Documentatie

-Sistem de gestionare a unei baze de date-

Nume: Balint

Prenume: Simina

Grupa: 30226

**Cuprins**

1. Obiective si cerinte . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
2. Analiza Problemei . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
3. Solutia aleasa si metode de implementare . . . . . . . . . . . . . . . . . . . . . .
4. Modalitati de utilizare . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
5. Diagrama UML si resurse. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
6. Dezvoltari ulterioare . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
7. Bibliografie . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . .. . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**1. Obiective si cerinte**

In acest Proiect se doreste realizarea unui sistem de gestionare a bazelor de date. Aceasta aplicatie ar trebui sa proceseze comenzile plasate de mai multi clienti spre un depozit. Bazele de date relationale sunt folosite pentru a stoca produsele, pe care le dispune acest depozit, clientii si comenzile trimise de catre acestia.

In principiu pentru realizarea unei astfel de aplicatie este necesara folosirea, cel putin, a urmatoarelor clase:

Clase Model - care reprezinta modele de baza ale aplicatiei.

Clase Business Logic - care contin logica aplicatiei.

Clase de prezentare - care contin interfata grafica.

Clase pentru accesul la date - unde se realizeaza accesul propiu zis al bazei de date.

Programul in sine nu este unul complicat, mai ales avand la dispozitie toate resursele necesare, iar obiectivul acestui proiect este acela de a intelege mult mai bine conectarea dintre un sistem de gestiune de baze de date si un mediu de programare orientata pe obiecte. Abilitatea unei clase Java de a examina si manipula datele chiar din interiorul sau, nu pare sa sune a a fi ceva extrem de complicat sau elaborat, dar in alte limbaje de programare aceasta trasatura lipseste sau chiar nu exista. De exemplu, intr-un program care utilizeaza limbaje precum: Pascal, C, C++ este imposibila obtinerea unor informatii referitoare la functile sau procedurile definite in interiorul programului.

**2. Analiza Problemei**

Pentru a putea proiecta aplicatia descrisa mai sus, trebuie mai intai sa intelegem ce inseamna cu adevarat o baza de date. O **bază de date**, uneori numita și **banca de date**, reprezinta o modalitate de stocare a unor informatii si date pe un suport extern, cu posibilitatea extinderii usoare si a regasirii rapide a acestora. La prima vedere sarcina poate parea banala. Totusi, în conditiile în care este vorba de a lucra cu milioane de elemente, fiecare putand consta din cantitati de date care trebuie accesate simultan prin Internet de catre mii de utilizatori raspanditi pe intreg globul; si în conditiile cand disponibilitatea aplicatiei si datelor trebuie sa fie permanenta, solutiile bune nu sunt de loc simple.

Elementul de baza al unei **BD** este tabelul. Un tabel la randul lui este alcatuit din linii si coloane, fiecare coloana avand un nume specific. Operatiile de baza care se pot realiza intr-un tabel, si cele pe care le-am descris si eu in proiectul meu sunt urmatoarele:

* + Insert - aceasta operatie va adauga un anumi rand nou in tabelul corespunzator;
  + Delete - aceasta operatie va sterge un anumit rand nou in tabelul corespunzator;
  + Update - aceasta operatie va modifica un anumit rand specificat cu valorile noi trimise ca parametri;
  + Find - acesta operatie va returna coloana dorita din tabel;
  + Select all - aceasta operatie va returna toate datele tabelului.

O baza de date functionala ar trebui sa implementeze si sa execute corect fiecare dintre aceste operatii. Baza de date in care atribuita pentru aplicatie este MySql Workbench. In plus fata de operatiile amintite mai sus, va trebui sa implementam metode pentru generarea unei facturi pentru fiecare comanda acceptata.

**3. Solutia aleasa si metode de implementare**

Solutia aleasa este una simpla si modulara. Am pornit rezolvarea cerinteai prin realizand cea mai evidenta problema a acesteia si anume: conectarea editorului Eclipse la baza de date. Acest lucru a fost posibil prin descarcarea unui jar compatibil cu versiunea MySql folosita si prin adaugarea acestuia in proiect. Dupa care am relizat conexiunea propiu zisa prin intermediul Clasei ConnectionFactory unde am importat biblioteca java.sql.

Pentru a implementa operatiile fundamentale pentru a lucra intr-un tabel am folosit MySql Statement. Clasele java vor fi stocate in pachete diferite pentru a mentine o logica ordonata a programului, realizand astfel o programare modulara. Fiecare tabel din baza de date va avea drept corespondent o clasa java, in acest caz cu acelasi nume, iar fiecare coloana cate un atribut in aceste clase.

**4. Modalitati de utilizare**

Fiecare utilizator are la dispozitie mai multe optiuni pe care le poate alege oricand. In tabele Client, Product, Order acesta poate sa introduca date sau sa le stearga, de asemenea daca acesta doreste poate si sa modifice valorile din tabele, iar la nevoie, el poate gasi anumite element. Pentru a realiza toate acestea utilizatorul trebuie sa scrie toate comenzile pe cate o linie separata, una dupa alta.

In cazul in care utilizatorul doreste sa creeze o comanda, programul este gandit in asa fel incat daca, comanda este plasata cu succes, iar stocurile sunt disponibile, se va returna o factura pe numele clientului in format PDF, dar in cazul in care stocul unui produs pe care doreste un client sa il comande nu dispune de cantitatea necesara, va returrna un fisier PDF in care se explica faptul ca, comanda nu poate fi plasata deaoarece nu mai sunt produse disponibile.

Insert client: Ion Popescu, Bucuresti

Insert client: Luca George, Bucuresti

Report client

Insert client: Sandu Vasile, Cluj-Napoca

Report client

Delete client: Ion Popescu, Bucuresti

Report client

Insert product: apple, 20, 1

Insert product: peach, 50, 2

Insert product: apple, 20, 1

Report product

Delete Product: peach

Insert product: orange, 40, 1.5

Insert product: lemon, 70, 2

Report product

Order: Luca George, apple, 5

Order: Luca George, lemon, 5

Order: Sandu Vasile, apple, 100

Report client

Report order

Report product

**5. Diagrama UML si resurse**

La fel cum am specificat si mai sus aplicatia este compusa din 4 pachete: Model, DataAcces, Presentation, Business.

Fiecare pachet este este alcatuit din mai multe clase:

-pachetul Model contine clasele Client, Product, Order in continutul carora se afla cate un constructor, dar si gettere si settere.

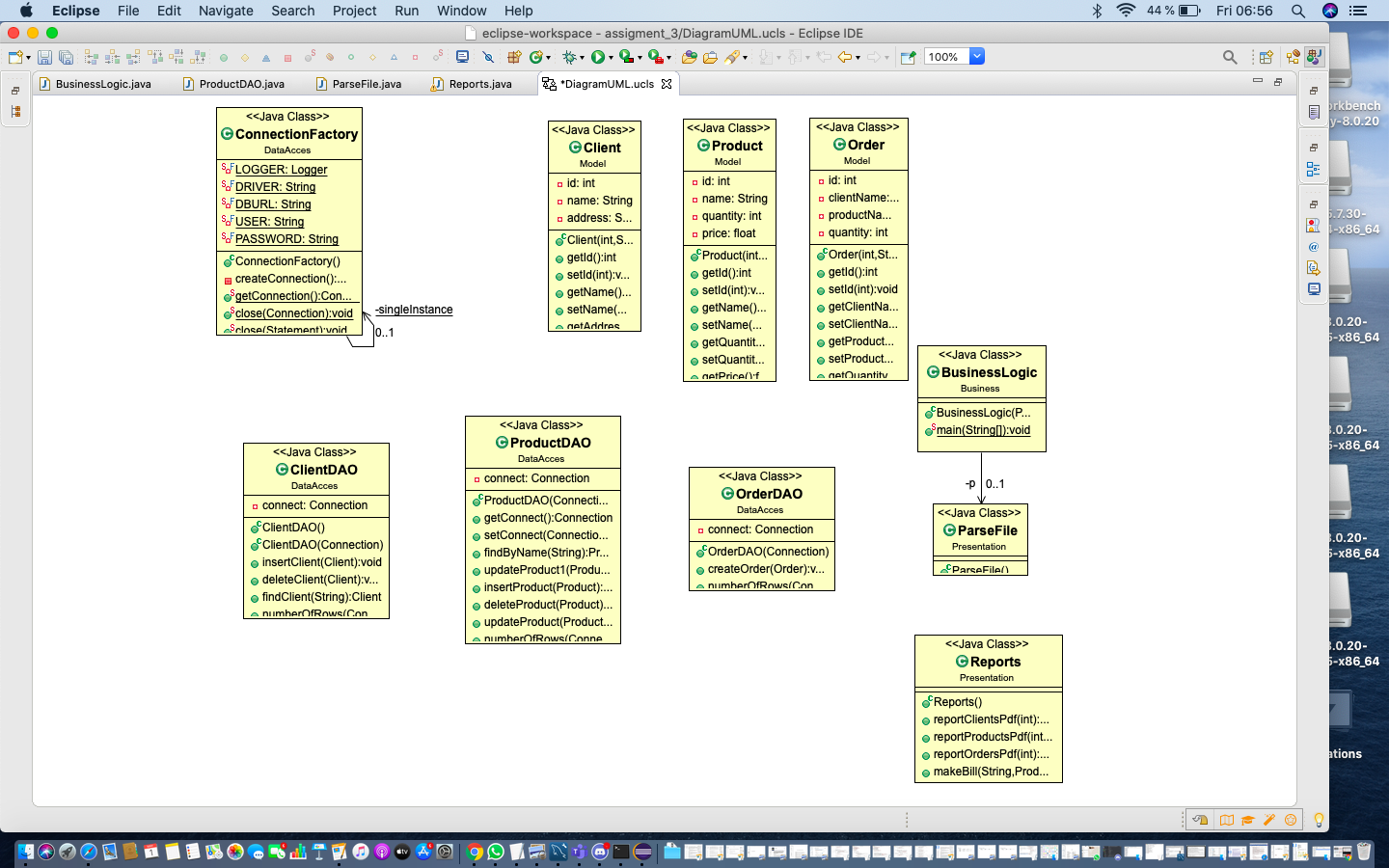
-pachetul DataAcces contine clasele ConnectionFactory (despre care am vorbit la punctele anterioare), ClientDAO, ProductDAO, respectiv OrderDAO; in clasele DAO descriu toate mtodele care au legatura cu operatiile fundamentale.

-pachetul Presentation are in componenta sa 2 clase: Repot si ParseFile. Clasa Report contine toate metodelele de generare a raporturilor PDF; pentru a realiza aceste raporturi a fost nevoie de importarea unui nou jar, care ne permite folosirea bibliotecii iText. Clasa ParseFile contine o metoda optimizata de parsare a unui fisier de intrare, orice fel de fisier de intrare, nu doar cel specificat in cerinta problemei.

-pachetul Business contine clasa BusinessLogic in care se realizeaza apelul functie readInstruction.

Dupa ce tot programul a fost implementat corect s-au generat javaDoc- uri si un jar file pentru a rula executia programului din terminal.

Diagrama UMl:



**6. Dezvoltari ulterioare**

Dupa cum am mai mentionat metoda de implementare aleasa este una simpla, asa ca fara dar si poate, aceasta se poate imbunattii.

Voi prezenta in continuare cateva dezavantaje ale acestei metode si posibile solutii pentru acestea:

- potentiale dezavantaje a utilizarii DAO ar fi acestea ca se foloseste cod duplicat si inversiunea abstracta., dar si ceea ce numin “leaky abstraction”;

leaky abstraction= o abstractie care permite patrunderea unor detalii care nu ar trebui sa apara;

- cand o aplicatie utilizeaza mai multe conexiuni de tip DAO, se pot repeta aceleasi comenzi de prea multe ori;

\*o solutie la acesta problema : implementarea unui DAO generic care sa se ocupe de toate operatiile comune.

-fiecare schimbare in interfata necesita editarea nu doar a unei implementari, ci a mai multor implementari in mai multe clase.

Se poate spune faptul ca aceasta problema poate fi imbunatatita si prin complexitate. Baza noastra de date ar putea sa contina mai multe tabele intermediare, iar operatiile pe tabele sa fie mai elaborate.

**7.Bibliografie**

<https://www.javatpoint.com/steps-to-connect-to-the-database-in-java>

<https://www.baeldung.com/simplifying-the-data-access-layer-with-spring-and-java-generics>

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

https://www.wikipedia.org/