DOCUMENTATIE

Tema 3

NUME STUDENT:BIRISAN DENISA OPHELIA

GRUPA:30226

# 

# CUPRINS

[1. Obiectivul temei 3](file:///C:\Users\Asus\Downloads\PT2021-2022_Documentation_Template%20(2).doc#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](file:///C:\Users\Asus\Downloads\PT2021-2022_Documentation_Template%20(2).doc#_Toc95297886)

[3. Proiectare 3](file:///C:\Users\Asus\Downloads\PT2021-2022_Documentation_Template%20(2).doc#_Toc95297887)

[4. Implementare 3](file:///C:\Users\Asus\Downloads\PT2021-2022_Documentation_Template%20(2).doc#_Toc95297888)

[5. Rezultate 3](file:///C:\Users\Asus\Downloads\PT2021-2022_Documentation_Template%20(2).doc#_Toc95297889)

[6. Concluzii 3](file:///C:\Users\Asus\Downloads\PT2021-2022_Documentation_Template%20(2).doc#_Toc95297890)

[7. Bibliografie 3](file:///C:\Users\Asus\Downloads\PT2021-2022_Documentation_Template%20(2).doc#_Toc95297891)

# 1.Obiectivul temei

Obiectul principal il constituie implementarea unei baze de date simplificate a unui magazin. In acest sens, am considerat cele 3 tabele: Client, Produs, Comanda, care descriu cele 3 entitati implicate in procesul de cumparare. Obiectivele secundare sunt urmatoarele:

* Implementarea operatiei de adaugare in tabel
* Implementarea operatiei de stergere din tabel
* Implementarea operatiei de updatare element din tabel
* Implementarea operatiei de afisare toate elemente din tabel
* Implementarea unei interfete grafice cu userul
* Implementarea operatiei de realizare a unei comenzi cu modificarile de rigoare aduse tabelelui Produs (decrementare stoc)
* Generarea javadoc
* Generarea unui fisier SQL de creare tabele

Toate aceste functionalitati vor fi descrise in detaliu in capitolul 3.

Userul va putea interactiona cu baza de date prin intermediul unei interfete grafice, care ii va permite sa selecteze operatia, rezultatele aparand in casetele corespunzatoare.

# 2. Analiza problemei, modelare, scenarii, cazuri de utilizare

Pentru a putea implementa operatiile, am utilizat tehnica Reflection: am implementat operatiile add,insert,update,findAll generalizat, in clasa AbstractDAO, apoi le-am apelat pentru tipul de obiect necesar noua (Client,Produs).

**2.1 Considerente teoretice clasa de tip Singleton**

O clasa Singleton e o clasa care poate avea un singur obiect(instanta) in acelasi timp. Dupa prima instantinere, daca mai incercam o data, noua instanta va pointa la prima instanta creata. Deci, orice modificari facem la orice variabila din interiorul clasei prin orice instanta, modifica aceleasi atribute si e vizibila prin orice instanta.  
 Puncte cheie:  
1. Constructorul trebuie sa fie privat  
2.Se va scrie o metoda statica care are ca return type obiectul acestei clase (Connection pt noi)=>conceptul de LAZY INITIALIZATION  
3. Instanta trebuie sa fie unica si sa fie stocata ca un atribut private static.

De ce avem nevoie de o clasa Singleton?

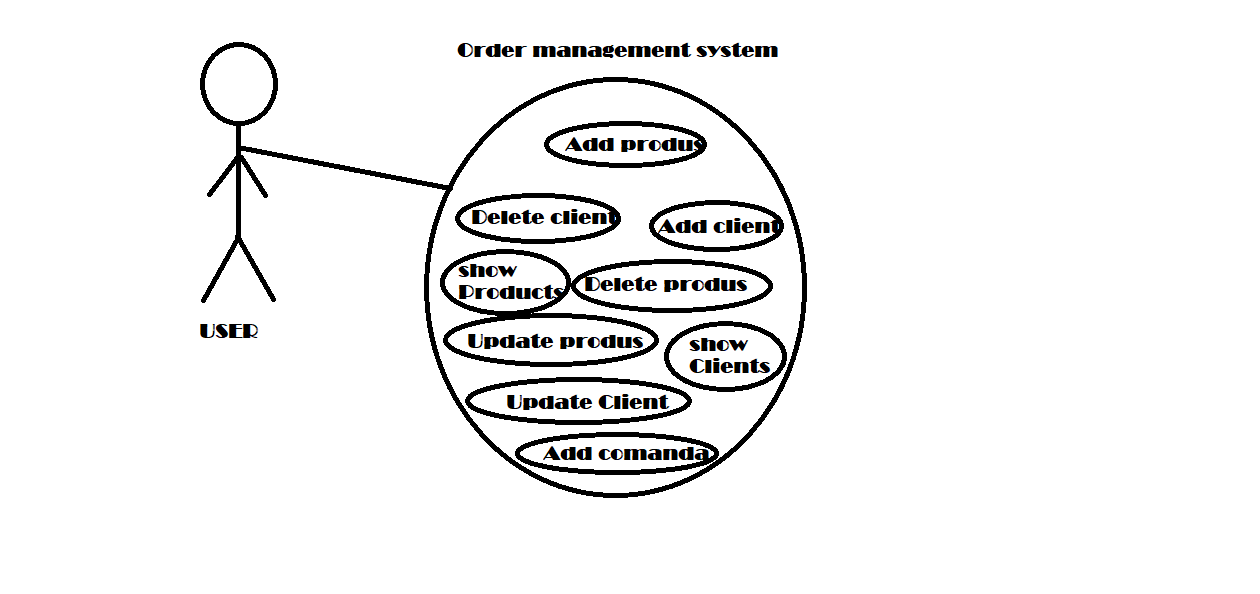
In principal, pentru ca vrem sa restrangem numarul de obiecte create la unul singur(folosit pt multithreading si database connection in Java). Clasa Singleton se asigura ca la un moment de timp un singur thread/o singura conexiune poate accesa conexiunea realizata.

* Exemple de clase Singleton: Runtime class, Action Servlet, Service Locator; constructori private si factory methods sunt alte exemple.
* Cum diferentiem o clasa normala de o clasa Singleton?  
   -pentru a instantia o clasa normala, folosim constructorul; pt a initializa o clasa Singleton folosim metoda getInstance();  
   -o clasa normala vanishes la finalul executiei unei aplicatii, dar una Singleton nu se distruge la final singura  
   -Exista 2 forme a ale desgin patternului Singleton:  
   -early instantiation= instantierea are loc la load time  
   -lazy instantiation=instantierea e facute conform requirementului

**2.2.Use Case**: adaugarea unui element in baza de date, in tabelul Client sau Produs

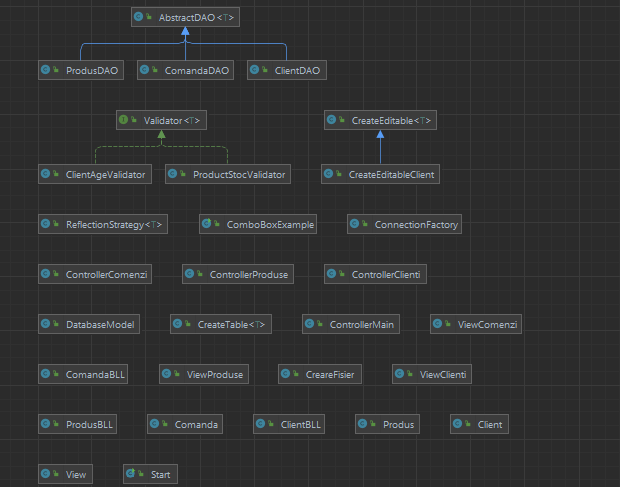
* Actorul principal:userul
* Scenariul de succes:
  + - userul completeaza campurile pt elemental de adaugat
    - userul seleteaza optiunea de a insera un nou produs
    - aplicatia introduce elementul in baza de date
* Scenariu alternativ: Utilizatorul insereaza valori invalide pentru Client sau Produs.
* Pentru Client, nu ii permit sa introduca un client cu varsta sub 18 ani (adica nu accepta interactiunea cu posibil cumparatori minori) sau peste 120 de ani.
* Pentru Produs, nu ii permit sa introduca un stoc numar negativ sau 0.
* Pnetru Comanda (functionalitate implementata direct la nivelul Controllerului pentru Comanda), nu ii permit sa cumpere 0 produse (comanda fara sens) => afisez un mesaj de eroare “Va rugam nu incercati sa cumparati 0 produse..” si resetez interfata cu userul. De asemenea, nu permit sa incerce sa cumpere un produs cand nu exista o cantitate suficienta in stoc=>s-ar ajunge la o cantitate negativa.

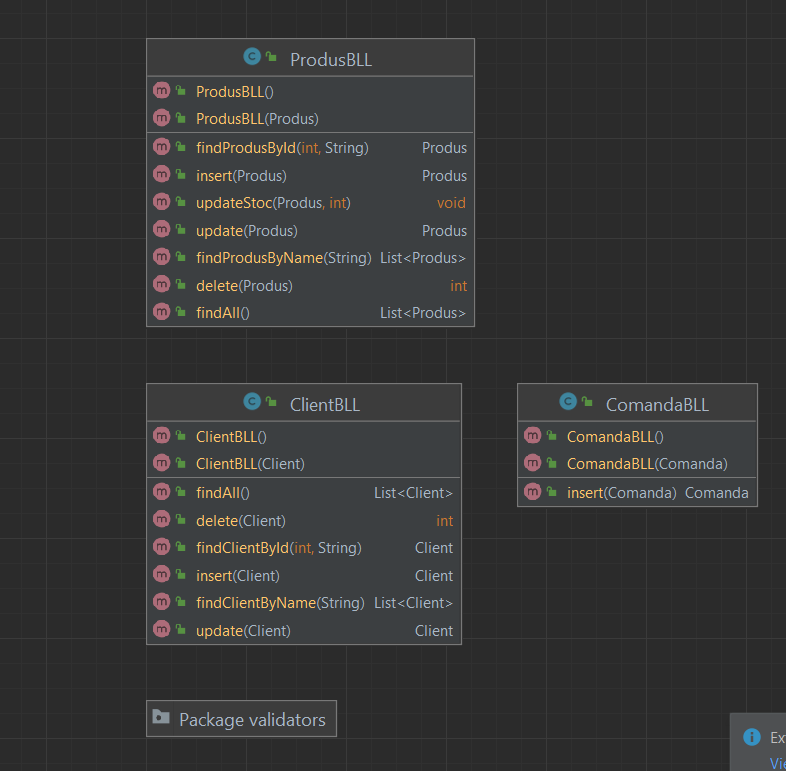
**2.3 Diagrama de use case**:

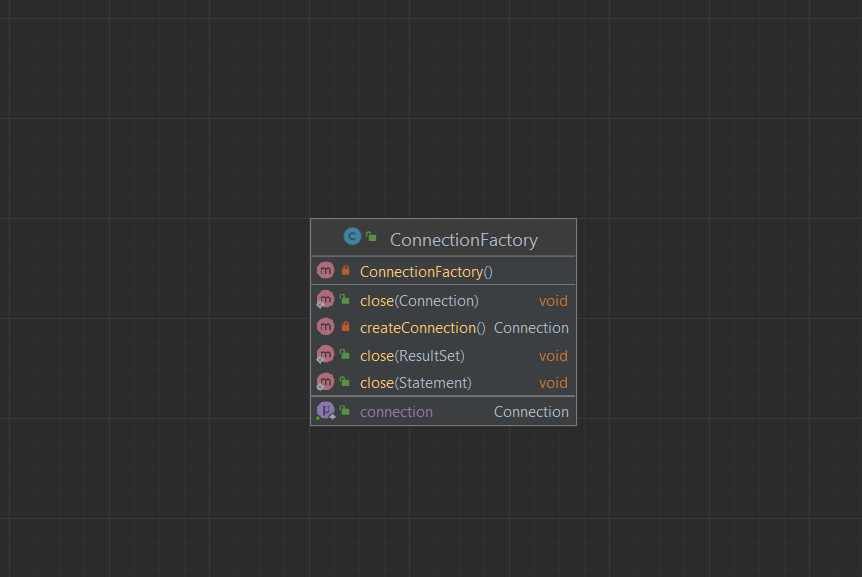


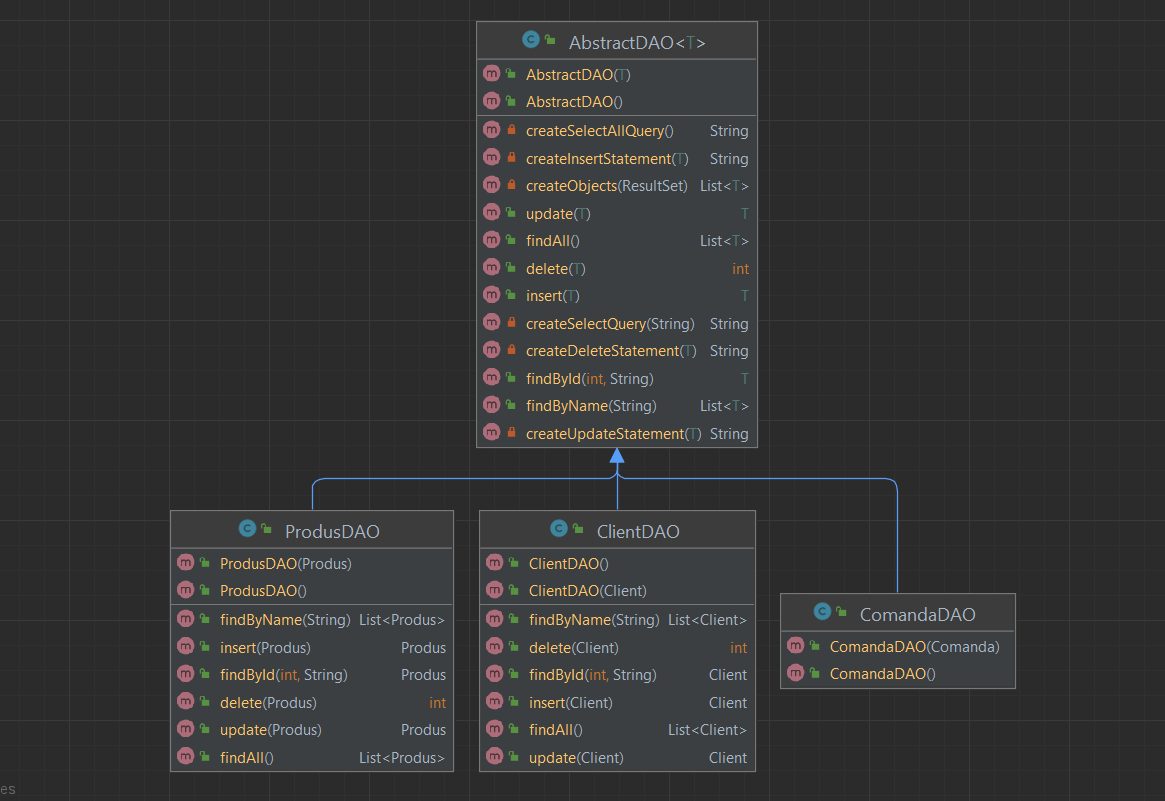
**3.Proiectare**

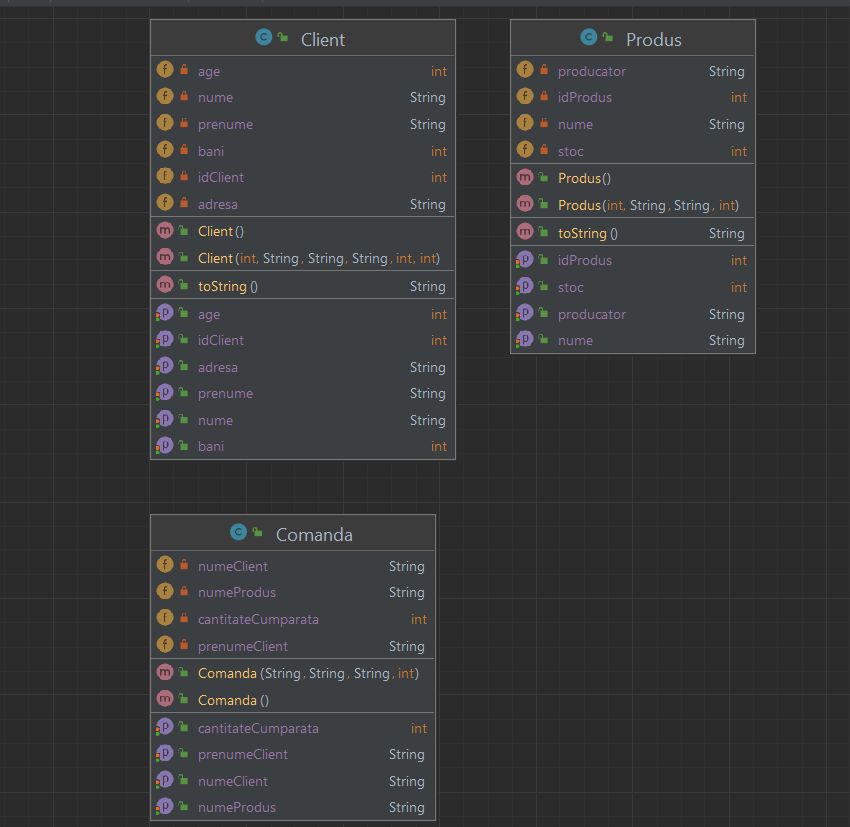
**3.1 Diagramele UML**:

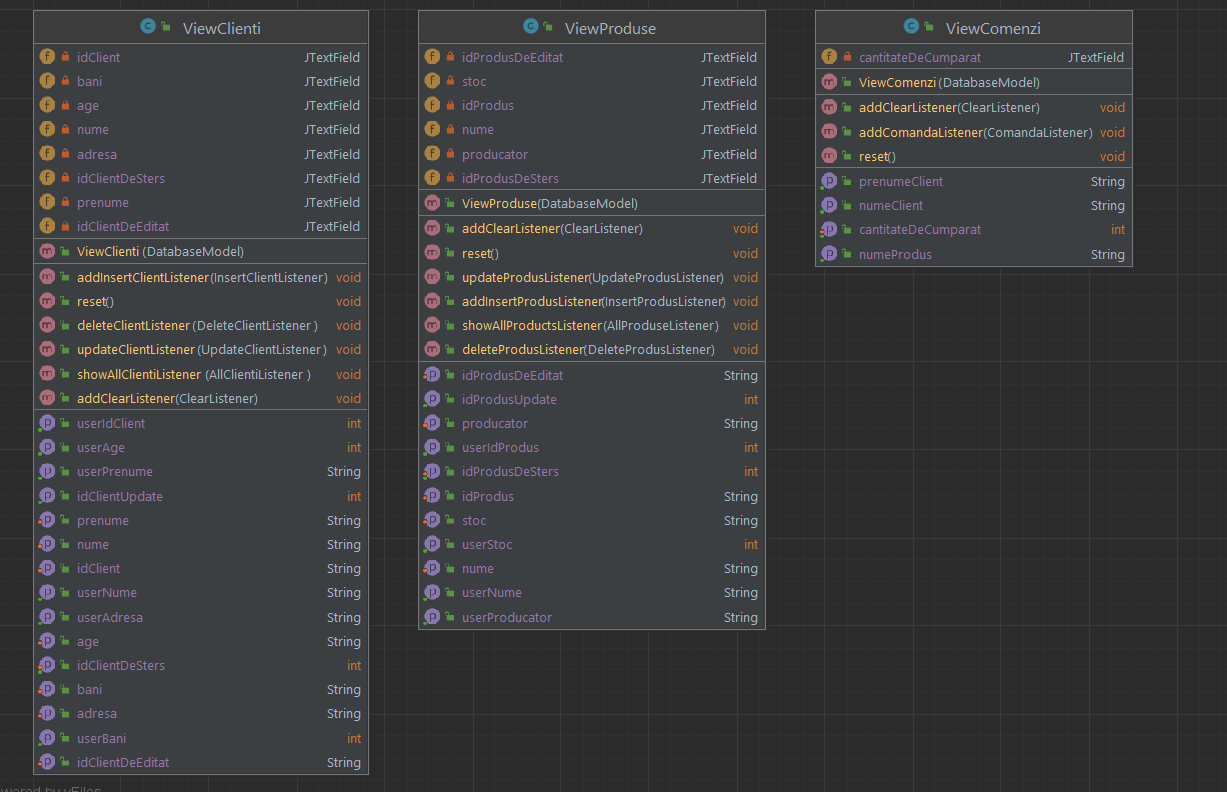


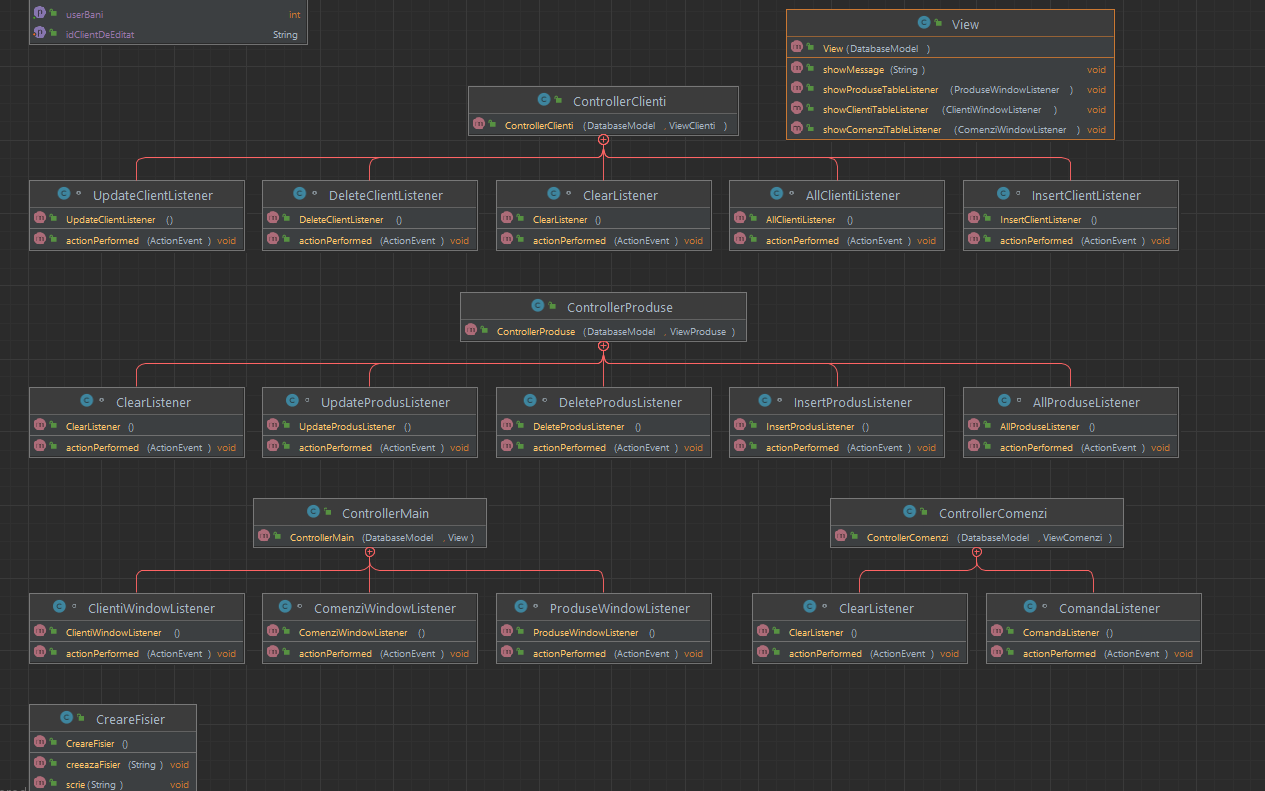


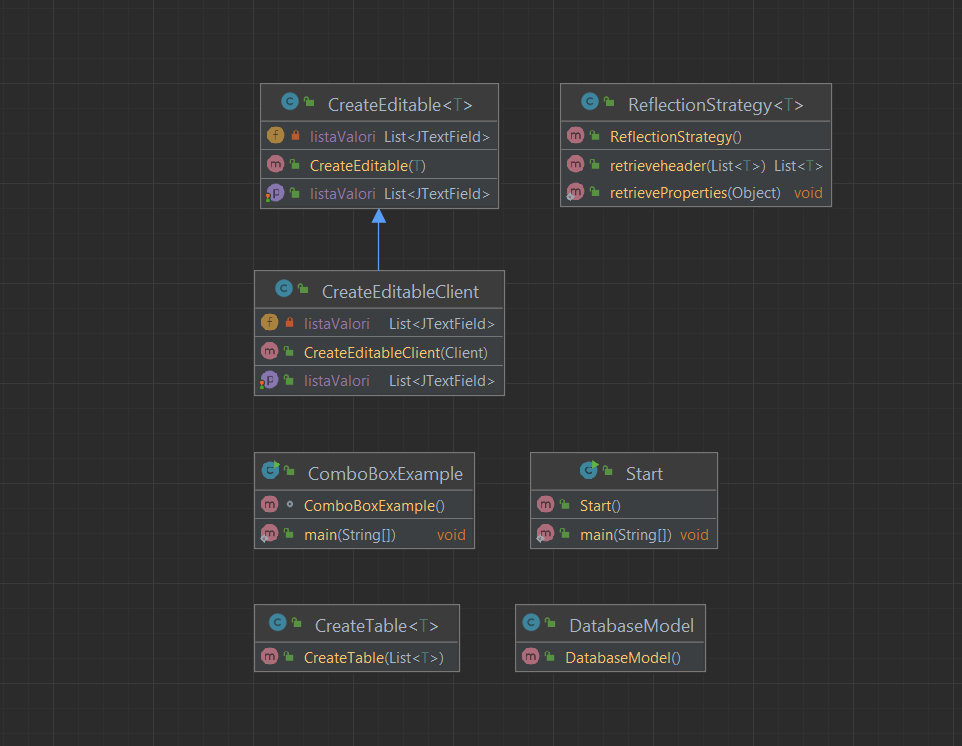








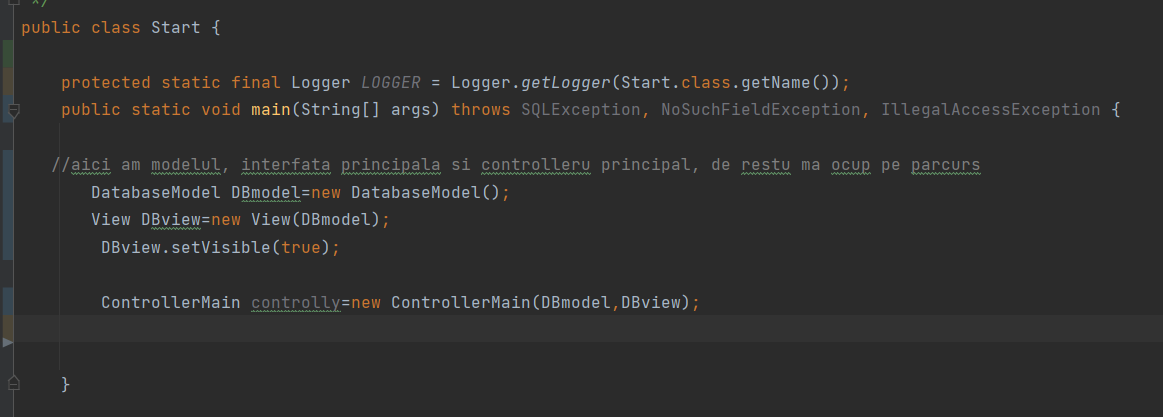




Am structurat codul conform binecunoscutei arhitecturi Layered Arhitecture. Astfel, am 4 pachete mari, Connection (pentru legarea la baza de date):

* Model=contine maparea claselor la baza de date
* Presentation Layer=interfata grafica,contine si elementele de Controller pentru fiecare interfata
* Bussiness Layer=contine clasele care incapsuleaza logica de aplicatie
* Data Access Layer= contine clasele continand interogarile si conexiuena la baza de date

Pe langa aceste 5 mari pachete, exista si pachetul Start, care contine clasa Start,unde avem o instanta de DatabaseModel ( ce include toate cele 3 tabele), una de View( setata vizibila, pentru a putea vedea GUI-ul) si o instata de controller principal. Apoi, in fiecare din cele 3 interfete pentru fiecare tabel in parte am alt Controller instantiat, care se ocupa de functionalitatile specifice.



**4.Implementarea**

**4.1.Pachetul Model**

**4.1.a Clasa Client**

Clasa Client are atributele:idClient, nume, prenume, adresa, bani si age. Pentru fiecare atribut, am definit settere si getter, corespunzator strategiei de incapsulare. Am definit atat constructor cu parametrii, cat si fara. De asemenea, tot aici am suprascris si metoda toString pentru a afisa datele unui client sub o forma usor de inteles.

**4.1.b Clasa Produs**

Clasa Produs are atributele:idProdus, nume, producator si stoc. Pentru fiecare atribut, am definit settere si getter, corespunzator strategiei de incapsulare. Am definit atat constructor cu parametrii, cat si fara. De asemenea, tot aici am suprascris si metoda toString pentru a afisa datele unui produs sub o forma usor de inteles.

**4.1.b Clasa Comanda**

Clasa Comanda are atributele:numeClient, prenumeClient,numeProdus si cantitateCumparata. Pentru fiecare atribut, am definit settere si getter, corespunzator strategiei de incapsulare. Am definit atat constructor cu parametrii, cat si fara.

**4.2.Pachetul BLL**

Fiecare clasa din pachet:ClientBLL, ProdusBLL si ComandaBLL verifica practice daca s-a reusit inserara corecta in baza de date, respective daca rezultatul constructorului din pachetul dao, clasa corespunzatoare este sau nu NULL, si afiseaza un mesaj de eroare,daca este null.

**4.3. Pachetul DAO**

Clasele ComandaDAO, ClientDAO si ProdusDAO doar apeleaza metodele din superclasa Abstract<DAO>. Aici intervine strategia de Reflection. Logica pentru inserare, stergere, updatare, findAll e scrisa o singura data, in clasa parametrizata, apoi in functie de tipul obiectului T, devina specifica pentru Client, Produs.

**4.3.a Clasa Abstract<DAO>**

Clasele ComandaDAO, ClientDAO si ProdusDAO doar apeleaza metodele din superclasa Abstract<DAO>. Aici intervine strategia de Reflection. Logica pentru inserare, stergere, updatare, findAll e scrisa o singura data, in clasa parametrizata, apoi in functie de tipul obiectului T, devina specifica pentru Client, Produs.

* Constructoul imi face rost de clasa specifica pe care lucrez, adica tabelul mapat in pachetul model.
* Metoda createSelectQuery= genereaza o interogare de tipul “SELECT \* from <numeTabel> WHERE <filed>=?” => de exemplu selectez clientii care au varsta 18
* Metoda createSelectAllQuery= genereaza un statement care imi face rost de toate tuplele din tabelul respective
* Metoda createInsertStatement= are drept valoare returnta statementul SQL necesar sub forma de String (generat printr-un obiect de tip StringBuilder, pentru care se apeleaza metoda append) pentru a insera o linie in tabel, pe baza tuplei primate ca parametru. Folosinf un obiect de tip Field, trece prin toate coloanele tabelului, pentru a putea stii care este ultima coloana pentru care nu mai trebuie “,” . Sintaza SQL: INSERT INTO <tableName> VALUES (valueColoana1,valueColoana2,….,ValueUltimaColoana).Setam field drept vizibil, deoarece fiecare atribut e private pentru clasa sa si alfel nu le-am putea accesa.

for(Field fiel Metoda createInsertStatement d: type.getDeclaredFields()) {  
 field.setAccessible(true);

* Metoda CreateObjects= desi poate parea complicate, face o treaba simpla: avand un obiect de tipResultSet genereaza o lista de obiecte de tip T. Pasii:

-creeaza o lista de obiecte de tipul T

-face rost de constructorii clasei T, folosind metoda: **Constructor[] ctors = type.getDeclaredConstructors().**

**-**Intr-o bucla, cat timp mai avem elemente in resultSet: resultSet.next()!=null:

**-**setam constuctorul ctor ca fiind accesibil si cfeam o noua instant a clasei T

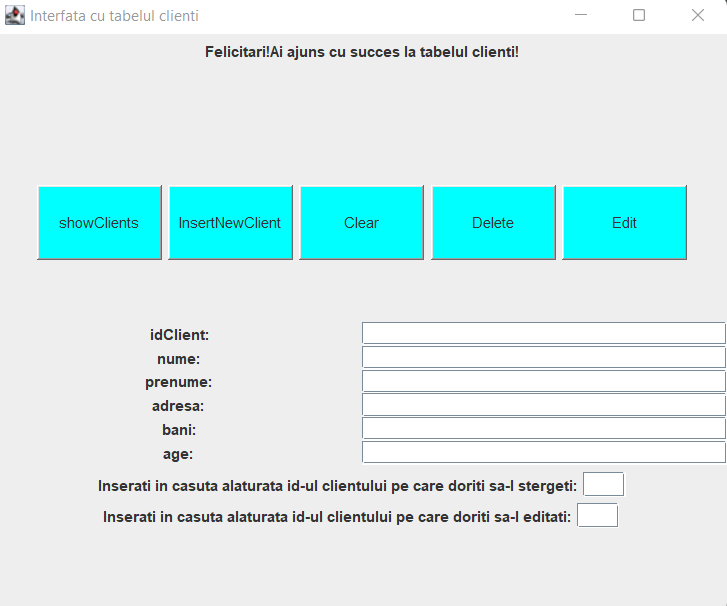
**-**pt fiecare field, obtin numele fieldului si valoarea din resultSet

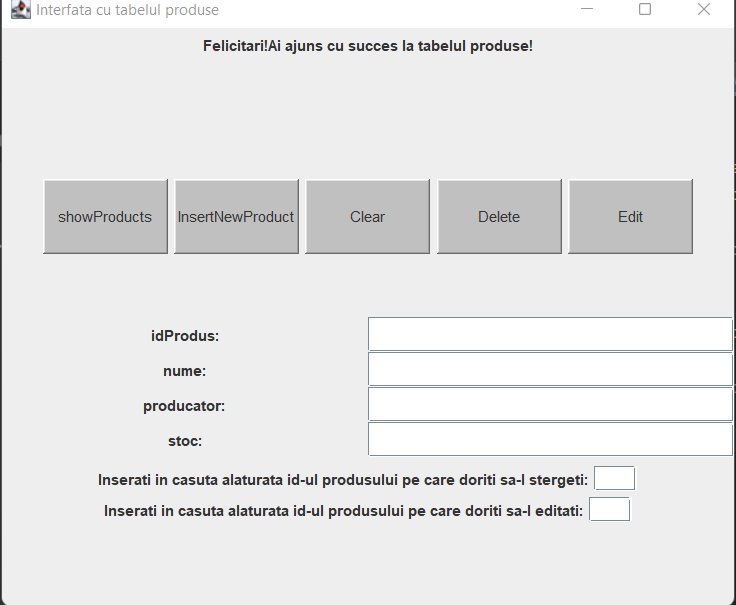
**-**la final ,adaug obiectul in lista si o returnez in cele din urma

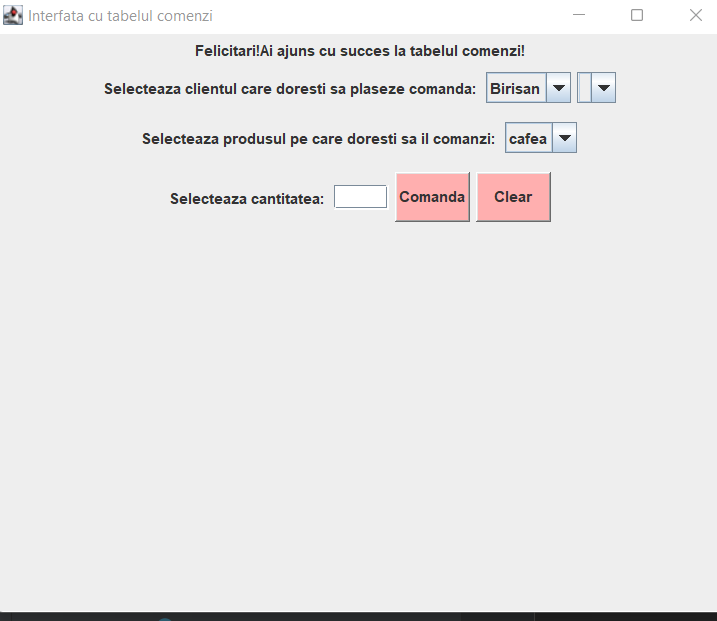
* Metoda createDeleteStatement= analog createInsertStatement. Sintaxa in SQL:DELETE FROM <tableName> WHERE <field>=?
* Metoda createUpdateStatement= analog createInsertStatement. Sintaxa in SQL:UPDATE <tableName> SET <field>=? WHERE <tuplaIdentifier>=?
* Metoda findAll=realizeaza o conexiune la baza de date folosind un obiect din clasa Connection definite in prealabil. Apoi, face rost de “matrita” pentru un selectAllQuery din metoda createSelectAllQuery si o salveaza intr-un String. ResultSet reprezinta rezultatul executiei statementului, pentru care apelam metoda createObjects pentru a genera lista de obiecte necesara ca return value pt metoda findAll. La final, nu uitam sa inchidem resultSetul, statementul si conexiunea.
* Metoda findById= lucreaza similar cu findAll doar ca trebuie sa setam parametrul idClient/idProdu: folosim metoda statement.setInt(Indexparametru, valaore) sau setString, in functie de tipul parametrului. Indexii parametrilor incep de la 1.Pentru ca numele fieldului e variabil dupa tabel, il transmitem ca parametru.
* Metodele delete si insert functioneaza analog cu metoda findById
* Metoda update= in plus furnizeaza un nou JFrame, unde putem introduce noile valori pentru campurile modificabile, selectand in prealabil id-ul Client/Produs care se doreste a fi modificat.

**4.4. Pachetul Presentation**

****

****

****

****

****

* Clasele View,ViewClienti,ViewProduse,ViewComenzi= sunt responsabile cu afisarea interfetelor.
* Clasele ControllerMain,ControllerView,ControllerClienti,ControllerComenzi= gestioneaza actiunile pe butoane. In plus, ControllerComenzi mai are rolul de a dispune generarea fisierului text al facturii.
* Clasa CreareFisier ma ajuta sa creez o factura text. Initial, instantiaza un obiect de tip File, pentru care apeleaza metoda file.exists()=determina daca fisierul exista sau nu filePath-ul proiectului meu. Daca nu, se apeleaza metoda file.createNewFile(), precum si se instantiaza un nou obiect de tip Writer,numit Scriitor, cu ajutorul caruia voi scrie efectiv datele in fisier. In metoda scrie,” Scriitorul” va scrie in fisier Stringul primit drept parametru, folosind metoda: Scriitor.write(String linieNoua)

**4.5. Pachetul Start**

Pe langa clasa Main a programului, denumita sugestiv Start, 2 alte clase prezinta interes:

* Clasa ReflectionStrategy= include 2 metode importante:

-retrieveProperties= metoda statica care functioneaza tot pe baza conceptului de Reflection: primeste o tupla de tip T, acceseaza fieldurile si face rost de valori, afisandu-le intr-un format user-friendly.

-retrieveHeader= tot o metoda static, dar care, accesand fieldurile listei de tuple lista de tip T, returneaza o List<T> care contine informatiile necesare pentru generarea capului de tabel.

* Clasa CreateTable= contine doar constructorul, care primind o lista de tuple, returneaza un tabel gata facut intr-un nou JFrame. Se foloseste de metoda retrieveHeader pentru a face rost de capul de tabel, pentru al doilea argument al metodei JTable(arg1,arg2). Petru primu argument, e nevoie de inregistrari: pentru fiecare element din lista primita ca parametru fac rost de valorile din fields,pe care le pun in linia corespunzatoare nr-lui elementului din lista => o matrice de Stringuri. Creez, la final un JScrollPane, adaugat la frame, pentru ca, in caz ca avem foarte multi client, sa putem derula direct, fara a fi nevoie sa marim fereastra.

**4.4. Pachetul Connection**

Contine o singura:Connection, care permite conectarea la baza de date, a carei explicare depaseste scope-ul acestui proiect.

**5.Rezultate**

Intrucat pentru aceasta tema nu a fost nevoie sa fie scrise teste, o modalitate de testare alternative o reprezinta verificarea grosiera in MySQL Workbench a functionarii operatiilor implementate, precum si a generarii istoricului Comenzilor in fisierul cu proiectul.

**6. Concluzii**

Aplicabilitate: utilizabil pentru a usura viata de zi cu zi, un reigstru de evidente basic, dar util. In plus, se poate extinde pentru a include mai multe tabele, putem sa scriem textFile-ul cu istoricul comenzilor intr-o alta pagina HTML specifica userului, pentru a-si documenta si el cheltuielile pe site-ul firmei. Pe viitor, putem imbunatati si interfata, sa o facem mai user-friendly, colorata si fun to use.

Alt upgrade ar fi sa putem anunta userul care a fost anuntat ca nu exista produsul pe care il doreste in cantitatea respective in stoc ce alte produse similar sunt disponibile, ceea ce ar putea mari in timp profitul firmei pentru care am scris programul.

O alta imbunatatire viitoare ar fi sa se poate inregistra timpul realizarii comenzii, precum si o comunicare cu providerul astfel incat userul sa poata primi informatia despre cat de rapid va primi comanda, eventual si cu firma de curierat pentru a putea urmari drumul pachetului spre sine.

Din punctul de vedere al cunostiintelor dobandite, personal, am invatat cum sa lucrez cu o clasa generic, cum sa apelez metodele din ea pentru diferite clase specific via Reflection. De asemenea, am inteles cum se realizeaza conexiunea cu baza de date, precum si importanta deschiderii unei singure conexiuni o data (motive de securitate) via conceptul Singleton. Am descoperit cum se pot scrie interogari pentru un tabel, iar din punctul de vedere Front-End am reusit sa generez JComboBox-uri, facand ca al doilea sa depinda de valoarea aleasa in primul. Concret, daca in baza de date am 2 clienti:Birisan Denisa si Mirela Dauer, sa nu pot select Birisan Mirela. Prin aceasta strategie, limitez sansele ca userul sa selecteze un client invalid si sa isi piarda timpul.

**7.Bibliografia**

* <https://regex101.com/>
* <https://stackoverflow.com/>
* <https://www.geeksforgeeks.org/>
* <https://www.w3schools.com/java/>
* <https://dsrl.eu/courses/pt/materials/A1_Support_Presentation.pdf>