)

* Gestionarea rezultatelor:

Se vor citi datele de intrare din fisierele in-test.txt, iar rezultatul generat se va scrie in documente pdf in cazul in care s-au dat comenzile respective precum si in baza de date(capitolul 5)

2.Analiza problemei:

2.1.USE CASE Diagram:



2.2. Scenarii:

O utilizare corecta a programului presupune introducerea unui fisier in-test.txt cu continut valid pentru a determina conditiile initiale de desfasurare. Un input valid contine instructiunile: insert, delete, order, report cu urmatoarea sintaxa: (VEZI SI EXPLICATIILE METODELOR DIN SECTIUNEA 4.3 LA DELETE!)

-pentru a insera un nou client in baza de date:

**Insert client: nume, adresa**

-pentru a sterge un client din baza de date:

**Delete client: nume, adresa**

-pentru a insera un produs in baza de date:

**Insert product: denumire, cantitate, pret**

-pentru a sterge un produs din baza de date:

**Delete product: denumire**

-pentru a crea o comanda:

**Order: nume, denumire, cantitate; denumire2, cantitate2; denumire3, cantitate3....**

-pentru a genera un raport care contine toti clientii:

**Report client**

-pentru a genera un raport care contine toate produsele existente in baza:

**Report product**

-pentru a genera un raport cu toate comenzile efectuate:

**Report orders**

Dupa procesarea inputului valid ca raspuns se vor actualiza tabelele din baza de date cu informatiile nou introduse. Se considera ca inainte de procesarea fisierului baza de date este goala. De asemenea, pentru fiecare comanda este generata cate o factura.

In cazul in care sintaxa instructiunilor existente nu este respectata( lipseste doua puncte : / lipseste virgula , / lipseste spatiul dintre caractere / numele instructiunii este scris gresit / cantitatile/preturile introduse nu sunt numere intregi/reale si pozitive / in cazul in care se comanda mai multe produse in acelasi order nu exista punct si virgula intre acestea si spatiu la sfarsitul sirului de produse si cantitati dorite scenariul de functionare este gresit iar ca si consecinta executia programului se va incheia, fiind generate exceptii. De aceea, se recomanda mare atentie la scrierea corecta a instructiunilor pentru o functionare fara impedimente!

1. Proiectare:

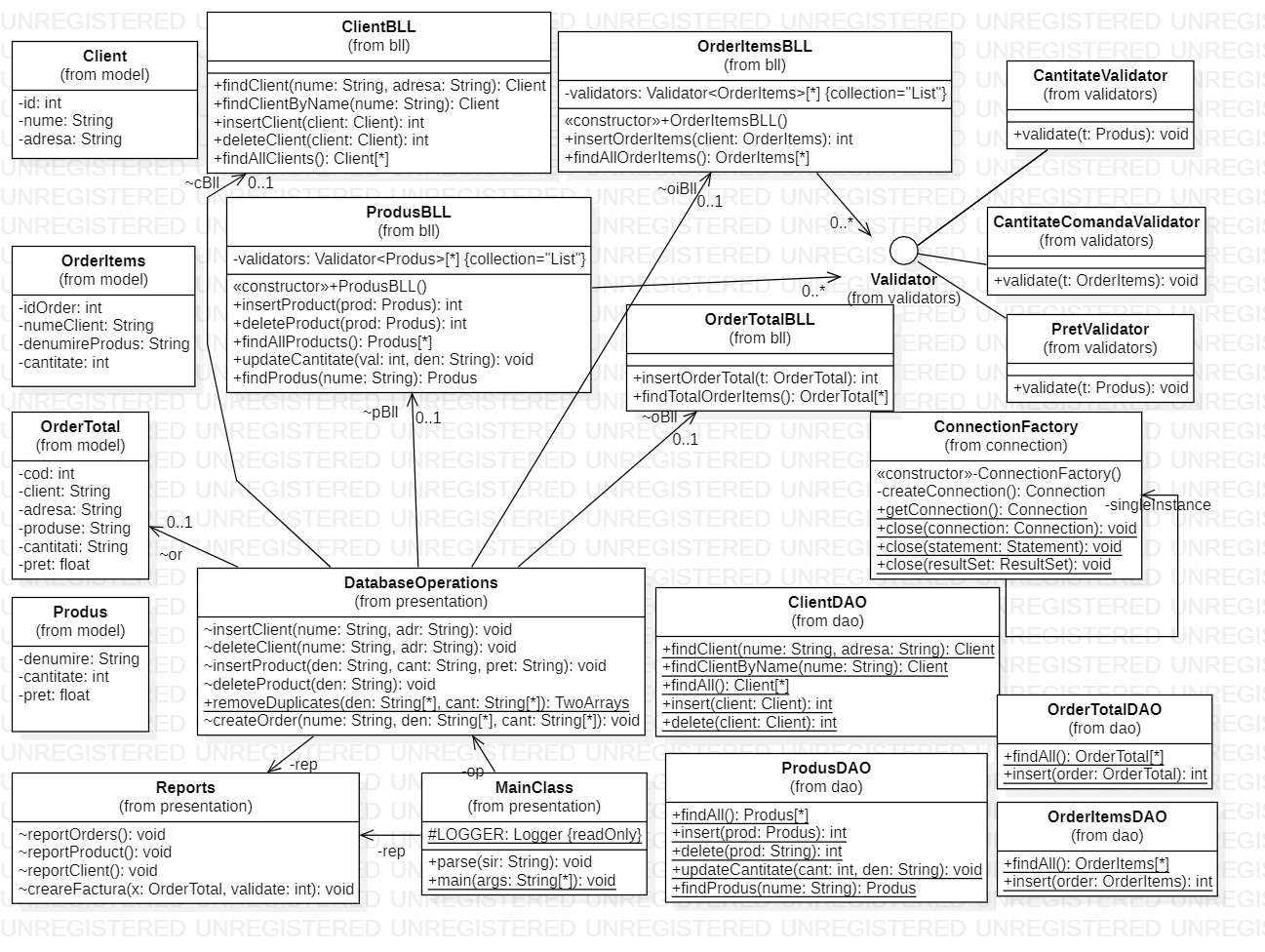
3.1.Structuri de date:

Deoarece multipli clienti/produse/comenzi trebuie procesati/procesate in mai multe metode adesea este folosita ca structura de date ArrayList deoarece faciliteaza lucrul cu vectori prin metodele preimplementate. De exemplu:

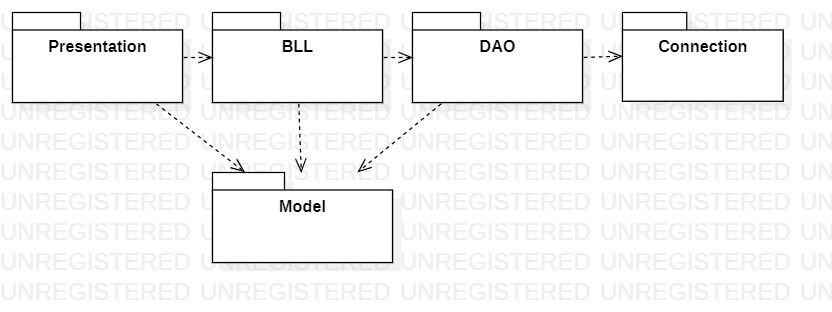
ArrayList<Client> list= **new** ArrayList<Client>();

3.2.Diagrama de clase

Diagrama de clase este folosita in modelarea orientata obiect pentru a descrie structura statica a sistemului si a modului in care este el structurat. De asemenea, aceasta ofera o notatie greafica pentru reprezentarea claselor (entitati care au caracteristici comune) si a relatiilor (legaturi dintre doua sau mai multe clase) dintre ele. Pentru lizibilitatea schemei nu au fost puse relatiile de dependenta intre clase, insa acestea exista intre: DatabaseOperations si Produs, DatabaseOperations si OrderItems, DatabaseOperations si Client, DatabaseOperations si TwoArrays, Reports si Produs, Reports si OrderTotal, Reports si Client, Reports si ProdusBLL, Reports si OrderTotalBLL, Reports si ClientBLL, ProdusDAO si Produs, ProdusDAO si ConnectionFactory, Produs BLL si Produs, Produs BLL si ProdusDAO, ProdusBLL si PretValidator, PretValidator si Produs, ProdusBLL si CantitateValidator, CantitateValidator si Produs, ClientDAO si Client, ClientDAO si ConnectionFactory, ClientBLL si ClientDAO, ClientBLL si Client, OrderTotalDAO si OrderTotal, OrderTotalDAO si ConnectionFactory, OrderTotalBLL si OrderTotal, OrderTotalBLL si OrderTotalDAO, OrderItemsDAO si OrderItems, OrderItemsDAO si ConnectionFactory, OrderItemsBLL si OrderItemsDAO, OrderItemsBLL si OrderItems, OrderItemsBLL si CantitateComandaValidator, CantitateComandaValidator si OrderItems.



3.3.Diagrama de pachete:



UML furnizeaza mijloace de grupare a elementelor din cadrul diagramelor numite pachete. Pachetele constituie baza pentru controlul configuratiei si stocare, contribuind la reutilizarea codului.

3.4.Algoritmi:

In cazul acestui proiect nu au fost utilizati algoritmi preimplementati insa exista o anumita strategie in ceea ce priveste crearea unei comenzi:

**function** creeazaComanda (Client c, Produs p, Cantitate x) **is**

**if**  c nu exista in baza de date **then**

generare mesaj de eroare

return //nu efectuam comanda

**if**  p nu exista in baza de date **then**

generare mesaj de eroare

return //nu efectuam comanda

**if**  x > cantitatea produsului x existenta in baza de date **then**

generare mesaj de eroare

return //nu efectuam comanda

//daca am ajuns aici inseamna ca nu au existant impedimente si comanda se poate efectua

**creeazaComanda** (c, p, x) + **generareFactura**

1. Implementare:

4.1.Descrierea claselor din pachetul Model:

-contine clasele care se mapeaza tabelelor din baza de date creata

* Clasa Client:

Are ca principala functionalitate stocarea datelor de identificare ale fiecarui client deoarece contine atributele:

**private** **int** id;

**private** String nume;

**private** String adresa;

Este asociata tabelului “client” in care se introduce fiecare noua persoana pe care o inseram cand parsam fisierul. Clasa contine contructorul specific cu campuri, metode get si set precum si doua metode printHeader() si printLine() care ajuta in cazul in care se doreste afisarea tuturor clientilor asemenea formei unui tabel. Id-ul de indentificare este unic pentru fiecare client si constituie cheie primara in cadrul tabelului asociat.

* Clasa Produs:

Are rolul de retine datele definitorii pentru un nou produs pe care dorim sa il adaugam in baza de date deoarece fiecare atribut este atribuit unui camp:

**private** String denumire;

**private** **int** cantitate;

**private** **float** pret;

Se remarca existenta contructorului specific cu campuri,a metodelor get si set precum si a doua metode printHeader() si printLine() care ajuta in cazul in care se doreste afisarea tuturor produselor asemenea formei unui tabel. Denumirea este unica pentru fiecare produs si constituie cheie primara in cadrul tabelului asociat.

* Clasa OrderItems:

Principala sa functionalitate este de a retine fiecare produs individual al comenzii, cantitatea produsului si clientul care il doreste. Ca si celelalte clase din pachetul “model” aceasta clasa are un echivalent in baza de date cu campuri asemenea atributelor:

**private** **int** idOrder;

**private** String numeClient;

**private** String denumireProdus;

**private** **int** cantitate;

Clasa contine constructorul specific, metode get si set pentru fiecare camp, precum si metodele printHeader() si printLine() pentru afisare mai facila in consola. Cheia primara in tabelul asociat este reprezentata de idOrder. Necesitatea acestui tabel apare in cazul in care procesam comenzi cu mai multe produse si dorim sa tinem evidenta si a produselor individuale, nu doar a unui total al comenzii.

* Clasa OrderTotal:

Clasa contine constructorul cu campuri, metode get si set si metodele printHeader() si printLine() care in acest caz sunt folosite la afisarea comenzii totale asociate unui fisier intr-un document pdf reprezentand factura. Ca si atribute specifice clasei se disting:

**private** **int** cod;

**private** String client;

**private** String adresa;

**private** String produse;

**private** String cantitati;

**private** **float** pret;

Desi contraintuitiv campurile produse si cantitati sunt stringuri, nu siruri de produse sau siruri de integer deoarece in cadrul tabelului asociat in baza de date tipul asociat acestor campuri este declarat va VARCHAR deoarece tipurile primitive alea coloanelor din MySql nu au notiunea de vector. Asadar campul produs va fi perceput ca si inlantuirea tuturor produselor dorite de un client intr-o comanda, iar cantitati contine pe pozitia echivalenta din sir cantitatea produsului. In cadrul tabelului, cheia primara este constituia de campul unic cod.

* 1. Descrierea claselor din pachetul Connection:
* Clasa ConnectionFactory:

Clasa contine numele driverului, locatia bazei de date, userul si parola pentru a accesa serverul MySql. Conexiunea la baza de date se va plasa intr-un obiect Singleton\*. Clasa contine metode pentru a realiza conexiunea si de a o inchide. Realizarea acesteia a fost preluata din sursa numarul (3) citata in bibliografie.

* 1. Descrierea claselor din pachetul Presentation:
* Clasa MainClass:

Principalul rol al acestei clase este de a prelua fisierul de intrare si de a formata datele din acesta. Astfel, se disting doua metode importante:

**public** **static** **void** main(String[] args) –primeste ca argument calea fisierului de intrare care trebuie prelucrat. Dupa deschiderea acestuia fiecare linie din fisier este adaugata intr-un ArrayList<String> pentru a fi ulterior manipulata. Pentru acest lucru se apeleaza metoda parse dupa ce fiecare linie a fost adaugata.

**public** **void** parse(String sir) - cu rol in prelucrarea sirului corespunzator unei linii din fisier trimis ca parametru. Fisierul valid contine pe fiecare linie o instructiune de tipul celor specificate in sectiunea 2.2 si de aceea sirul se parseaza facand split intre semne de punctuatie definitorii precum doua puncte, virgula sau punct si virgula pentru a identifica numele instructiunii precum si parametrii acesteia. Dupa ca am gasit toti parametrii cazului identificat se apeleaza functia asociata instructiunii din clasele DatabaseOperation sau Reports.

* Clasa DatabaseOperations:

Clasa are rolul de a instantia noi clienti/produse/comenzi si de a apela metodele responsabile de prelucrarea bazei de date dupa verificarile necesare. Se disting deopotriva trei atribute contor folosite in instantierea unica:

**private** **static** **int** *i* = 0; //contor pt id Client

**private** **static** **int** *j* = 0; //contor pt id OrderItems

**private** **static** **int** *k* = 0; //contor pentru cod OrderTotal

Metodele sale au fost apelate din metoda parse prezentata anterior, acestea fiind:

**void** insertClient(String nume, String adr) –daca clientul cu numele dat exista deja in tabel se afiseaza un mesaj corespunzator iar in caz contrar se apeleaza metoda din BLL responsabila cu inserarea in tabel

**void** deleteClient(String nume, String adr) –deoarece exista posibilitatea ca un potential client sa faca niste comenzi, sa ii fie generate facturi pentru acestea apoi ulterior sa nu mai doreasca sa fie procesat exista posibilitatea stergerii lui din baza de date. Stergerea sa nu implica ca i se anuleaza comenzile anterioare deoarece acestea au fost finalizate odata cu generarea facturii asa ca atunci cand se doreste stergerea clientului se apeleaza doar metoda din BLL responsabila cu stergerea din tabelul client, nu si din order.

**void** insertProduct(String den, String cant, String pret) – daca produsul cu denumirea data exista deja in tabel se face update la cantitatea existenta adaugand-o si pe cea trimisa ca parametru iar in caz contrar se apeleaza metoda din BLL responsabila cu inserarea in tabel a noului produs. In cazul in care se incearca inserarea unui produs care exista deja in tabel, insa are pret diferit se afiseaza in consola un mesaj de eroare deoarece denumirea trebuie sa fie unica!

**void** deleteProduct(String den) –aplicand acelasi rationament ca si in cazul stergerii clientului, stergerea unui produs din baza de date inseamna ca acesta nu va mai putea fi procesat in comenziile viitoare, nu ca nu a existat nicio comanda care sa contina acel produs asa ca se apeleaza doar metoda din BLL responsabila cu stergerea din tabelul produs, nu si din order.

**public** **static** TwoArrays removeDuplicates(ArrayList<String> den, ArrayList<String> cant) –utilitatea acestei metode intervine in cazul in care un client face o comanda de genul:

Order: Sandu Vasile, apple, 5; lemon, 6; apple, 4

Dorim ca in comanda finala a clientului Sandu Vasile sa apara apple 9; lemon, 6. Asadar functia acestei metode este de a verifica daca in aceeasi comanda existta acelasi tip de produs si daca da sa se scrie o singura data cu cantitatea insumata.

**void** createOrder(String nume, ArrayList<String> den, ArrayList<String> cant) –metoda aplica strategia mentionata la sectiunea 3.4. Dupa ce se verifica cazurile de anulare a comenzii pentru ca datele de intrare au fost eronate: nu exista clientul cu numele nume, nu exista unul dintre produse cu denumirea den, nu exista stoc suficient pentru unul dintre produse se elimina duplicatele din sirul de denumiri de produse si sirul de cantitati asociate primite ca parametru utilizand functia anterior prezentata. Daca se trece de toti acesti pasi inseamna ca se efectueaza o comanda valida asa ca se scade din stocul produselor existente cantitatea dorita in comanda, apoi se instantiaza in noi OrderItems pentru fiecare produs separat al comenzii precum si un OrderItem total pe care le inseram in tabelele corespunzatoare prin apelul functiei din BLL. De asemenea, se genereaza o factura pentru comanda totala atestand ca aceasta a fost finalizata.

* Clasa Reports:

Dupa cum a fost prezentat si in sectiunea 2.2 exista posibilitaea generarii unor rapoarte care sa contina toate comezile efectuate, toate produsele existente in stoc si toti clientii. Aceasta clasa contine acest tip de operatii precum si generarea facturilor, fiecare avand cate un contor specific pentru generare diferita a numelui:

**private** **static** **int** *countRepClient* = 1;

**private** **static** **int** *countRepProdus* = 1;

**private** **static** **int** *countRepOrder* = 1;

**private** **static** **int** *countFactura* = 1;

**void** reportOrders()–se pastreaza intr-un sir de comenzi toate liniile existente in tabelul OrderTotal plin aplicarea functiei: ArrayList<OrderTotal> list = oBll.findTotalOrderItems(); Pentru acest sir se genereaza un tabel intr-un document cu format PDF folosind functiile specifice de creare de documente.

**void** reportProduct()–se pastreaza intr-un sir de comenzi toate liniile existente in tabelul Produs plin aplicarea functiei: ArrayList<Produs> list = pBll.findAllProducts(); Pentru acest sir se genereaza un tabel intr-un document cu format PDF folosind functiile specifice de creare de documente.

**void** reportClient()–se pastreaza intr-un sir de comenzi toate liniile existente in tabelul Client plin aplicarea functiei: ArrayList<Client> list = cBll.findAllClients(); Pentru acest sir se genereaza un tabel intr-un document cu format PDF folosind functiile specifice de creare de documente.

**void** creareFactura(OrderTotal x, **int** validate) –metoda genereaza o factura in format PDF pentru fiecare comanda in care se scrie un mesaj de eroare in caz ca datele facturii nu au fost adecvate sau componenta comenzii in caz contrar. Realizarea validarii se face in functie de parametrul validate primit, valorile 1, 2 si 3 reprezentand cazuri de eroare.

* Clasa TwoArrays:

Singura sa utilitate implementata a fost sa cream un nou obiect care contine doua ArrayList<String> deoarece avem nevoie de un astfel de obiect returnat in cadrul metodei removeDuplicates din clasa DatabaseOperations.

* 1. Descrierea claselor din pachetul BLL:

-contine clasele care incapsuleaza logica aplicatiei

* Clasa ClientBLL:

Clasa responsabila cu apelarea metodelor echivalente din pachetul DAO, clasa ClientDAO, continand metodele:

**public** Client findClient(String nume, String adresa)

**public** Client findClientByName(String nume)

**public** **int** insertClient(Client client)

**public** **int** deleteClient(Client client)

**public** ArrayList<Client> findAllClients()

* Clasa ProdusBLL:

Clasa responsabila cu apelarea metodelor echivalente din pachetul DAO, clasa ProdusDAO. Se remarca existenta unor validatori care in caz ca datele produsului nu sunt corecte arunca o exceptie. Principalele metode sunt:

**public** **int** insertProduct(Produs prod) –cu validarea produsului inainte de inserare pentru nu a avea date inconsistente in tabel

**public** **int** deleteProduct(Produs prod) –cu validarea produsului pentru a nu sterge un produs inexistent

**public** ArrayList<Produs> findAllProducts()

**public** **void** updateCantitate(**int** val, String den)

**public** Produs findProdus(String nume)

* Clasa OrderItemsBLL:

Clasa responsabila cu apelarea metodelor echivalente din pachetul DAO, clasa OrerItemsDAO. Se remarca existenta unui validator care in caz ca datele itemului din order nu sunt corecte arunca o exceptie. Metodele existente sunt:

**public** **int** insertOrderItems(OrderItems client) –cu validarea cantitatii comandate

**public** ArrayList<OrderItems> findAllOrderItems()

* Clasa OrderTotalBLL:

Clasa responsabila cu apelarea metodelor echivalente din pachetul DAO, clasa OrderTotalDAO, continand metodele:

**public** **int** insertOrderTotal(OrderTotal t)

**public** ArrayList<OrderTotal> findTotalOrderItems()

* 1. Descrierea claselor din pachetul BLL.Validators:
* Clasa PretValidator:

Implementeaza interfata Validator<Produs> si contine metoda **public** **void** validate(Produs t) care verifica daca pretul unui produs este un numar pozitiv. In caz ca nu se respecta conditia se arunca o exceptie.

* Clasa CantitateValidator:

Implementeaza interfata Validator<Produs> si contine metoda **public** **void** validate(Produs t) care verifica daca cantitatea unui produs este un numar pozitiv. In caz ca nu se respecta conditia se arunca o exceptie.

* Clasa CantitateComandaValidator:

Implementeaza interfata Validator<OrderItems> si contine metoda **public** **void** validate(OrderItems t) care verifica daca cantitatea unui item al comenzii este un numar pozitiv. In caz ca nu se respecta conditia se arunca o exceptie.

* 1. Descrierea claselor din pachetul DAO:

-contine clasele care au in componenta lor interogarile specifice bazei de date

* Clasa ClientDAO:

Are ca atribute statice query-uri specifice pentru fiecare operatie executata pe tabelul “client”:

**private** **static** **final** String ***insertStatementString*** = "INSERT INTO client (idClient, nume, adresa)" + " VALUES (?,?,?)";

**private** **final** **static** String ***deleteStatementString*** = "DELETE FROM client where nume = ? and adresa = ?";

**private** **final** **static** String ***findStatementString*** = "SELECT \* FROM client where nume = ? and adresa = ?";

**private** **final** **static** String ***findNameStatementString*** = "SELECT \* FROM client where nume = ?";

**private** **final** **static** String ***findAllStatementString*** = "SELECT \* FROM client";

Se individualizeaza metodele:

**public** **static** Client findClient(String nume, String adresa) –responsabila cu cautarea in tabel a clientului cu numele si adresa specificate in parametri si returnarea acestuia daca s-a gasit

**public** **static** Client findClientByName(String nume) –responsabila cu cautarea in tabel a clientului cu numele specificat in parametri si returnarea acestuia daca s-a gasit

**public** **static** ArrayList<Client> findAll()–functia returneaza intr-un sir toti clientii existenti in tabelul “client” si datele lor de identificare

**public** **static** **int** insert(Client client) –insereaza clientul primit ca parametru in baza de date

**public** **static** **int** delete(Client client) –sterge clientul primit ca parametru din baza de date

* Clasa ProdusDAO:

Are ca atribute statice query-uri specifice pentru fiecare operatie executata pe tabelul “produs”:

**private** **static** **final** String ***insertStatementString*** = "INSERT INTO produs (denumire, cantitate, pret)" + " VALUES (?,?,?)";

**private** **final** **static** String ***deleteStatementString*** = "DELETE FROM produs where denumire = ?";

**private** **final** **static** String ***findAllStatementString*** = "SELECT \* FROM produs";

**private** **final** **static** String ***updateCantitateStatementString*** = "UPDATE produs SET cantitate = ? WHERE denumire = ?";

**private** **final** **static** String ***findProdusStatementString*** = "SELECT \* FROM produs WHERE denumire = ?";

Metodele existente sunt:

**public** **static** ArrayList<Produs> findAll()–functia returneaza intr-un sir toate produsele existente in tabelul “produs” si detaliile acestora

**public** **static** **int** insert(Produs prod) –insereaza produsul primit ca parametru in baza de date

**public** **static** **int** delete(String prod) –sterge produsul primit ca parametru din baza de date

**public** **static** **void** updateCantitate(**int** cant, String den) –are rolul de a actualiza cantitatea produsului cu denumirea den in baza de date

**public** **static** Produs findProdus(String nume) –responsabila cu cautarea in tabel a produsului cu numele specificat in parametri si returnarea acestuia daca s-a gasit

* Clasa OrderItemsDAO:

Are ca atribute statice query-uri specifice pentru fiecare operatie executata pe tabelul “orderitems”:

**private** **static** **final** String ***insertStatementString*** = "INSERT INTO orderitems (id, nume, denumire, cantitate)" + " VALUES (?,?,?,?)";

**private** **final** **static** String ***findAllStatementString*** = "SELECT \* FROM orderitems";

Avem ca metode:

**public** **static** ArrayList<OrderItems> findAll()–functia returneaza intr-un sir toate itemele din comenzi existente in tabelul “orderitems” si detaliile acestora

**public** **static** **int** insert(OrderItems order) –insereaza itemul din comanda primit ca parametru in baza de date

* Clasa OrderTotalDAO:

Are ca atribute statice query-uri specifice pentru fiecare operatie executata pe tabelul “ordertotal”:

**private** **static** **final** String ***insertStatementString*** = "INSERT INTO ordertotal (cod, client, adresa, produse, cantitate, pret)" + " VALUES (?,?,?,?,?,?)";

**private** **final** **static** String ***findAllStatementString*** = "SELECT \* FROM ordertotal";

Exista metodele:

**public** **static** ArrayList<OrderTotal> findAll()–functia returneaza intr-un sir toate comenzile existente in tabelul “ordertotal” si detaliile acestora

**public** **static** **int** insert(OrderTotal order) –insereaza comanda primita ca parametru in baza de date

1. Rezultate:

Pentru realizarea verificarii rezultatelor programul este configurat sa ruleze din terminal si sa citeasca un parametru: calea fisierului de intrare In acest sens s-a configurat un fisier .jar. De asemenea, a fost creat un dump al bazei de date pentru a avea aceeasi definire a tabelelor. Corectitudinea rezultatelor se poate verifica prin analizarea datelor din baza de date sau a pdf-urilor generate

.

1. Concluzii:

In concluzie, aceasta tema mi-a oferit oportunitatea de lucra cu baze de date in contextul limbajului Java si de a descoperi noi tehnici de impartire a proiectului in layere. Astfel, am sedimentat o parte din conceptele de baza ale lucrului cu acestea si am aprofundat alte cunostinte in legatura cu POO . Proiectul poate avea alte dezvoltari ulterioare prin adaugarea de noi tabele cu noi functionalitati sau prin creerea unei interfete grafice care sa usureze setarea datelor initiale.

7. Bibliografie:

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

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

3. <https://utcn_dsrl@bitbucket.org/utcn_dsrl/pt-layered-architecture.git>