**Order Management**

**Assignment 3**

Nume: Boghian Ciprian

An II

Gr 30224

**Cuprins**

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

**1.Obiectuvul Temei**

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

**Obiectivul problemei**: Se doreste realizarea unei aplicatii Java, care sa fie utilizata intr-un mod cat mai bun posibil, eificient si placut din punctul de vedere al utilizatorului in vederea gestionareii unei firme cu diferite produse, un lucru foarte normal in ziua de azi. 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 functional, este nevoie si se doreste folosirea unei baze de date un se vor stoca informatii necesare sistemului nostru de gestiuni a magazinului, crerea unei aplicatii care sa acceseze, stocheze si afiseze in mod cat raoid si eficientdatele stocate in baza de date, dar in acelasi timp se vor folosi tehnici noi de programare mai avansare, cum ar fi reflectia.

2.Analiza problemei:

Depozitele, magazinele impreuna cu celelate unitati de comert au nevoie de o metoda de gestionare si inventariere cat mai rapida si usoara pentru a face experineta clinetului cat mai placuta si pentru a avea o eficienta cat mai mare in preluarea si respectarea comenziulor. Aceasta aplicatie simuleaza introducerea de noi clienti , de un angajat, care va introduce toti clientii intr-o baza de date, a tuturor produselor care se afla in momentul de fata in inventarul magazinului si a comenzilor selectate de fiecare client in parte. 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.

Din principii de eficienta si de rapiditate trebuie sa facem o interfata cat mai prietenoasa si usor de folosit pentru orice angajat pentru ca aplicatia noastra sa ajunga l o eficienta maxina.

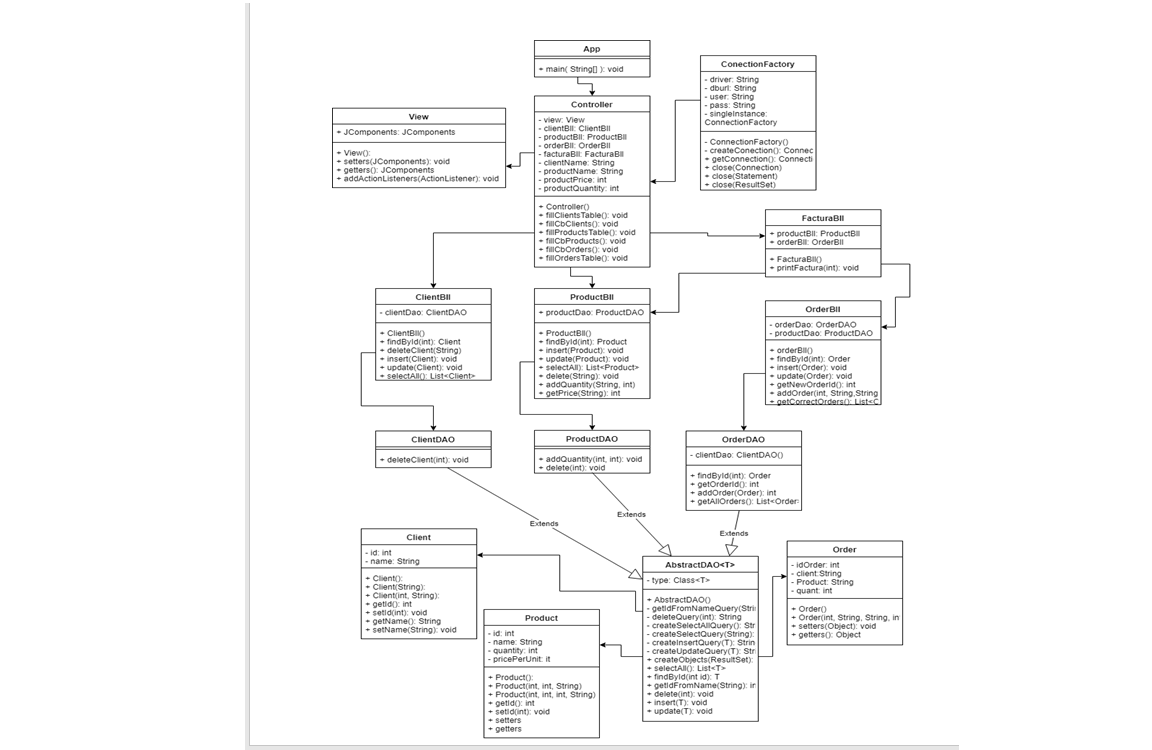
**3.Proiectare:**

Am decis ca o comanda poate fi realizata doar de un singur client asupra unui singur produs, iar odata ce comanda a fost realizata, aceasta sa nu mai poata fi stearsa din sitem prin metode uzuale, disponibile utilizatorului normal, din motive de gandire economica si management, pentru a detine toate datele in cazul unei verificari, sau a fiscalizarii finale lunare sau anuale. De asemenea, modificarile unei comenzi por fi efectuate doar de catre un administrator.

Din punct de vedere al modelariiam incercat ca apliatia mea sa fie cat mai bine segmentata, pana in momentul in care fiecare componenta logica sa fie separata 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.

In concluzia tuturor mentionate mai sus am decis sa minimizez pe cat posibil datele pe care utilizatorul este nevoit sa le introduca in interfata aplicatie, pentru a face un GUI cat mai sumpu de inteles astfel incat aceasta are de introdus in principal date legate de noi produse sau de noi clienti deoarece exista deja in siste, in baza de date, vechii clienti si vechile produse care vor putea fi foarte usor de accesat. De exemplu, in care in care se doreste plasarea unei noi comenzi cu produse si clienti existenti, se acceseza direct paretea de comenzi in care va exista 2 comboBoxuri in care sunt introdusi toti clientii si toate produsele existente, astfel ramanand sa se introduca doar cantitatea dorita.

Diagrama UML:



Pe langa modalitati de programare des intalnite, cel mai mult am folosit o serie de colentii, in special ArrayList-ul, pentru stocarea datelor extrase din tabela bazei de date si pentru afisarea tabelelor in interfata grafica, GUI.

Structurearea aplicatiei:

Clasa Client : contine id-ul si numele fiecarui client introdus in baza de date. Este clasa care sta la baza aplicatiei, si reprezinta modelul clientului din viata reala.

Clasa Product : contine idul, numele, pretul pe bucata si o cantitate existenta in acel moment in magazin. Toate aceste caracteristici se refera strict la un anumit produs. Este clasa care sta la baza aplicatiei, si reprezinta modelul produsului din viata reala.

Clasa Order: contine un idOrder, id-ul clientului, un id produs si cantitatea din produsul respectiv care se doreste a fi comandata. Aceasta clasa doreste simularea cat mai in detaliu a unei comenzi existe in ziua de azi in majoritatea magazinelor.

Clasa AbstractDAO: clasa ce va face o legatura dintre modelele din baza de date cu clasle model din aplicatie , astfel incat, in aceasta clasa extragem si sa 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

Clasa ClientDAO: face legatura dontre clientul din baza de date cu clasa Client din aplicatia noastra, iar in aceasta clasa noi reusim sa extragem si sa modificam sau adagam informatii la baza noastra de date. Diferenta intre aceasta clasa si clasa AbstractDAO, care contine functionalitati comune este ca aceasta clasa contine si functionalitatea principala a clientilor.

Clasa OrderDAO: face legatura intre comenzile din baza de date si clasa Order din aplicatie.

Clasa ProductDAO: face legatura dintre produsele din baza de date cu clasa Product din aplicatia noastra, iar tot in aceasta clasa extragem si modificam informatii la baza noastra de date asupra unui produs. Diferenta intre aceasta clasa si clasa AbstractDAO, care contine functionalitati comune este ca aceasta clasa contine si functionalitatea principala a produselor.

Clasa ClinetBll: realizeaza executiile logice din spatele clientilor, utilizand clasa ClientDAO, care face legatura cluentlui cu baza de date.

Clasa ProductBll: : realizeaza executiile logice din spatele produselor, utilizand clasa ProductDAO, care face legatura cluentlui cu baza de date.

Clasa OrderBll: realizeaza executiile logice din spatele comenzilor, utilizand clasa OrderDAO, care face legatura cluentlui cu baza de date.

**4.Implementare:**

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.

Acest mecanism ne este oferit de catre pachetul java.lang.reflect. Reflectia inseamna autoexaminare, adica permite determinarea structurii clasei. In limitele managerului de securitate putem afla metodele clasei, constructorii clasei si restul membrilor clasei. Cateodata putem sa modificam starea obiectlui prin apelul metodelor specifice sau putem construi obiecte noi . Mecanismul de reflectie este utilizat de componentele Java (Java beans) pentru determinarea  capabilitatilor obiectelor pe timpul executiei. Membrii clasei sunt atributele si metodele. Metodele se impart la randul lor in doua categorii, constructori si metode obisnuite.  Metoda returneaza o referinta la clasa corespunzatoare. Cu ajutorul acestei referinte putem apela metoda getMethod(). Aceasta metoda primeste ca prim argument numele metodei, iar al doilea argument este un array de tip Class specificand signatura metodei.  In exemplul de mai sus am apelat o metoda fara parametri, deci al doilea argument este un array vid. Daca am fi dorit sa apelam o metoda cu argumente, acest array ar fi trebuit initializat cu un array cu clasele argumentelor. Pentru tipuri primitive se utilizeaza o constructie Clasa.TYPE. De exemplu pentru tipul primitiv int se va utiliza Integer.TYPE. Fiecare tip primitiv are o clasa corespunzatoare in pachetul java.lang  Accesul metodelor se rezolva similar cu tehnica poinetrilor la functii in limbajul C standard. Clasa Method invoke() se utilizeaza pentru apelul unei metod cu argumente specificate. Exemplul urmator va utiliza aceasta facilitate prin apelul unei metode statice fara nici  un argument. Numele clasei si numele metodei se primesc de la linia de comanda.

Pentru a evita redundanta, clasele care reprezinta variante le acelorasi structuri abstracte de date sunt organizate in structuri numite ***ierarhii de clase***. ***Mostenirea*** este proprietatea prin care o clasa derivata dintr-o alta (numita acum superclasa) poseda toate atributele de stare si comportamentul superclasei. Avantajul unei asemenea organizari consta in aceea ca proiectantii pot stapini mai bine complexitatea unor sisteme soft pe de o parte iar pe de alta parte se creaza posibilitatea de a reutiliza componentele soft prin mostenire.

Clasa derivata dintr-o alta clasa (superclasa) va mosteni toate proprietatile acesteia la care putem adauga si altele sau, de ce nu, le putem restringe pe cele dintii. Restringerea sau modificarea unor metode mostenite poarta numle de rescriere.

POO, in general, pune un accent deosebit in reutilizarea codului si in dezvoltarea de elemente soft cu aplicabilitate cit se poate de generala. Mostenirea este o tehnica *de implementare* (si nu una de design cum ar parea) deosebit de utila din acest punct de vedere.

Prin mostenire intelegem proprietatea prin care instantele unei clase pot sa acceseze atit datele cit si comportamentul, adica metodele asociate cu clasa din care clasa instantiata a fost derivata. Aceasta din urma se mai numeste si superclasa. Mostenirea este totdeauna tranzitiva. Asa cum o clasa poate mosteni trasaturile de la superclasa din care este derivata, superclasa la rindul ei poate fi clasa derivata dintr-o alta superclasa. Cu alte cuvinte, daca electronul este o subclasa a clasei leptonilor iar clasa leptonilor este la rindul ei o sublclasa a clasei particule ideale, atunci electronul va mosteni proprietati (date si metode) atit de la clasa ParticulaIdeala cit si de la clasa Lepton.

Este evident ca atunci cind derivam o clasa dintr-o alta clasa, codul clasei din care facem derivarea nu mai trebuie rescris. (Implicatiile sunt deosebit de impotante nu numai sub aspectul economiei de timp si energie dar si prin aspectul cresterii lizibilitatii codului si scaderii costului de intretinere.) Aceleasi clase pot fi folosite in proiecte diferite, din clasele respective putindu-se deriva clasele de care este nevoie. Vom putea privi clasele ca pe niste componenete soft.

Faptul ca putem deriva mai multe clase dintr-o aceeasi superclasa ne asigura ca, cel putin in mare, comportamantul va fi acelasi pentru toate clasele. Interfetelevorfi similare pentru acel grup de obiect ceea ce implica faptul ca nu se ofera utilizatorului o colectie eclectica de obiecte care sunt aproape identice dar care interactioneaza diferit.

Proiectul a fost strucutrat folosind arhitecutra MVC ( Model – View – Controller ) si din acest motiv aplicatia a fost conceputa sub forma a mai multor pachete: Models, care reprezinta datele, obiectele, folosite de aplicatie: Clienti, ClientDAO, ClientBLL, View ( unde este implementata interfata grafica ) si Controllers( controlorul care preia intrarile de la utilizator si le transpune in modelele cu care lucram, mai apoi ocupandu-se si de operatiile effectuate ) .

**5.Concluzii si dezvoltari ulterioare:**

Datorita acestei teme am aprofundat si mi-am intarit lucrul cu baze de date in Java si MySql. Proiectul acesta te ajuta in dezvoltarea ulterioara in initirea noastra ca viitori programatori. Aceast proiect a necesitat cunostintele de la proiectul anterior si astfel s-au intarit si mai bine bazele noastre de cunostinte in domeniul POO folosint diferite colectii cum ar fi ArrayList-ul si folosirea diferitelor ustensile din Jframe, cum ar fi Table si modalitatea de umplere a acestuia folosind informatii din baza de date si crearea capetelor de tabel.

Ca modalitaiti de dezvolatare ulterioara, se pot introduce mai multe detali despre produse, clienti sau comenzi, se poate introduce un sistem de utilizatori, astfel incat sa se faca diferenta intre ei si fiecare tip sa aiba acces doar la un anumit set de operatii, Administrator, Client, Manager Comenzi etc, si nu un ultimul rand un client sa poata forma o comanda pentru mai multe produse din diferite categorii simultan.

**6.Bibliografie**

http://www.ms.sapientia.ro/~manyi/teaching/oop/oop\_romanian/curs16/curs16.html#Reflectie

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