# 

**DOCUMENTATIE TEMA 3**

**Order Management**

**Varvara Raluca Ana-Maria**

**Grupa 30222**

**Profesor Laborator: Oana Andreea Marin**

Contents

[1](#_Toc69723038)

[**1.** **Obiectivul temei** 3](#_Toc69723039)

[**2.** **Analiza** 3](#_Toc69723040)

[**3.** **Proiectarea sistemului** 4](#_Toc69723041)

[3.1 Decizii de proiectare 4](#_Toc69723048)

[3.2 Divizarea in subsisteme/pachete 4](#_Toc69723049)

[3.3 Împărțirea în clase 5](#_Toc69723050)

[3.4 Diagrama de clase 8](#_Toc69723051)

[3.5 Structuri de date 9](#_Toc69723052)

[3.6 Interfețe folosite/definite 9](#_Toc69723053)

[3.7 Interfața 10](#_Toc69723054)

[4. Implementare 12](#_Toc69723055)

[5. Rezultate 18](#_Toc69723056)

[6. Concluzii 18](#_Toc69723057)

[7. Bibliografie 18](#_Toc69723058)

# **Obiectivul temei**

Obiectivul principal al proiectului este de a creea o aplicație care să pună la dispoziție utilizatorului un șistem de management al comenzilor. Aceste sisteme se folosesc pretutindeni, fiecare site se foloseste de un asemenea sistem pentru vanzari online. Clientul se poate loga, poate vizualiza produsele si poate sa isi adauge in cosul de cumparaturi oricate produse atat timp cat sunt in stoc destule. dup ace s-a incheiat comanda, clientul primeste o facture pe care trebuie sa o plateasca. Toate aceste date sunt tinute in niste baze de date, iar aplicatia trebuie sa comunice cu aceasta.

Sub-obiectivele principale ale temei reprezintă pașii care au fost urmați pentru atingerea obiectivului final:

* Proiectarea unui client;
* Proiectarea unei produs;
* proiectarea unei comenzi;
* Conexiunea la baza de date;
* Crearea query-urilor pentru accesarea din baza de date a datelor;
* Implementarea interfeței cu utilizatorul.

# **Analiza**

**Cerințele funcționale** vor include următoarele:

* introducerea, editarea si stergerea unui client;
* afisarea listei de client;
* introducerea, editarea si stergerea unui produs;
* afisarea listei de produse;
* introducerea de comenzi;
* facturarea comenzii.
* vizualizare comanda

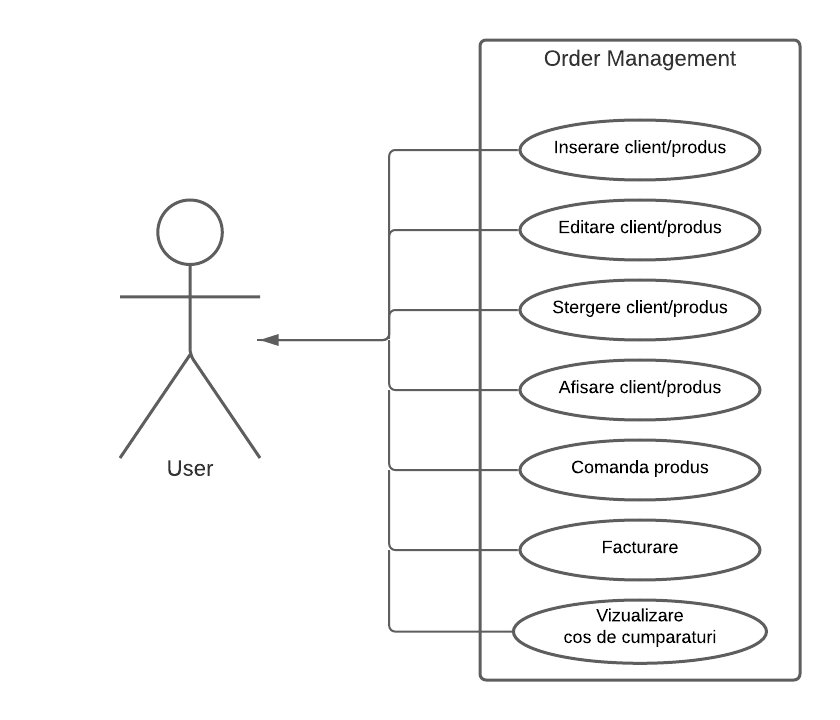


Fig.1.

**Cerințele non-funcționale** vor fi următoarele:

* Aplicatia ar trebui să fie intuitiva și ușor de folosit de către user;
* Incapsularea datelor;
* Eficientizarea scrierii codului si crearea de clase cat mai abstracte cu Reflection.

**Use case:** crearea unei comenzi

**Actorul principal:** user-ul

**Scenariul principal de success:**

1. User-ul introduce datele necesare în interfața grafică (clientul pentru care se face comanda, produsul si cantitatea);
2. User-ul repeat primul pasa pana isi finalizeaza comanda;
3. User-ul primeste bonul/factura;

**Secvența alternativă:** date introduce incorect

Userul introduce în unul dintre câmpurile de date ceva ce nu este un număr, sau lasă gol un câmp. Acesta va fi atenționat cu mesajul necesar.

**Secventa alternativa:** Nu exista destule produse pentru comanda lui, acesta va fi atentionat.

**Secvența alternativă:** un alt client incearca sa faca o comanda inainte sa isi fi facturat clientul de dinainte comanda. Se va primi un mesaj de atentionare, si se va putea efectua urmatoarea comanda de abia dupa facturarea celei de dinainte.

**Secventa alternativa:** nu exista clientul sau produsul introduce: se va primi un mesaj de atentionare.

# **Proiectarea sistemului**



## Decizii de proiectare

Din punct de vedere al proiectării am luat următoarele decizii:

* Crearea unei clase abstracte AbstractTable<T> care creaza headerul si continutul unui table in functie de o lista de obiecte cu tehnica reflection;
* Crearea clasei Factura pentru a retine datele care tin de bon, pentru crearea sa mai usoara la finalul unei comenzi (se retine produsul cu cantitatea ceruta si id-ul clientului);

## Divizarea in subsisteme/pachete

Am decis sa folosesc o arhitectura de tip Layered care imparte sistemul in 4/5 layers. Layerul pentru Model unde se retin clasele care definesc obiectele concrete cu care se lucreaza, structurile de date. Layerul de prezentare unde sunt clasele ce implementeaza interfata grafica. Business Logic layer ce continue clasele care se ocupa de logica aplicatiei. Data Access Layer ce se ocupa de accesul la baza de date, crearea queriurilor. Layerul de conexiune ce creaza conexiunea la baza de date.

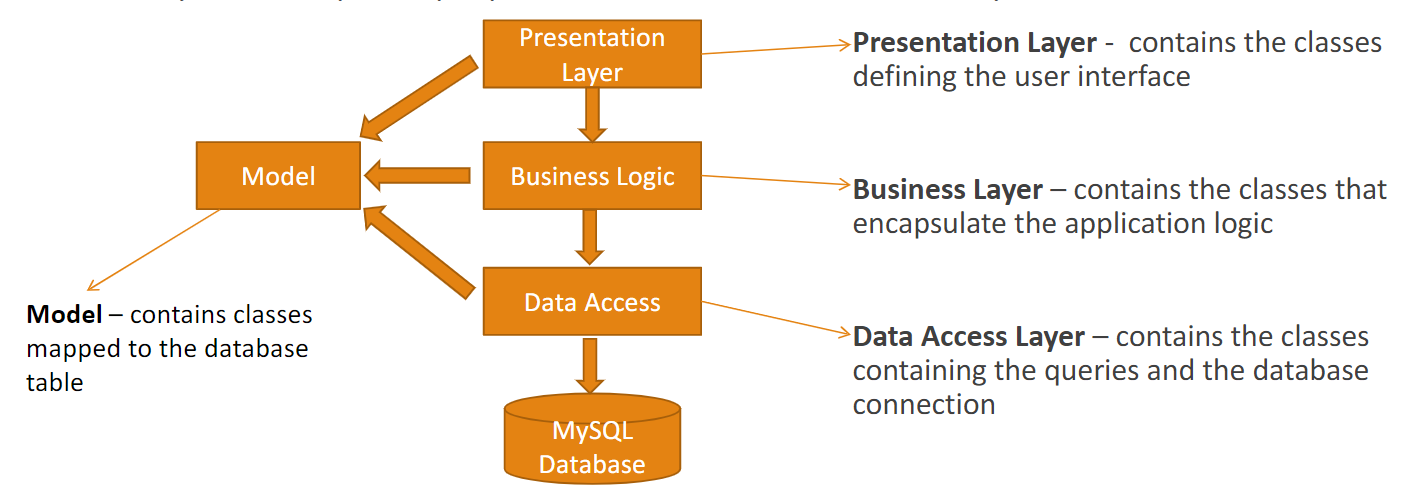
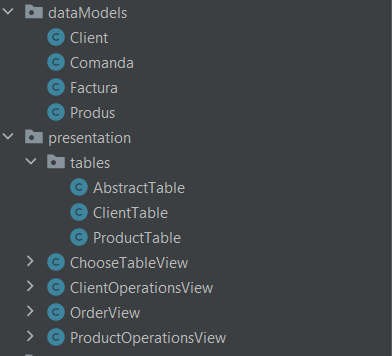
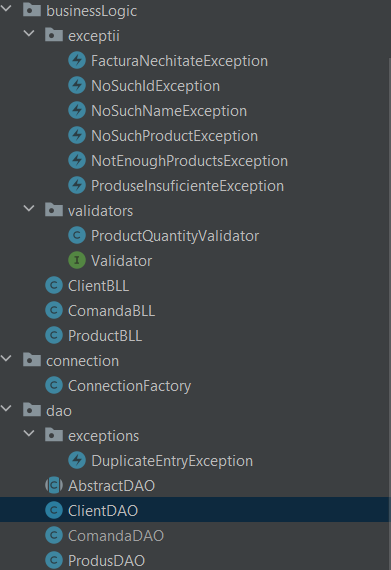


Fig. 2

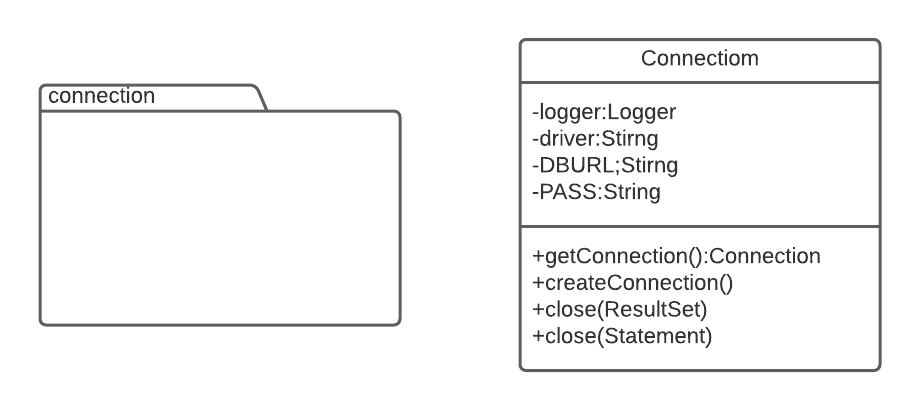
Este un mod ușor și intuitiv de implementare și împărtire în pachete, și nu există confuzii în legătură cu interacționarea dintre pachete.

## Împărțirea în clase

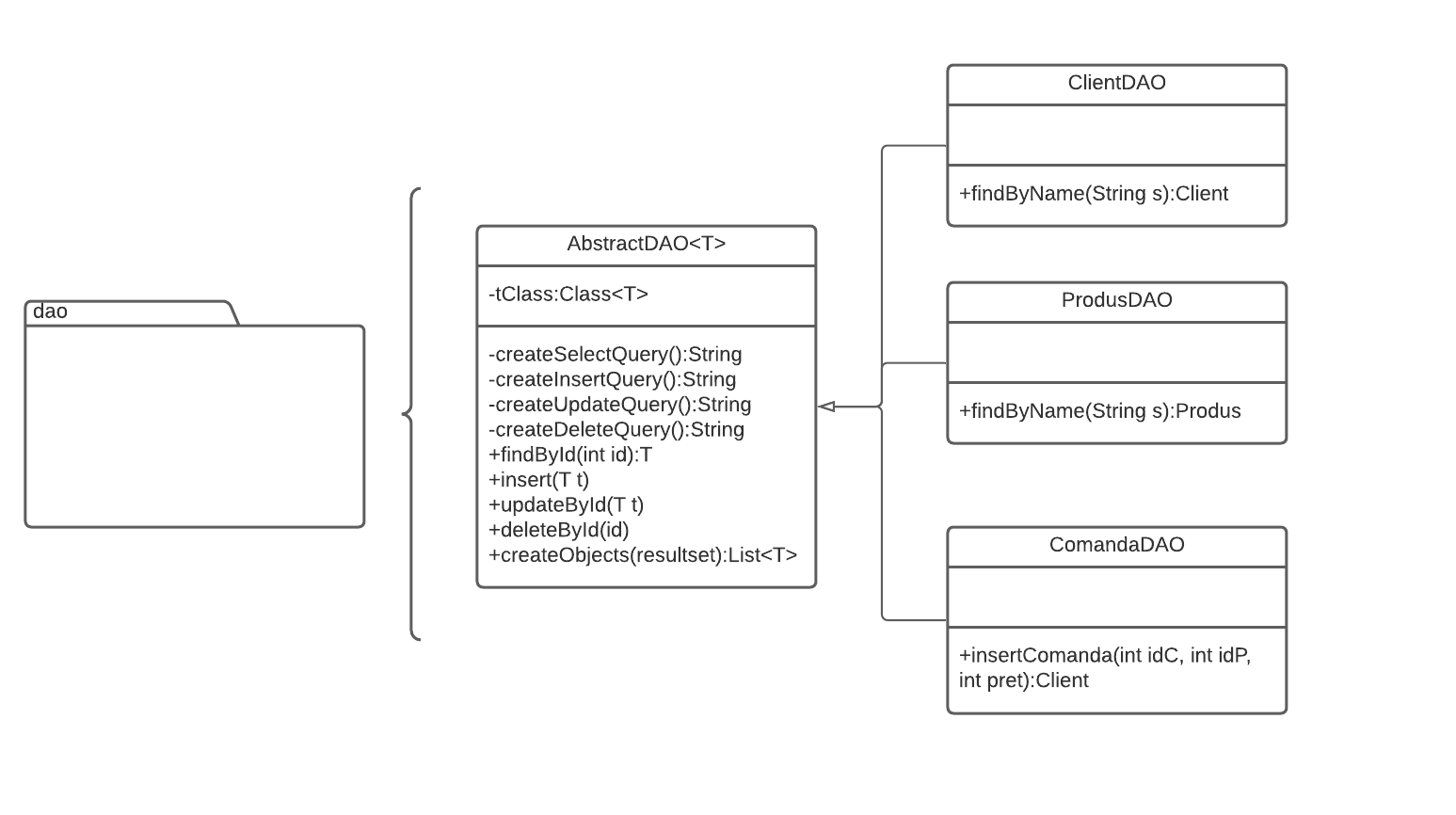




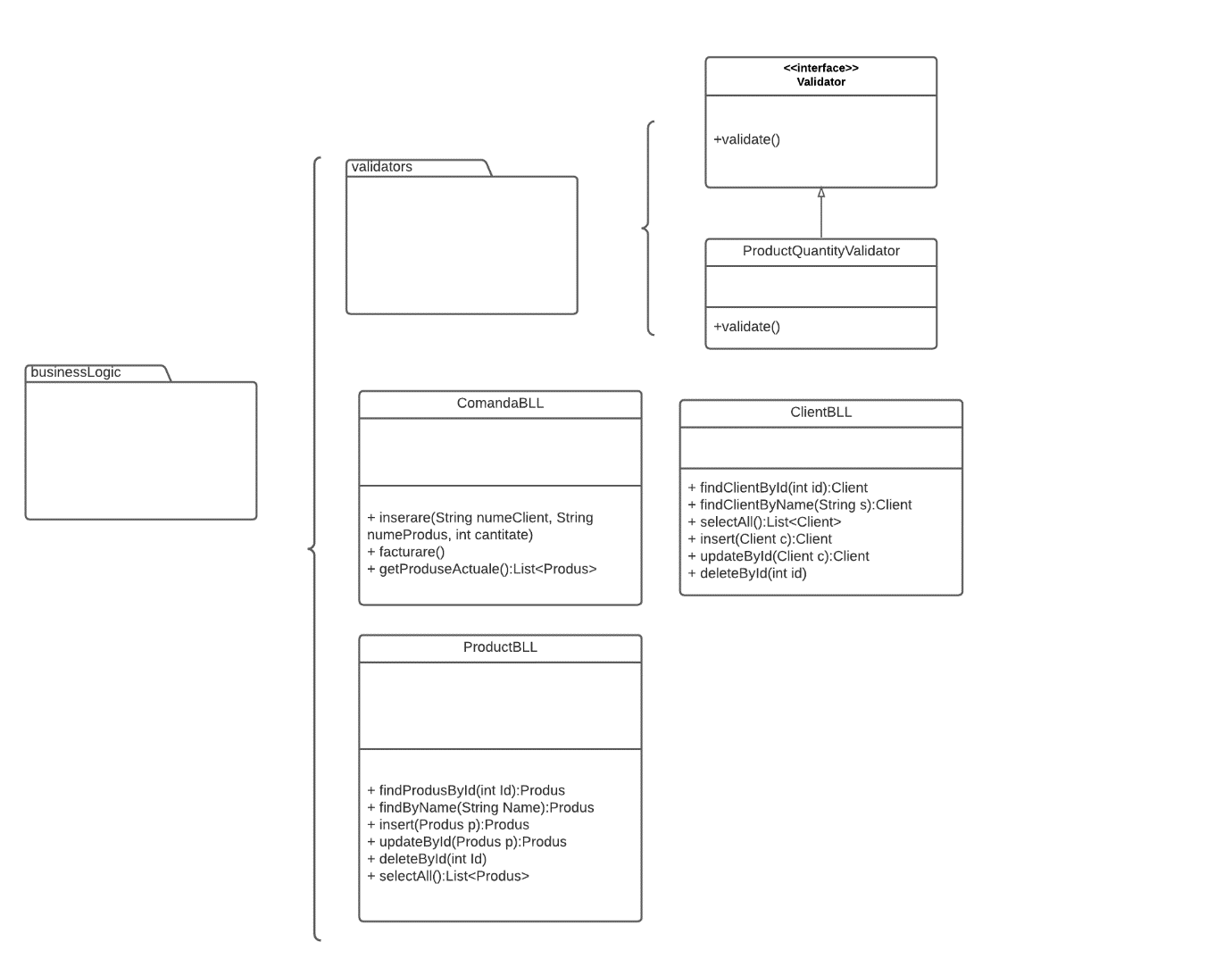
**Pachetul dataModel** va reține clasele care definesc modelele de date și care le vor putea stoca persistent astfel avem urmatoarele structure: Client, Factura, Comanda, Produs. Clienti vor retine datele importante despre ei pentru procesare, la fel si Produsele, si Comenzile. Factura mai retine id-urile clientului care a inceput comanda, pretul total si lista de produse, dar are si un parametru care retine daca comanda este sau nu in procesare.



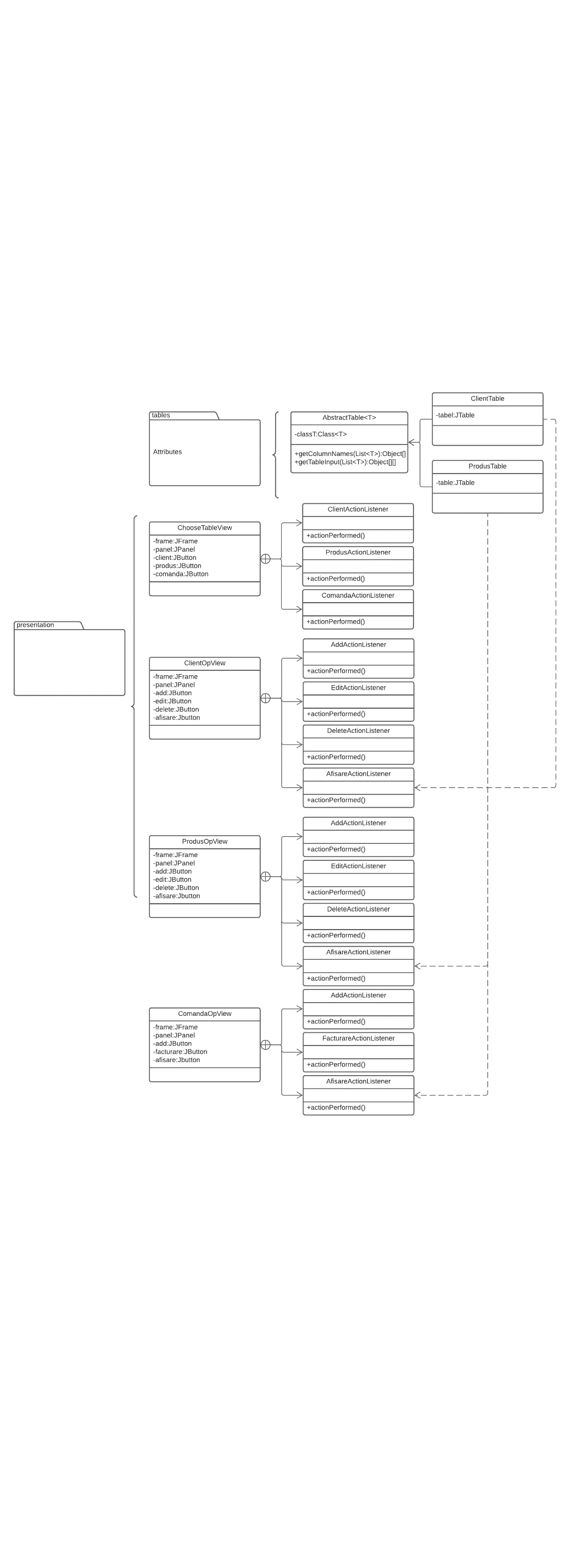
**Pachetul Connection** este pachetul care se ocupa cu conexiunea la baza de date si care inchide conexiunea, dar inchide si conexiunea statementului si rezultatului la baza de date.



**Pachetul dao** detine clasele care se ocupa cu accesul la baza de date. Are o clasa abstracta care a create prin reflection metode de inserare stergere updatare si interogare. Fiecare clasa care extinde aceasta clasa abstracta mai are propriile metode de access la baza de date specifice tabelului interogat.



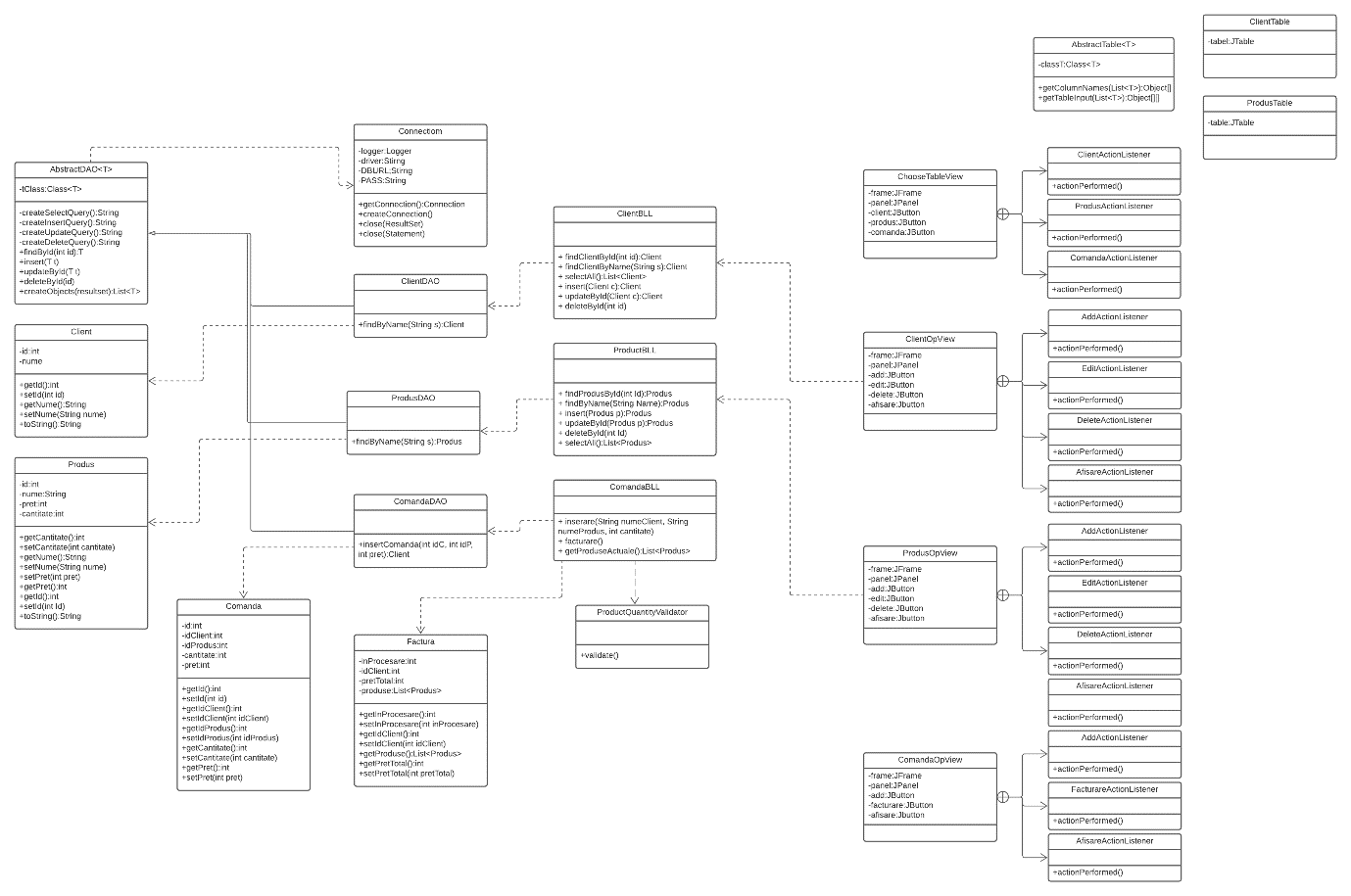
**Pachetul businessLogic** retine toata logica din spatele operatiei, creaza toata operatiile cu ajutorul claselor de access la date si valideaza imputurile.



**Pachetul presentation** detine 4 clase pentru fiecare fereastra si inca 3 pentru JTables, una dintre ele fiind abstracta pentru crearea tabelelor in mod abstract cu reflection.

## Diagrama de clase

Unified Modeling Language (prescurtat UML) este un limbaj standard pentru descrierea de modele și specificații software. Diagrama de clase UML este folosită pentru reprezentarea vizuală a claselor și a interdependențelor, taxonomiei și a relațiilor de multiplicitate dintre ele. Diagramele de clasă sunt folosite și pentru reprezentarea concretă a unor instanțe de clasă, așadar obiecte și a legăturilor concrete dintre acestea.



## Structuri de date

Structurile de date predefinite pe care le-am folosit sunt: ArrayList, List. List este folosita pentru rezultatele queriurilor, iar arraylist este folosita pentru a retine lista de produse de pe un bon.

De asemenea, clasele din Model pot fi văzte ca niste structure de date ce definesc atributele importante pentru fiecare obiect in parte, deoarece clasele din model definesc clasele concrete, palpabile, care exista si in realitate.

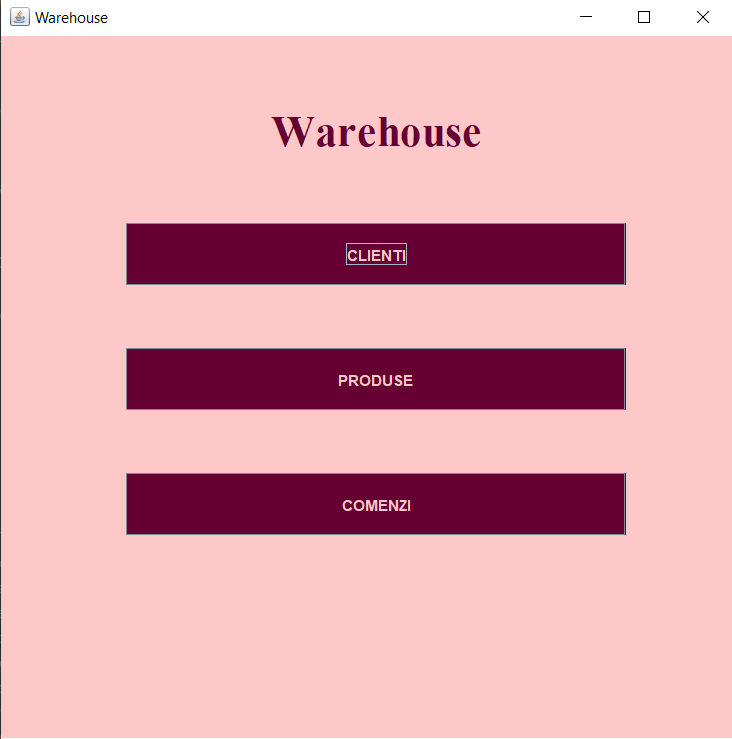
## Interfețe folosite/definite

Interfata gata definita pe care am folosit-o este List, care se poate găsi în biblioteca java util. O alta interfata predefinita folosita este ActionListener, ce se gaseste in biblioteca java.awt, pe care am implementat-o in clasele interne din presentationLayer.

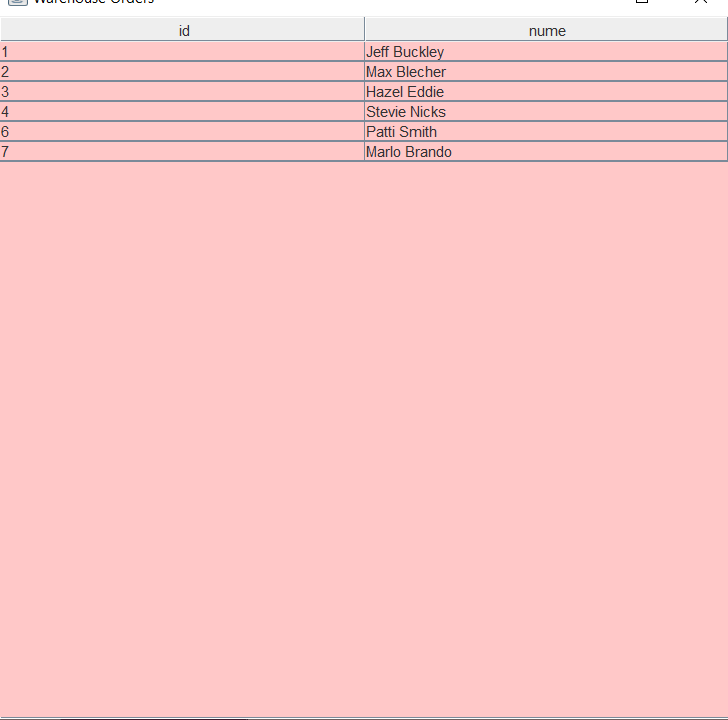
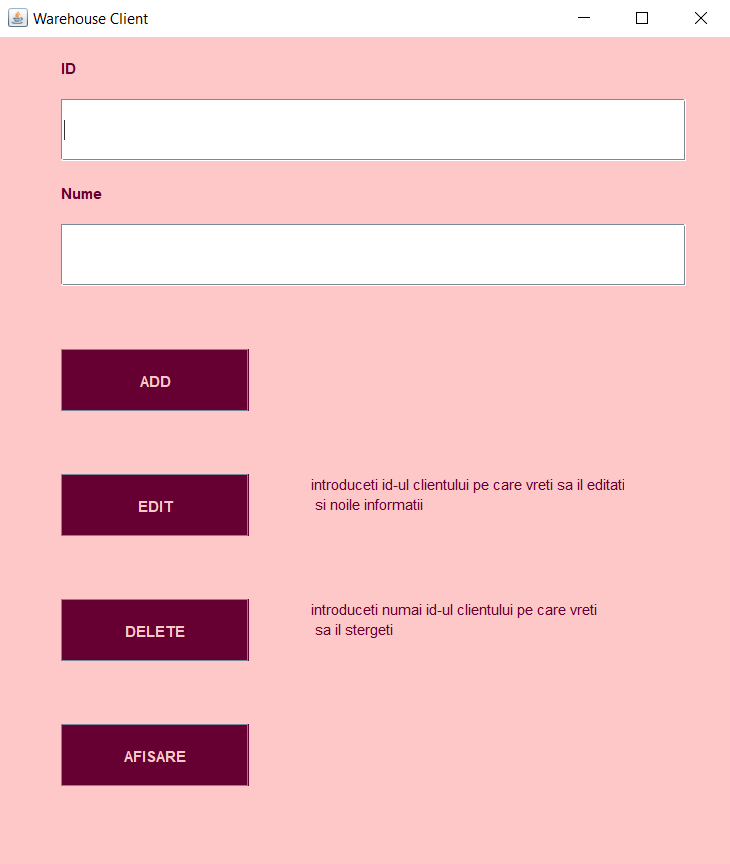
Am definit o interfață pentru validarea datelor, intrucat la o dezvoltare ulterioara ar putea fi nevoie de mai multi validatori pentru mai multe campuri. Momentan Validator este o interfata simpla ce are o singura metoda de validare si este implementata de o singura clasa ce valideaza daca este numarul necesar de produse in stoc.

## Interfața

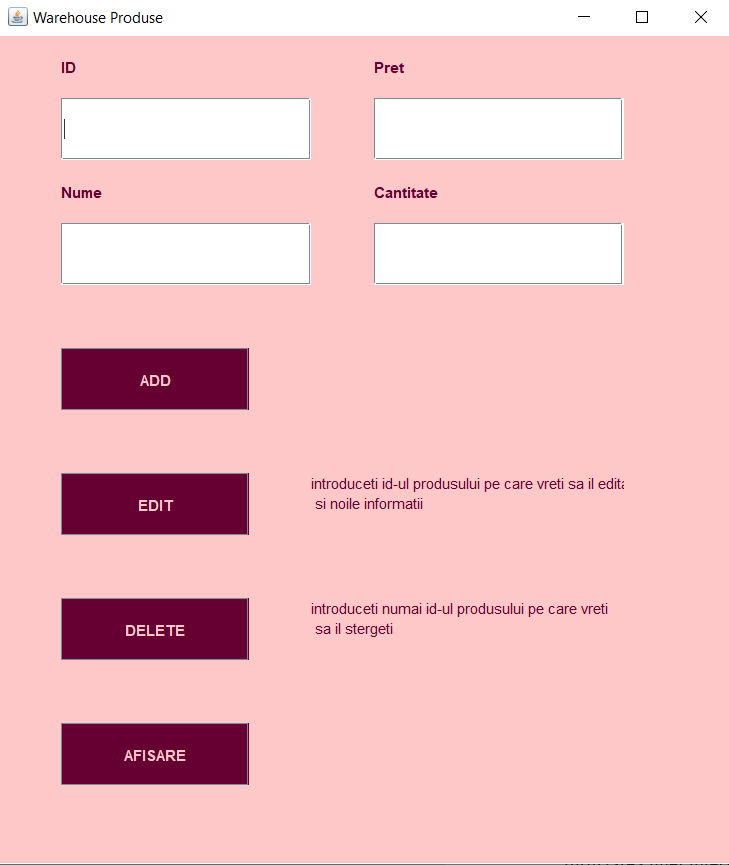
Am ales să implementez o interfata simpla și intuitivă, ușor de folosit de către user. Prima pagina are 3 butoane ce te redirectioneaza catre cele 3 pagini principale, pagina care se ocupa cu gestionarea clientilor, pagina ce se ocupa cu gestionarea produselor si pagina ce se ocupa cu crearea comenzilor.



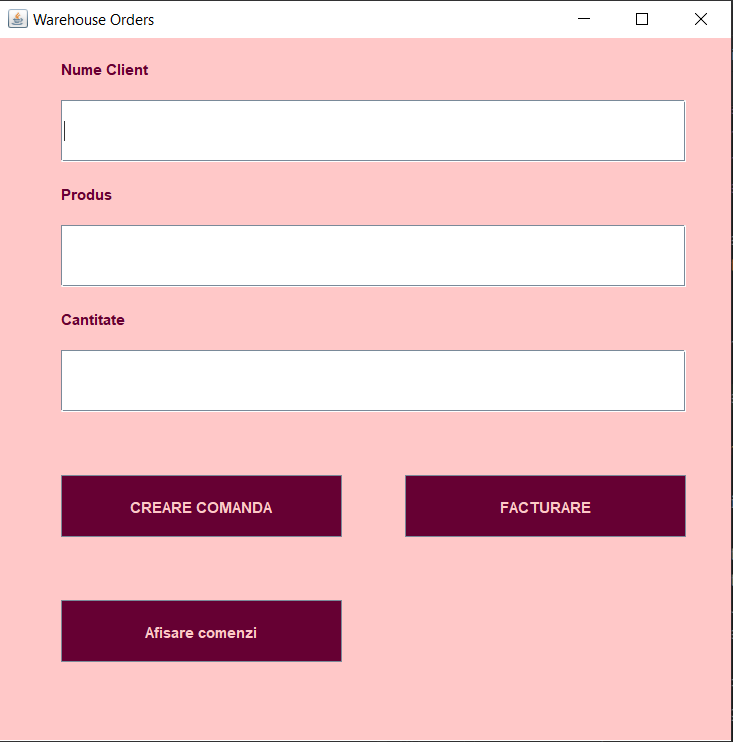
Pagina ce se ocupa cu gestionarea clientilor are 2 textfielduri pentru introducerea datelor legate de client si 4 butoane pentru manipularea datelor. Are in dreapta fiecarui buton cate o informatie ajutatoare legata de folosirea lor. Butonul add va adauga un client in baza de date, edit va updata in functie de id, delete va sterge in functie de id si afisare va afisa toti clientii cu ajutorul unui Jtable.



Meniul pentru gestionarea produselor arata astfel, are aceleasi butoane ca meniul pentru gestionarea clientilor dar are in plus campuri suplimentare, deoarece un produs are mai multe date de care este definit.



Meniul pentru comenzi are acum 3 campuri numele clientului, produsul si cantitatea si 3 butoane, de creare comanda, facturare si afisare ce este in “cosul de cumparaturi”.



Afisare comenzi va arata produsele care sunt pana acum comandate, creare comanda vac rea o comanda si Facturare va crea facture, un fisier txt pe care scrie fiecare produs si totalul.

Am ales să implementez interfața cu Java Swing. Swing este un set de instrumente widget GUI pentru Java. Face parte din Oracle Java Foundation Classes (JFC) - un API pentru furnizarea unei interfețe grafice de utilizator (GUI) pentru programele Java. De asemenea pentru a nu încarca threadul principal cu printarea în text fielduri am folosit SwingWorker care permite crearea unui thread worker pentru interfața care să o updateze la intervalul necesar.

# Implementare

**Pachetul dataModel**

Pentru toate clasele din acest pachet avem cate un tabel in mysql workbench cu fiecare atribut.

De aceea, pentru a crea obiectele cu ajutorul tehnicii reflection avem nevoie de gettere si settere pentru fiecare atribut si de cate un constructor gol pentru fiecare clasa.

**Clasa Client**

Clasa Client este folosită pentru a descrie clienții care stau la cozi, având atributele importante pentru procesare

**Atribute:**

* **private int id; -** folosit pentru a identifica mai bine clienții
* **private String nume;**

Toate atributele sunt **private**, folosind conceptul specific progrămarii orientate pe obiect de incapsulare a datelor.

**Metode:**

* **public int getId()**
* **public int getName()** – getter pentru nume si id
* **public int setName(String s) --**settere pentru nume si id
* **public void setId(int id)**

**Clasa Produs**

Clasa produs este clasa care retine datele specific ale unui produs acestea fiind id-ul numele, cantitatea existenta si pretul. Avem si un table in baza de date pentru acest obiect cu toate atributele lui, si cu aceleasi nume.

**Atribute:**

* **private int id;**
* **private String nume;**
* **private int pret;**
* **private int cantitate;**

**Metode:**

* **public int getId()**
* **public int getName()** – getter pentru nume si id
* **public int setName(String s) --**settere pentru nume si id
* **public void setId(int id)**
* **public int getCantitate();**
* **public void setCantitate();**
* **public int getPret();**
* **public void setIdpret (int id);**

**Clasa Comanda**

Este clasa care definește structura comenzi, ea are nevoie de un id pentru a pastra unicitatea unei chei primare in baza de date. in rest are datele specific unei comenzi, un client care a facut comanda, un produs, cantitatea de produse care a fost comandata si pretul total al comenzii care este calculate ca cantitate\*pret\_produs.

**Atribute:**

* **private int id;**
* **private int idClient;**
* **private int idProdus;**
* **private int cantitate;**
* **private int pret;**

**Metode:**

* **public int getId();**
* **public void setId(int id);**
* **public int getIdClient();**
* **public void setIdClient();**
* **public int getIdProdus();**
* **public void setIdProdus(int id);**
* **public int getCantitate();**
* **public void setCantitate();**
* **public int getPret();**
* **public void setIdpret (int id);**

**Clasa Factura**

Este o clasa pe care am ales eu sa o implementez pentru a putea retine mai multe obiecte intr-o comanda, mai multe produse. Avem nevoie de id-ul clientului pentru care se face factura deoarece nu ar putea avea 2 persoane cu 2 comenzi diferite aceiasi factura. De asemenea avem un flag de procesare, care ne va ajuta sa verificam daca este in procesare sa verificam si id-ul clientului care face urmatoarea comanda.

**Atribute:**

* **private int inProcesare = 0;**
* **private int idClient;**
* **private int pretTotal = 0;**
* **private List<Produs> produse = new ArrayList<Produs>();**

**Metode:**

* **getInProcesare();**
* **setInProcesare();**
* **getProduse();**
* **getIdClient();**
* **getPretTotal();**
* **setPretTotal();**

**Pachetul connection**

**Clasa Connection**

Este clasa care se ocupa cu conexiunea la baza de date din mysql si se foloseste de fiecare data cand vrea sa se accese baza de date cu un query.

**Atribute:**

* **private static final Logger LOGGER = Logger.getLogger(ConnectionFactory.class.getName());**
* **private static final String DRIVER = "com.mysql.cj.jdbc.Driver";**
* **private static final String DBURL = "jdbc:mysql://localhost:3306/depozit";**
* **private static final String USER = "root";**
* **private static final String PASS = "root";**

**Metode:**

* **private Connection createConnection()**
* **public static Connection getConnection()**
* **public static void close(Statement statement)**
* **public static void close(ResultSet resultSet)**

**Pachetul dao**

**Clasa AbstractDAO**

Este clasa abstracta care creeaza prin reflection toate interogarile necesare pentru a accesa baza de date.

**Metode:**

* **protected String createSelectQuery(String field) –** este metoda care se ocupa cu crearea stringului ce reprezinta interogarea in functie de un anumit field specificat. de exemplu daca fieldul este id va fi asa “select \* from table where id=?”
* **private String createInsertQuery()** este metoda care se ocupa cu crearea stringului ce reprezinta inserarea in tabel.
* **private String createDeleteQuery(String field)** este metoda care se ocupa cu crearea stringului ce reprezinta stergerea in functie de un anumit field specificat. de exemplu daca fieldul este id va fi asa “delete from table where id=?”
* **private String createUpdateQuery(String str)** este metoda care se ocupa cu crearea stringului ce reprezinta updatarea unei tuple din table in functie de un field.
* **public T findById(int id) –** este metoda care se ocupa cu executarea queryului construit mai sus, si se cauta in functie de id.
* **protected List<T> createObjects(ResultSet resultSet) -**este metoda care cu ajutorul tehnicii reflection creaza o lista de obiecte dintr-un rezultat la o interogare. se apeleaza constructorul gol, apoi se invoca getterele si stterele cu fiecare tuple gasita in rezultat.
* **public T insert(T t) throws DuplicateEntryException –** este metoda care executa inserarea construita mai sus si poate arunca o exceptie in cazul in care cheia primara exista deja. Cu ajutorul tehnicii reflection se completeaza stringul create mai sus.
* **public void deleteById(int id) throws NoSuchIdException –** este metoda care executa stergerea construita mai sus. Poate sa arunce o exceptie daca nu exista id-ul specificat.
* **public T updateById(T t) throws NoSuchIdException –** se executa updatarea create mai sus si se completeaza campurile cu campurile lui T
* **public List<T> selectAll()**

**Clasa ClientDAO**

Aceasta clasa extinde clasa abstracta descrisa mai sus si ii aduce in plus cateva metode specific tabelului client, care se vor folosi mai departe.

**Metode:**

* **public Client findByName(String name)** – in loc sa se caute dupa un id, se cauta in nume specific in tabela bazei de date.

**Clasa ProdusDAO**

Aceasta clasa extinde clasa abstracta descrisa mai sus si ii aduce in plus cateva metode specific tabelului client, care se vor folosi mai departe.

**Metode:**

* **public Produs findByName(String name)** – in loc sa se caute dupa un id, se cauta in nume specific in tabela bazei de date.

**Clasa ComandaDAO**

Aceasta clasa extinde clasa abstracta descrisa mai sus si ii aduce in plus cateva metode specific tabelului client, care se vor folosi mai departe.

**Metode:**

* **public Comanda insertComanda( int idClient, int idProdus, int cantitate, int pret)** – se insereaza toate caracteristicile in afara de id, care se va genera automat intrucat in baza de date id ul este auto incrementat.

**Pachetul businessLogic**

**Clasa Client BLL**

Este clasa care se ocupa cu logica din spatele aplicatiei pentru partea de gestionare a clientilor si operatiile pe baza de date.

**Metode:**

* **public Client findClientByName(String s) throws NoSuchNameException**
* **public List<Client> selectAll()**
* **public Client insert(Client c) throws DuplicateEntryException**
* **public Client updateById(Client c) throws NoSuchIdException**
* **public void deleteById(int id) throws NoSuchIdException**

**Clasa ProdusBLL**

Este clasa care se ocupa cu logica din spatele aplicatiei pentru partea de gestionare a produselor si operatiile pe baza de date.

**Metode:**

* **public Produs findProdusByName(String s) throws NoSuchNameException**
* **public List<Produs> selectAll()**
* **public Produs insert(Produs c) throws DuplicateEntryException**
* **public Produs updateById(Produs c) throws NoSuchIdException**
* **public void deleteById(int id) throws NoSuchIdException**

**Clasa ComandaBLL**

Este clasa care se ocupa cu logica din spatele aplicatiei pentru partea de gestionare a comenzilor si operatiile pe baza de date.

**Metode:**

* **public Comanda inserare(String numeClient, String numeProdus, int cantitate) throws NoSuchIdException, NoSuchNameException, NoSuchProductException, ProduseInsuficienteException, FacturaNechitateException** – aceasta metoda cauta prima oara clientul produsul iar apoi verifica daca mai exista cantitatea necesara din produsul specificat. Daca nici una dintre aceste conditii nu e indeplinita, sau daca este alt client care vrea sa faca comanda se arunca exceptie, in caz contrar se creaza o comanda si se adauga produsul cu cantitatea dorita in lista de produse din factura.
* **public void facturare() –** se vor scrie intr-un fisier txt toate produsele din comanda si se va reinitializa factura.
* **public List<Produs> getProduseActuale()**

**Interfata Validator**

Este descrisa o interfata pentru posibile dezvoltari ulterioare, momentan are o singura metoda de validare a numarului de produse.

**Clasa ProductQuantityValidator**

Implementeaza interfata de mai sus pentru clasa produs.

**Pachetul presentation**

**Clasa AbstractTable<T>**

Este clasa care creeaza cu ajutorul tehnicii reflection headerul si interiorul unui table in functie de o Lista de obiecte.

**Metode:**

* **public Object[] getColumnsNames(List<T> l)** – este metoda ce creaza numele fieldurilor din table
* **public Object[][] getTableInput(List<T> l)** – este metoda ce creaza interiorul tabelului.

**Clasa ChooseTableView**

Este clasa ce creaza frameul principal pentru alegere a sectiuniii in care se vrea sa se intre

**Atribute:**

* **private JButton client**
* **private JButton produse**
* **private JButton comenzi**

Are cate o clasa interna care implementeaza ActionListener pentru fiecare button si deschide frameul necesar.

**Clasa ClientOperationsView**

Este clasa care descrie frameul pentru gestionarea clientilor, are cate un buton si cate o clasa interna corspunzatoare de tip ActionListener pentru fiecare operatie: adaugare, editare stergere si inserare.

Mai avem cate o clasa de acest tip si pentru produse si pentru comenzi. Cea de produse este identica, are numai mai multe textFielduri deoarece un produs are mai multe atribute si clasa de comanda are numai 3 actiuni posibile: adaugare de comanda, facturare si afsiarea cosului de cumparaturi.

# Rezultate

La fiecare inserarea updatare sau stergere se poate consulta atat jTableul caracteristic paginii cat si baza de date pentru a garanta siguranta si corectitudinea datelor inserate sau sterse.

# Concluzii

În concluzie realizarea acestui proiect a fost foarte utilă , amintind de o mare parte din principalele concepte ale programării orientate pe obiecte învățate semestrul trecut.

Am învățat să lucrez cu tehnica Reflection care este o tehnica utila mai ales cand se vor aceleasi functionalitati pentru mai multe clase si am invatat sa lucrez mai bine cu abstractizarea datelor.

O dezvoltare ulterioară ar fi adaugarea unui email clientului plus un validator pentru email, sa se poata updata produsele prin incrementarea cantitatii, si sa se creeze un pop-up cu factura. De asemenea se mai poate adauga un cont bancar pentru clienti si sa poata face comanda numai daca au destui bani in cont.

# Bibliografie

* <https://lucid.app/documents>
* <https://app.diagrams.net/>
* <https://docs.oracle.com/javase/tutorial/uiswing/components/table.html>
* <https://stackoverflow.com/questions/27582757/catch-duplicate-entry-exception>
* <https://docs.oracle.com/javase/tutorial/uiswing/components/table.html>
* ASSIGNMENT\_3\_SUPPORT\_PRESENTATION