**ORDER**

**MANAGEMENT**

**IRIMIA LAURA-MARIA**

**GRUPA 302210**

**PROFESOR LABORATOR: MITREA DAN**

Cuprins

[Obiectivul temei 3](#_Toc69802902)

[Obiectivul principal 3](#_Toc69802903)

[Obiective secundare 3](#_Toc69802904)

[Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc69802905)

[Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfața utilizator) 4](#_Toc69802906)

[Diagrama UML: 7](#_Toc69802907)

[Interfața: 7](#_Toc69802908)

[Interfața pentru tabelul client: 7](#_Toc69802909)

[Interfața pentru tabelul product: 7](#_Toc69802910)

[Interfața pentru tabelul comandă: 8](#_Toc69802911)

[Implementare 8](#_Toc69802912)

[Clasa „ConnectionFactory”: 8](#_Toc69802913)

[Clasa „AbstractDAO”: 9](#_Toc69802914)

[Clasa „ClientBLL”: 12](#_Toc69802915)

[Clasa „Controller”: 12](#_Toc69802916)

[Clasa „ReflectionTechnique”: 14](#_Toc69802917)

[Rezultate (testare) 15](#_Toc69802918)

[Concluzii 15](#_Toc69802919)

[Bibliografie 15](#_Toc69802920)

# Obiectivul temei

## Obiectivul principal

Obiectivul principal al acestei teme este proiectarea și implementarea unei aplicații responsabile cu procesarea comenzilor unor clienți într-un depozit, utilizând baze de date relaționale pentru stocare clienților, a produselor și a comenzilor. În plus, aplicația trebuie structurată după o anumită arhitectură, layered architecture.

## Obiective secundare

|  |  |
| --- | --- |
| **Obiectiv secundar** | **Descriere** |
| Dezvoltarea de use case-uri și scenarii | Căutarea de scenarii în care aplicația dezvoltată o să fie folosită și de către cine va fi folosită. |
| Alegerea structurilor de date | Proiectarea unor structuri eficiente și potrivite cerințelor date. |
| Împărțirea pe clase | Alegerea unor clase care vor ușura proiectarea problemei. |
| Dezvoltarea algoritmilor | Implementarea algoritmilor necesari rezolvării problemei și rezolvării cazurilor speciale. |
| Implementarea soluției | Combinarea tuturor algoritmilor implementați in vederea rezolvării problemei. |
| Testare | Găsirea cazurilor care pot crea probleme aplicației dezvoltate și testarea acestora. |

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Procesarea comenzilor unor clienți este un proces extrem de întâlnit în viața de zi cu zi. Pentru implementarea acestei aplicații am folosit o bază de date cu trei tabele aferente clienților, produselor și comenzilor. Am creat o interfață grafică ușor de folosit atât de către utilizatorii experimentați cât și de către cei neexperimentați. Utilizatorul poate introduce un client nou în tabelul corespunzător din baza de date, poate modifica numele unui client deja existent din tabelul aferent, poate modifica adresa unui client deja existent din tabelul corespunzător, poate șterge un client deja existent din tabelul corespunzător, poate găsi un client după numărul de identificare asociat acestuia din tabelul corespunzător și poate afișa toți clienții aflați la un moment dat în tabelul aferent.

De asemenea, utilizatorul poate introduce un produs nou în tabelul corespunzător din baza de date, poate modifica numele unui produs deja existent din tabelul aferent, poate modifica cantitatea unui produs deja existent din tabelul corespunzător, poate modifica prețul unui produs deja existent din tabelul corespunzător, poate șterge un produs deja existent din tabelul corespunzător, poate găsi un produs după numărul de identificare asociat acestuia din tabelul corespunzător și poate afișa toate produsele aflate la un moment dat în tabelul aferent.

Deopotrivă, utilizatorul poate selecta din clienții și produsele deja existente în tabelele corespunzătoare un client și un produs asupra căruia poate plasa o comandă pentru o anumită cantitate. Un client poate plasa o comandă pentru un singur produs odată. Fiecare comandă este afișată într-un fișier text asociat comenzii respective.

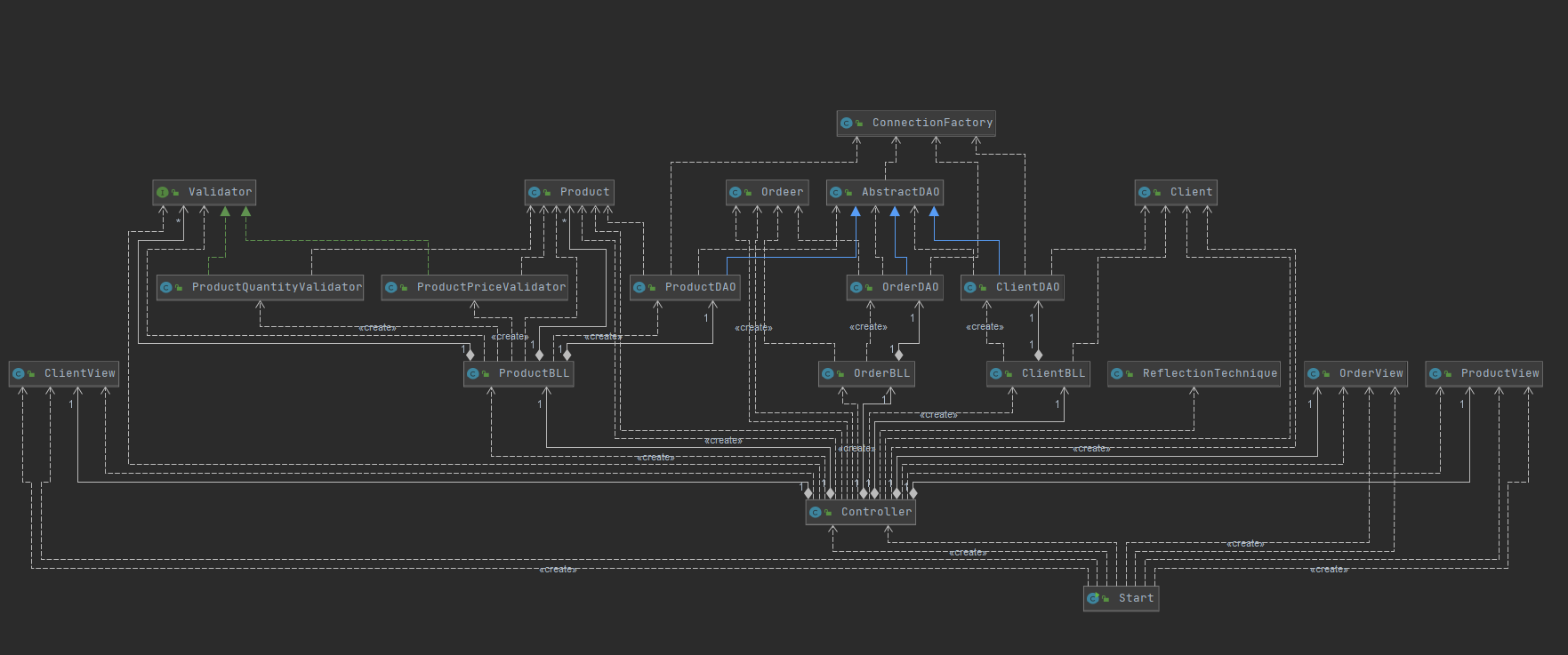
Fiecare tabel are propria lui fereastră grafică cu metodele asociate descrise mai sus pentru o mai ușoară înțelegere a aplicației. Produsele pe care vrea utilizatorul să le introducă în tabel sunt mai întâi verificate dacă corespund cu intervalul de valori setat.

# Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfața utilizator)

|  |  |
| --- | --- |
| **Package** | **Class** |
| main -> java -> **connection** | ConnectionFactory |
| main -> java -> **model** | Client  Product  Order |
| main -> java -> **dao** | AbstractDAO  ClientDAO  ProductDAO  OrderDAO |
| main -> java -> **bll** | ClientBLL  ProductBLL  OrderBLL |
| main -> java -> bll -> **validators** | Validator (**Interface**)  ProductPriceValidator  ProductQuantityValidator |
| main -> java -> **presentation** | ClientView  ProductView  OrderView  Controller |
| main -> java -> **start** | ReflectionTechnique  Start |

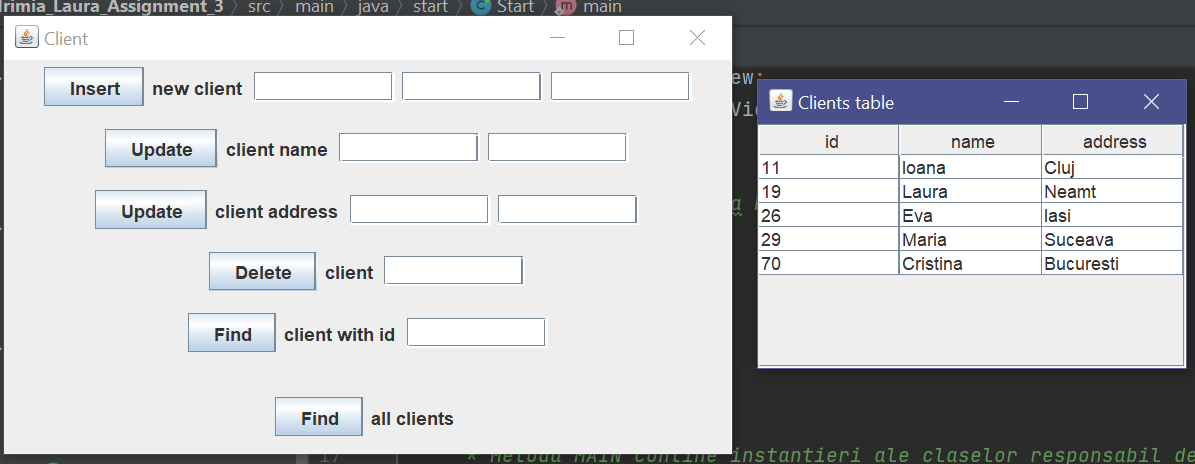
|  |  |
| --- | --- |
| **Class** | **Descriere** |
| ConnectionFactory | Clasa are atributele necesare conectării la baza de date: numele driver-ului, locația bazei de date, numele de utilizator și parola pentru accesarea server-ului MySQL. Ca și metode avem: metoda care creează conexiunea cu baza de date, metoda care returnează conexiunea creată, metoda care închide o conexiune, metoda care închide un query și metoda care închide rezultatul returnat de un query. |
| Client | Clasa are atributele care descriu un client: id, nume și adresă. Pe lângă metodele prestabilite de set și get și de constructori, mai avem metoda suprascrisă toString pentru o afișare mai plăcută. |
| Product | Clasa are atributele care descriu un produs: id, nume, cantiate și preț. Pe lângă metodele prestabilite de set și get și de constructori, mai avem metoda suprascrisă toString pentru o afișare mai plăcută. |
| Order | Clasa are atributele care descriu o comandă: id, numele produsului comandat, numele clientului care a făcut comanda, cantiate și preț. Pe lângă metodele prestabilite de set și get și de constructori, mai avem metoda suprascrisă toString pentru o afișare mai plăcută. |
| AbstractDAO | Clasa generică care conține metodele de accesare specifice tuturor tabelelor din baza de date. Avem un parametru important care ne spune tipul clasei obiectului cu care a fost apelată o anumită metodă. Ca și metode avem: metode care formează query-urile de care avem nevoie, metoda care returnează toate elementele aflate la un moment dat într-un anumit tabel, metoda care găsește un anumit element după id-ul aferent, metoda care adaugă un element într-un anumit tabel, metoda care șterge un element după id-ul aferent dintr-un anumit tabel, metoda care modifică numele unui client sau a unui produs din tabelul aferent și o metodă care creează o listă de obiecte de tipul clasei din model asociate tabelului corespunzător. |
| ClientDAO | Clasa care conține metodele de accesare a bazei de date specifice tabelului client. Avem o singură metodă specifică tabelului client: modificarea adresei unui client deja existent în tabelul aferent în funcție de id-ul dat corespunzător. |
| ProductDAO | Clasa care conține metodele de accesare a bazei de date specifice tabelului produs. Avem două metode specifice tabelului produs: modificarea cantității unui produs deja existent în tabelul aferent în funcție de id-ul corespunzător și modificarea prețului unui produs deja existent în tabelul aferent în funcție de id-ul corespunzător. |
| OrderDAO | Clasa care conține metodele de accesare a bazei de date specifice tabelului comandă. Avem patru metode specifice tabelului comandă: modificarea numelui unui client care a plasat o comandă deja existentă în funcție de id-ul corespunzător, modificarea numelui unui produs deja existent într-o comandă în funcție de id-ul corespunzător, modificarea cantității unui produs dintr-o comandă deja existentă în funcție de id-ul corespunzător și modificarea prețului deja existent într-o comandă în funcție de id-ul corespunzător. |
| ClientBLL | Clasa care se ocupa cu logica aplicației legată de tabelul client. Clasa are un singur atribut principal și anume un obiect de tipul ClientDAO care asigură conectarea logicii cu metodele care manipulează baza de date. Toate metodele care se află aici apelează metodele care accesează baza de date prin prisma atributului. |
| ProductBLL | Clasa care se ocupa cu logica aplicației legată de tabelul produs. Clasa are un singur atribut principal și anume un obiect de tipul ProdusDAO care asigură conectarea logicii cu metodele care manipulează baza de date. Toate metodele care se află aici apelează metodele care accesează baza de date prin prisma atributului. Clasa mai are un atribut de tipul listă de validatori care ne ajută să validăm un produs care se vrea introdus la un moment dat. |
| OrderBLL | Clasa care se ocupa cu logica aplicației legată de tabelul comandă. Clasa are un singur atribut principal și anume un obiect de tipul OrderDAO care asigură conectarea logicii cu metodele care manipulează baza de date. Toate metodele care se află aici apelează metodele care accesează baza de date prin prisma atributului. |
| Validator (**Interface**) | Interfața care conține metoda de validare care trebuie suprascrisă de toate clasele care implementează interfața. |
| ProductPriceValidator | Clasa care implementează interfața descrisă mai sus și care suprascrie metoda de validare astfel încât să valideze produsul doar dacă prețul se află între capetele intervalului declarat prin atributele finale declarate. |
| ProductQuantityValidator | Clasa care implementează interfața descrisă mai sus și care suprascrie metoda de validare astfel încât să valideze produsul doar dacă cantitatea se află între capetele intervalului declarat prin atributele finale declarate. |
| ClientView | Clasa are ca atribute toate componentele de la nivel de interfață care necesită manipularea ulterioară pentru fereastra aferentă tabelului client. Conține constructorul interfeței, metode care returnează textul din TextField-uri, metode care adaugă ascultători pentru fiecare buton aferent și metoda care setează textul unei etichete cu clientul căutat sau cu mesajul excepției în caz că nu se găsește niciun client. |
| ProductView | Clasa are ca atribute toate componentele de la nivel de interfață care necesită manipularea ulterioară pentru fereastra aferentă tabelului produs. Conține constructorul interfeței, metode care returnează textul din TextField-uri, metode care adaugă ascultători pentru fiecare buton aferent și metoda care setează textul unei etichete cu produsul căutat sau cu mesajul excepției în caz că nu se găsește niciun produs. |
| OrderView | Clasa are ca atribute toate componentele de la nivel de interfață care necesită manipularea ulterioară pentru fereastra aferentă tabelului comandă. Conține constructorul interfeței, metodă care returnează textul din TextField și metode care adaugă ascultători pentru fiecare buton aferent. |
| Controller | Clasa care leagă clasele din țin de logica aplicației cu clasele ce țin de interfața aplicației. Are șase atribute principale, trei atribute care reprezintă obiecte de tipul fiecărei ferestre grafice implementate și trei atribute care reprezintă obiectele care se ocupă cu logica bazei de date. În cadrul constructorului inițializez atributele de tip fereastră grafică cu cele primite ca și parametrii, inițializez atributele care se ocupă cu logica bazei de date și asociez metoda ascultătoare pentru fiecare buton din fiecare fereastră grafică în parte. |
| ReflectionTechnique | Clasa care folosește tehnica reflexiei pentru a extrage proprietățile si valorile proprietăților unui obiect. Are două metode: una care extrage proprietățile obiectului primit ca parametru și le returnează sub formă de vector și a doua care extrage valorile proprietăților obiectului primit ca parametru și le returnează sub formă de vector. |
| Start | Clasa care conține metoda main. Aici instanțiez fiecare fereastră grafică pe care o primește controller-ul ca parametru în constructorul aferent și setez fiecare fereastră grafică ca fiind visibilă. |

## Diagrama UML:

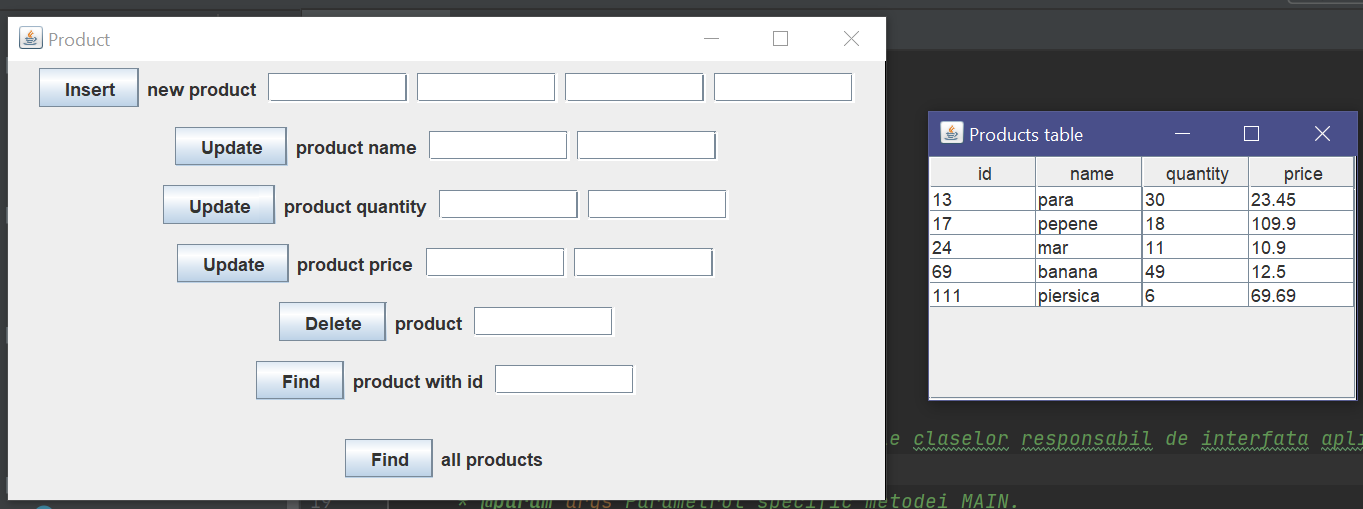


## Interfața:

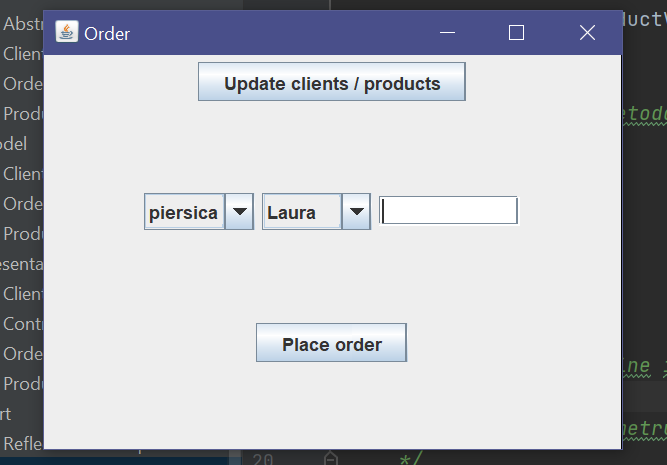
### Interfața pentru tabelul client:



### Interfața pentru tabelul product:

****

### Interfața pentru tabelul comandă:

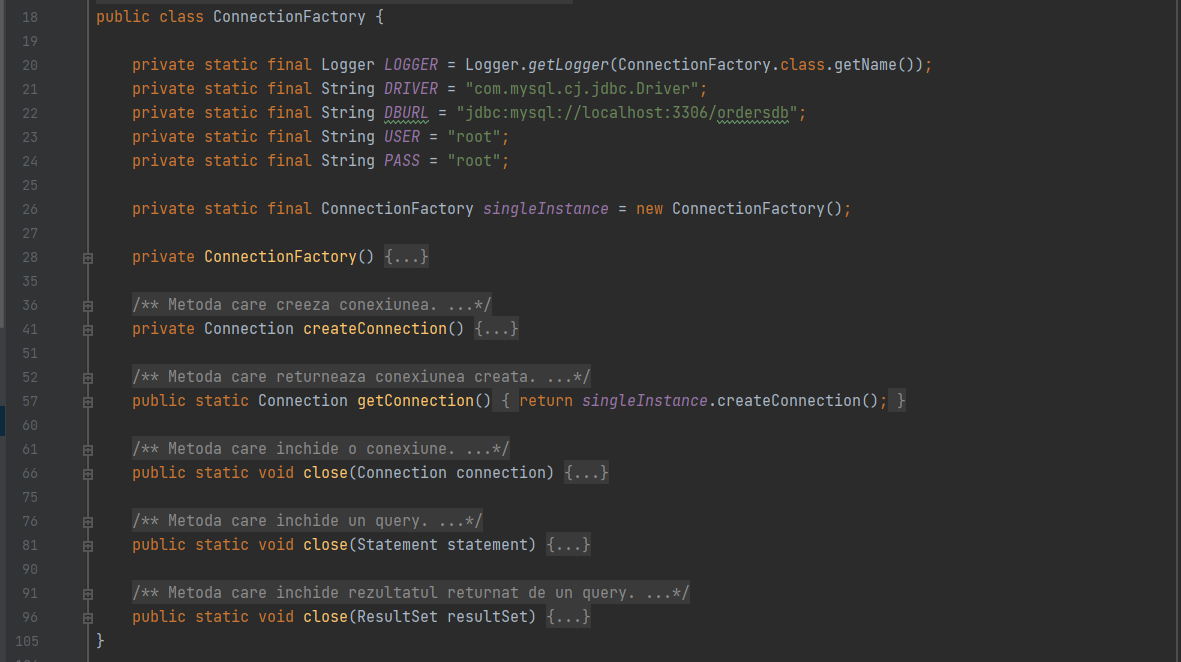


# Implementare

Voi adăuga doar anumite clase și metode care mi se par mie cele mai importante pentru a nu încărca inutil documentația.

## Clasa „ConnectionFactory”:

Clasa „ConnectionFactory” este clasa care se ocupă cu conexiunea dintre aplicație și server-ul MySQL care conține baza de date. Atributele sunt atributele necesare conectării la baza de date: numele driver-ului, locația bazei de date, numele de utilizator și parola pentru accesarea server-ului. În cadrul metodei care creează conexiunea, se declară o conexiune locală asupra căreia se încearcă egalarea cu conexiunea rezultată din apelarea metodei care returnează o conexiune pe baza locației bazei de date, a numelui de utilizator și a parolei. Restul metodelor au la bază conexiunea creată, un query și un rezultat returnat de un query.



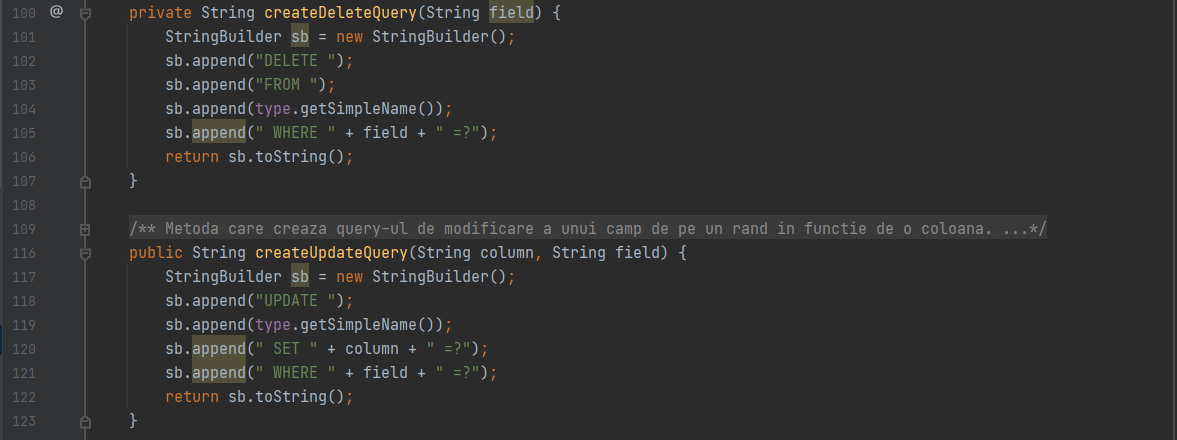
## Clasa „AbstractDAO”:

Clasa care conține metodele asociate tuturor tabelelor pentru accesarea bazei de date. Mai jos am adăugat metoda care creează query-ul de adăugare a unui element într-un tabel. Se poate vedea cum am luat pe rând fiecare câmp din obiectul cu care se apelează metoda și am salvat numele în String după sintaxa specială a query-ul și apoi finisat sintaxa cu câte valori trebuie introduse.



Urmează metodele care creează query-ul de ștergere și de modificare a unui element dintr-un tabel. Pentru metoda de ștergere mi-am creat String-ul și în el am adăugat numele câmpului după care se șterge, iar pentru metoda de modificare am adăugat numele câmpului după care se caută elementul care trebuie modificat și numele câmpului care se modifică.

Restul metode care creează query-uri sunt asemănătoare.



Pentru metoda de extragere a tuturor elementelor dintr-un tabel m-am folosit de o conexiune, un query și rezultatul obținut de query. Metoda returnează lista de obiecte de tipul corespunzător celor din tabel și apoi închide toate variabilele folosite.



Metoda de inserare într-un tabel am implementat-o pe baza unei conexiuni cu baza de date și a unui query, fără a mai avea nevoie de rezultatul returnat de query pentru că nu este necesar. Pentru fiecare ? din query am luat pe rând fiecare câmp din obiectul cu care s-a apelat metoda și am extrast metodele de set și get asociate câmpului respectiv. Apoi am căutat doar metoda de get pentru a obține valoarea câmpului și am adăugat valoarea în locul ? asociat ei în query. După ce am terminat de editat query-ul, l-am executat și apoi am închis toate variabilele de care m-am folosit.



Metode de ștergere în funcție de id am implementat-o tot pe baza unei conexiuni și a unui query. Mi-am creat query-ul și apoi am adăugat pe locul ? valoarea id-ul care trebuie căutat pentru a fi șters din tabel și apoi am închis variabilele folosite.

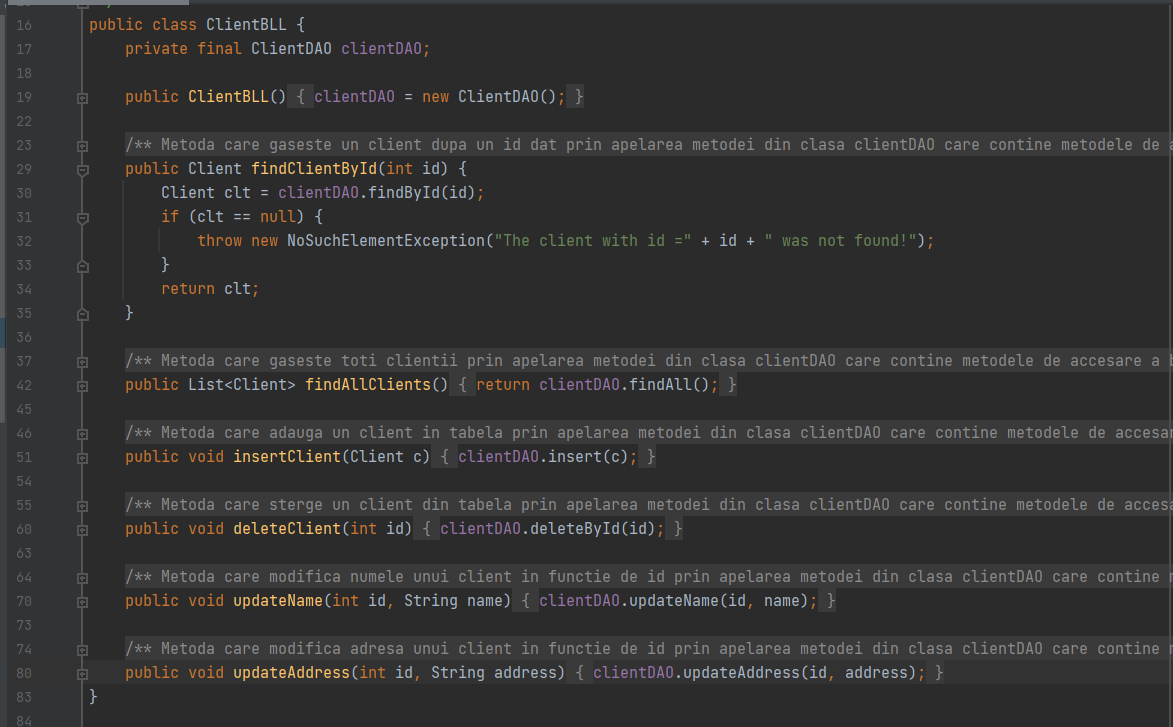


Metoda de modificare a numelui din această clasă este folosită doar pentru tabelele client și produs, pentru tabelul comandă avem metodele de modificare foarte asemănătoare în clasa DAO corespunzătoare comenzii. Mi-am deschis o conexiune cu baza de date și am editat query-ul de modificare astfel încât pe locul valorii cu care vreau să modific valoarea curentă am adăugat valoarea primită ca parametru și pe locul valorii după care se caută în tabelul respectiv am adăugat valoarea id-ului primit ca parametru și am executat query-ul. După am închis toate variabilele locale de care m-am folosit.



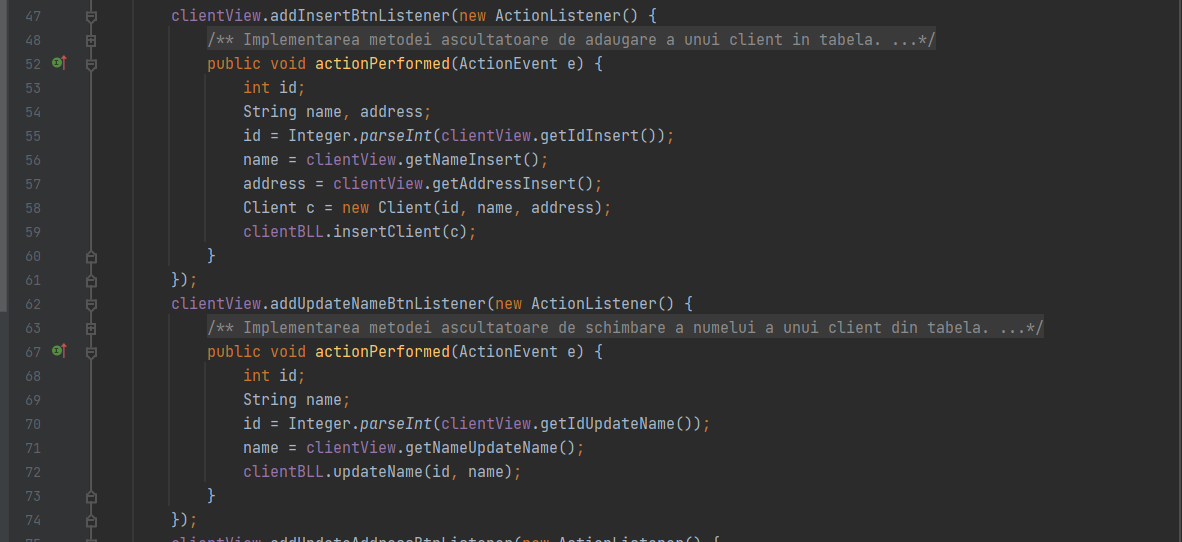
## Clasa „ClientBLL”:

Clasa care se ocupă cu logica aplicației legată de tabelul client. În fiecare metodă din această clasă am apelat metodele aferente care accesează tabelul din baza de date cu parametrii necesari și am returnat valorile corespunzătoare.

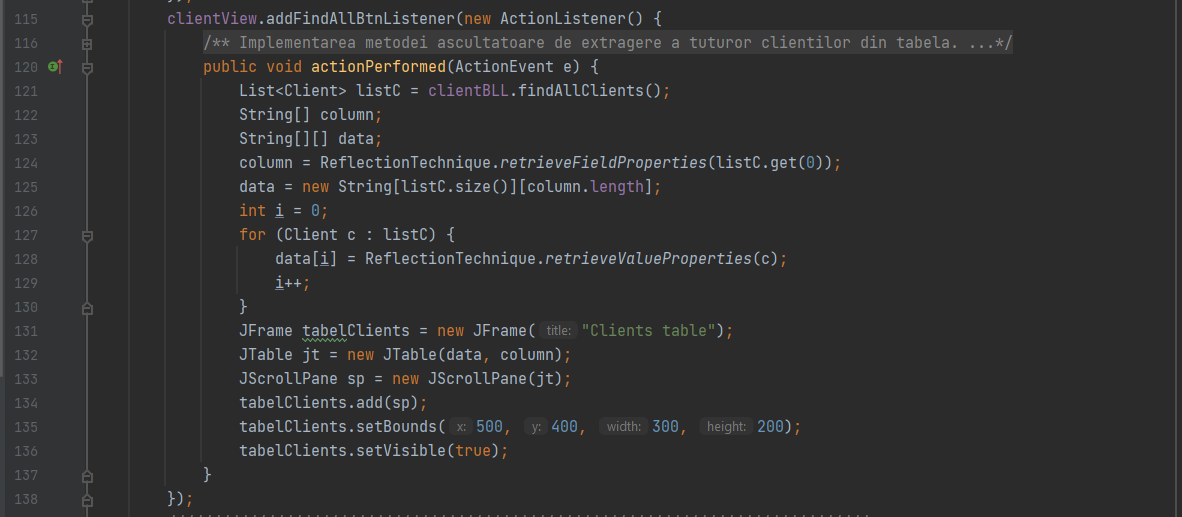


## Clasa „Controller”:

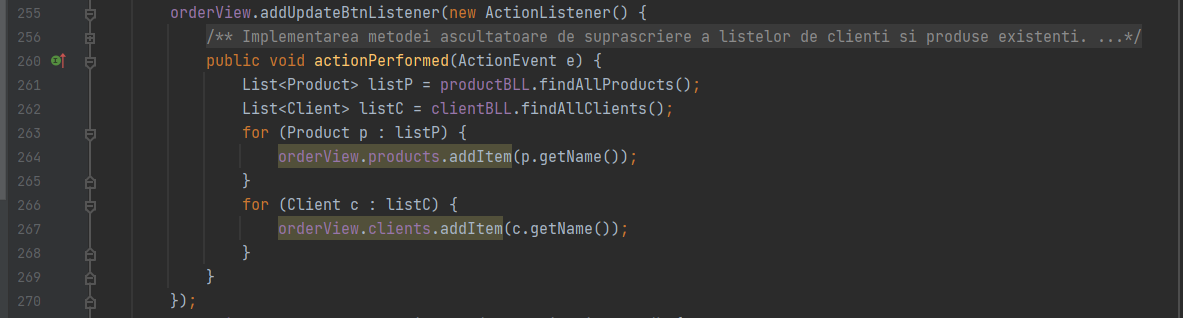
Clasa care se ocupă cu legăturile dintre interfețe și logica aplicației dezvoltate. Mai jos am adăugat metodele implementate de adăugare de ascultători pentru butoanele de adăugare client și de modificare a numelui unui client, restul metodelor care nu au fost adăugate fiind foarte asemănătoare. Pentru metoda de adăugare client mi-am declarat variabilele corespunzătoare fiecărui câmp din obiectul client și le-am inițializat cu valorile date în fereastra grafică de către utilizator. Mi-am creat un client nou cu valorile de mai sus și apoi am apelat metoda care adaugă clientul în tabel din clasa de logică a clientului. Metoda de modificare a numele este asemănătoare, eu inițializând id-ul clientului pe care vrem să îl modificăm și valoarea cu care se vrea să fie modificată valoarea curentă și apoi apelez metoda corespunzătoare din clasa de logică a tabelului.

****

Urmează metoda care adaugă un client în tabelul corespunzător. Aici m-am folosit de metoda reflexiei pentru a extrage câmpurile și valoarea unui obiect și pentru a crea tabelul de afișare. Având în vedere că un JTabel are nevoie de un vector de nume de coloane și o matrice de elemente care vor reprezenta elementele de pe fiecare linie din tabel, mi-am declarat o listă de clienți pe care o creez cu ajutorul metodelor care extrag câmpurile și valorile câmpurilor. Vectorul de numele de coloane îl inițializez cu vectorul cu nume de câmpuri returnat de metoda respectivă. Matricea am declarat-o de dimensiunea listei de elemente în care se află toți clienții din tabel și de dimensiunea vectorului de coloane. Apoi, fiecare linie din matrice a fost asociată cu vectorul de valori corespunzător câmpurilor returnat de metoda aferentă. Apoi mi-am creat o nouă fereastră care conține tabelul implementat.

****

Mai jos este metoda care adaugă în ComboBox-uri numele clienților și produselor aflați la un moment dat în tabelele aferente. Îmi extrag toți clienții și toate produsele în două liste cu ajutorul metodei care extrage toate elementele dintr-un tabel și apoi adaug numele fiecărui element în ComboBox-ul asociat.

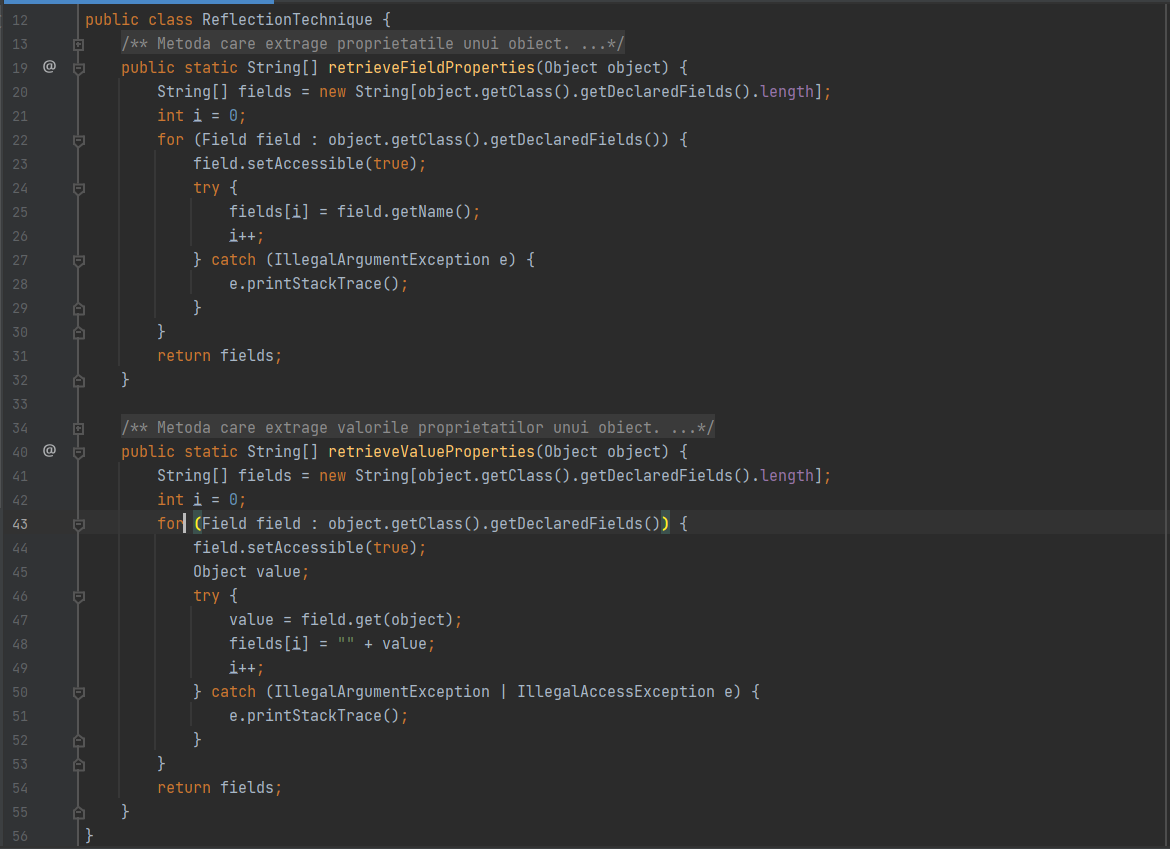
****

Metoda de plasare a unei comenzi am implementat-o pe baza clienților și produselor aflate la un moment dat în tabele. Mi-am inițializat numele produsului selectat în ComboBox și apoi l-am căutat în lista de produse din tabel pentru a îi extrage prețul și cantitatea. Dacă cantitatea introdusă de utilizator în fereastra grafică este mai mare decât cantitatea actuală, îmi afișez o fereastră cu mesajul de eroare corespunzător. Dacă cantitatea din stoc este suficientă, îmi creez o nouă comandă pe care o afișez și în fereastra grafică și în fișierul corespunzător acesteia și apoi modific cantitatea care trebuie să rămână în stoc.

****

## Clasa „ReflectionTechnique”:

Mai jos am adăugat metodele care se folosesc de tehnica reflexiei pentru a extrage câmpurile și valorile câmpurilor unui obiect primit ca parametru.

****

# Rezultate (testare)

Toate rezultatele asociate cu această aplicație sunt date de ferestrele grafice, tabelele din baza de date și fișierele text generate.

# Concluzii

În urma rezolvării acestei teme, pot spune că mi-am însușit mai bine cunoștințele din semestrul întâi despre Java și am acumulat și informații noi, cum ar fi conexiunea aplicației implementate cu o bază de date și lucrul cu metode specifice bazei de date. Dezvoltarea treptată a aplicației m-a ajutat să vin cu idei cât mai creative și eficiente pentru fiecare problemă care apărea. Ca și dezvoltări ulterioare, aș mai adăuga câteva tabele și câteva metode care să permită o aplicație cât mai realistă cu mai multe tipuri de produs pe o comandă și poate aș face și o interfață mai plăcută vizual, nu doar minimalistă.

# Bibliografie

Nu m-am ajutat de nicio sursă de inspirație pentru redactarea documentație. Tot ce am scris aici sunt ideile care mi-au venit pe moment. Nu pot garanta că în aceste idei nu se găsesc rezumate a altor idei citite pe diverse site-uri de domeniu sau în diverse cărți.