**DOCUMENTATIE TEMA 3**

**Gestionarea comenzilor**

**Nume prenume: Man Larisa Silvana**

**Grupa 30224**

**Profesor Laborator Assist Pop Claudia**

Contents

[1. Cerinte Functionale 3](#_Toc481099601)

[2. Obiective 3](#_Toc481099602)

[2.1. Obiectiv Principal: 3](#_Toc481099603)

[2.2. Obective Secundare: 3](#_Toc481099604)

[3. Analiza Problemei 4](#_Toc481099605)

[4. Proiectare 5](#_Toc481099606)

[4.1. Structuri de date 8](#_Toc481099607)

[4.2. Diagrama de clase 11](#_Toc481099608)

[4.3. Algoritmi 11](#_Toc481099609)

[5. Implementare 12](#_Toc481099610)

[6. Concluzii si Dezvoltari Ulterioare 15](#_Toc481099611)

[7. Bibliografie 15](#_Toc481099612)

# Cerinte Functionale

Cerinta proiectului este de a crea o aplicatie de procesare a comenzilor. O baza de date va stoca produsele, clientii si comenzile facute . Aplicatia va contine minim aceste clase:

1. Clasele specifice: Order, Customer, Product
2. Business Logic pentru fiecare clasa in parte
3. Clasa pentru prezentare: GUI
4. Clase de acces la baza de date

# Obiective

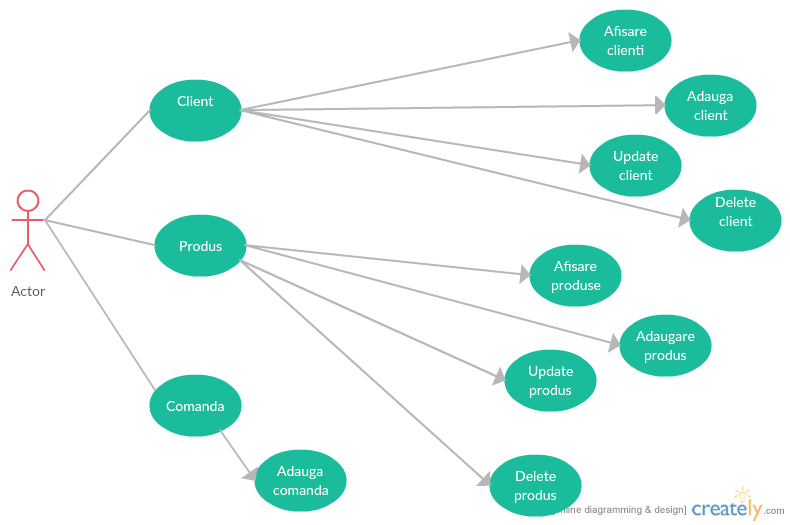
## Obiectiv Principal:

**Obiectivul** temei este de a implementa o aplicatie cu interfata grafica pentru a fi folosita pentru gestionarea comenzilor. Aplicatia permite vizualizarea, modificarea, stergerea si adaugarea de produse, comenzi si clienti, din perspectiva unui administrator. De asemenea, pentru stocarea produselor, comenzilor si clientilor am folosit o baza de date, creata in limbajul de programare SQL, cu o structura a tabelelor asemanatoare cu structura claselor implementata in Java.

## Obective Secundare:

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Dezvoltarea de use case-uri si scenarii | Am ales operatiile care se pot utiliza pe baza de date, respectiv adaugarea, stergerea, update si vizualizarea informatiilor, atat pentru client, produs si comanda. | 3 |
| Impartirea pe clase | Am folosit Layered Architecture pentru o structura adecvata unei astfel de aplicatii. | 4 |
| Dezvoltarea algoritmilor | Am creat operatiile de adaugare, stergere si update pentru fiecare clasa. | 4 |

# Analiza Problemei



**Analiza problemei** presupune modul in care percepem ceea ce trebuie sa implementam. Astfel, analiza cerintei trebuie facuta riguros inca de la inceput pentru a ne feri de omiterea anumitor pasi sau anumitor detalii legate de implementare. Din cerinta se poate deduce ca o sa avem de implementat o aplicatie care sa pastreze datele intr-o baza de date si care trebuie sa se foloseasca de acestea. Prin urmare, urmatoarea etapa pe care trebuie sa o atingem este **modelarea** ei si anume ce clase avem si care o sa fie capabilitatile acestora. Pe langa clase un aspect important pe care trebui sa il atingem este folosirea Layered Architecture care imparte aplicatia in diferite layere, fiecare dintre acestea avand un anumit scop. In proiectul meu am ales sa am urmatoarele pachete:

* Model: contine clasele representative pentru fiecare table din baza de date in parte
* Presentation: contine clasa care defineste interfata cu userul
* BLL (Business Logic): contine clasele care incapsuleaza logica aplicatiei
* DAO (Data Access): contine clasele care au queries, pentru comanda facuta in baza de date, in limbaj SQL
* Connection: legatura propriu-zisa dintre Java si SQl, baza de date

# Proiectare

Un aspect important in realizarea acestui proiect este modalitatea de a face legatura intre o baza de date, create in MySQL si mediul de dezvoltare open-source Eclipse. MySQL este un sistem de gestiune a bazelor de date relaționale, produs de compania suedeza MySQL AB și distribuit sub Licența Publică Generală GNU. Este cel mai popular SGBD open-source la ora actuală, fiind o componentă cheie a stivei LAMP (Linux, Apache, MySQL, PHP).

Deși este folosit foarte des împreună cu limbajul de programare PHP, cu MySQL se pot construi aplicații în orice limbaj major. Există multe scheme API disponibile pentru MySQL ce permit scrierea aplicațiilor în numeroase limbaje de programare pentru accesarea bazelor de date MySQL, cum ar fi: C, C++, C#, Java, Perl, PHP, Python, FreeBasic, etc., fiecare dintre acestea folosind un tip specific API. O interfață de tip ODBC denumită MyODBC permite altor limbaje de programare ce folosesc această interfață, să interacționeze cu bazele de date MySQL cum ar fi ASP sau Visual Basic. În sprijinul acestor limbaje de programare, unele companii produc componente de tip COM/COM+ sau .NET (pentru Windows) prin intermediul cărora respectivele limbaje să poată folosi acest SGBD mult mai ușor decât prin intermediul sistemului ODBC. Aceste componente pot fi gratuite (ca de exemplu MyVBQL) sau comerciale.

Acum voi descrie cum se face accesul la baza de date din aplicatiile Java**. JDBC (Java Database Connectivity)** este o interfaţă standard (API) de programare a accesului la baze de date fiind constituită dintr-un set de clase si interfeţe scrise în Java, furnizând mecanisme standard pentru dezvoltarea aplicaţiilor Java cu baze de date . Folosind facilitățile oferite de JDBC este usor să transmitem secvențe SQL către baze de date relaționale (MYSQL ,MS-SQL,Oracle) . Pachetele Java care ofera suport pentru lucrul cu baze de date sunt incluse in **java.sql** , ce reprezinta nucleul tehnologiei JDBC oferind posibilitatea lucrului cu instrucțiuni SQL și procesarea rezultatelor obținute în urma interogărilor , respectiv **javax.sql** ,ce oferă capabilități pe parte de server cum ar fi tranzacții distribuite și conexiuni multiple, în condițiile în care într-o arhitectură client-server baza de date se poate afla pe aceeași masină sau pe o altă masină cu care clientul este conectat dintr-un intranet sau Internet. Utilizând API-ul JDBC aplicațiile pot fi programate în mod independent de SGBD, iar apelurile SQL către baza de date vor fi trimise ,preluate si transformate de către diverse drivere scrise specific pentru fiecare SGBD. Specificația JDBC pune la dispoziția producătorilor de drivere reguli de construire a driverelor, de-alungul timpului atât specificația cât și driverele au cunoscut o evoluție continuă, sub aspectul funcționalităților oferite.Se vor verifica caracteristicile/functionalitatile oferite de fiecare driver pentru a putea dezvolta in mod corect aplicatia client Java.

Accesarea unei baze de date folosind JDBC este simplă şi implică următorii paşi:

1. Obţinerea unui obiect de tip **Connection** ce încapsulează conexiunea la baza de date (în acest pas se realizează deci conexiunea la baza de date).

2. Obţinerea unui obiect **Statement** dintr-un obiect de tip Connection. Acest obiect este folosit pentru a transmite spre execuţie comenzi SQL către baza de date. Prin intermediul acestui obiect sunt efectuate operaţii de interogare şi modificare a bazei de date.

3. Obţinerea unui obiect **ResultSet** dintr-un obiect Statement. Obiectul ResultSet încapsuleaz rezultatele operaţiilor de interogare.

4. Procesarea rezultatelor încapsulate în obiectul ResultSet.

Procesul de conectare la o bază de date implică înregistrarea unui driver corespunzător si realizarea unei conexiuni cu baza de date. O conexiune (sesiune) la o baza de date reprezintă un context prin care sunt trimise secvente SQL catre baza de date si primite rezultate la nivel de aplicație.

JDBC furnizează acces orientat obiect la bazele de date, prin definirea de clase și interfețe ce modelează concepte abstracte, semnificatia celor mai importante fiind urmatoarea:

Drivere - clasa Driver so Driver Manager.

Conexiunea la baza de date – clasa Connection

Interogări SQL clasele Statement, Prepared Statement, Callable Statement

Multimi rezultat al executiei - clasa Result Set

Specificarea accesului la o baza de date create si gazduita de un **SGBD** , dintr-o aplicatie Java presupune mai multe etape, ce trebuie parcurse intr-o secventiere stricta si anume: inregistrarea unui driver ce va fi utilizat pentru translatarea catre baza de date a frazelor SQL , respectiv stabilirea conexiunii cu baza de date , corect specificata prin intermediul unui string de conexiune specific fiecarei baze de date.

Clasele si interfetele responsabile cu realizarea unei conexiuni sunt:

DriverManager - este clasa ce se ocupă cu înregistrarea driverelor ce vor fi folosite în aplicatie;

Driver - interfata pe care trebuie sa o implementeze orice clasă ce descrie un driver;

Connection – interfata ce descrie obiectele ce modeleazaă o conexiune propriu-zisă cu baza de date.

In procesul de conectare la o baza de date, aplicația trebuie sa înregistreze la masina virtuala ce ruleaza aplicatia (JVM) ,driverul JDBC responsabil cu comunicarea cu respectiva baza de date. Acest lucru presupune încarcarea în memorie a clasei ce implementeaza driver-ul , si poate fi realizată în mai multe moduri, dupa cum urmeaza :

a. Folosirea clasei **DriverManager**:

DriverManager.registerDriver(new TipDriver());

b. Folosirea metodei Class.forName ce apeleaza ClassLoader-ul masinii virtuale:

Class.forName("TipDriver"); Class.forName("TipDriver").newInstance();

c. Setarea proprietatii sistem **jdbc.drivers**, care poate fi realizată în doua moduri :

— De la linia de comanda: java -Djdbc.drivers=TipDriver Aplicatie

— Din program: System.setProperty("jdbc.drivers", "TipDriver");

Folosind aceasta ultimă metodă, specificarea mai multor drivere se face separând numele claselor cu punct si virgula. Intr-o aplicatie pot exista simultan mai multe conexiuni la baze de date diferite sau la aceeasi baza de date. Cea mai larg utilizată metodă este metoda b.

După ce un driver JDBC a fost înregistrat, acesta poate fi folosit la stabilirea unei conexiuni cu o bază de date. Avand în vedere faptul că pot exista mai multe drivere încărcate în memorie, trebuie să avem posibilitea de a specifica pe lângă un identificator al bazei de date și driverul folosit pentru a accesa baza de date. Aceasta se realizează prin intermediul unei structuri specifice, numită **JDBC URL**, ce are următorul format general :

***Jdbc : sub-protocol : identificator DB***

unde

• sub-protocol denumeste tipul de driver ce trebuie folosit pentru realizarea conexiunii si poate fi *odbc, Microsoft SQL, Oracle, sybase, db2, etc.*

• Identificatorul bazei de date este un indicator specific fiecărui driver corespunzator bazei de date cu care aplicatia doreste să interactioneze. In functie de tipul driver-ului , acest identificator poate *include numele unei masini gazda, un numar de port, numele unui fișier* , etc. La primirea unui JDBC URL, DriverManager-ul va parcurge lista driverelor înregistrate în memorie, pana cand unul dintre ele va recunoaste URL-ul respectiv. Daca nu exista nici unul potrivit, atunci va fi lansata o exceptie de tipul SQLException, cu mesajul "no suitable driver".

Realizarea unei conexiuni.Metoda folosita pentru realizarea unei conexiuni este **getConnection** din clasa DriverManager și poate avea deasemenea mai multe forme:

**Connection c = DriverManager.getConnection(url);**

**Connection c = DriverManager.getConnection(url, username, password);**

Folosirea diferitelor tipuri de drivere implica doar schimbarea numelui clasei ce reprezinta driverul si a modalitatii de specificare a bazei de date.

Pentru MySQL – **String url = “jdbc:mysql://host\_name:port/NumeBD”**

unde host\_name este adresa completa a hostului (masinii fizice ) ce gazduieste baza de date, ea poate fi chiar sistemul local, caz in care notatia simplificata este **localhost**.

O conexiune creata va fi folosita pentru transmiterea de secvente SQL utilizate pentru interogarea sau actualizarea bazei de date. Pentru eliberarea resurselor alocate mecanismelor descrise anterior , este necesară închiderea unei conexiuni ,prin metoda **close**. De asemenea, clasa *Connection* asigura facilitati avansate pentru controlul tranzactiilor efectuate la nivelul bazei de date prin metodele specifice ***commit, rollback, setAutoCommit.***

Pentru **SGBD MySQL** secvența de inițializare este:

**Driver d = (Driver)Class.forName("com.mysql.jdbc.Driver").newInstance();**

Si respectiv *Connection URL*:

**con = DriverManager.getConnection("jdbc:mysql://localhost: 3033", "userName", "password")**

Ambele secvente prezentate realizează accesul la baza de date găzduită de sistemul de gestiune corespunzător ce rulează pe aceeaşi maşină (host) pe care este executată aplicaţia client Java.

După realizarea conexiunii cu metoda **DriverManager.getConection**, se poate folosi obiectul Connection rezultat pentru a se crea obiecte de tip **Statement**, **PreparedStatement** sau **CallableStatement** cu ajutorul cărora putem trimite secvenţe SQL către baza de date. Obținerea şi prelucrarea rezultatelor unei interogări este realizată prin intermediul obiectelor de tip ResultSet.

Cele mai uzuale comenzi SQL sunt cele folosite pentru: interogarea bazei de date: SELECT, actualizarea datelor: INSERT, UPDATE, DELETE, sau chiar actualizarea schemei de date: CREATE, ALTER, DROP . oricare din acestea pot fi transmise către baza de date, ca şi string, întru execuţie ca un „statement “, printr-una din metodele prezentate anterior.

## Structuri de date

In realizarea acestei aplicatii am folosit Layered Arhitecture, o arhitectura stratificata, multi-strat, 3-tier, de tip server-client. Straturile sunt folosite pentru a aloca responsabilitati in aplicatie, structurand-o logic si eficient. Se imparte in partea de prezentare, procesarea aplicatiei si managementul datelor. Stilul Layers (pe niveluri) este adecvat aplicatiilor care pot fi descompuse in grupuri de subtaskuri, fiecare grup reprezentand  un anumit nivel de abstractizare. Fiecare nivel utilizeaza servicii furnizate de nivelul(urile) de sub el.

Arhitectura este o abstractizare a sistemului​. Ea omite anumite detalii: se exclud acele proprietati ale elementelor care nu afecteaza modul cum elementele pot fi utilizate de alte elemente sau interactiunile intre elemente.

Proprietati vizibile din exterior: servicii furnizate, caracteristici de performanta, utilizarea resurselor partajate de mai multe elemente, etc.​

Un Layer poate accesa direct doar componentele publice din Layer- ul imediat subordonat. De exemplu, Presentation Layer poate accesa doar componentele publice din Application Layer dar nu si din Data Layer. Application Layer poate accesa componentele publice din Data Layer dar nu si din Presentation Layer. Facand asta se poate minimiza dependenta dintre un strat fata de altele. Aceasta minimizare a dependentei va aduce beneficii pentru mentenanta/ dezvoltarea, imbunatatirea si scalabilitatea Layer-ului respectiv. Facand acest lucru, devine posibila si imbunatatirea securitatii unui Tier. De exemplu, un Client Layer nu poate accesa Data Layer in mod direct ci doar prin Application Layer deci Data Layer are un nivel de securitate ridicat.

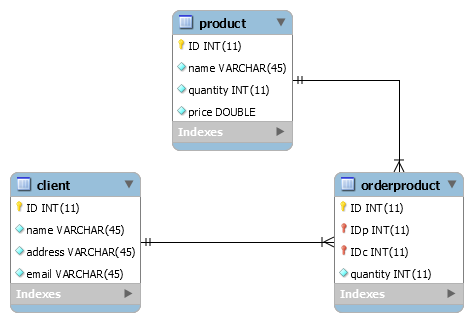
Pentru a obtine o arhitectura 3-Tier completa, toate cele 3 Layere ar trebui sa ruleze pe calculatoare separate.

Presentation Layer reprezinta un strat pe care utilizatorii il pot accesa direct cum ar fi, interfata unui utilizator de desktop sau o pagina web etc. Mai este numit si Client Layer.

Application Layer : acest strat incapsuleaza logica de business(cum ar fi regulile de business si validarile de date), conceptul de domeniu, logica accesului la date si asa mai departe. Se mai numeste si Middle Layer.

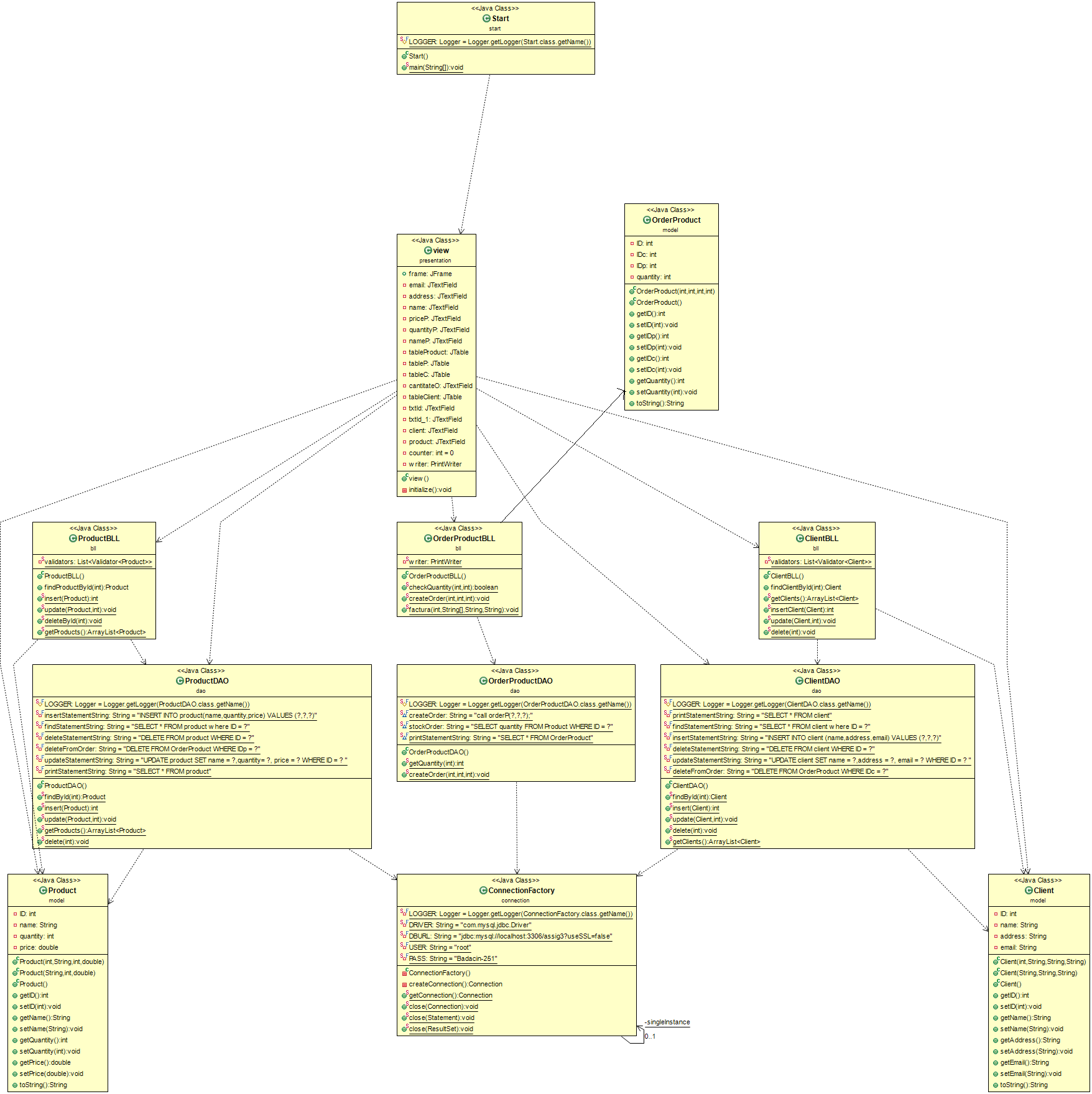
Data Layer : reprezinta data sursa externa care inmagazineaza datele aplicatiei respective cum ar fi serverul bazei de date, sistemul CRM, sistemul ERP, sistemele de mostenire etc. Cel pe care il intalnim cel mai des este serverul bazei de date. Pentru o arhitectura N-Tier trebuie folosite baze de date care nu sunt auto-incluse cum ar fi SQL server , Oracle, DB2, MySQL sau PostgreSQL. Acestea pot sa ruleze pe un singur calculator.

Am creat baza de date in MySQL, ajungand la diagrama:



Avem trei tabele: client, product, respectiv orderProduct, fiecare avand un ID de tip Integer ca si cheie primara., dupa care va fi identificat. Tabelul orderProduct are doua chei straine, care fac legatura intre el si product, respectiv client. Pe langa ID, tabelul client mai are ca si atribute un nume, o adresa si un email de validare a comenzii; tabelul product are o denumire, cantitate si pret pentru fiecare produs in parte, iar tabelul orderProduct are cantitatea comandata. Pentru a realiza o comanda, am creat o procedura in SQL, care realizeaza aceasta operatie dupa ID, actualizand cantitatea produselor.

## Diagrama de clase



## Algoritmi

Operatiile implementate pentru acest proiect sunt: adaugare, update, delete pentru clienti si produse, iar cea mai importanta operatie – comanda, realizata printr-o procedura scrisa in SQL ai apelata din Java cu un Callable Statement. Aceste metode vor fi descrise detaliat in urmatorul capitol.

# Implementare

Dupa cum am mentionat mai sus, acest proiect respecta arhitectura Layered. Astfel, pachetele vor fi: **model, BLL, DAO, presentation, connection** si **start**. In urmatoarele randuri voi detalia ce contin acestea.

Pachetul „model” contine trei clase: Client, Product si OrderProduct. Fiecare dintre acestea au un tabel reprezentant in baza de date. Atributele fiecarei clase sunt liniile tabelelor. Asadar, clasa „Client” va avea un ID, name, address si email; clasa „Product” va avea un ID, name, quantity si price; iar clasa „OrderProduct”, un ID, quantity, IDc ( ID client ), IDp ( ID produs ) , cele din urma fiind cheile straine care realizeaza relatiile dintre tabele. Toate acestea au un constructor, getters si setters pentru toate campurile.

Pachetul „dao” (Data Access) contine din nou trei clase, cate una pentru fiecare tabel. Clasa ClientDAO contine Statementurile pe care acest tabel le foloseste, respectiv un statement de afisarea a informatiilor din tabelul client, de cautare dupa ID, de inserare, stergere, update. Ca si metode avem metoda findById(), care gaseste un client dupa ID, metoda insert(), care adauga un client in baza de date, metodele update() si delete(), care fac update la informatiile nou introduse, respectiv sterg un client din baza de date. Mai avem si o metoda getClients() care returneaza un ArrayList de clienti, adica lista clientilor din baza de date.

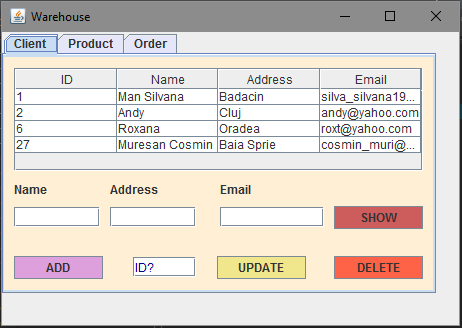
Clasa ProductDAO contine aceleasi metoda si statementuri, dar adaptate la produse. In aceste metode am folosit ResultSet-uri, PreparedStatement-ui din pachetul java.sql si am realizat legatura cu baza de date, cu ajutorul metodei ConnectionFactory, din pachetul Connection.

Clasa OrderProductDAO are o metoda getQuantity() are rolul de a prelua cantitatea produselor si a verifica daca exista destule bucati pentru a se realiza comanda. Metoda createOrder() creeaza comanda apeland prodecura din MYSQL.

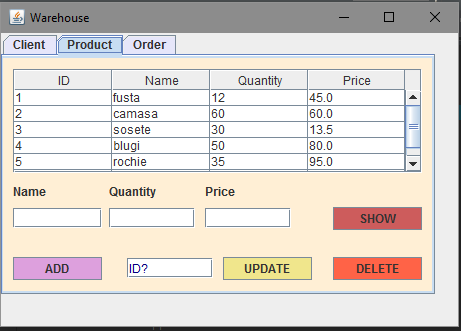
Pachetul „bll” (Business Logic) cu cele trei clase, cate una pentru fiecare tabel, contine metodele pe care le gasim si in DAO. Daca in DAO le-am implementat, aici le vom apela. Ce gasim nou este metoda care scrie factura cu ajutorul unui PrintWriter. Factura va si plasata in fisierul dorit, sub forma unui document text, care conține datele clientului, denumirea produsului si prețul final. Aceasta metoda se afla in clasa OrderProductBLL.

Pachetul „connection” contine clasa ConnectionFactory, are face legatura dintre mediul Eclipse si baza de date MySQL. Gasim pe langa constructor, o metoda createConnection(), care face legatura cu baza de date dupa URL, user si parola serverului, metoda getConnection() si metodele de close() pentru conexiune, Statement si ResultSet.

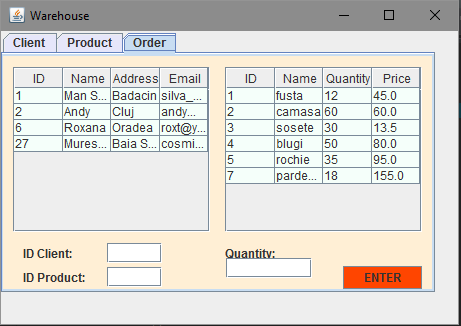
Pachetul „presentation” contine clasa View, adica implementarea interfetei grafice GUI. Am folosit extensia programului Eclipse, WindowBuilder, pentru a face munca mai usoara. Fereasta contine trei taburi: unul pentru operatiile facute pe tabela Client, unul pentru Product si al treilea pentru Order.



Tabul „Client” contine un tabel care afiseaza implicit ce se afla in baza de date, insa avem si un buton SHOW care face acest lucru, daca dorim sa vedem ce modificari au aparut. Butonul ADD este folosit pentru a adauga un nou client in baza de date, dupa ce ii vom introduce datele: numele, adresa, email-ul. ID-ul se va genera automat. Butonul de UPDATE va modifica informatiile unui client, pe care il selectam introducand ID-ul in field-ul potrivit. Se vor introduce si noile date, respectiv noul nume, adresa, email. Butonul DELETE va sterge clientul al carui ID il vom introduce.



Tabul „Product” contine un tabel care afiseaza implicit ce se afla in baza de date, insa avem si un buton SHOW care face acest lucru, daca dorim sa vedem ce modificari au aparut. Butonul ADD este folosit pentru a adauga un nou produs in baza de date, dupa ce ii vom introduce datele: denumirea, cantitatea, pretul. ID-ul se va genera automat. Butonul de UPDATE va modifica informatiile unui produs, pe care il selectam introducand ID-ul in field-ul potrivit. Se vor introduce si noile date, respectiv noua denumire a produsului, cantitatea si pretul. Butonul DELETE va sterge produsul al carui ID il vom introduce.



Tabul „Order” va afisa implicit doua tabele, unul in care putem vizualiza toti clientii din baza de date si altul in care vom vedea produsele disponibile. Pentru a comanda un produs trebuie sa introducel ID-ul clientului in cauza si ID-ul produsului, dar sa nu uitam si cantitatea. Mai apoi se va apasa butonul ENTER. Daca cantitatea dorita se afla in stoc, se va crea factura si plasa sub forma unui fisier text intr-un director dorit de noi, iar daca nu se va afisa un mesaj de atentionare.

Pachetul „start” contine metoda in care gasim main-ul programului.

# Concluzii si Dezvoltari Ulterioare

Ca o concluzie, acest proiect m-a ajutat sa imi consolidez cunostintele de programare orientata pe obiect dobandite in primul semestru si sa imi organizez munca in baza paradigmelor POO. Totodata, felul in care am structurat proiectul si am creat clasele si metodele m-a ajutat sa eficientizez codul din punct de vedere al lungimii si al usururintei intelegerii acestuia, lucruri vitale in cazul programelor complexe care necesita mai multe clase sau care presupun un numar mai mare de programatori.

In cadrul dezvoltarilor ulterioare se pot aminti urmatoarele: imbunatatirea interfetei grafice, posibilitatea de a vizualiza produsul prin anumite fotografii, posibilitatea de a comanda mai multe produse deodată, de a exista un cos cu produse, de a primi factura prin email.

# Bibliografie

* <https://en.wikipedia.org/wiki/Multitier_architecture>
* <https://ftp.utcluj.ro/pub/users/civan/Introducere_in_BazedeDate/2_LABORATOR/09_10_Aplicatii_cu_BD/JDBC/L9_IBD_JDBC.pdf>
* <https://ro.wikipedia.org/wiki/MySQL>
* <http://stackoverflow.com/>
* <http://www.coned.utcluj.ro/~salomie/PT_Lic/3_Lab/HW3_Tema3/HW3_Indications.pdf>