

**FACULTATEA: Automatică şi Calculatoare**

**SPECIALIZAREA: Calculatoare şi tehnologia informaţiei**

**DISCIPLINA: Tehnici de Programare**

**APLICATIE BAZA DE DATE**

**Îndrumător Laborator: Realizator:**

Cristina Bianca Pop Oneţiu Alexandru Lucian

Grupa 30228

***Cuprins***

1.Cerinte Functionale

2. Analiza Problemei

3. Obiective

4. Proiectare

4.1 Diagrama de utilizare

4.2 Diagrama de clase

4.3 Diagrama de secventa

5. Implementare

6. Concluzii si Dezvoltari Ulterioare

7. Bibliografie

# Cerinte Functionale

**Cerinta:**

Propuneti, proiectati si implementati o palicatie de gestionare a datelor, capabila sa realizeze toate operatiile specifice administrarii unei baze de date : creare, update, selectie, stergere .

**Consideratii generale**

Problema propusa este una extrem de populara .Aplicatiile pentru managementul bazelor de date sunt realizate in general in Java , fiind folosite pe scara larga : aproape fiecare afacere are o baza de date pentru angajati , clienti , produse , etc. Privind problema din punctul de vedere a logicii folosite in spatele aplicatiei , aceasta este de dificultate medie . Provocarea vine defapt de la nevoia de a face clasele si metodele cat mai reutilizabile cu putinta . Asadar , aspectele tehnice sunt , de aceasta data , cele problematice data fiind abordarea ceimplica folosirea genericelor si a reflectiei .

Pentru a putea sa modelam aceasta problema , trebuie sa fie discutat layer –ul pe care aplicatia este bazata , altfel spus baza de date . Cerintele problemei impun folosirea unui numar de 3 tabele : Client , Product ( Produs ) , Order ( Comanda ) . Fiecare dintre aceste tabele contine campuri ce faciliteaza lucrul cu baza de date : id pentru clienti, comenzi si produse , cantitatea pentru fiecare produs , pret si asa mai departe .

Pentru implementarea propriu – zisa a problemei in Java , dat fiind ca aplicatia foloseste resursele unei aplicatii straine , este necasara o impartire pe layere a logicii si a entitatilor ce lucreaza impreuna pentru realizarea produsului final . Asadar , clasele sunt impartite in pachete , fiecare dintre acestea fiind responsabil pentru anumite operatii . Asadar, avem un pachet responsabil pentru stabilirea conexiunilor dintre baza de date si aplicatie , un altul pentru accesarea campurilor bazei de date .

Data access layer trimite informatii business logic layer-ului , care la randul sau acceseaza DAO, returnand o entitate cunoscuta . Aceste entitati reprezinta clase operationale care au ca si corespondent clientii, comenzile si produsele , adica principalii operanzi ai aplicatiei . Un alt layer este cel folosit pentru interfata grafica , implementata respectand principiile modelului MVC . Acesta rezulta intr-un nou pachet care izoleaza logica de functionalitate a programului , entitatile grafice si controllerii . Acest ultim layer , extrem de important este numit , sugestiv , presentation layer .

Printre cele mai importante parti ale aplicatiei , intalnim implementarile DAO ( Data Access Object ) , care sunt responsabile pentru a putea sa accesam obiectele bazei de date . Un DAO reprezinta un obiect care asigura o interfata abstracta pentru un obiect al bazei de date . Altfel spus , aplicand bazei de date interogari ( query-uri ) pentru obtinerea informatiilor necesare , DAO asigura cateva operatii specifice fara a expune detaliat rezultatul obtinui din baza de date . Acesta este transmis bll ( business logic layer ) care selecteaza din baza de date ceea ce se cere prin DAO .

Cazurile de utilizare fiind destul de clare , atentia ar trebui sa fie focusata pe erorile care pot sa apara accesand sau modificand baza de date , si situatiile care le genereaza . Spre exemplu , trebuie sa fim atenti sa nu stergem sau sa accesam un produs inexistent , sau sa cumparam mai multe produse decat poate sa suporte stocul . De asemenea , o atentie deosebite trebuie sa fie acordata in sincronizarea stergerior sau update-ului facute in aplicatia Java , respectiv in baza de date . Aceasta necesita un efort suplimentar . CCand un produs este cumparat , sau orice alta tranzactie este facuta , aplicatia ofera o chitanta in care sunt prevazute detalii referitoare la comanda . Totul este apoi trecut intr-un fisier text . .

# Analiza Problemei

Prin analiza problemei, ne referim la un prim set abstract de operații și proprietăți prin care încercăm să depistăm eventualele însușiri și comportamente ale proceselor necunoscute. Programarea orientată ne oferă aici un avantaj clar, tocmai fiindcă ea permite să taclăm problema de la un nivel superiror, fără a mai fi constrâși, într-o așa măsură, de caracteristicile tehnice.

Această strategie de conceptualizare, mai poartă numele și de bottom-up design. Este foarte avantajoasă din prisma găsirii componentelor constituente, deoarece pot fi găsite, relativ ușor, structuri cu o legătură directă în lumea reală( obiecte, acțiuni etc.). Din păcate această versatilitate vine cu prețul complexității, ea crescând spre măsură ce se avansează pe nivelele inferioare.

De cele mai multe ori se pornește de la specificația proiectului, căutându-se:

* Substantive, care devin eventuale clase candidat
* Verbe ce ar putea juca rolul metodelor din clasă.

# Obiective

Obiectivul principal al acestei teme este dezvoltarea unei aplicatii Java capabila sa realizeze toate operatiile specifice administrarii unei baze de date : creare, update, selectie, stergere . Baza de date este creata folosind MySQLWorkbench . Aplicatia trebuie sa permita unui user sa realizeze o comanda a unor produse alese din tabelul de produse al bazei de date , urmand ca la final sa primeasca o chitanta a comenzii .

Aplicatia urmareste sa ofere un model general de manageriere a unui sistem pentru procesarea comenzilor clietilor unui depozit . In cazul de fata, produsele apartin unei categorii generale : Product , insa acest model poate sa fie imbunatatit cu usurinta . Este utiizata structurarea aplicatiei “ pe straturi ” , cunoscuta sub numele de “ layering model ” . De asemenea , este folosita Reflectia , cu scopul de a obtine o buna intelegere a acestei tehnici , prin aplicarea ei . . . . . . . . . . . . . . . . . . . . . . . .

Aplicatia ofera in acelasi timp si o interfata grafica , menita sa faciliteze lucrul cu baza de date prin aplicatie . Aceasta este impartita in 2 parti , in functie de persoana care o acceseaza : administratorul ( admin ) , sau clientul ( user ) . In functie de persoana care acceseaza baza de date ( user / admin ) , aceasta ofera un set de comenzi : adaugare , stergere , update pentru datele din interiorul tabelelor bazei de date . Aceasta interfata este implementata folosind pachetul Swing , oferit de Eclipse IDE pentru Java .

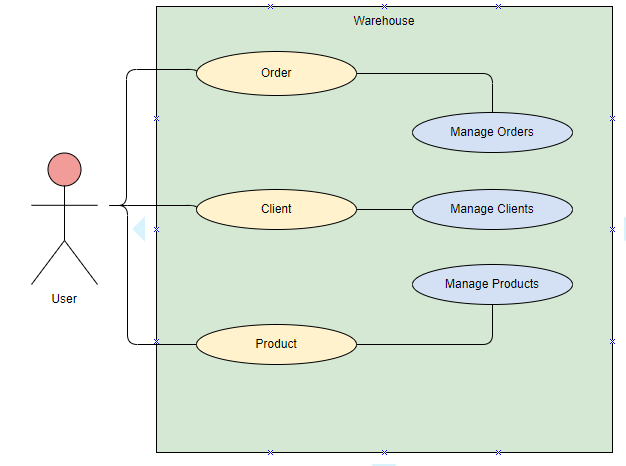
Toate aceste concepte trebuie sa fie implamentate in concordanta cu principiile paradigmei POO : incapsulare , abstractizare , mostenire si polimorfism .

# Proiectare

**4.1 Diagrama de utilizare**

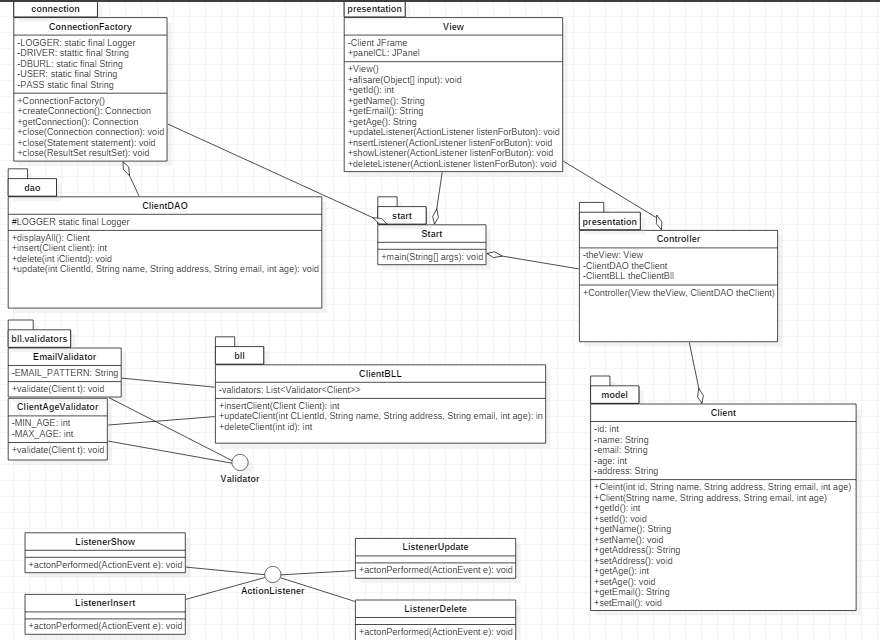
Unified Modeling Language (prescurtat UML) este un limbaj standard pentru descrierea de [modele](https://ro.wikipedia.org/w/index.php?title=Modelare_orientat%C4%83_pe_obiect&action=edit&redlink=1) și [specificații](https://ro.wikipedia.org/w/index.php?title=Specifica%C8%9Bie&action=edit&redlink=1) pentru [software](https://ro.wikipedia.org/wiki/Software) , in programarea POO . Acestea ne ofera o intelegere mai usoara a modelarii problemei , a felului in care clasele se raporteaza unele la altele , si a felului in care principiile POO sunt aplicate . Evident , nu toate metodele si toate atributele sunt reprezentate pentru fiecare clasa . Dimpotriva , am ales sa “ ascund ” unele atribute sau metode , mai putin relevante pentru imagine de ansamblu a problemei , tocmai pentru ca relatiile stabilite intre clase sa fie cele evidentiate .

Diagrama *use case* permite prezentarea posibilităților pe care utilizatorul le are atunci când folosește aplicația. Se distinge o singura operație care poate fi executata de utilizator, deoarece aplicația are un rol de simulare demonstrativa. Diagrama este exemplificată în schema următoare.



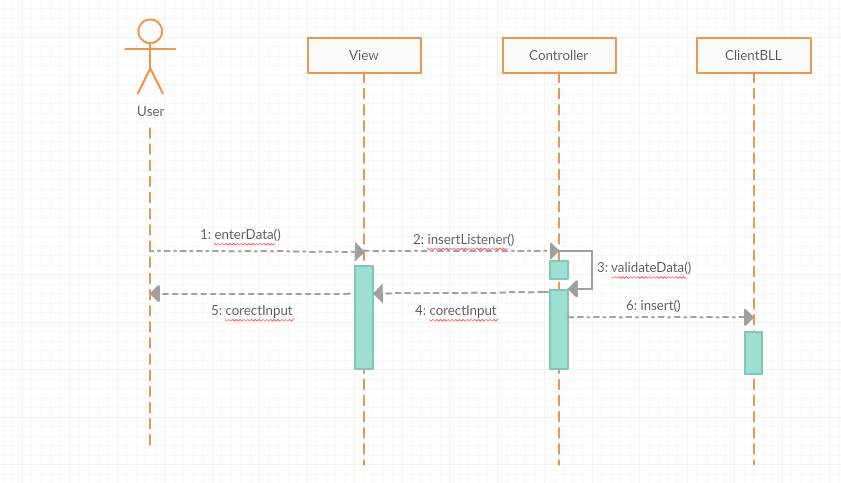
**4.1 Diagrama de clase**

Diagrama de clasă prezintă principalele clase necesare pentru realizarea aplicației, fiecare element al diagramei având numele clasei, variabilele de clasă, cât și metodele aferente clasei respective. În plus, se conturează relațiile dintre clase (de dependență, de agregare), cât și vizibilitatea datelor și a metodelor (publice, private sau protejate). Pentru a nu supraaglomera schema, în diagramă au fost menționate doar datele definitorii ale claselor. Pentru o perspectivă completă asupra proiectului, se vor vizualiza sursele cu cod. Diagrama de clasă este reprezentată prin schema următoare.

****

**4.1 Diagrama de secventa**

Diagrama de secvență arată cum interacționează obiectele unul cu celălalt, și în ce ordine. Aceste interacțiuni sunt aranjate în ordine temporală și prezintă scenariile posibile care pot apărea în urma interacțiunii utilizatorului cu aplicația.

****

# Implementare

**Pachete:**

Intregul proiect este impartit intr-o serie de pachete , fiecare continand clase ce implementeaza anumite functionalitati . Dupa tipul claselor continute , se disting urmatoarele pachete :

- dao : contine clase care asigura realizarea accesului la o baza de date care sta la baza intregii aplicatii , contine clasele ProductDAO, ClientDAO, ComenziDAO ;

- bll : realizeaza legatura dintre dao si baza de date , contine clasele : ComenziBLL, ClientBLL, ProductBLL

-bll.validatorslocul unde se fac verificarile datelor de intrare si contine ClientAgeValiator, EmailValidator, ProductValidator, QunatityValidator.

- connection : pentru conexiunea dintre baza de date ( realizata in workbench ) si aplicatia java , contine clasa ConnectionFactory ;

- model : contine clasele : Client , Product , Comenzi , pe baza carora este realizata baza de date ;

- presentation : este pachetul ce contine clase responsabile de realizarea unei interfete grafice

- dal: este pachetul responsabil pentru data acces layes si contine clasa Reflection

- main : contine clasa Main ;

**Clase**

Intregul proiect este impartit in clase , prezentand , pe scurt , scopul crearii fiecareia si relatiile , mai semnificative , stabilite intre acestea .

**Reflectia** utilizata pentru a descrie un cod capabil sa utilizeze un alt cod din sistem , sau chiar pe sine insusi . Spre exemplu , avem un obiect de tip necunoscut in Java , pe care se apeleaza o metoda existenta . Chiar daca sistemul nu poate sa apeleze static metoda respectiva pentru acel obiect , folosind reflectia, obiectul poate sa fie inspectat si daca exista acea metoda sa fie apelata ori de cate ori este necesar .

**Clasa ClientDAO**

Este folosita pentru a asigura accesul la baza de date ( Database Access Object ) . Sunt realizate apoi interogari aplicabile bazei de date , folosite pentru a primi informatiile necesare : Select , Update , SelectAll , Insert , Delete.

**Clasa ClientDAO**

Sunt realizate apoi interogari aplicabile bazei de date , folosite pentru a primi informatiile necesare : Select , Update , SelectAll , Insert , Delete.

**Clasa ComenziDAO**

Clasa ClientDAO particularitatea acestei clase consta in faptul ca pentru stabilirea orderuri-lor avem nevoie , in plus doar de id-ul clientului care a realizat comanda , respectiv de id-ul produsului ( produselor ) comandate . In acest sens , mosteirea reprezinta cea mai facila metoda de implementare .

Este folosita pentru a asigura accesul la baza de date ( Database Access Object ) . Sunt realizate apoi interogari aplicabile bazei de date , folosite pentru a primi informatiile necesare : Select , Update , SelectAll , Insert , Delete .

**Clasa ConnectionFactory**

Clasa ConnectionFactory este utilizata pentru a realiza conexiunea cu baza de date, careia ii sunt aplicate interogarile . Sunt necesari o serie de parametri pentru a realiza aceasta conexiune , si anume : driver-ul bazei de date , adresa url si parola pentru userul care acceseaza aceasta baza de date . Dupa ce aceste detalii sunt stabilite , conexiunea poate sa fie realizata . Este folosit un design pattern : singleton care restrictioneaza instantierea unei clase si asigura faptul ca o singura instanta a clasei exista in masina virtuala java . Prin metoda getConnection( ) este asigurata o accesare la nivel global acestei clase . Date fiind caracteristicile pattern- ului singleton , putem sa targem concluzia ca este extrem de necesar pentru aplicatii care presupun logarea in diferite aplicatii . Nu dorim ca doua instante diferite sa faca schimbari in baza de date pe care o manageriem , deoarece acest comportament ar cauza anumite conflicte . In final , este nevoie sa inchidem cnexiunea cu baza de date folosita : statement si resultSet .

**Clasele Client, Product si Comenzi**

Ofera modele pentru obiectele de tipul client , produs si comanda . Specific acestor clase este id-ul comenzii , produsului , respectiv comenzii , unic generat si realizat prin autoincrementare . Fiecare dintre aceste clase contine o serie de atribute care reprezinta campurile tabelelor cu acelasi nume care constituie baza de date .

**Clasele pachetului presentation**

Clasele folosite prntru reaizarea interfetei grafice sunt incluse in pachetul presentation , acestea asigurand realizarea unei interfete grafice accesibile oricarui utilizator . Clasele implementeaza metode asemanatoare , utilizate pentru a prelua texte din campurile de text , pentru a adauga diferiti ascultatori( listeneri ) butoanelor sau pentru a realiza frame- uri .

**Clasa View**

Reprezinta clasa care contureaza principala fereastra a interfetei grafice. Aici sunt construite cele 3 JFrame care sunt afisate utilizatorului CLinet Product si Order.

**Clasa Contoller**

Reprezinta clasa care contine clasele de tip ActionListener care va interaction cu clasa View si prin care utilizatorul poate face operatile necesare.

1. **Concluzii si Dezvoltari Ulterioare**

In concluzie, putem spune ca am reusit sa implememtam o aplicatie pentru procesarea de comenzi .Aceasta aplicatie este usor de utilizat datorita interfetei accesibile. In urma realizarii acestui proiect, am invatat o multitudine de lucruri noi : folosirea unui design pattern n-tier, impartirea claselor in pachete in functie de functionalitate, realizarea conexiunii dintre o baza de date si Java, scrierea unor interogari SQL in Java si trimiterea acestora spre baza de date in scopul executarii lor, realizarea unei interfete grafice mai complexa decat cele de pana acum.

**Dezvlotari ulterioare:** S-ar putea adauga implementarea unei interfete de Login cu username si password, precum si o interfata grafica mai complexa, implementarea de oferte, vizulizare profitului magazinului etc.

# Bibliografie

**Java de la 0 la expert ( editia a II -a ) , Stefan Tanasa , Stefan Andrei , Cristian Olaru , Editura : Polirom**

**Cursul de Tehnici de programare al domnului profesor Ioan Salomie**

**Forum-uri.**

**Tutoriale Youtube**

**Cursul domnului profesor Marius Joldos**

**http://docs.oracle.com/javase/7/docs/api/**

**http://stackoverflow.com/**