**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**

**DEPARTAMENTUL CALCULATOARE**

**TEMA 3**

**Order Management**

**Documentație**

**Nume: Litu Saviana**

**Grupa: 30221**

**An academic 2020 – 2021**

**Cuprins**

[**Obiectivul temei** 3](#_Toc70426266)

[**Analiza problemei, modelare, scenarii, cazuri de utilizare** 4](#_Toc70426267)

[**Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfața utilizator)** 5](#_Toc70426268)

[**Implementare** 11](#_Toc70426269)

[**Rezultate** 24](#_Toc70426270)

[**Concluzii** 27](#_Toc70426271)

[**Bibliografie** 27](#_Toc70426272)

# **Obiectivul temei**

Obiectivul acestei teme este de a propune și implementa o aplicație Java ce are scopul de a procesa comenzile clienților unui depozit. Se va folosi o bază de date relațională pentru a stoca date despre produse, clienți si comenzi. În plus, aplicația va fi structurată în pachete, conform unei arhitecturi stratificate.

Pentru a realiza acest obiectiv principal, definim următoarele obiective secundare:

|  |  |  |
| --- | --- | --- |
| **Obiectiv** | **Descriere** | **Capitol** |
| Analiza problemei | Se analizează enunțul problemei și se dezvoltă un design inițial | **2** |
| Alegerea structurilor de date potrivite | Se aleg structuri de date care să faciliteze accesarea datelor din baza de date | **3** |
| Proiectare | Se vor crea clase și pachete în concordanță cu design-ul dorit și se vor scrie algoritmi pentru realizarea prin metode reflexive a operațiilor asupra datelor din baza de date | **3** |
| Implementare | Se va realiza conexiunea la baza de date și implementarea operațiilor asupra datelor din baza de date | **4** |
| Realizarea interfeței grafice | Se va realiza interfața utilizator utilizând JavaFX | **4** |
| Testarea | Se vor efectua teste pentru diferite input-uri | **5** |

# **Analiza problemei, modelare, scenarii, cazuri de utilizare**

O bază de date relațională este un tip de bază de date care stochează și oferă acces la punctele de date care sunt legate între ele. Bazele de date relaționale se bazează pe modelul relațional, o modalitate intuitivă și simplă de reprezentare a datelor în tabele. Într-o bază de date relațională, fiecare rând din tabel este o înregistrare cu un cod unic numit cheie. Coloanele din tabel au atribute ale datelor și fiecare înregistrare are, de obicei, o valoare pentru fiecare atribut, facilitând stabilirea relațiilor dintre punctele de date. Așadar, folosind o bază de date relațională putem facilita stocarea datelor referitoare la comenzile din cadrul unui depozit, în cazul de față.

Analizând enunțul temei propuse, observăm că trebuie să începem prin a crea o bază de date relațională (warehouse), care să conțină cel puțin trei tabele – product, client și order. Apoi, în cadrul proiectului Java, creăm pachetele model (unde se vor afla clasele care vor fi mapate la tabelele din baza de date), presentation (care conține clasele unde se dezvoltă interfața utilizator), bll (adică Business Logic Layer – unde se vor afla clasele care încapsulează logica aplicației), dao (adică Data Access Layer – unde se află clasele care conține interogări asupra bazei de date și conexiunea cu aceasta) și connection (care va conține clasa în care se face conexiunea inițială cu baza de date).

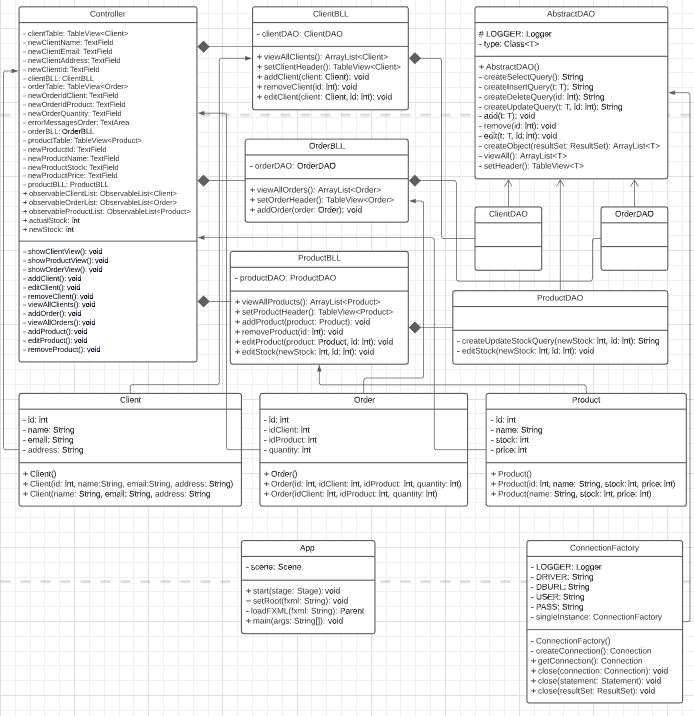
Trecând la partea de modelare, după tabelele din baza de date ne dăm seama care vor fi clasele din pachetul model, și anume Client, Product și Order. Clasele din pachetul dataAccessLayer – AbstractDAO (pentru a implementa prin metode reflexive accesul datelor din baza de date) și ClientDAO, ProductDAO, OrderDAO – clase care vor extinde clasa AbstractDAO, preluând toate metodele ei. Clasele din businessLogicLayer vor fi ClientBLL, ProductBLL și OrderBLL. În pachetul presentation se va afla clasa Controller, iar în pachetul connection vom avea clasa ConnectionFactory.

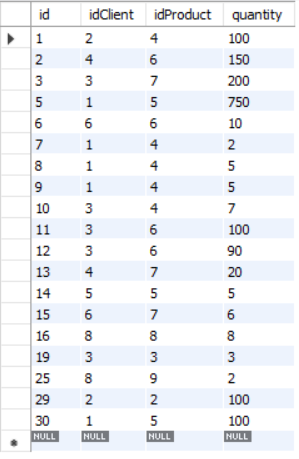
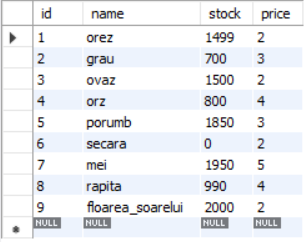
În continuare, voi prezenta cazul de utilizare ideal:

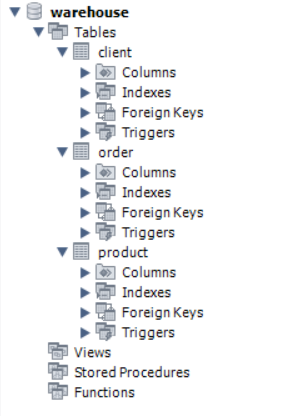
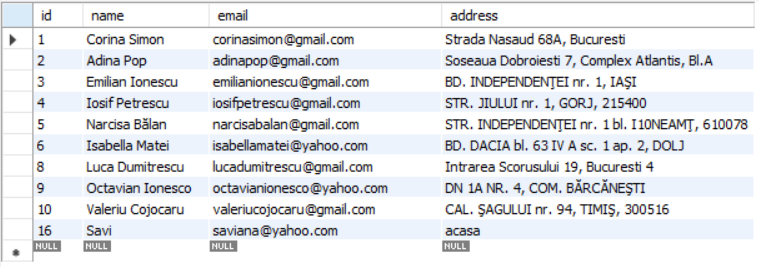
1. Utilizatorul alege tabelul pe care vrea să îl vadă/modifice apăsând unul dintre butoanele Client View, Product View sau Order View.
2. Utilizatorul alege acțiunea pe care vrea să o realizeze asupra tabelului respectiv – View all (pentru toate cele trei tabele – îi arată toate datele din tabel), Add (pentru toate cele trei tabele – adaugă o nouă linie in tabel, după ce se completează câmpurile care se doresc a fi introduse), Remove (pentru tabelele Client și Product – șterge o intrare din tabel, în funcție de id-ul oferit) și Edit (pentru tabelele Client și Product – editează un rând din tabel în funcție de datele oferite spre a fi modificate).
   1. Dacă apasă View all, va vedea toate datele din tabelul ales.
   2. Pentru a adăuga o nouă linie în oricare dintre tabele, va trebuie să completeze câmpurile cu datele cu care vrea să populeze tabelul, apoi să apese butonul Add pentru a adăuga în baza de date și View all pentru a vedea tabelul modificat.
   3. Pentru a șterge o linie din tabelul Client sau Product, va trebuie să furnizeze id-ul liniei care să fie ștearsă, într-un text field din inferfața grafică și să apese butonul Remove, apoi View all pentru a vedea tabelul modificat.
   4. Pentru a edita o linie din tabelul Client sau Product, va trebuie sa furnizeze id-ul liniei care să fie editată și datele care să se modifice, să apese butonul Edit și apoi View all pentru a vedea tabelul modificat.
3. După ce a terminat operațiile asupra unui tabel, utilizatorul poate alege alt tabel sau să închidă interfața grafică.

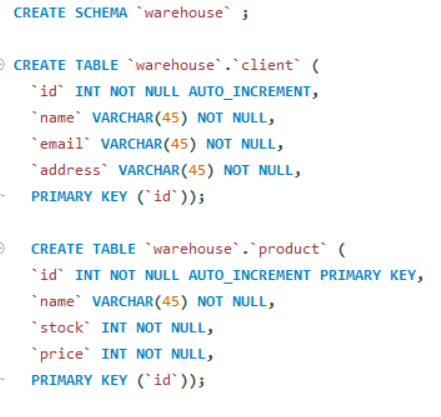
# **Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfața utilizator)**

* **Diagrama UML**

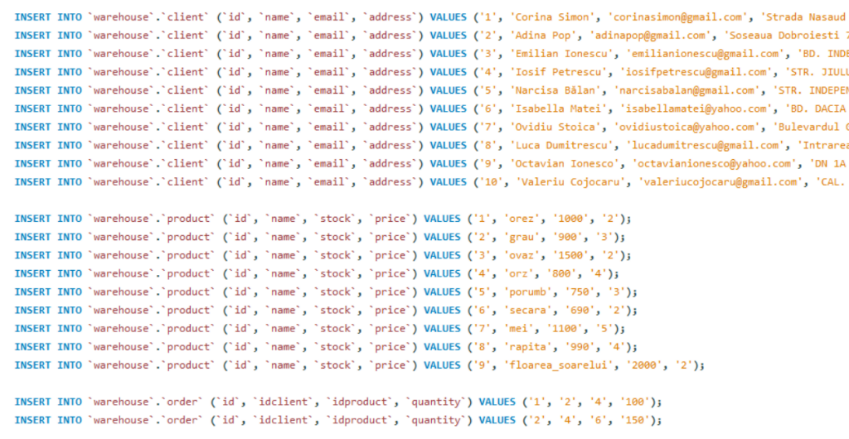


* **Structura bazei de date**







Am creat o nouă bază de date, am creat tabelele, cu primary key și foreign keys unde este cazul și am populat tabelele cu câteva date pentru început.

* **Structuri de date**

Structurile de date folosite sunt:

* Int
* ArrayList
* String
* TextField, TextArea, TableView și TableColumn – specifice JavaFX, folosite pentru a declara text field-urile, respectiv tabelele și coloanele acestora.
* Logger, Connection, ResultSet, PreparedStatement – pentru a realiza conexiunea cu baza de date, respectiv a lucra cu query-uri asupra datelor din aceasta.
* **Proiectarea claselor**

*Clasa App*: clasă în care se află metoda main, deci din care se rulează aplicația.

*Clasa Client*: clasă în care se află atributele specifice unui client, regăsite și în coloanele tabelului client – id, name, email și address, pentru care se scriu getters și setters. În plus, avem și mai mulți constructori, pentru diferite utilizări ulterioare.

*Clasa Order*: clasă în care se află atributele specifice unei comenzi, regăsite și în coloanele tabelului order – id, idClient, idProduct, quantity, pentru care se scriu getters și setters. În plus, avem și mai mulți constructori, pentru diferite utilizări ulterioare.

*Clasa Product*: clasă în care se află atributele specifice unui produs, regăsite și în coloanele tabelului product – id, name, stock și price, pentru care se scriu și getters și setters. În plus, avem și mai mulți constructori, pentru diferite utilizări ulterioare.

*Clasa AbstractDAO*: clasă în care se realizează, prin metode reflexive, query-uri generalizate (folosind tipul generic T) asupra datelor din baza de date – viewAll (adică SELECT \*), add (adică INSERT INTO), remove (DELETE FROM) și edit (UPDATE), respectiv conexiunea cu baza de date înainte executării query-urilor, prin intermediul unui obiect din clasa Connection și metoda getConnection din clasa ConnectionFactory.

*Clasa ClientDAO*: clasă care extinde clasa AbstractDAO, preluând toate metodele sale, care se vor realiza asupra obiectelor de tip Client.

*Clasa OrderDAO*: clasă care extinde clasa AbstractDAO, preluând toate metodele sale, care se vor realiza asupra obiectelor de tip Order.

*Clasa ProductDAO*: clasă care extinde clasa AbstractDAO, preluând toate metodele sale, care se vor realiza asupra obiectelor de tip Product. În plus, această clasă conține o metodă care creează un query asupra coloanei stock, care se regăsește doar în tabelul product, pentru a o modifica. Acesta ne va ajuta la modificarea stocului unui produs atunci când se face o nouă comandă validă.

*Clasa ClientBLL*: clasă în care se aplică logica dezvoltată în clasa AbstractDAO asupra tabelului cu clienți – se implementează operațiile viewAll, add, remove, edit asupra clienților.

*Clasa OrderBLL*: clasă în care se aplică logica dezvoltată în clasa AbstractDAO asupra tabelului cu comenzi – se implementează operațiile viewAll, add asupra comenzilor.

*Clasa ProductBLL*: clasă în care se aplică logica dezvoltată în clasa AbstractDAO asupra tabelului cu produse – se implementează operațiile viewAll, add, remove, edit și editStock asupra produselor.

*Clasa ConnectionFactory*: clasă în care se pregătește conexiunea cu baza de date, prin furnizarea username-ului, parolei etc. Prin intermediul metodei publice getConnection din această clasă se poate realiza conexiunea și extragerea datelor din tabele.

*Clasa Controller*: clasă în care se implementează funcționalitatea butoanelor și text field-urilor din interfața grafică. Atributele de tip TextField sunt folosite pentru ca utilizatorul sa poată introduce noi date în baza de date, respectiv să selecteze id-ul obiectelor pe care vrea să le șteargă. Atributul de tip TextArea se referă la chenarul în care se afișează mesajul de eroare atunci când se realizează o comandă asupra unui produs cu stoc insuficient. De asemenea, folosesc și trei atribute de tip TableView, acestea reprezentând tabelele cu date extrase din baza de date. Metodele din această clasă descriu funcționalitatea butoanelor View all, Add, Remove, Edit, Client View, Product View, Order View din interfața grafică.

* **Algortimi**

Algoritmii din cadrul acestui proiect se referă la dezvoltarea logicii de accesare a datelor din baza de date, din clasa AbstractDAO, folosind metode reflexive.

Inițial ne definim o clasă generică pentru a putea abstractiza toate metodele din această clasă. Aceasta va putea fi inlocuită ulterior cu una dintre clasele model – Client, Order, Product, pentru a specifica asupra cărui tabel se vor realiza operațiile.

Implementăm 4 metode pentru crearea query-urilor de realizat asupra tabelelor din baza de date – createSelectQuery, createInsertQuery, createDeleteQuery, createUpdateQuery. Acestea vor fi generalizate, pentru a se potrivi oricărui tabel. Prima metodă efectuează SELECT \* asupra unui tabel, pentru a prelua toate datele sale. Metoda pentru operația delete este la fel de simplă ca prima, însă aceasta se realizează în funcție de id-ul obiectului, adică primul field din fiecare clasă.

Metodele pentru insert și update sunt mai complexe – ele preiau numele tabelei, apoi accesează toate field-urile unei clase, pentru că operațiile acestea au structurile următoare:

INSERT INTO table\_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);

UPDATE table\_name  
SET column1 = value1, column2 = value2, ...  
WHERE condition;

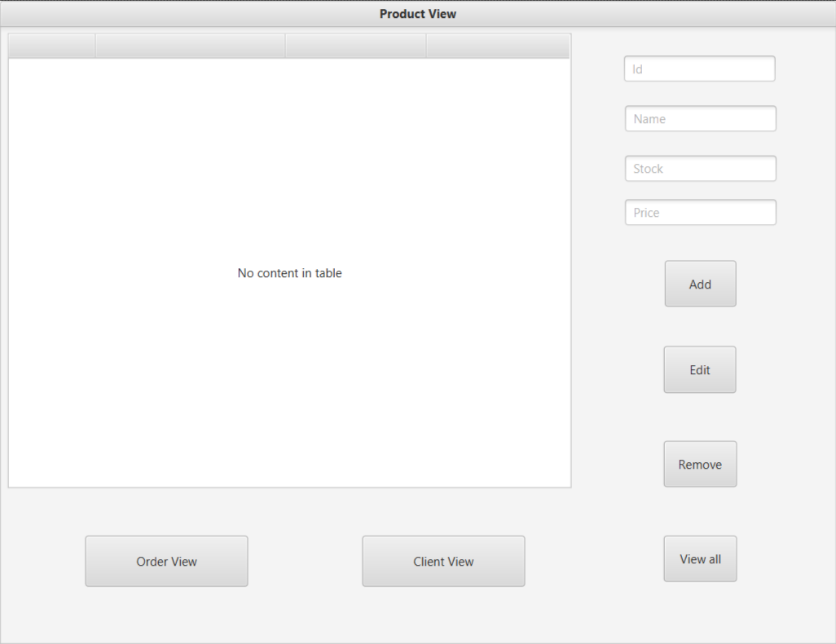
Valorile care sunt inserate, respectiv atribuite coloanelor asupra cărora se face update sunt realizate folosind obiecte de tipul PropertyDescriptor, Method și Object, pentru a invoca metodele de preluare a datelor.

După crearea query-urilor, acestea sunt apelate în funcțiile viewAll, add, remove, edit, în care se realizează conexiunea cu baza de date și executarea interogărilor. Metoda viewAll returnează o listă de obiecte obținute prin apelarea funcției createObjects asupra obiectului de tip ResultSet rezultat în urma interogării datelor dintr-un tabel.

Altă metodă importantă este cea prin care se genereaza reflexiv header-ul pentru fiecare tabel – setHeader. Aceasta returnează TableView<T>, unde T va fi una dintre clasele Client, Product, Order, după caz.

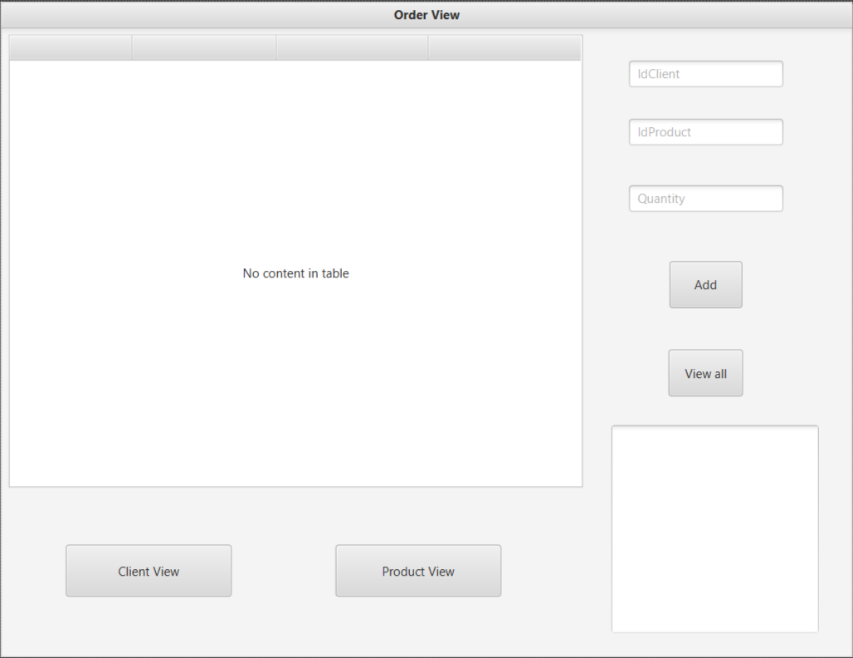
* **Interfața utilizator**

Pentru a realiza interfața grafică folosind JavaFX, am creat un proiect Java de tip Maven și am dezvoltat interfața incluzând dependințele specifice, apoi modelând aspectul în SceneBuilder. Am ales cadranul care să stea la bază, de tipul TitledPane, apoi am adăugat un tabel de tipul TableView și în jurul său TextField-uri, TextArea și butoane pentru schimbarea window-ului sau pentru efectuarea operațiilor.



Window-ul care apare la apăsarea butonului Product View.

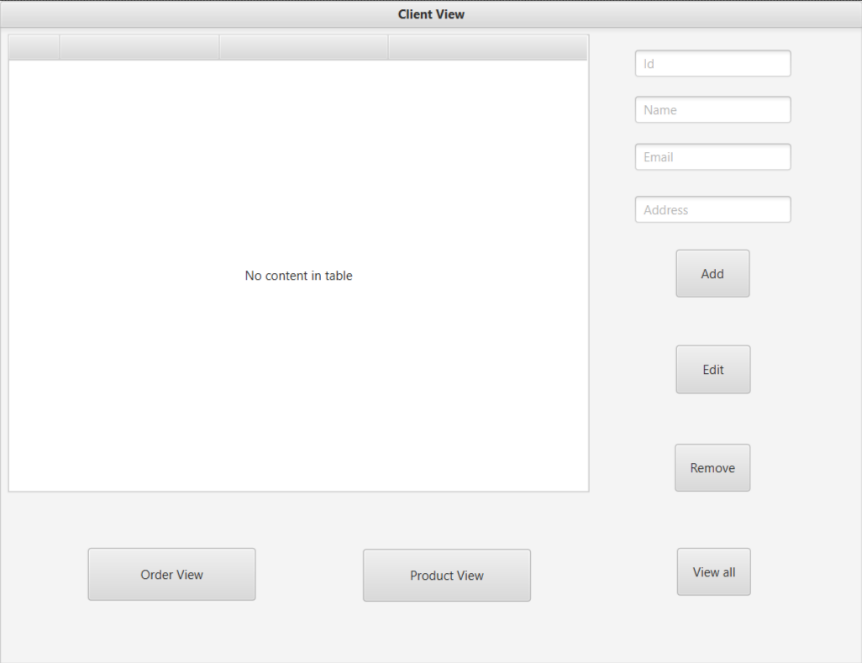
TextField-urile și TableView-ul au cate un fx:id după care pot fi identificate în clasa Controller, iar butoanele au câte o metodă care să le corespundă, unde este implementată funcționalitatea.



Window-ul care apare la apăsarea butonului Order View.

TextField-urile și TableView-ul au câte un fx:id după care să fie identificate în Controller, iar butoanele câte o metodă în care se implementează funcționalitatea.

Cadranul din colțul drept de jos este TextArea în care se afișează mesajul de eroare în cazul în care un produs are stoc insuficient.

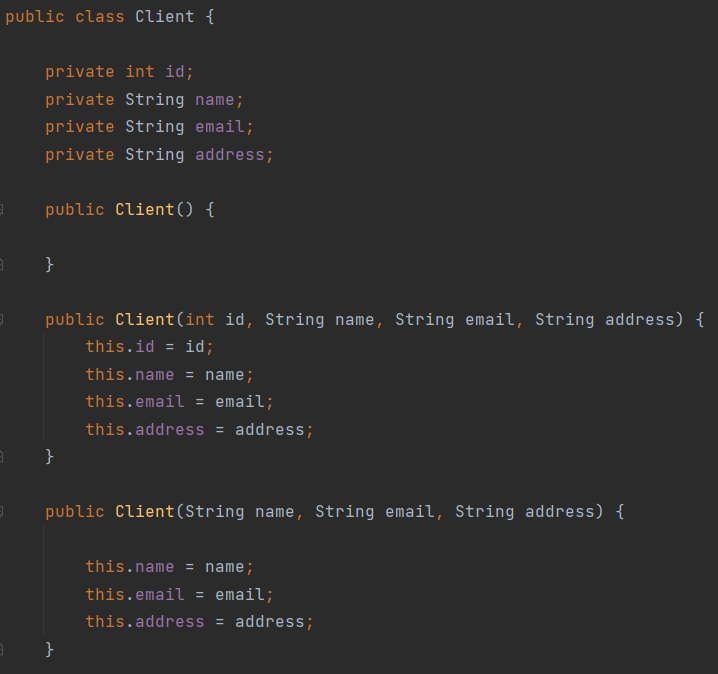
Window-ul care apare când se apasă butonul Client View.

TextField-urile și TableView-ul au câte un fx:id după care sunt identificate în Controller, iar butoanele au câte o metodă în aceeași clasă.

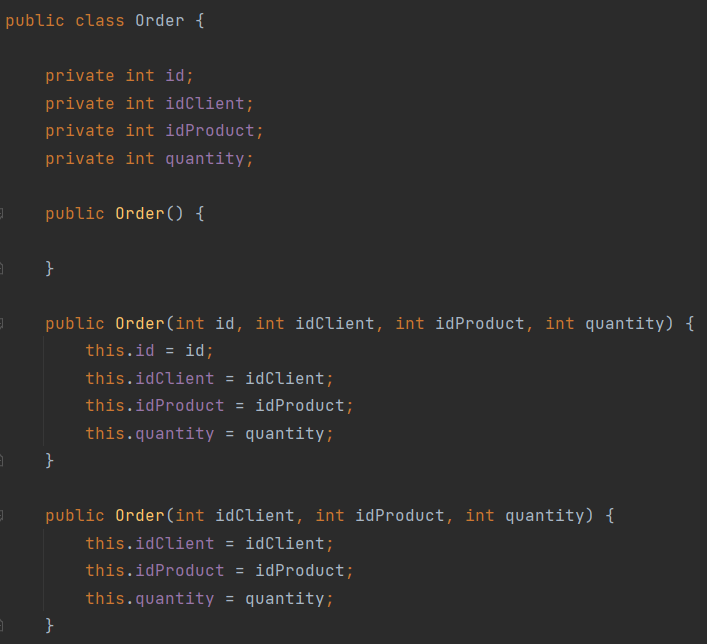
Atât metodele care corespund butoanelor, cât și celelalte componente sunt mereu precedate de adnotația @FXML.

# **Implementare**

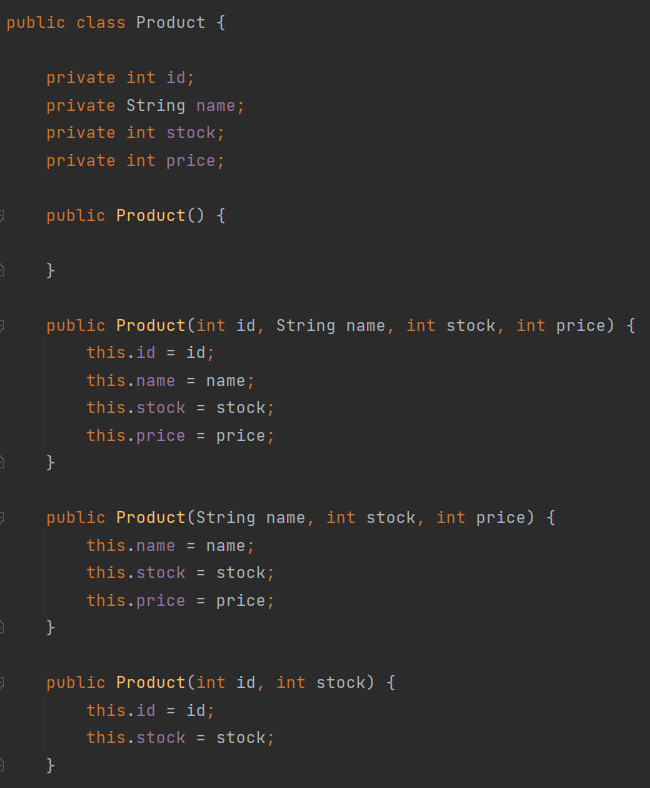
Încep prin a prezenta clasa Client. Acestea sunt atributele, care corespund cu denumirile coloanelor din baza de date. De asemenea, am folosit 3 constructori și în rest getters și setters pentru toate atributele:



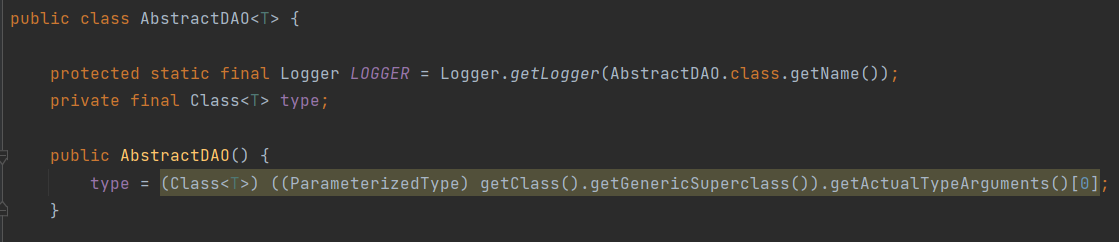
În clasa Order am aceste atribute, care corespund cu coloanele din tabelul order și 3 constructori, getters și setters:



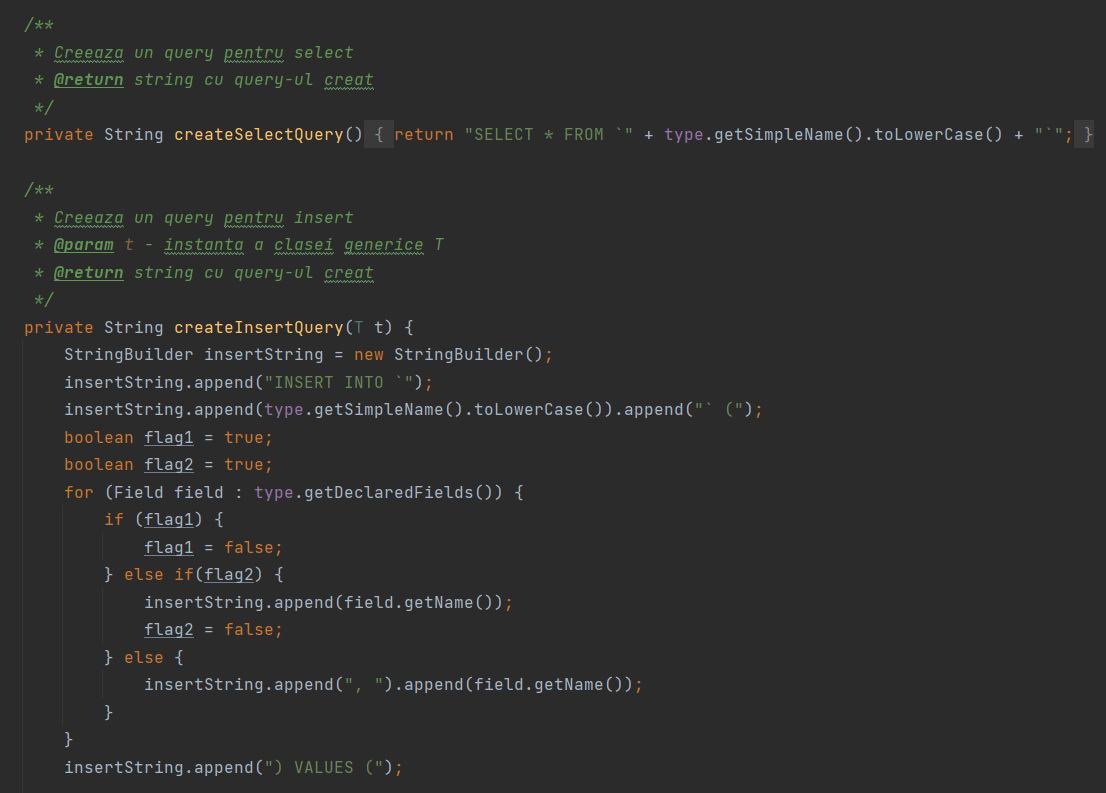
În clasa Product am aceste atribute, care corespund cu coloanele din tabelul product, 4 constructori și getters și setters:

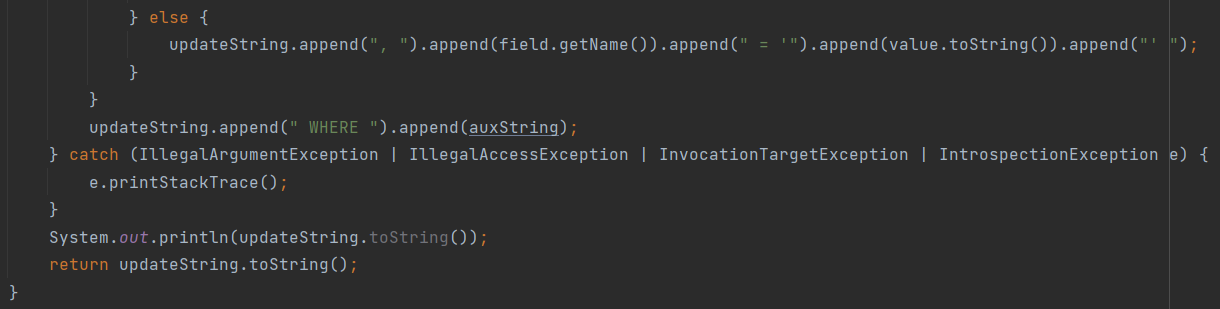
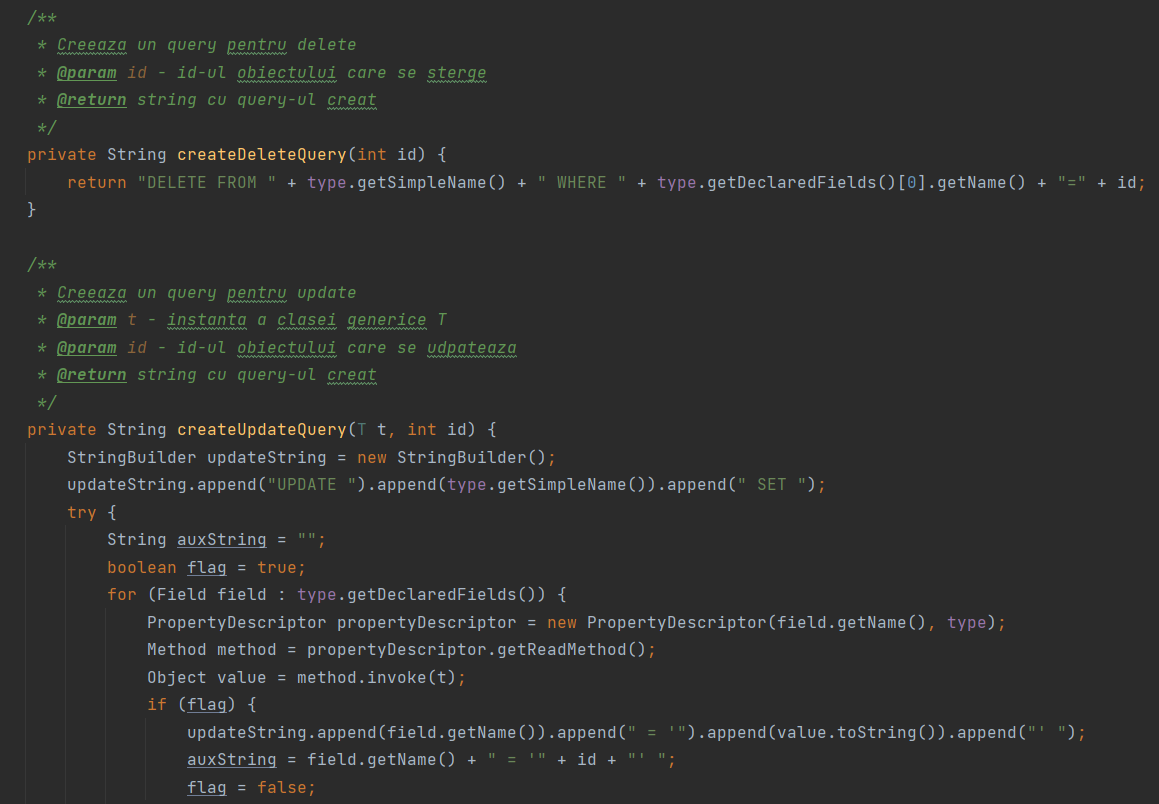


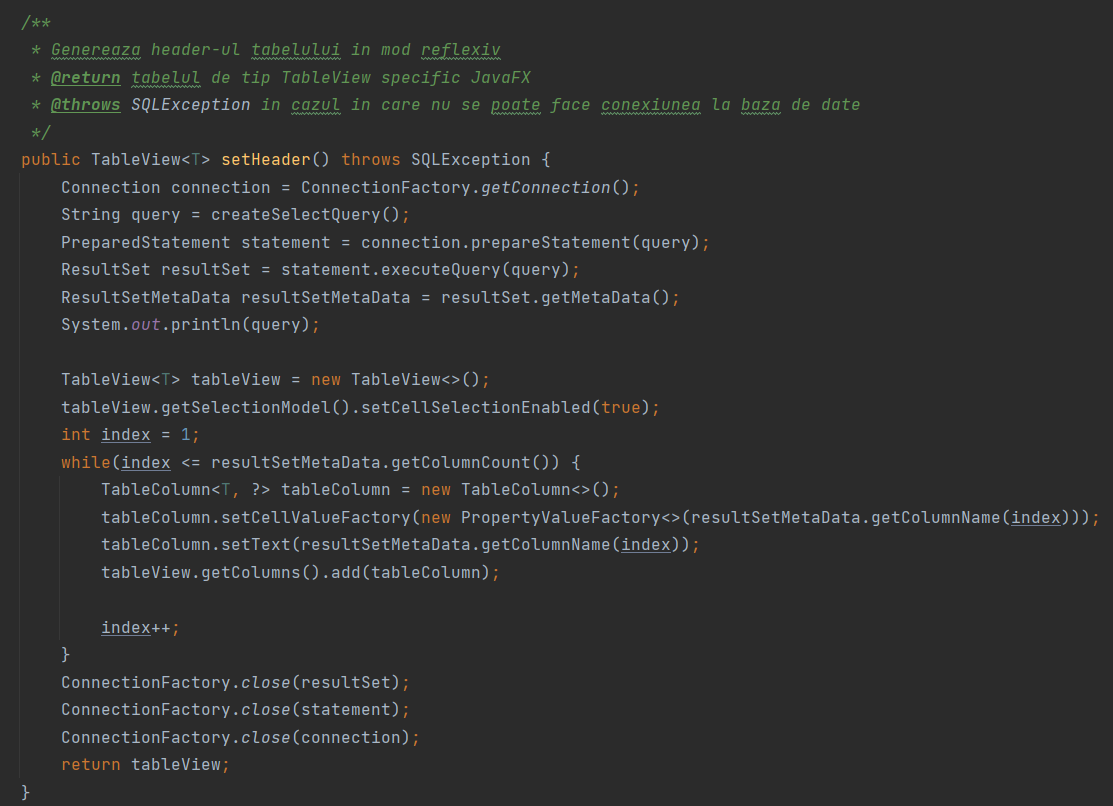
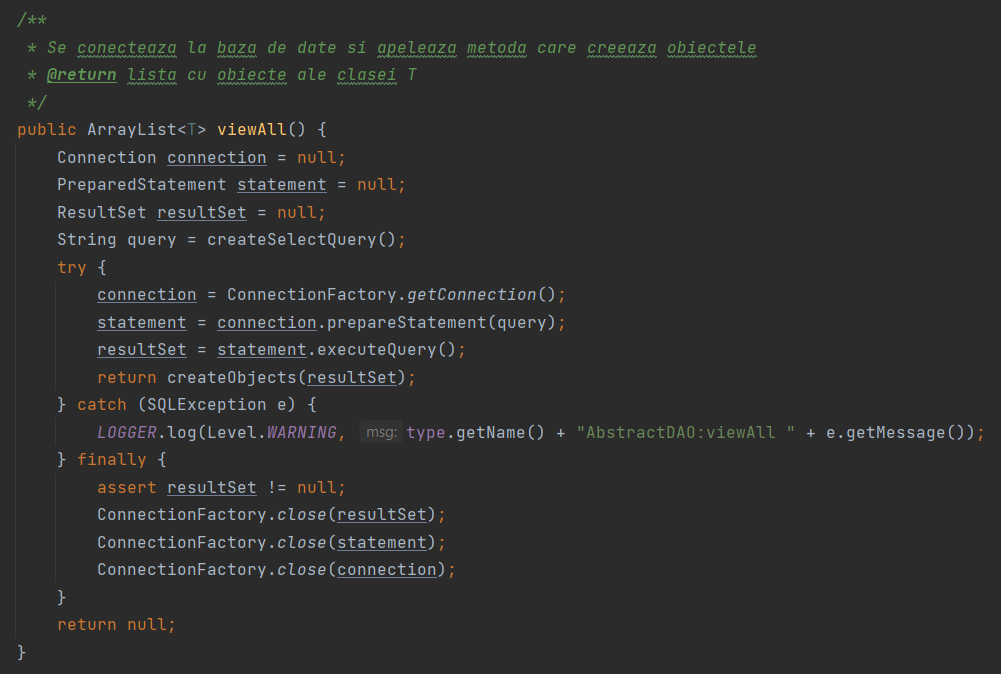
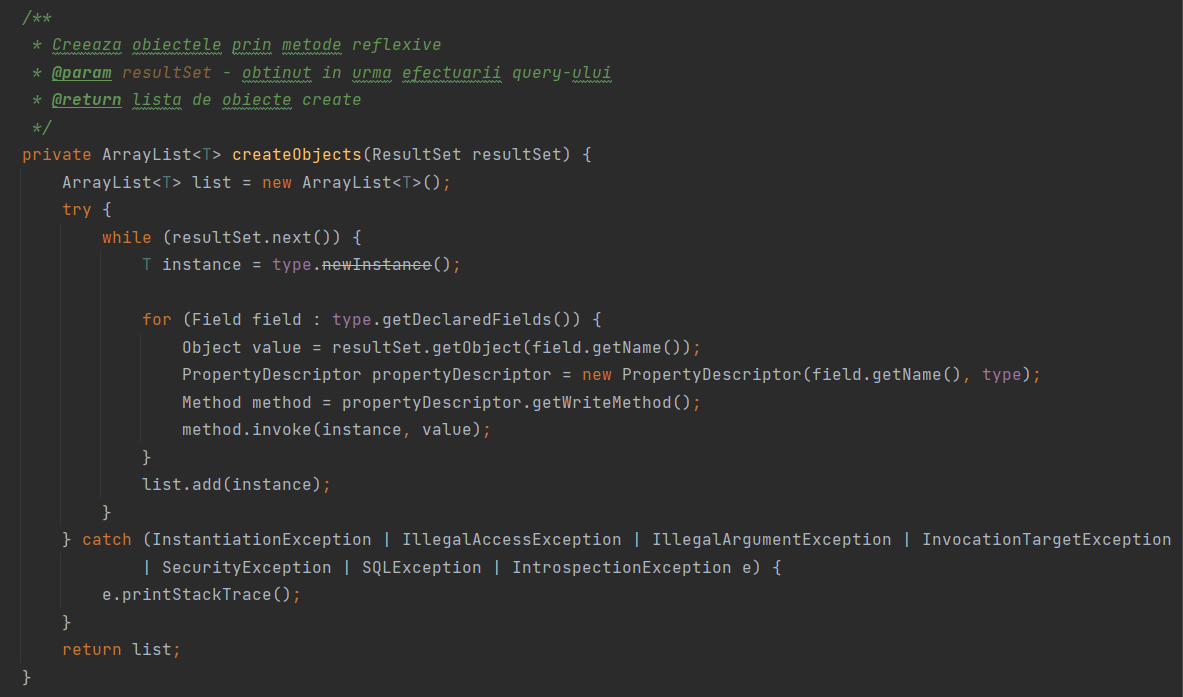
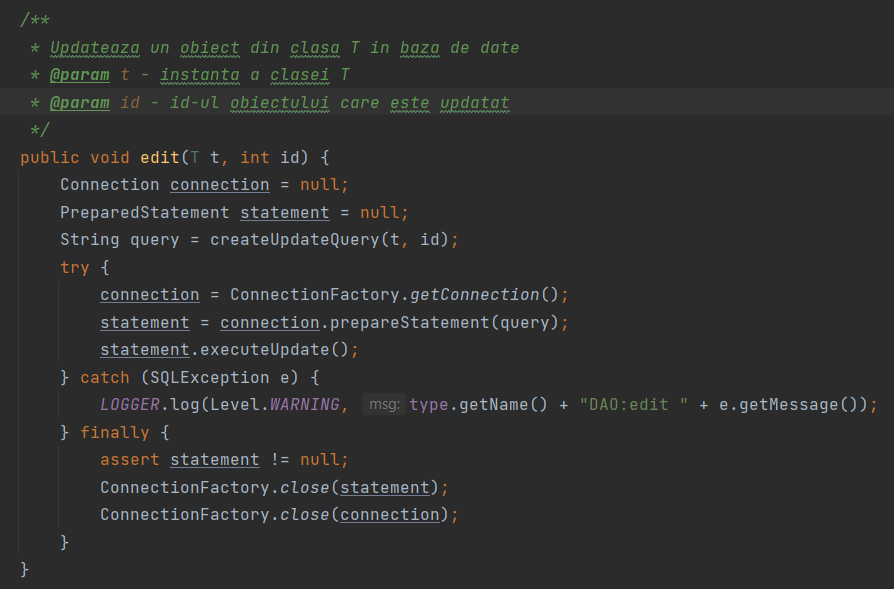
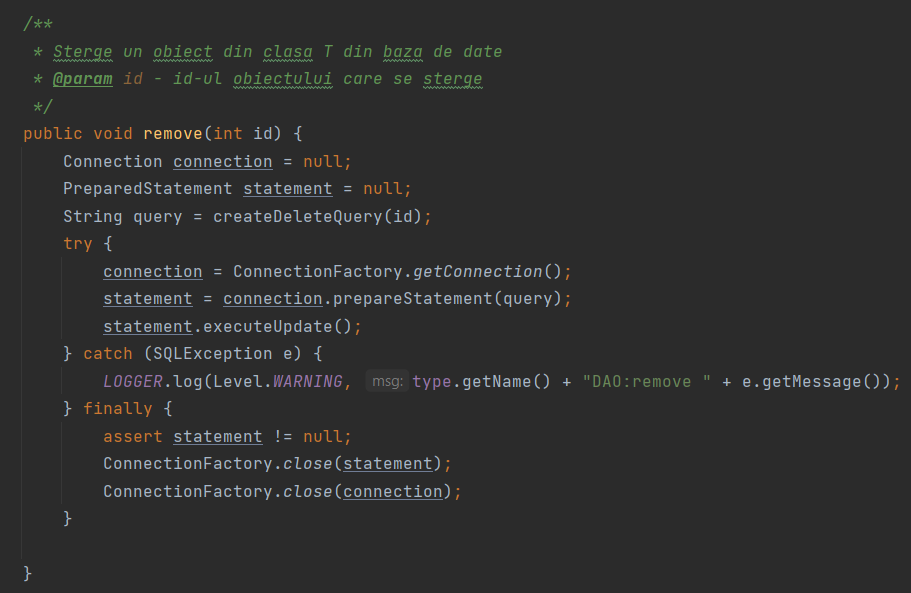
În clasa AbstractDAO acestea sunt atributele. Type este clasa generică care va fi inlocuită cu denumirea uneia dintre clasele Client, Product, Order, după caz:



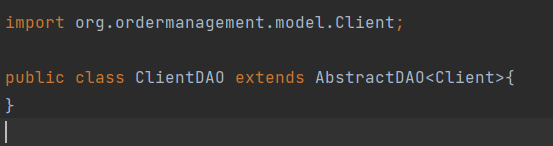
Metodele au fost documentate folosind JavaDoc:



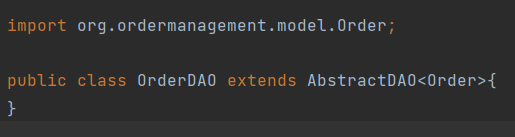




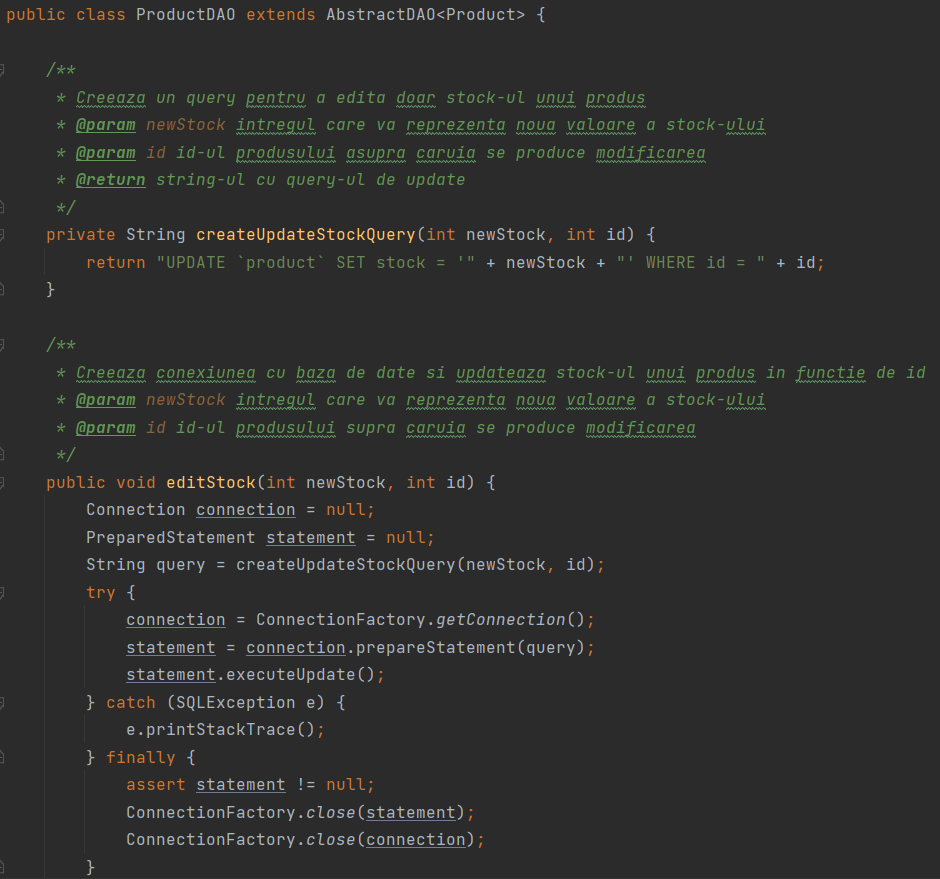
Clasa ClientDAO doar extinde clasa AbstractDAO:



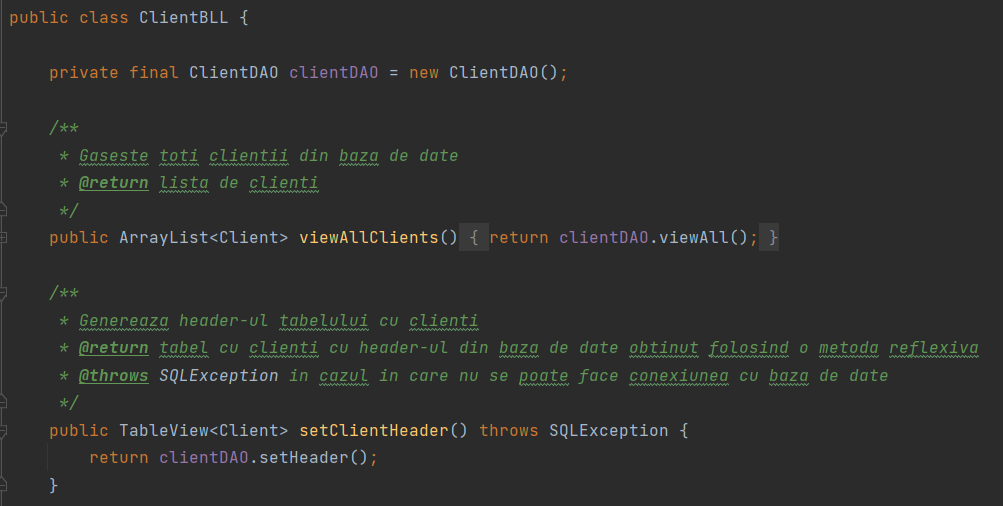
Clasa OrderDAO extinde clasa AbstractDAO:

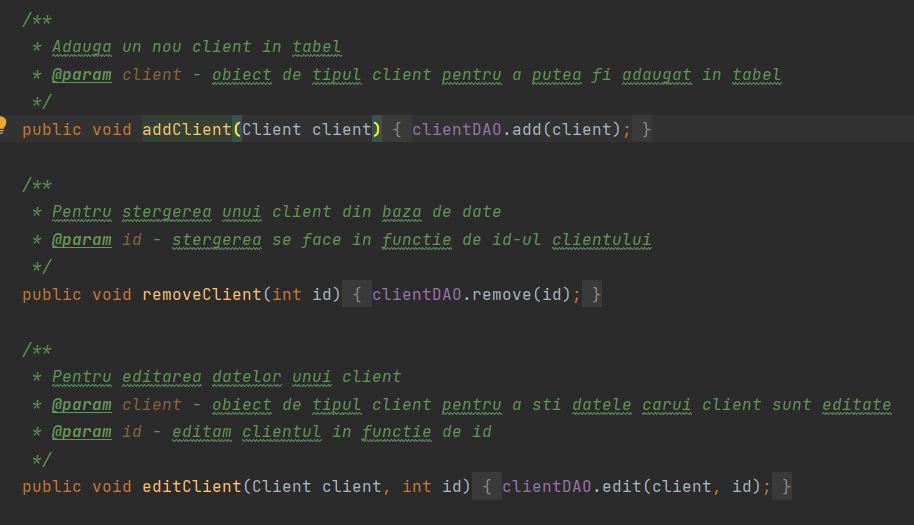


Clasa ProductDAO extinde clasa AbstractDAO, dar implementează în plus și metode pentru a face update asupra coloanei stock din tabelul product:

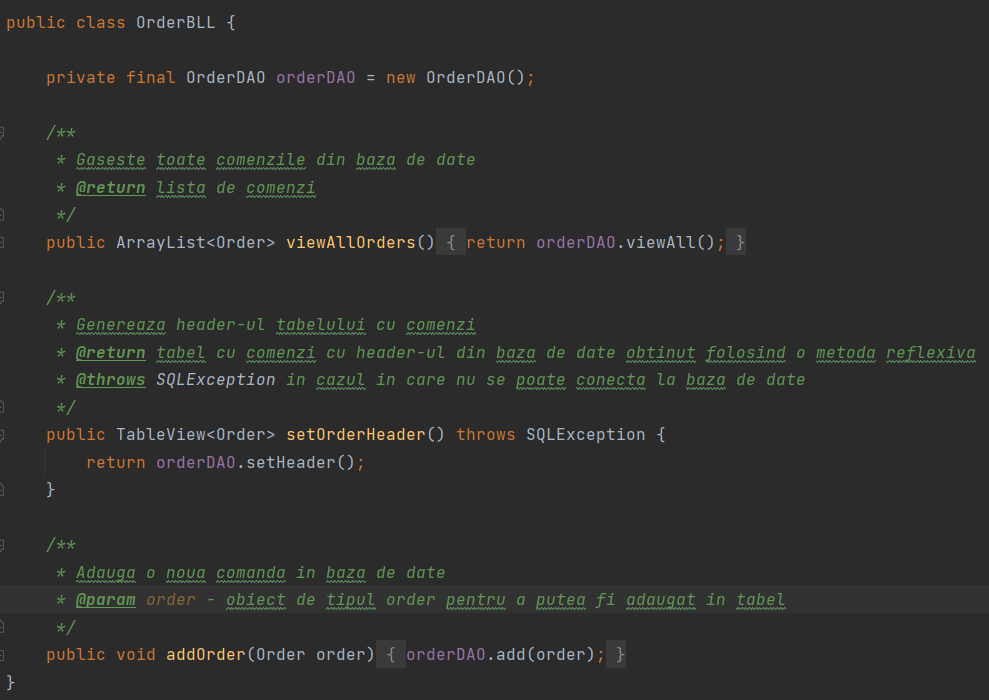


Clasa ClientBLL are drept atribut un obiect de tipul ClientDAO pentru a putea accesa metodele acestei clase, iar toate metodele sunt documentate folosind Java Doc:

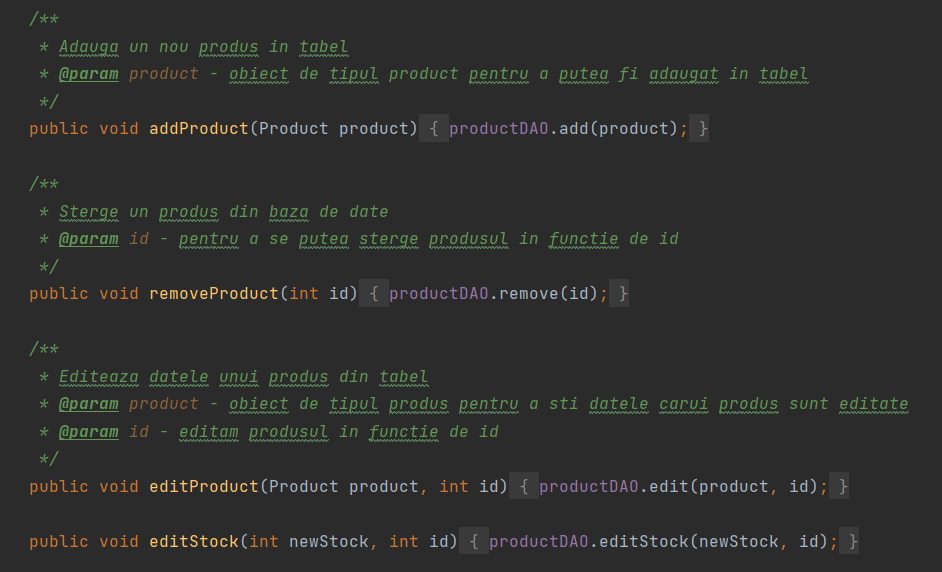




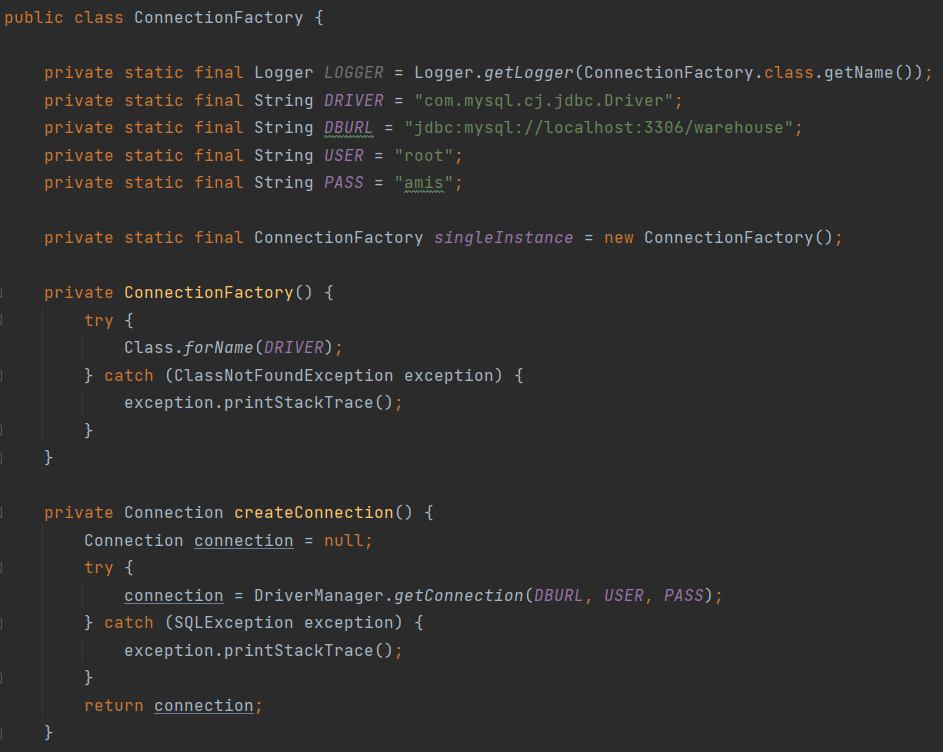
Clasa OrderBLL are un atribut de tipul OrderDAO pentru a accesa metodele acestei clase, iar metodele sunt documentate folosind Java Doc:



Clasa ProductBLL are atributul de tip ProductDAO pentru a accesa logica dezvoltată în pachetul dao, iar toate metodele sunt documentate cu Java Doc:



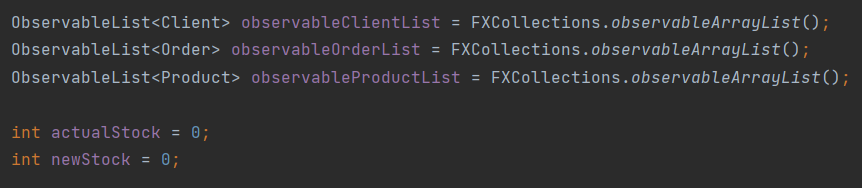
Clasa ConnectionFactory conține date pentru accesarea bazei de date warehouse și metode care fac posibilă conexiunea cu aceasta:



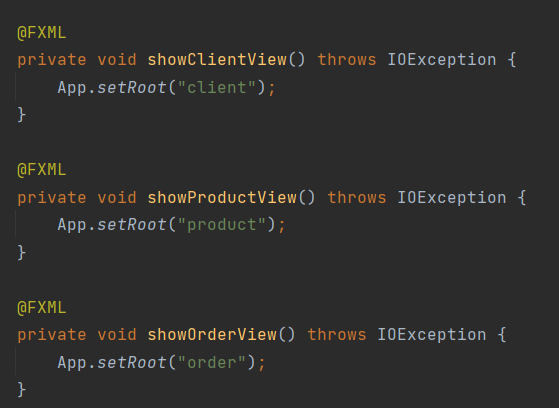
În clasa Controller avem declarate inițial elementele din interfața grafică, având câte o adnotație @FXML, dar si obiecte din clasele ClientBLL, OrderBLL, ProductBLL pentru a apela metodele implementate anterior:



Avem 3 liste ObservableList, folosite pentru popularea tabelelor, iar întregii actualStock și newStock sunt folosiți pentru a modifica stocul unui produs după înregistrarea unei noi comenzi:



Prin intermediul acestor metode se realizează schimbarea scenei atunci când se apasă unul dintre butoanele Client View, Product View, Order View:



Aceste metode reprezintă funcționalitatea butoanelor din window-ul Client View:

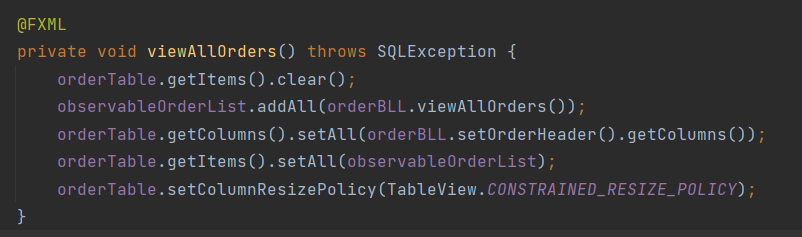


Aceste metode reprezintă funcționalitatea butoanelor din window-ul Product View:



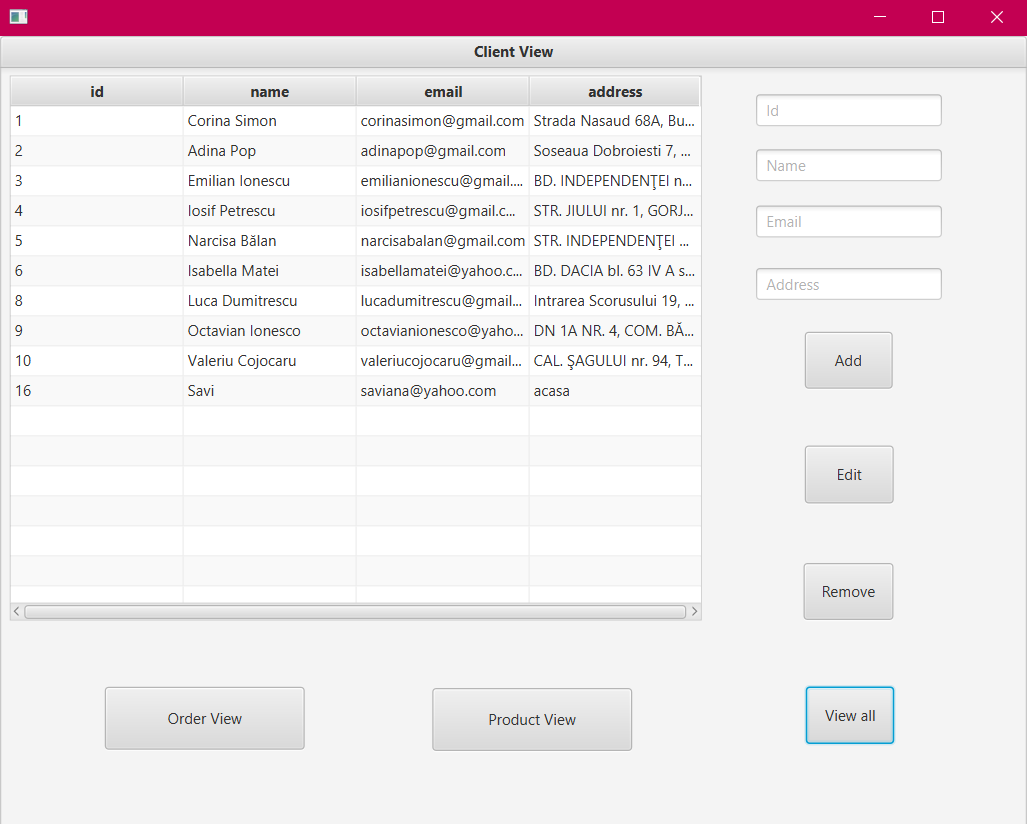
Acestea sunt metodele de adăugare a unei noi comenzi și vizualizare a tuturor comenzilor. Prima metodă este mai complexă deoarece aici realizez verificarea datelor – trebuie să se realizeze comenzi doar folosind clienți și produse existente. De asemenea, tot aici verific și modific stocul produsului după efectuarea unei comenzi:

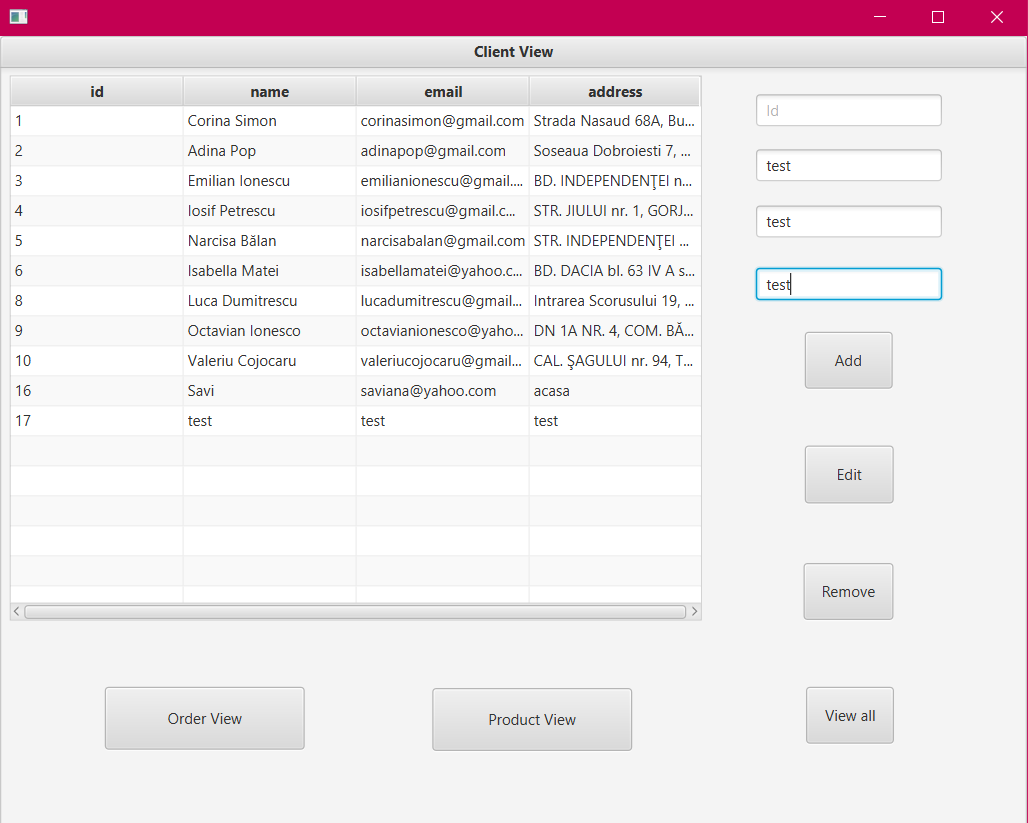




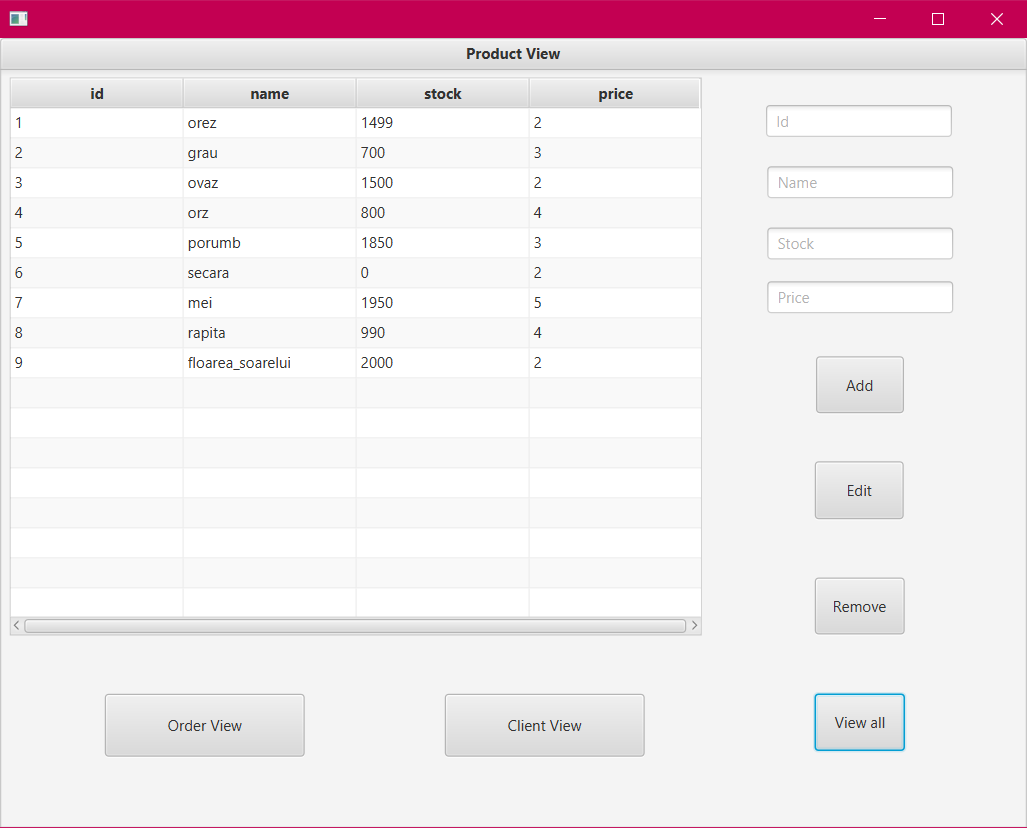
# **Rezultate**

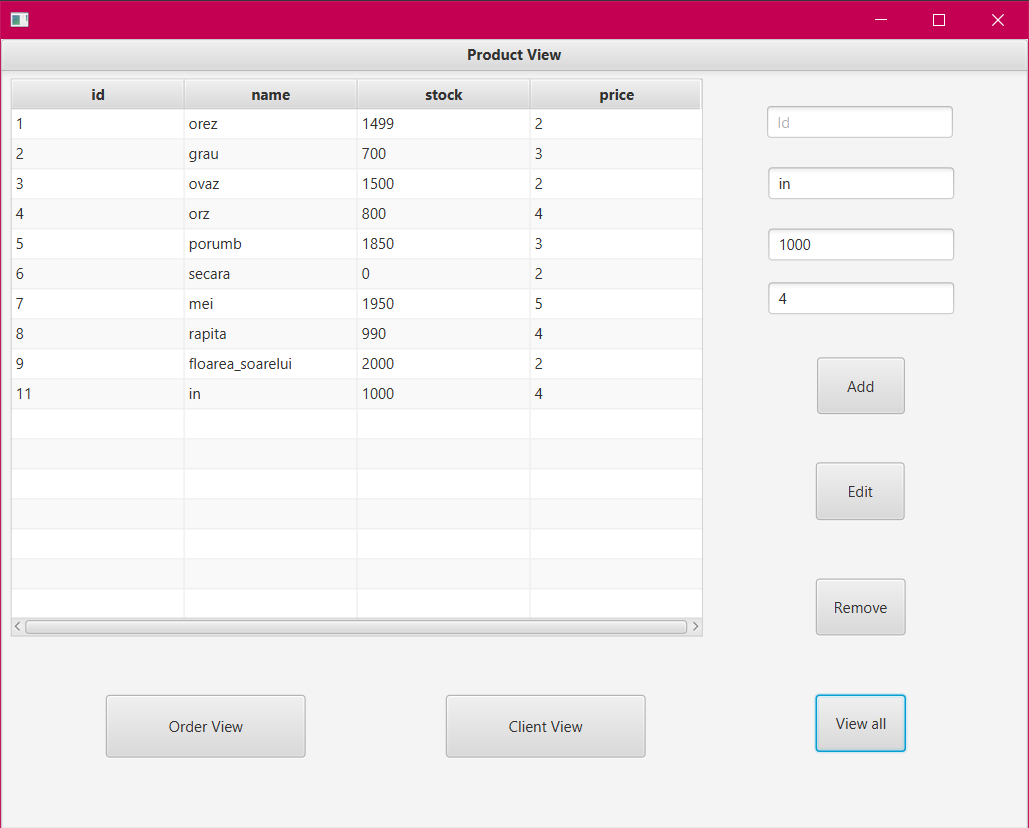
Teste din Client View:



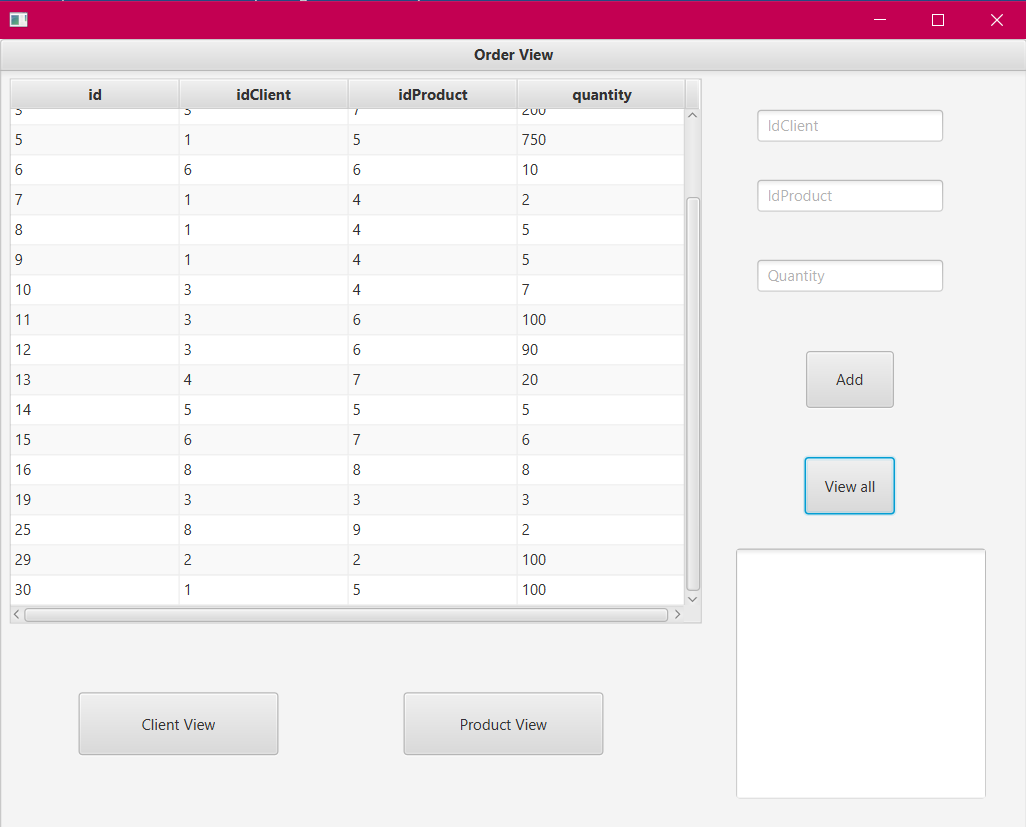


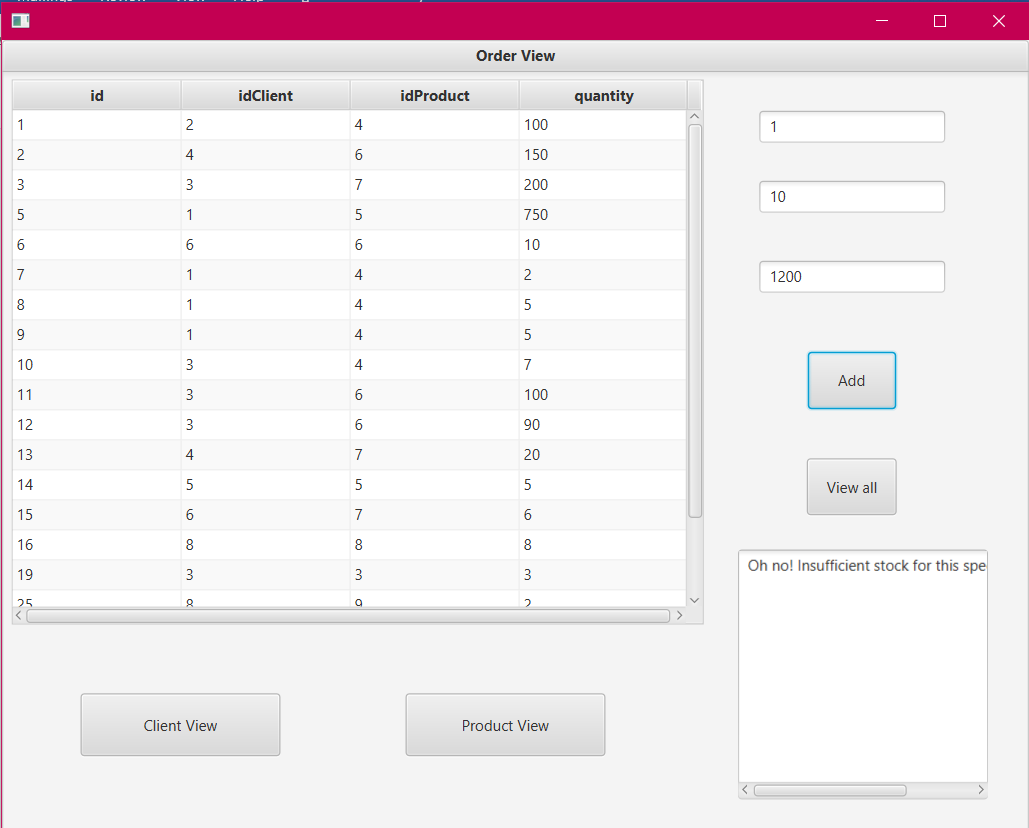
Teste din Product View:





Teste din Order View:



Aici se afișează mesajul de eroare deoarece am introdus quantity = 1200, iar stocul este doar 1000 pentru produsul cu id-ul 10.

# **Concluzii**

În concluzie, această temă m-a ajutat să exersez lucrul cu baze de date și să înțeleg mai bine cum se realizează conexiunea între aplicația mea Java și baza de date creată, dar și preluarea datelor din tabele. De asemenea, prin intermediul acestei teme am început să mă familiarizez cu conceptul de reflexivitate în Java și m-a ajutat să înțeleg cât de folositor poate fi acesta, mai ales în cazul în care lucrăm cu baze de date.

Ca dezvoltare ulterioară, aș vrea să introduc și funcționalitatea de a genera o factură în urma efectuării unei comenzi, dar și mai multe metode reflexive în clasa AbstractDAO.

# **Bibliografie**

<https://www.w3schools.com/sql/>

<https://gitlab.com/utcn_dsrl/pt-layered-architecture>  
<https://lucid.app/lucidchart>

<https://www.jetbrains.com/help/idea/working-with-code-documentation.html#generate-javadoc>

<https://www.oracle.com/technical-resources/articles/java/javareflection.html>

<https://www.youtube.com>

<http://tutorials.jenkov.com/java-reflection/index.htmlhttps://dzone.com/articles/layered-architecture-is-good#:~:text=What%20Is%20Layered%20Architecture%3F,the%20particular%20concern%20it%20represents>