**Order Management**

**Assignment 3**

**Bilc Sergiu Ion**

**30224**

**An: II**

**Cuprins**

1. Obiectivul temei
2. Analiza problemei
3. Proiectare
4. Implementare
5. Concluzii
6. Bibliografie

1.Obiectiul temei

**Cerinta**: Consider an application OrderManagement for processing customer orders for a warehouse.

**Obiective:** Se doreste crearea unei aplicatii Java, care sa poata fi utilizata intr-un mod cat mai eficient si mai placut pentru rezolvarea unei chestiuni des intalnite in ziua de astazi, si anume gestionarea unei firme de gestiune a anumitor produse. Aplicatia ar trebui sa permita adaugarea de noi produse, de noi clienti, dar si crearea de comenzi pentru fiecare client. Se doreste crearea unui noi comenzi, ceea ce consta in selectarea unui produs existent, selectectrarea unui client existent, inserarea unei cantitati dorite, verificarea daca aceasta cantitate exista, iar daca aceasta conditie este indeplinita, comanda este finalizata, un bon fiscal fiind generat pentru clientul care a cerut aceasta achizitie.

Din punct de vedere al functionalitatii, se doreste utilizarea unei baze de date pentru a stoca informatiile necesare sistemului nostru de gestiune a magazinului, crearea unei aplicatii care sa acceseze, stocheze si afiseze intr-un mod cat mai eficient datele stocate in baza de date, utlizand in acelasi timp noi tehnici de programare mai avasnate, cum ar fi reflectia. De asemenea, se doreste ca aplicatia sa fie aranjata in patru pachete, numite si layere, acestea fiind: model layer, reprezentand modelele de date ale aplicatiei, business logic layer, continand logica aplicatiei, data access layer, referindu-se la clasele prin care se acceseaza baza de date, si presentation layer, ce va contine interfata grafica.

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

2. 1. Analiza problemei

Magazinele, impreuna cu gestiunea intr-un mod cat mai eficient al acestora, sunt o problema constanta pentru oameni inca de la aparitia comertului, pana in zilele noastre, necesitatea aceasta fiind din ce in ce mai mare in zilele noastre, deoarece magazinele ajung sa comercializeze un numar extraordinar de mare de produse. O buna parte din populatia globului se ocupa cu comertul, gestionarea magazinelor si cu comertul, in timp ce cu totii suntem implicati, in fiecare zi, in cel putin o tranzactie cu rol comercial. Din acest motiv, odata cu dezvoltarea tehnologica, se incearca computerizarea si eficientizarea acestui proces de comercializare a tutoror produselor. Astfel, au aparut o serie ampla de aplicatii cu rol de gestionare a stocului unui magazin si de inregistrare a tuturor tranzactilor facute, impreuna cu stocarea, iar mai apoi accesarea rapida a informatilor legate de clienti, produse sau comenzile livrate sau in curs de livrare.

Tema aceasta doreste sa vina tocmai in directia rezolvarii acestei probleme, simuland gestiunea unui magazin, aceasta aplicatie putand fi folosita, in mod fictiv, de un angajat al unei companii care foloseste sistemul implementat . Precum am spus anterior, se doreste accesul rapid la toate informatiile stocate de baza noastra de date, cum ar fi lista tuturor clientilor, impreuna cu toate informatiile stocate privindui pe acestia, la lista tuturor produselor, impreuna informatile despre aceste produse , dar mai ales posibilitatea modifica si de a sterge in mod rapid informatii despre acestia, utilitatea pentru operator fiind o prioritate . Deasemenea, se doreste crearea unui meniu de gestionare a comenzilor din sistem , ceea ce reprezinta vizualizarea tuturor comenzilor din sistem , sau adaugarea unei noi comenzi.

2.2. Asumptii si decizii de proiectare

Asumpțiile au rolul de a simplifica complexitatea problemei și implicit a soluției acesteia. O asumpție este constituită de presupunerea faptului că limitele de intervalelor trebuie să fie introduse corespunzător: limita inferioară, apoi limita superioară.

Asumptiile luate de mine in cadrul acestei teme sunt: o comanda poate fi realizata doar asupra unui singur client, ce va comanda un singur produs, iar odata ce o comanda a fost realizata, aceasta nu mai poate fi stearsa din system prin metode uzuale, disponibile utilizatorului usual, din motive de business si pentru a mentine datele stocate in cazul unei verificari din partea unei institutii de natura fiscala, acestea neputand fi sterse la voia oricui. Stergerea comenzilor poate fi doar de catre administrator, prin acesul bazei de date. De asemenea, modificarea unei comenzi pote fi efectuata doar de catre un administrator.

2.3. Modelare

In ceea ce priveste modelarea problemei, metoda de organizare logic, clasele, metodele si datele trebuie alese astfel incat aplicatia sa fie cat mai bine segmantata, pana in momentul in care fiecare componenta logica sa fie separate de celelalte componente cu care nu poseda o functionalitate comuna.

Prin urmare, am ales sa sa segmentez proiectul in 4 straturi, sau layere: stratul de modele, care o sa fie format din clase cu rol de sintetizare a elementelor pe care dorim sa le modelam, cu care dorim sa lucram. Acestea o sa fie Clientii, Produsele, si Comenzile. Tot in aceasta sectiune ar putea sa intre si Factura, mai exact documentul cu datele legate de o comanda proaspat incheiata; un alt strat este stratul de acces la baza de date, ce implica atat o functionare comuna pentru toate clasele ce formeaza acest layer(ClientDAO, ProductDAO, OrderDAO), cum ar fi selectarea tuturor elemtentelor din oricare dintre aceste tabele, dar si o functionare particulara pentru fiecare dintre aceste clase; un alt strat este stratul de business logic al aplicatiei, ce va indica modul in care aceasta va functiona, iar ultimul strat este cel de prezentare al aplicatiei, stratul interfetei grafice, fiecare layer accesand pentru a putea functiona, layerul precedent.

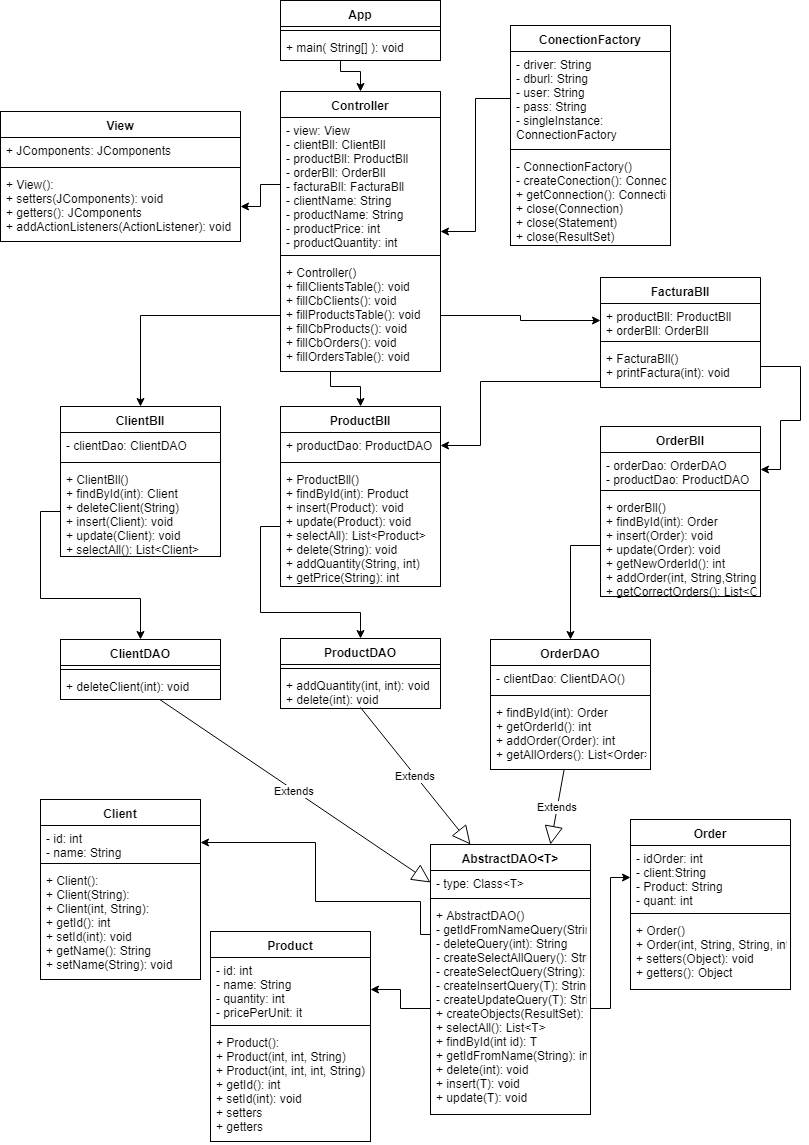
3. Proiectare

3.1. Decizii de proiectare

Am decizia sa minimezez pe cat posibil datele pe care utilizatorul este nevoit sa le introduca, astfel incat acesta are de introdus in principal date legate dnoi produse sau de noi clienti pe care acesta doreste sa ii introduca in sistem, in timp de datele care deja se afla in bazele de date o sa ii fie usor de accesat. De exemplu, in cazul in care utilizatorul doreste sa stearga un client, numele tuturor clientilor o sa fie afisat intr-un JcomboBox, in dupa ce utilizatorul va alege clientul dorit, acesta o sa poata sters din baza de date fara probleme.

3.2. Diagrama UML

Diagrama UML poate fi analizata in Anexa 1.



3.3. Structuri de date

Pe langa tipurile de date primitive, am folosite si o serie de colectii, in special ArrayList-ul, pentru stocarea datelor extrase din tabelele bazei de date si pentru generarea tabelelor prezente in interfata grafica.

3.4. Proiectare clase

Layerul models:

* Clasa Client
  + Este o clasa ce formeaza nivelul cel mai de jos al aplicatiei, clasa ce reprezinta modelul din spatele clientului din viata reala.
  + Contine un id si un nume pentru fiecare client
* Clasa Product
  + Este o clasa ce formeaza nivelul cel mai de jos al aplicatiei, reprezentand modelul din spatele produselor din viata reala.
  + Este formata dintr-un id, un nume, un pret si o cantitate existenta in momentul de fata in magazin.
* Clasa Order
  + Este o clasa ce formeaza nivelul cel mai de jos al aplicatiei, reprezentand modelul din spatele comenzilor din viata reala, componenta ce va face legatura dintre clienti si produse.
  + Aceasta clasa este formata din un idOrder, un nume al clientului/id-ul, un id pentru produs si cantitatea din produsul respectiv care se doreste a fi comandata.

Layerul data access:

* Clasa ConnectionFactory:
  + Clasa in care se realizeaza literlamente legatura dintre aplicatia noastra si baza de date
  + Aceasta conexiune va fi formata dintr-un url pentru baza de date, de numele unui utilizator si de parola acestuia pentru baza de date, impreuna cu Singletonul care va contine obiectul de tipul Connection ce va transmis mai departe aplicatiei.
* Clasa AbstractDAO:
  + Clasa ce face legatura dintre modelele din baza de date cu clasele model din aplicatia noastra, astfel incat, in aceasta clasa extragem, modificam sau adaugam informatii la baza noastra de date, utilizandu-ne de legatura dintre java si baza de date si prin executarea de query-uri. Executa operatii de baza, comune pentru toate modelele: selectAll, insert, update, delete, findById, etc.
* Clasa ClientDAO:
  + Clasa ce face legatura dintre clientul din baza de date cu clasa Client din aplicatia noastra, astfel incat, in aceasta clasa extragem, modificam sau adaugam informatii la baza noastra de date. Fata de clasa AbstractDAO, care contine functionalitatile comune dintre clienti si celelalte modele, aceasta clasa va contine si functionalitatea particulara a clientilor.
* Clasa ProductDAO:
  + Clasa ce face legatura dintre produsul din baza de date cu clasa Product din aplicatia noastra, astfel incat, in aceasta clasa extragem, modificam sau adaugam informatii la baza noastra de date. Fata de clasa AbstractDAO, care contine functionalitatile comune dintre produse si celelalte modele, aceasta clasa va contine si functionalitatea particulara a produselor.
* Clasa OrderDAO:
  + Clasa ce face legatura dintre comenzile din baza de date si clasa Order din aplicatia noastra.

Layerul business logic:

* Clasa ClientBll:
  + Clasa ce va realiza executia logicii din spatele clientilor, utilizand clasa ClientDAO, care face legatura dintre clienti si baza de date.
* Clasa ProductBll:
  + Clasa ce va realiza executia logicii din spatele produselor, utilizand clasa ProductDAO, care face legatura dintre produse si baza de date.
* Clasa OrderBll:
  + Clasa ce va realiza executia logicii din spatele order, utilizand clasa OrderDAO, care face legatura dintre comenzi si baza de date.

Layerul de prezentare:

* Clasa View:
  + Reprezinta interfata aplicatiei
* Clasa Controller:
  + Clasa in care este realizata functionarea aplicatiei, in care sunt utilizate componentele din layerul business logic, informatile extrase din acestea fiind afisate in interfata grafica.

4. Implementare

In implementarea acestei teme am incercat sa ma folosesc in egala masura si de functionalitatea bazei de date, astfel incat am preferat sa folosesc o serie de proceduri stocate pentru a minimiza efortul de a crea query-uri in mod explicit in interiorul aplicatiei mele. Acestea sunt: addOrder, getOrders, addCLient, addProduct.

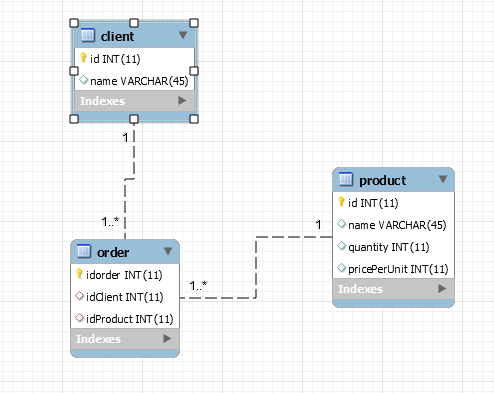
Cel mai important aspect legat de aceasta tema este folosirea reflectiei in analizarea claselor si a bazei de date. Utilizand acest procedeu, am reusit sa creez o clasa ce descrie comportamentul comun al tuturor modelelor din aceasta aplicatie, si naume, AbstractDAO. In aceasta clasa am implementat o metoda denumita createObjects(), care preia ca parametru ResultSet-ul obinut in urma executiei unui query( select \* from client), iar pentru fiecare linie din acesta, metoda creeaza un obiect nou de tipul clasei pentru care se doreste executarea acestui query(ex: Clasa Client), se analizeaza field-urile acesteia, iar pentru fiecare field, se incearca obtinerea informatiei din acest ResultSet, informatie care va fi mai apoi setata in obiectul proaspat creat. Fiecare obiect este adaugat la o lista de obiecte de tipul clasei care apeleaza in acel moment metoda, iar aceasta lista este mai apoi returnata metodei respective. Celelalte functionalitati comune sunt:

* Metoda de selectAll(): aceasta este o metoda care executa un query de selectare particularizat a tuturor datelor despre un anumit model, creaza o lista cu rezultatul aceleui Query, iar mai apoi returneaza lista de obiecte amintita mai sus.
* Metoda findByid(int): primeste ca parametru id-ul unui obiect din baza de date, iar in functie de tipului obiectului pentru care sa doreste cautarea, aceasta metoda creeaza un query personalizat de returnare e tuturor datelor despre clientul, produsul sau comanda a carui id este cel dat ca parametru. Metoda va returna obiectul ce va contine toate informatiile despre obiectul cu id-ul respectiv.
* Metoda getIdFromName(String): aceasta metoda primeste ca parametru numele unui obiect, cauta in baza de date elementul care contine acest nume, si returneaza id-ul acestuia.
* Metoda delete(int): aceasta metoda primeste ca parametru id-ul unui obiect, pe care il sterge din baza de date
* Metoda insert(T): primeste ca parametru un obiect care se doreste a fi insertat in tabela din baza de date corespunzatoare tipului corespunzator
* Metoda update(T): primeste ca parametru un obiect si face modificarile in baza de date a elementului care are acelasi id cu obiectul transferat ca parametru metodei.

In ceea ce priveste functionalitatile particulare ale claselor, am decis sa implementez mai multe metode care totusi ar putea fi considerate cu functionalitate comuna, cum ar fi stergerea unui produs sau a unui client. Acest lucru poate reprezenta o problema din cauza forreign key-urilor existente in baza de date pentru aceste tabele, astfel am decis sa ma folosesc de procedurile stocate amintite mai sus si sa creez o funcitonalitate diferita pentru cele doua mdolee. Acest lucru a fost realizat in clasele clientDAO si ProductDAO. Clasa ProductDAO contine metodele: addQuantity(int, int), care updateaza cantitatea existenta in stoc pentru produsul cu id-ul transmis ca parametru, si metoda Delete(int), care sterge produsul cu id-ul transmis ca parametru. Clasa ClientDAO contine metoda de deleteClient(int), in timp ce clasa OrderDAO contine metoda de gasirea a unei comenzi dupa id( findById() ), metoda de obtinerea unui nou id care sa poata fi folosit pentru o comanda nou ( getOrderId() ), metoda addOrder(Order) de adaugare unei noi comenzi si metoda getAllOrders care returneaza toate comenzile existente in baza de date.

Toate informatile necesare pentru operatiile de sterge, modificare si adaugare a unei noi comnezi au fost introdue intr-un JComboBox, astfel incat utiliator nu va fi nevoit sa le introduca el.

Baza de date:



5. Concluzii si dezvoltari ulterioare

Din această temă am aprofundat lucrul cu baze de date în Java. Din punctul meu de vedere, proiectul acesta este unul foarte folositor, mai ales în privința dezvoltării noastre, a studenților, ca viitori programatori, existand o mare aplicabilitate practic a unei aplicatii care sa lucreze in acelasi timp si cu o baza de date.

Ca si dezvoltari ulterioare pot fi luate in considerare urmatoarele posibilitati:

* Introducerea mai multor detalii despre produse, clienti sau comenzi
* Introducerea unui sistem de utilizator, astfel incat fiecare utilizator sa poata efectua doar anumite operatii
* Introducerea posibilitatii ca o singura comanda sa poata contine mai multe produse.

6. Bibliografie

[1] [http://coned.utcluj.ro/~salomie/PT\_Lic/4\_Lab/](http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/%20)