

**ORDER MANAGEMENT**

# Obiectivul temei

Principalul obiectiv al acestei teme a fost implementarea unui sistem care procesează comenzile clienților pentru un depozit. În vedere acestui scop, au existat multiple obiective secundare precum:

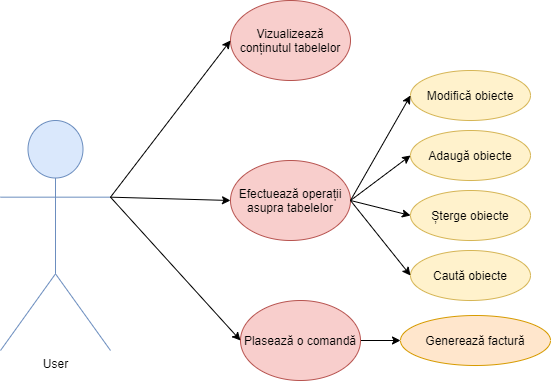
* Crearea unor baze de date pentru stocarea clienților, a produselor și a comenzilor efectuate
* Proiectarea unui sistem prin care se va plasa corect o comandă, verificându-se existența produselor pe stoc
* Dezvoltarea unei aplicatii grafice destinată fiecărui tip de utilizator, pentru a ușura efectuarea unor acțiuni asupra bazei de date
* Generarea unei facturi la fiecare plasare a unei comenzi

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Este necesară stocarea tuturor produselor, tuturor clienților și tuturor comenzile într-o bază de date pentru a nu se pierde inforamțiia caracteristică a acestora și pentru a se realiza o clasificare. Cerințele funcționale ale acestei teme sunt reprezentate de operațiile de ștergere, de modificare, de selectare sau de adăugare a noi obiecte în fiecare dintre aceste tabele.

Acest proiect are rolul de a simula functionarea unei depozit pe baza cerințelor funcționale enumerate mai sus. Având în vedere complexitatea sistemului este necesară o modelare obiectuală. Fiecărui tip de obiect îi va corespunde un cod unic de indentificare în tabela destinată lui în baza de date.

Problema poate fi transpusă în mediul real prin asocierea sistemului cu funcționarea unui depozit. Clienții care vor putea realiza comenzi îsi vor salva anumite date specifice pentru a realiza o comandă. La plasarea fiecărei comenzi este necesară compararea tabelelor stocate pentru a verifica existența clientului în baza de date a depozitului. Având în vedere faptul că este imposibilă comandarea unui produs inexistent, atunci când clientul dorește să comande un produs, este necesară verificarea apartenenței produsului în baza de date. Având ca scop o funcționarea cât mai realistă, pe lângă verificarea apartenenței produsulu, mai este necesară și compararea cantității dorite de către client cu cea de pe stocul curent. În urma verificărilor efectuate, daca toate datele introduse vor corespunde cerințelor, comanda va fi plasată și se va genera automat o factură în format PDF care va conține numărul de facturare, numele clientului care a efectuat comanda și numele produsului comandat.

Diagrama use-case a sistemului:

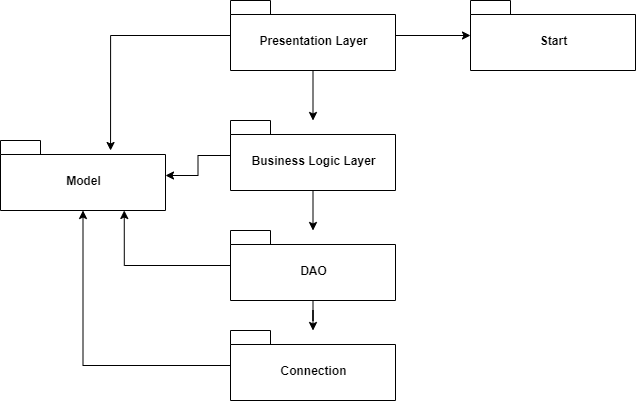
# 

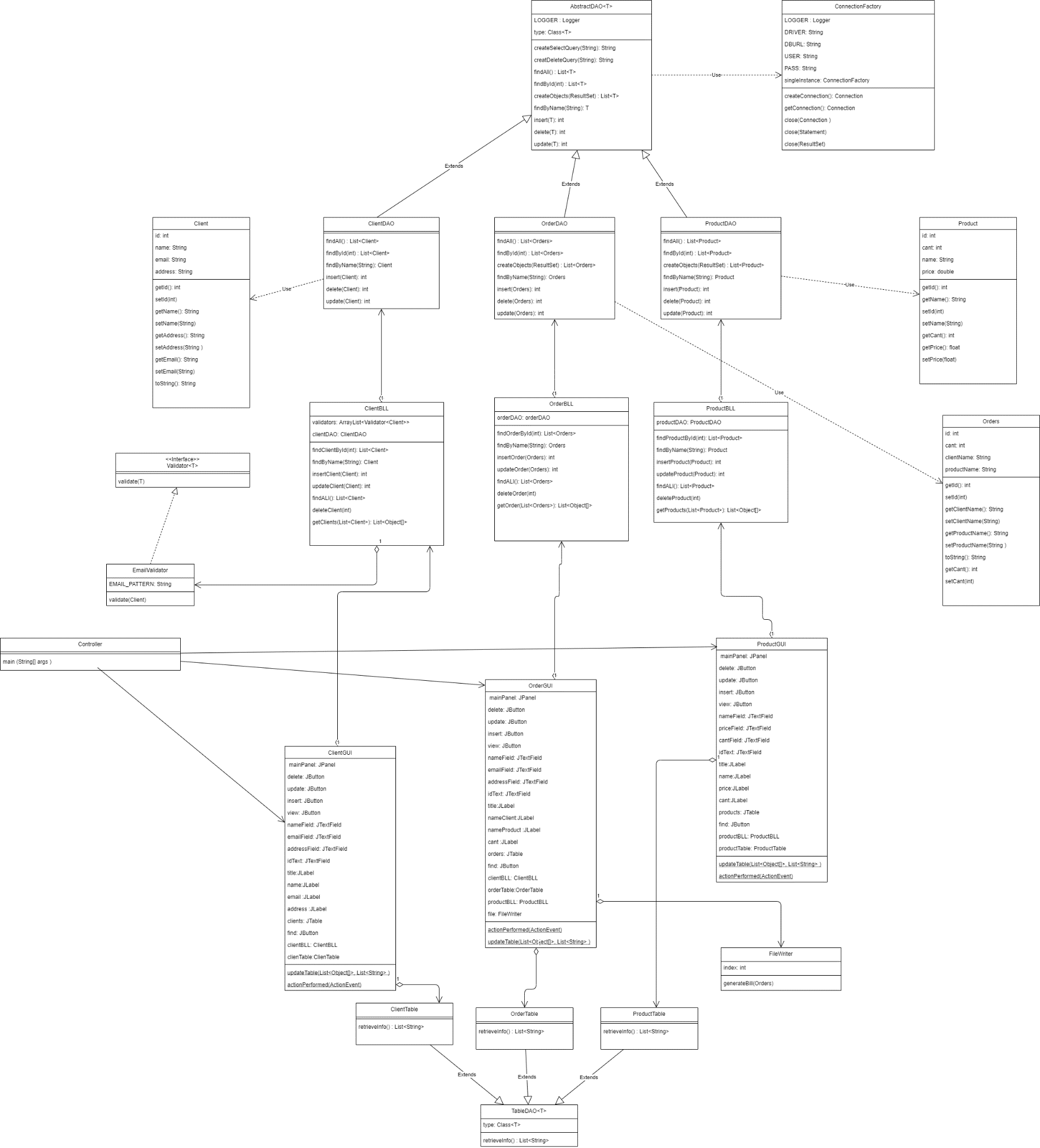
# Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

Pentru stocarea produselor, clienților care vor putea realiza comenzi și a comenzilor propriu-zise am utilizat baze de date relaționale. Fiecare dintre datele introduse în tabele vor avea un cod de identficare unic în cadrul tabelei pentru a diminua riscul de suprascriere a datelor.

Proiectarea sistemului a fost realizată după modelul layered architecture pattern. Componentele din această arhitectură sunt organizate în layere orizontale, fiecare layer indeplinind un rol specific în cadrul aplicației: presentation layer în care sunt implementate clase legate de GUI, acest pachet are rolul de manipula interfața grafică unde vor fi prezentate tabelelepentru fiecare tip de obiect, business logic layer unde sunt implementate clasele care conțin logica aplicației, acest pachet va realiza legătura între presentațion layer și DAO layer (se vor implementa regulile de funcționare ale aplicației), data access layer în care se vor implementa clasele care vor acesa și realiza operații baza de date și tabelele acesteia, model layer în care se vor implementa clasele corespunzătoare fiecărui tip de dată pe care aplicația îl va modela, connection layer în care se vor implementa clasele care vor realiza conexiunea la baza de date și vor crea canalul de comunicare între aplicație și server și start layer unde vor fi implementate metode care realizează operațiile tehnice ale aplicației (generarea facturii, generarea headerelor pentru interfața grafică).

Interacționarea cu baza de date a fost implementată print tehnici de reflecție, astfel în pachetul DAO am declarat o clasă abstractă care va construi în mod dinamic prin această tehincă interogările dorite pentru fiecare tabel. Această tehnică face posibilă inspectarea claselor, interfețelor, câmpurilor și metodelor în timpul rulării, fără a cunoaște numele claselor sau metodelor la compilare, de asemenea este posibilă instanțierea unor obiecte noi, invocarea de metode și chiar obținerea sau setarea unor valori ale câmpurile obiectelor. Această tehnică a fost folosită și pentru a obține în mod dinamic headerele pentru tabelele care vor fi transpuse în interfața grafică și care vor conține datele salvate în baza de date.

Diagrama de pachete:

 Diagrama de clase:

## Implementare

## Pachetul BLL

În acest pachet au fost implementate clasele care vor defini modul logic de funcționare al întregii aplicații. Acest pachet conține trei clase, specifice fiecărui tip de clasă și un subpachet de validare a datelor.

### Subpachetul Validators

În acest pachet au fost implementate clasele care vor verifica corectitudinea anumitor date introduse de către utilizator.

#### Interfața Validator

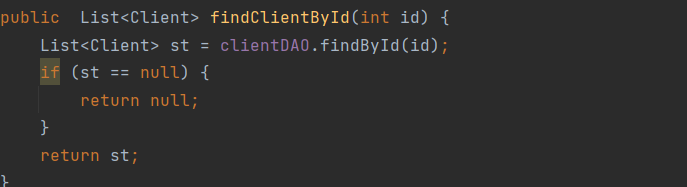
În această clasă este definită o singură metodă numită validate și care are un singur parametru generic, putând fi implementată de orice altă clasă care implementează interfața pentru a valida orice tip de dată dorită.

#### Clasa EmailValidator

Această clasă implementează interfața Validator pentru a verifica corectitudinea structurii emailului introdus pentru un nou client ce va urma a fi adăugat în baza de date. Este definit un singur atribut de clasă care reprezintă pattern-ul emailului. Este implementată metoda de validare care verifică dacă obiectul din clasa Client primit ca argument conține un email care corespunde cu pattern-ul dorit, în caz de necorespondență va fi aruncată o excepție care va semnala incorectitudinea datelor introduse.

### Clasa ClientBLL

În această clasă sunt implementate metode care definesc funcționalitățile clienților. La nivel de clasă sunt definite două atribute: o listă care conține validatori pentru obiectele de tip client și un obiect de tipul ClientDAO pentru a se putea realiza legătura cu partea de accesare a dateor, corespunzătoare pachetului DAO. Metodele implementate în această clasă corespund operațiilor care se vor putea realiza asupra tabelelor unde sunt stocate informațiile despre clienți. Metoda findClientById va apela metoda findById din clasa ClientDAO care va returna obiectele de tip Client care au id-ul corespunzător celui trimis ca parametru, în cazul în care nu există obiecte cu acest id se va returna null.



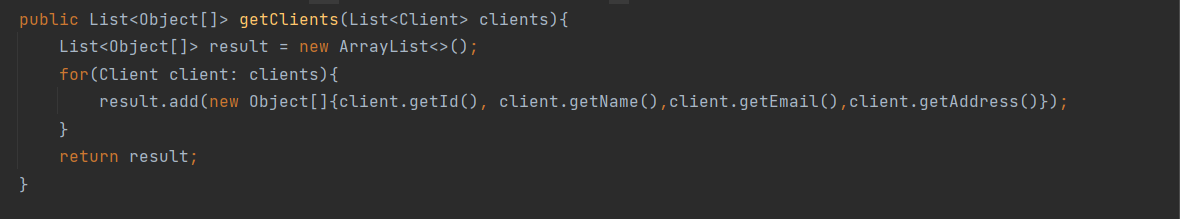
Metoda findByName va funcționa asemănător metodei findById, doar că de această dată se va returna o listă ce va conține toate obiectele de tipul Client care vor avea numele specificat în Stringul trimis ca parametru.

Metoda insertClient va efectua o verificare asupa corectitudinii datelor obiectului de tip Cient transmis ca parametru, iar dacă datele transmise au respectat pattern-ul impus se va apela metoda din clasa ClientDAO care va adăuga un nou client în baza de date.

Metoda updateClient are o funcționalitate asemănătoare cu insertClients, doar că de această dată se va apela metoda de update din ClientDAO care va modifica datele clientului, transmis ca parametru, în baza de date.

Metoda deleteClient va primi ca parametru un id specific unui client care va urma a fi șters din baza de date.

Metoda findAll are rolul de a returna o listă de clienți care va conține toti clienții stocați în baza de date.

Metoda getClients are rolul de a converti o listă formată din Clienți într-o matrice de tip Object, această metodă are rolul de a furniza datele corespunzătoare tabelului afișat în interfața destinată clienților.

### Clasa ProductBLL

În această clasă sunt implementate metode care definesc funcționalitățile produselor. La nivel de clasă este definit un singur atribut, un obiect de tipul ProductDAO pentru a se putea realiza legătura cu partea de accesare a dateor, corespunzătoare pachetului DAO. Metodele implementate în această clasă corespund operațiilor care se vor putea realiza asupra tabelelor unde sunt stocate informațiile despre clienți. Metoda findProductById va apela metoda findById din clasa ProductDAO care va returna obiectele de tip Product care au id-ul corespunzător celui trimis ca parametru, în cazul în care nu există obiecte cu acest id se va returna null.

Metoda findByName va funcționa asemănător metodei findById, doar că de această dată se va returna o listă ce va conține toate obiectele de tipul Product care vor avea numele specificat în Stringul trimis ca parametru.

Metoda insertProduct va apela metoda din clasa ProductDAO care va adăuga un nou produs în baza de date. Metoda updateProduct are o funcționalitate asemănătoare cu insertProduct, doar că de această dată se va apela metoda de update din ProductDAO care va modifica datele produsului, transmis ca parametru, în baza de date.

Metoda deleteProduct va primi ca parametru un id specific unui produs care va urma a fi șters din baza de date.

Metoda getProducts are rolul de a converti o listă formată din produse într-o matrice de tip Object, această metodă are rolul de a furniza datele corespunzătoare tabelului afișat în interfața destinată produselor.

Metoda findAll are rolul de a returna o listă de produse care va conține toate produsele din baza de date.

### Clasa OrderBLL

În această clasă sunt implementate metode care definesc funcționalitățile comenzilor. La nivel de clasă este definit un singur atribut, un obiect de tipul OrderDAO pentru a se putea realiza legătura cu partea de accesare a dateor, corespunzătoare pachetului DAO. Metodele implementate în această clasă corespund operațiilor care se vor putea realiza asupra tabelelor unde sunt stocate informațiile despre comenzi. Metoda findOrderById va apela metoda findById din clasa ProductDAO care va returna obiectele de tip Order care au id-ul corespunzător celui trimis ca parametru, în cazul în care nu există obiecte cu acest id se va returna null.

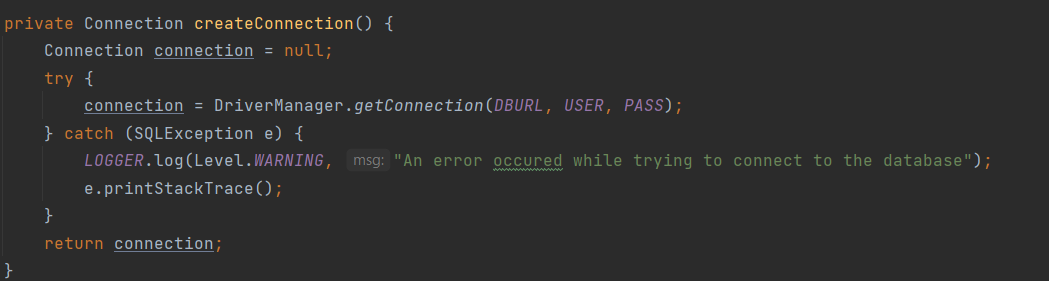
Metoda insertOrder va apela metoda din clasa OrderDAO care va adăuga un nou produs în baza de date. Metoda updateOrder are o funcționalitate asemănătoare cu insertOrder, doar că de această dată se va apela metoda de update din OrderDAO care va modifica datele produsului, transmis ca parametru, în baza de date.

Metoda deleteOrder va primi ca parametru un id specific unei comenzi care va urma a fi ștearsă din baza de date.

Metoda getOrders are rolul de a converti o listă formată din comenzi într-o matrice de tip Object, această metodă are rolul de a furniza datele corespunzătoare tabelului afișat în interfața destinată comenzilor.

## Pachetul Connecțion

În acest pachet este implementată funcția care va realiza calea de comunicare între aplicație și baza de dare. Clasa ConnectionFactory are ca atribute de clasă date specifice conexiunii la server. Metodele implementate în această clasă crespund conectării sau deconectării la serverul unde este implementată baza de date.



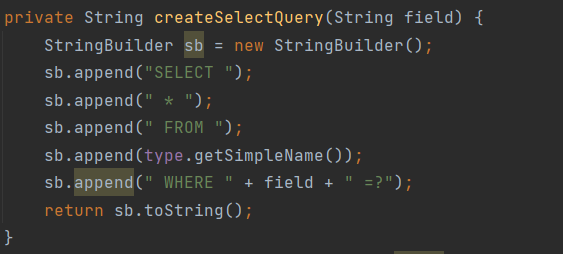
## Pachetul DAO

În aces pachet sunt implementate metodele care vor accesa datele stocate în baza de date și vor efectua operații asupra acestora.

### Clasa AbstractDAO

Această clasă este implementată prin tehnici de reflexie și are scopul de a defini operațiile propriu zise care vor fi realizate asupra datelor. Unul dintre atributele de clasă reprezintă tipul clasei care va moșteni clasa.

Metodele createSelectQuery și createDeleteQuery vor returna câte un Sting corespunzător fiecărui tip de interogare dorită, interogare care va urma a fi executată.



Metoda findAll va construi un query care va selecta din tabela dorită toate elementele. În urma executării acestui query informațiile extrase din baza de date sunt returnate într-o variabilă de tip SetResult, iar pentru parsarea și convertirea obiectelor la tipul dorit se va apela metoda createObject.

Metodele findById și findByName vor apela metoda createSelectQuery, dar cu parametrii corespunzători fiecărui tip de selecție. În urma executării queryului corespunzător rezultatul va fi returnat într-o variabilă de tipul ResultSet , iar pentru parsarea și conversia obiectelor la tipul dorit se va apela metoda createObject. În cazul în care în baza de date nu au fost găsite date corespunzătoare condtițiilor impuse, aceste metode vor returna null.

Metoda createObject va returna o listă de obiecte corespunzătoare clasei prin parsarea parametrului de tipul ResultSet. Această metodă va accesa câmpurile clasei dorite prin reflexie și va crea dinamic noi obiecte din clasa corespunzătoare pe care le va adăuga în listă.

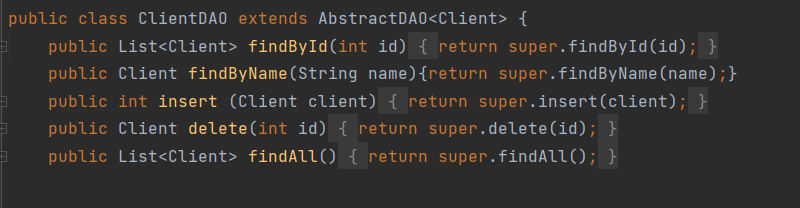
Metoda insert va construi dinamic un query în urma executării căruia se va adăuga în baza de date datele obiectului primit ca parametru.

Metoda delete va apela metoda createDeleteQuery și prin executarea query-ului corespunzător va șterge din baza de date obiectul al cărui id a fost primit ca parametru.

Metoda update va construi dinamic un query în urma executării căruia se va căuta în baza de date obiectul dorit după id și se vor modifica datele sale, fiind înlocuite cu datele obiectului primit ca parametru.

### Clasa ClientDAO

Această clasă moștenește clasa AbstractDAO și va apela metodele clasei părinte care prin reflexie se vor executa pentru tipuri de date de tipul Client.



### Clasa ProductDAO

### Această clasă moștenește clasa AbstractDAO și va apela metodele clasei părinte care prin reflexie se vor executa pentru tipuri de date de tipul Product.

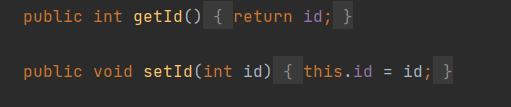
### Clasa OrderDAO

Această clasă moștenește clasa AbstractDAO și va apela metodei clase părinte care prin reflexie se vor executa pentru tipuri de date de tipul Order.

## Pachetul Model

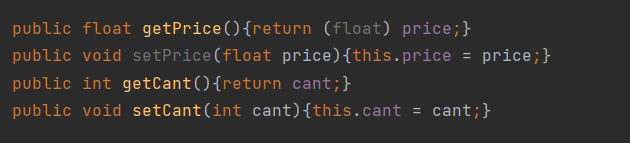
În acest pachet sunt implementate clasele corespunzătoare tipurilor de date modelate de către aplicație

### Clasa Client

Această clasă modelează obiectele de tipul clienților care sunt stocate în baza de date, definindu-le caracteristicile: id, nume, email, adresă. Metodele acestei clase sunt metode de obținere a informației despre obiectele de tipul Client și metode de setare ale câmpurilor caracteristice acestui tip de obiecte.

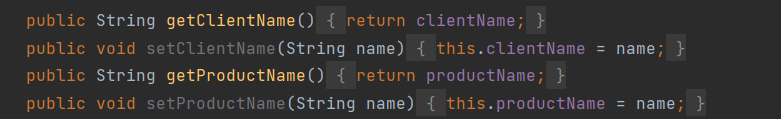
### Clasa Product

Această clasă modelează obiectele de tipul produselor care sunt stocate în baza de date, definindu-le caracteristicile: id, nume, preț, cantitate. Metodele acestei clase sunt metode de obținere a informației despre obiectele de tipul Product și metode de setare ale câmpurilor caracteristice acestui tip de obiecte.



### Clasa Order

### Această clasă modelează obiectele de tipul comenzilor care sunt stocate în baza de date, definindu-le caracteristicile: id, numele clientului care a efectuat comanda, numele produsului comandat, cantitatea de produs comandată. Metodele acestei clase sunt metode de obținere a informației despre obiectele de tipul Order și metode de setare ale câmpurilor caracteristice acestui tip de obiecte.



## Pachetul Presentațion

În cadrul acestui pachet este implementată interfața grafică pentru fiecare tip de obiect.

### Clasa ClientGUI

În această clasă se va implementa interfața destinată obiectelor de tip client și se vor seta acțiunie efectuate la apăsarea butoanelor definite.

### Clasa OrderGUI

În această clasă se va implementa interfața destinată obiectelor de tip order și se vor seta acțiunile efectuate la apăsarea bunoanelor definite.

### Clasa ProductGUI

În această clasă se va implementa interfața destinată obiectelor de tip prodct și se vor seta acțiunile efectuate la apăsarea bunoanelor definite.

### Clasa Controller

În această clasă este definită metoda main care va da start întregii aplicații.

## Pachetul Start

În acest pachet sunt definite clasele care au rol tehnic în cadrul aplicației.

### Clasa TableDAO

Această clasă este construită prin tehnici de reflexie și are o singură metodă care va construi o listă de stringuri ce va reprezenta headerul pentru tabelele afișate în interfața grafică.

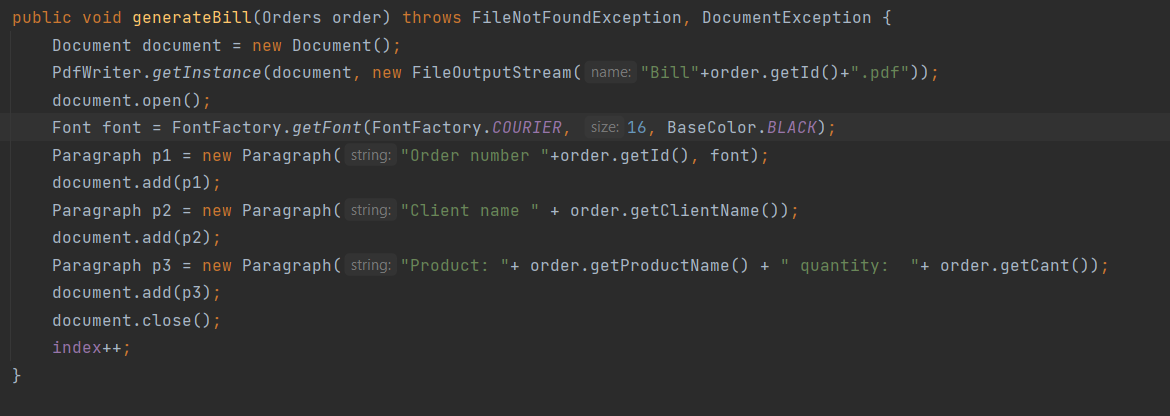


### Clasele ClientTable, ProductTable și OrderTable

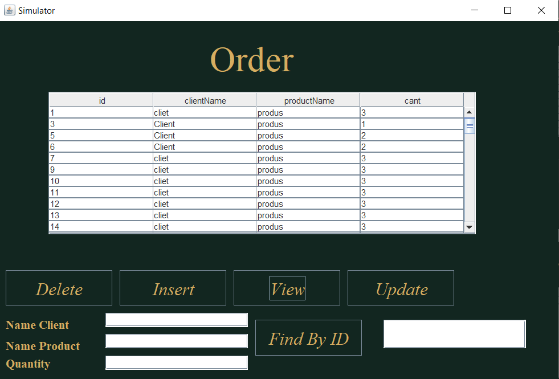
Aceste clase vor moșteni clasa TableDAO.

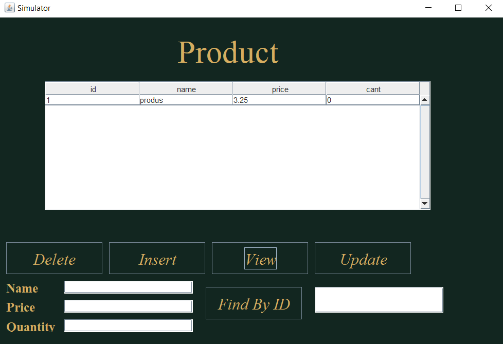
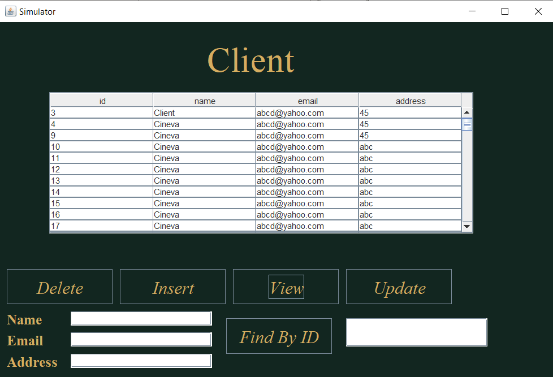
### Clasa FileWriter

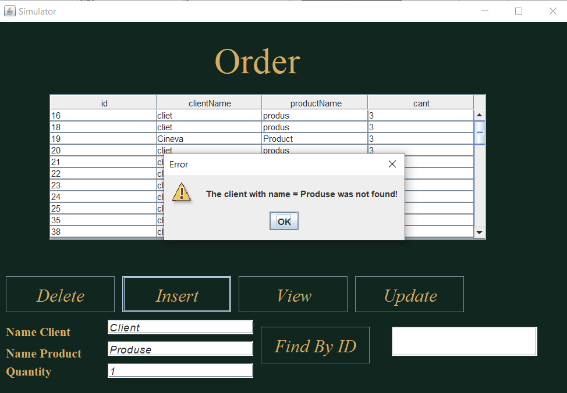
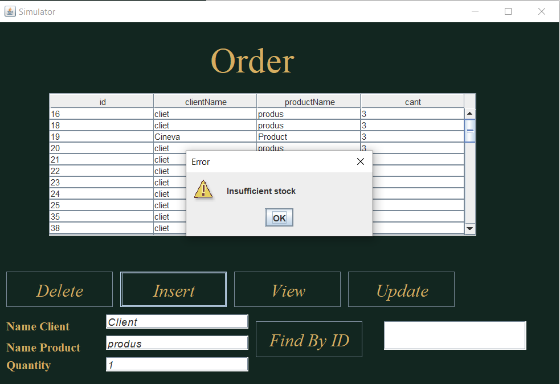
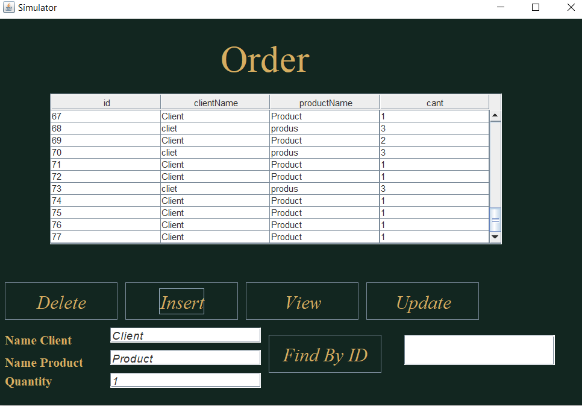
În această clasă se va genera factura corespunzătoare fiecărei comenzi prin metoda generateBiLL care va crea un nou document de tip PDF în care se vor scire datele obiectului d tip Order primit ca parametru.



# Testare

Vizualizarea datelor:



Efectuarea unei comenzi:

# Concluzii

În cadrul acestei teme s-a aprofundat notiuni caracteristice tehnicilor de programare precum tehnicile de reflexie, structrarea unui program pe layere etc.

# Bibliografie

**Connect to MySql from a Java application**

https://www.baeldung.com/java-jdbc

http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/

**Layered architectures**

https://dzone.com/articles/layers-standard-enterprise

**Reflection in Java**

http://tutorials.jenkov.com/java-reflection/index.html

**Creating PDF files in Java**

https://www.baeldung.com/java-pdf-creation

**JAVADOC**

https://www.baeldung.com/javadoc

SQL dump file generation

https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html