**Facultate: Automatică și Calculatoare**

**Specializare: Calculatoare și Tehnologia Informației**

**Disciplina: Tehnici de Programare**

**Tema: Gestionarea Comenzilor**

Student : Herman Felician-Nicu

Seria: B

Grupa: 30229

**Cuprins**

1.Obiectivul temei

1.1 Obiectivul Principal

1.2 Obiective Secundare

2.Analiza problemei, modelare, scenarii, cazuri de utilizare

3. Proiectare

3.1 Structuri de Date

3.2 Proiectare Clase

3.3 Diagrama UML

4. Implementare

4.1 Pachetul Connection

4.1.1 Clasa ConnectionFactory

4.2 Pachetul Model

4.2.1 Clasa Client

4.2.2 Clasa Product

4.2.3 Clasa Order

4.3 Pachetul DAO

4.3.1 Clasa ClientDAO

4.3.2 Clasa ProductDAO

4.3.3 Clasa OrderDAO

4.4 Pachetul BLL

4.4.1 Clasa ClientBLL

4.4.2 Clasa ProductBLL

4.4.3 Clasa OrderBLL

4.5 Pachetul Presentation

4.5.1 Clasa FileParser

4.5.2 Clasa GeneratePDF

4.6 Pachetul Start

4.6.1 Clasa Main

5. Rezultate

6. Concluzii

7. Bibliografie

**1.Obiectivul temei**

**1.1 Obiectivul principal:**

Obiectivul principal al temei este reprezentat de proiectarea unui sistem de gestionare a comenzilor. Aceasta gestiune a comenzilor incepe inca de la creearea bazei de date, a tabelelor. Mai apoi, in tabele se introduc date conform unor instructiuni citite din fisier. In final, pe baza informatiilor din tabele, clientilor li se pot asocia o comanda in limita produselor disponibile in baza de date. Un raport este generat in fisiere PDF in functie de cerinta utilizatorului.

**1.2 Obiective secundare**:

|  |  |  |
| --- | --- | --- |
| Obiectiv Secundar | Descriere | Capitol |
| Scenarii | In computere, un scenariu este o naratiune a interactiunilor dintre roluri ( actori in UML ) si sistem. In cazul nostru generarea fisierului de iesire pe baza fisierului de intrare. | 2 |
| Alegerea structurilor de date | Structurile de date folosite pentru a duce la capat obiectivul principal | 3 |
| Impartirea pe clase | Am ales o impartire pe clase foarte bine structurata, in care se creeaza legaturi intre operatiile de insert, update, pe baza unor conditii. | 3 |
| Dezvoltarea algoritmilor | Vor fi descrise structurile de date necesare pentru atingerea obiectivului principal, schema UML ( diagrama de clase ) precum si algoritmii folositi pentru realizarea actiunilor inserare, update si raport. | 4 |
| Implementarea solutiei | Vor fi descrise fiecare clasa cu campurile si metodele importante . | 4 |
| Testare | O sa fie citite din fisier diferite operatii, precum inserarea unui client in baza de date, stergerea acestuia sau creearea unei comenzi pe numele lui. Pe langa clienti, o sa fie disponibile si operatii cu produse. | 5 |

**2.Analiza Problemei**

Datorita evolutiei tehnologiei intr-un ritm atat de alert, magazinele offline o sa dispara incet, incet deci din ce in ce mai multe firme o sa se bazeze pe site-uri in vanzarea produselor. Aceasta aplicatie este exact partea de „back-end” a unui site, partea care se ocupa de baza de date a magazinului. Aceasta baza de date memoreaza informatii referitoare la clientii care fac cumparaturi, la produsele disponibile(nume, cantitate, pret) si la comenzile facute de clienti. In cazul unei insuficiente a unui produs pe care un client doreste sa-l achizitioneze, o sa fie generat un PDF de eroare.

Aplicatia o sa functioneze astfel, in fisierul de intrare o sa fie introduse pe rand comenzi, ca de exemplu:

Insert client: Nume, Adresa – aceasta comanda o sa duca la introducerea unui client pe nume „Nume” care locuieste la adresa „Adresa” in tabelul „clients” din baza de date.

Insert product : Denumire, cantitate, pret – comanda respectiva o sa introduca in tabelul „product” un produs „Denumire”, in cantitatea respectiva, cu pretul respectiv. In cazul in care exista deja introdus un produs „Denumire”, acest produs o sa fie updatat, noua cantitate fiind calculata ca „cantitate” + vechea cantitate. Deasemenea, pretul o sa fie cel mai tarziu pret introdus.

Delete client: Nume, Adresa – o sa fie cautat clientul respectiv in baza de date, iar daca acesta exista, o sa fie sters

Delete product : Denumire – o sa fie sters produsul cu denumirea respectiva

Report Client, Product, Order – genereaza un tabel si-l pune intr-un PDF generat de noi. Fiecare Report sau Update Esuat o sa genereze un PDF independet.

Order : Nume, Produs, Cantitate – o comanda pe numele clientului Nume o sa fie creeata avand produsul „Produs” cu cantitatea transmisa din fisier

**3.Proiectarea**

**3.1 Structuri de date**

In ceea ce priveste structurile de date, am incercat sa le folosesc pe cele care s-ar potrivi cel mai bine in functie de contextul utilizarii lor.

Fiecare clasa are un loc bine-stabilit in pachetul propriu, astfel clasele care se ocupa de operatiile cu baza de date sunt structurate in acelasi pachet, cele legate de afisare, la fel...etc.

**3.2 Proiectare Clase**

Tema contine 6 pachete care la randul lor contin un anumit numar de clase. Se incepe cu pachetul **Connection** in care este implementata clasa ConnectionFactory unde este realizata conexiunea cu baza noastra de date de pe un host local folosind un username si o parola specifica.

Avand conexiunea la baza de date, trecem la implementarea tabelelor in clasele din pachetul **Model.** Tabelele fiind instatiate si in proiectul nostru java, urmeaza sa efectuam operatiile specifice. Pachetul **DAO** este strict raspunzator de implementarea operatiilor de baza cu tabele(inserare, update, select, stergere).

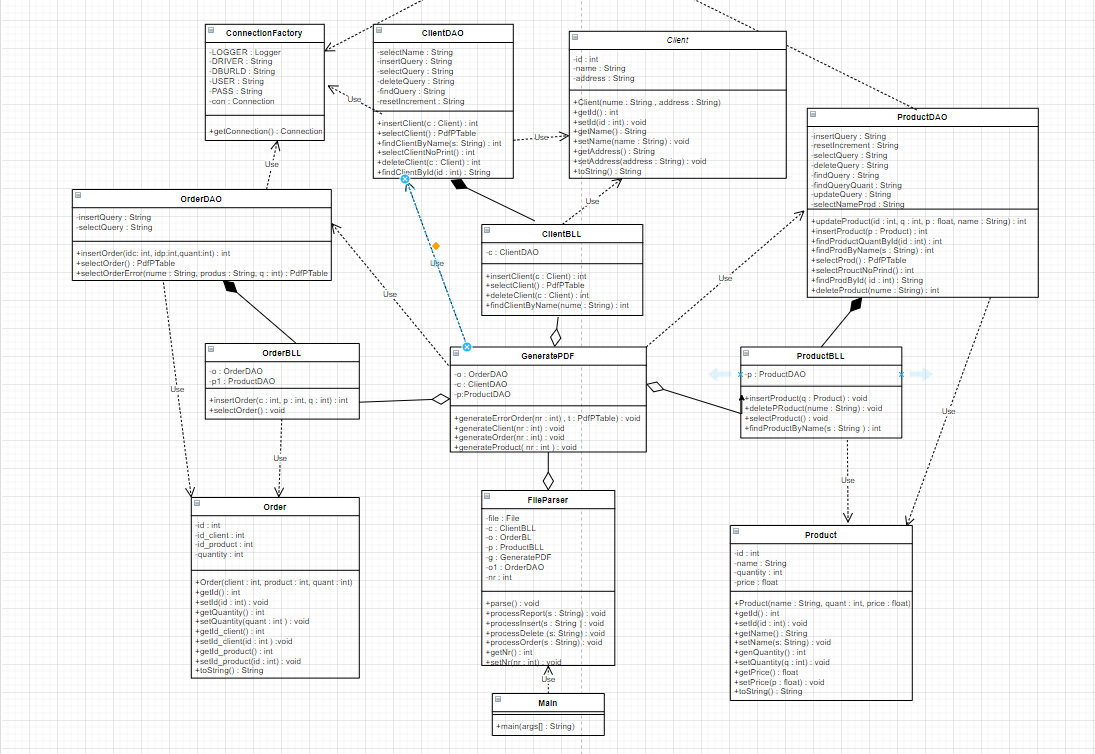
In cazul nostru avem nevoie de pachetul **BLL** doar pentru a creea o comanda, verificand daca exista numarul de produse dorite de client in baza de date. Dar in alte proiecte mai mari, acest pachet implementeaza clase de validatori pentru anumite operatii.

Odata avand introduse date in tabele, trebuie sa le prelucram ca sa fie vizibile utilizatorului. Acest proces este implementat in pachetul **Presentation** care contine clasele de generare a unui PDF si de scriere in acel document.

In final, toate aceste pachete cu clasele respective sunt compilate si executate din clasa Main care se gaseste in pachetul **Start.**

Mai departe o sa prezint fiecare clasa din fiecare pachet cu intrebuintarea ei, functiile si utilitatile pe care le realizeaza.

**3.3 Diagrama UML**

****

**4.Implementare**

**4.1 Pachetul Connection**

**4.1.1 Clasa ConnectionFactory**

Se realizeaza conexiunea la baza de date de pe serverul local MYSQL. Aici e implementata functia getConnection() care returneaza conexiunea in caz de succes si eroare altfel

**4.2 Pachetul Model**

**4.2.1 Clasa Client**

Sunt declarate campuri cu numele coloanelor din baza de date si este implementat un constructor care o sa creeze un Client cu un nume si o adresa.

**4.2.2 Clasa Product**

Sunt declarate campuri cu numele coloanelor din baza de date si este implementat un constructor care o sa creeze un Produs cu un nume si o adresa.

**4.2.3 Clasa Order**

Sunt declarate campuri cu numele coloanelor din baza de date si este implementat un constructor care o sa creeze o comanda cu id-ul clientului, id-ul produsului si cantitatea. Am ales sa memorez comanda folosind id-uri deoarece in baza de date acestea reprezeinta **CHEI STRAINTE** pentru tablea Orders care fac referinta la tabela clients si product. Astfel, cand incercam sa inseram o comanda pentru o persoana care nu exista, o sa primim o exceptie.

**4.3 Pachetul DAO**

**4.3.1 Clasa ClientDAO**

Am inceput prin definirea unor stringuri care reprezinta query-uri pe baza carora o sa lucram cu datele din baza de date. Functia insertClient() imi insereaza in baza de date cate un client trimis ca parametru al functiei, id-ul acestuia in baza de date autoincrementandu-se automat. Functia selectClient returneaza un PdfPTable deoarece returnul functiei o sa fie adaugat in tabelul generat in PDF. Acest PdfPTable se creeaza pe baza un select \* din tabelul Client.O sa avem nevoie si de functia findClientByName() care returneaza id-ul unui client cu numele transmis ca parametru deoarece in momentul creeari unei comenzi, noi ii dam numele si avem nevoie de id. Functia selectClientNoPrint() am implementat-o pentru functia deleteClient() astfel incat in momentul in care acest selectClientNoPrind() returneaza 0(nu mai avem clienti in baza de date) sa se reseteze incrementul la 0, in cazul in care stergem pe parcurs toti clientii sa inceapa inserarea de la 1. Iar functia delete, sterge un client din baza de date. Functia findClientById() o sa ne returneze numele clientului de la id-ul din parametru, avem nevoie de aceasta functie pentru a scris in PDF numele, nu id-ul clientului inserat.

**4.3.2 Clasa ProductDAO**

Deasemenea, am salvat query-urile in cate o variabila private final static String.Prima functie implementata e updateProduct() in care ii fac update unui produs, inlcuint fiecare camp cu campurile primite ca parametru. Mai apoi, am facut inserarea unui produs in baza de date prin functia insertProduct(). Pentr a verifica daca un produs poate sa fie introdus in comanda, am folosit functia findProductQuantById() care ne returneaza cantitatea unui produs cu un id anume, trimis ca parametru.Ca si in cazul clientilor, avem functia findProductByName() care ne returneaza id-ul unui produs cu un nume anume ca sa putem realiza comanda. Deasemenea, ca si in cazul clasei Client, am implementat selectProductNoPrint() pentru a stii in momentul in care apelam deleteProduct() daca mai sunt sau nu produse in baza de data, iar daca nu mai sunt sa resetam incrementul la 0. Functia de afisare, selectPRoduct() returneaza deasemenea un PdfPTable pe care o sa il folosim in momentul generarii raportului produselor. Functia findProdById() o sa ne returneze numele produsului de la id-ul din parametru, avem nevoie de aceasta functie pentru a scris in PDF numele, nu id-ul produsului inserat.

**4.3.3 Clasa OrderDAO**

Aceasta clasa este capabila sa introduca o comandata noua in tabela Order, de a selecta comenzile si de a selecta comenzile in cazul erorii. Am facut 2 functii de selectare a comenzilor, deoarece in PDF, inc cazul erorii o sa fie afisat un mesaj special. Verificarea daca o comanda se poate efectua sau nu o sa se faca in clasa OrderBLL care o sa se foloseasca de aceasta clasa pentru verificarile aferente. Ca variabile, avem aceleasi String-uri care reprezinta query-urile.

**4.4 Pachetul BLL**

**4.4.1 Clasa ClientBLL**

In aceasta clasa doar am facut functii care returneaza rezultatul celorlalte functii implementate in pachetul DAO.

**4.4.2 Clasa ProductBLL**

Functia insertProduct() o sa verifice daca produsul pe care vrem sa il inseram este deja inserat in baza de date, daca este inserat o sa modificam eventual pretul si cantitatea.

Restul functiilor sunt aceleasi ca si in pachetul DAO.

**4.4.3 Clasa OrderBLL**

Legat de creearea unei noi comenzi, o sa se verifice daca cantitatea pe care clientul doreste sa o achizitioneze este in stoc, daca este o sa facem comanda, daca nu, o sa returnam 0, iar pe baza acestuia o sa fie generat un pdf de eroare.

**4.5 Pachetul Presentation**

**4.5.1 Clasa FileParser**

Pentru inceput, am declarat o variabila de tip File in care am deschis fisierul de intrare din care o sa citim si o sa „parsam” informatiile.

Functia parse() face „treaba”, aceasta verifica pe rand daca primul cuvant este „insert”, „delete”,”report” sau „order” iar pe baza acestei verificari apeleaza una din functiile implementate.

Functia processInsert() primeste ca parametru un string S(care stim ca incepe cu Insert) si extrage pe rand informatiile. Adica, daca este cazul inserarii unui client, sau al unui produs. In cazul unui client tocanizam sirul si extragem numele, respectiv adresa, dupa care instantiem un nou Client avand numele si adresa acestuia. In final, pur si simplu inseram Clientul si afisam cu ce id a fost acesta inserat. In cazul unui produs, se procedeaza la fel, doar ca este extras din sir numele, cantitatea si pretul produsului. Este deasemanea instatiat un nou produs si inserat in baza de date.

Functia processDelete() primeste deasemenea un String s ca parametru si verifica daca este cazul sa eliminam un client sau un produs. Se extrag informatiile pentru fiecare caz si se sterge clientul/produsul in functie de cuvantul ce urmeaza dupa „Delete” in string-ul citit din fisier.

Functia processOrder() extrage informatiile din string, dupa care verifica daca se poate face noua comanda(pe baza conditiei de existenta a produselor in stoc), daca se poate face comanda, se face, daca nu, o sa generam un PDF de eroare cu elementele trimise in string-ul parametru.

In final, functia processRaport() de creeaza cate un PDF pentru fiecare caz de raport dorit, client, order sau product. Functiile de generare PDF sunt implementate in clasa ce urmeaza.

**4.5.2 Clasa GeneratePDF**

Dupa cum ii spune si numele, aceasta clasa genereaza PDF-urile necesare pe baza comenzilor furnizate de functiile din FileParser.

In cazul functilor generateClient()/generateOrder()/generateProduct() se genereaza cate un PDF sub forma de tabel in care sunt inserate toate datele din tabelele respective. Functiile primesc ca parametru un contor luat de mine pentur a stii PDF-ul cu ce numar urmeaza sa fie creeat, pentru a face cate un PDF pentru fiecare raport in parte.

Functia putin mai speciala este generateErrorOrder() primeste ca parametru acel contro si un Tabel t care contine informatii cu comenzile care nu au putut fi inserate si motivul pentru care inserarea a esuat.

Fiecare clasa isi creeaza propriul PDF cu nume diferit(generat pe baza contorului) iar aceste PDF-uri sunt vizibile in folderul temei.

**4.6 Pachetul Start**

**4.6.1 Clasa Main**

Instantiem un FileParser care face toata treaba, isi cheama metodele necesare pentru obtinerea rezultatelor dorite.

**5.Rezultate**

Rezultatele programului sunt foarte usor vizibile in PDF-urile generate, astfel pe baza comenzilor din fisierul „text.txt” o sa fie generate rezultatele in cate un pdf. Pentru fiecare Report o sa fie vizibile datele cerute, deasemenea in cazul erorii de inserare a unei comenzi o sa fie generat un pdf de eroare.

**6. Concluzii**

Avand in vedere ca in trecut am mai realizat un proiect in care am lucrat cu propria baza de date si cu selecturi, pot spune ca aceasta tema m-a ajutat sa ma perfectionez si sa aprofundez. Am folosit acest nou concept in care am legat pachetele intre ele + generarea de PDF uri. Am invatat aceste 2 lucruri noi. Am implementat tema destul de basic, avand doar 3 tables in baza de date, nefiind foarte eficient in cazul comenzilor, insa asa functioneaza perfect.

Ca extinere ulterioara ar putea fi introduse in baza de date imagini pentru produse, deasemenea pentru clienti. Pentru clienti s-ar putea face o logare, mai apoi putand doar sa comande lucruri. Deasemenea poate fi implementata si o interfata grafica in care utilizatorul sa fie liber in a vedea toate produsele din baza de date, comenzile si de a efectua operatii. In cazul administratorului, acesta sa poate introduce produse si de a le modifica.

**7.Bibliografie**

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

<http://tutorials.jenkov.com/java-reflection/index.html>

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

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

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

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

https://app.diagrams.net