

ORDERS MANAGEMENT

DOCUMENTATIE - TEMA 3 - TP



Păcurar Irina

UTCN, CTI, SERIA A, grupa 30223

ÎNDRUMĂTORI: PROF. DR. ING. TUDOR CIOARĂ, OVIDIU MAJA

- mai 2024 -

***Cuprins***

[***1. Obiectivul temei 4***](#_Toc166466157)

[***2. Analiza problemei, modelare, scenarii, cazuri de utilizare 5***](#_Toc166466158)

[***2.1.1. Analiza Problemei 5***](#_Toc166466159)

[***2.1.2. Modelare 5***](#_Toc166466160)

[***2.1.3. Scenarii 5***](#_Toc166466161)

[***2.1.4. Cazuri de Utilizare 5***](#_Toc166466162)

[***3. Proiectare 0***](#_Toc166466163)

[***4. Implementare pachete 1***](#_Toc166466164)

[***4.1. Org.example – Main 1***](#_Toc166466165)

[***4.2. Model 2***](#_Toc166466166)

[***4.2.1. Bill 2***](#_Toc166466167)

[***4.2.2. Client 2***](#_Toc166466168)

[***4.2.3. Order 4***](#_Toc166466169)

[***4.2.4. Product 5***](#_Toc166466170)

[***4.3. Bll 6***](#_Toc166466171)

[***4.3.1. Validators 6***](#_Toc166466172)

[***4.3.2. BillBll 9***](#_Toc166466173)

[***4.3.3. ClientBll 10***](#_Toc166466174)

[***4.3.4. OrderBll 11***](#_Toc166466175)

[***4.3.5. ProductBll 11***](#_Toc166466176)

[***4.4. Connection – ConnectionFactory 12***](#_Toc166466177)

[***4.5. Dao 13***](#_Toc166466178)

[***4.5.1. BillDao 13***](#_Toc166466179)

[***4.5.2. ClientDao 14***](#_Toc166466180)

[***4.5.3. GenericDao 14***](#_Toc166466181)

[***4.5.4. OrderDao 15***](#_Toc166466182)

[***4.5.5. ProductDao 16***](#_Toc166466183)

[***4.6. Presentation 16***](#_Toc166466184)

[***4.6.1. BillController 16***](#_Toc166466185)

[***4.6.2. BillView 17***](#_Toc166466186)

[***4.6.3. ClientController 17***](#_Toc166466187)

[***4.6.4. ClientView 18***](#_Toc166466188)

[***4.6.5. MainView 19***](#_Toc166466189)

[***4.6.6. OrderController 19***](#_Toc166466190)

[***4.6.7. OrderView 20***](#_Toc166466191)

[***4.6.8. ProductController 21***](#_Toc166466192)

[***4.6.9. ProductView 22***](#_Toc166466193)

[***4.6.10. TableModelUtils 24***](#_Toc166466194)

[***5. Rezultate 25***](#_Toc166466195)

[***6. Concluzii și dezvoltări ulterioare 25***](#_Toc166466196)

[***7. Bibliografie 26***](#_Toc166466197)

# Obiectivul temei

Proiectul solicită dezvoltarea unei aplicații de management al comenzilor pentru un depozit, care urmează arhitectura pe straturi și utilizează baze de date relaționale. Structura aplicației include patru tipuri principale de clase: modele de date care reprezintă structurile de date ale aplicației, clase de logică de afaceri pentru gestionarea operațiunilor aplicației, clase de prezentare pentru interfața grafică a utilizatorului și clase de acces la date pentru interacțiunea cu baza de date.

Sunt necesare: implementarea unui design orientat-obiect cu respectarea limitelor de lungime pentru clase și metode, utilizarea adecvată a JavaDoc pentru documentarea claselor, implementarea unei baze de date cu cel puțin trei tabele (Client, Produs, și Comandă), crearea unei interfețe grafice care permite gestionarea clienților și produselor, precum și plasarea comenzilor. Se solicită folosirea tehnicilor de reflecție pentru generarea dinamică a interfețelor și a query-urilor în baza de date, precum și definirea unei clase imutabile Bill în pachetul de model, utilizând înregistrările Java, pentru reprezentarea facturilor stocate într-o tabelă Log dedicată, care permite doar inserții și citiri.

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Pentru aplicația de gestionare a comenzilor într-un depozit, care folosește baze de date relaționale și arhitectura pe straturi, este crucial să se efectueze o analiză detaliată a problemei, modelarea sistemului, și să se definească scenariile și cazurile de utilizare relevante.

### Analiza Problemei

Aplicația trebuie să permită gestionarea eficientă a clienților, produselor și comenzilor. Trebuie să suporte operațiuni cum ar fi adăugarea, editarea și ștergerea datelor clienților și produselor, și procesarea comenzilor. Un aspect cheie este gestionarea stocurilor, aplicația trebuie să verifice disponibilitatea produselor și să actualizeze stocul după fiecare comandă. Toate aceste date trebuie să fie stocate într-o bază de date relațională, organizată în cel puțin trei tabele principale: Client, Produs și Comandă.

### Modelare

Modelul de date include trei entități principale:

* Client: Conține informații despre clienții depozitului.
* Produs: Deține detalii despre produsele disponibile în depozit, inclusiv stocul.
* Comandă: Asociază clienții și produsele cu cantitățile comandate.

De asemenea, va exista o tabelă Bill pentru a stoca facturi imutabile generate automat la fiecare comandă, utilizând recorduri Java.

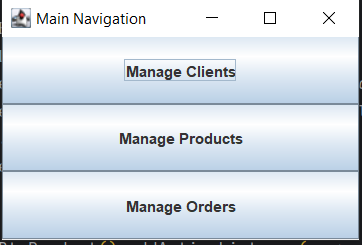
### Scenarii

Un scenariu tipic implică un utilizator al aplicației (angajat al depozitului sau client direct) care accesează interfața grafică pentru a vizualiza, adăuga, sau modifica informații despre produse sau clienți. De exemplu, un utilizator poate dori să adauge un nou produs în stoc, să actualizeze informațiile unui client, sau să plaseze o comandă pentru un client existent.

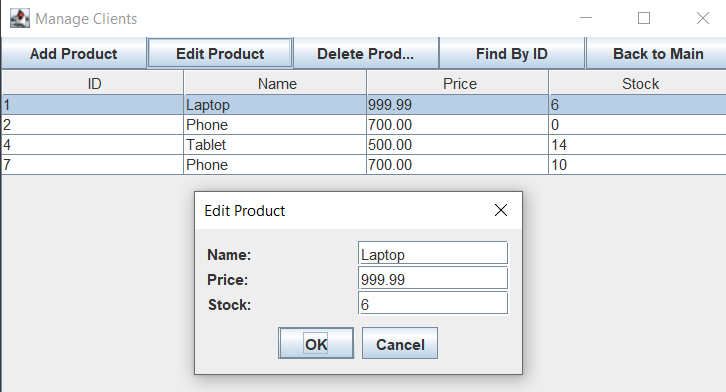
### Cazuri de Utilizare

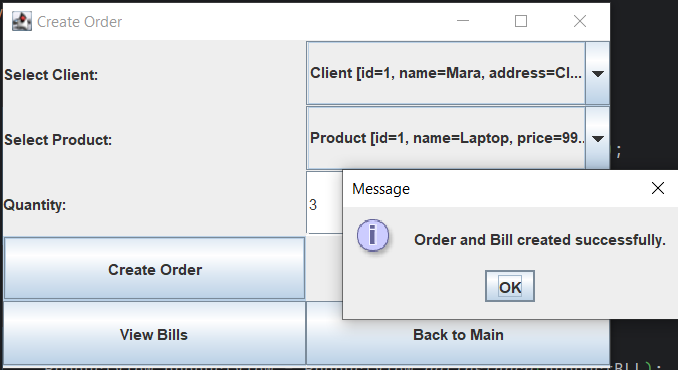
1. Adăugarea unui nou client: Utilizatorul introduce detaliile unui nou client într-un formular și le salvează în baza de date.
2. Editarea detaliilor unui client: Utilizatorul selectează un client dintr-o listă, modifică detaliile necesare și actualizează înregistrarea în baza de date.
3. Ștergerea unui client: Utilizatorul selectează un client și îl șterge din sistem.
4. Adăugarea unui nou produs: Detaliile produsului sunt introduse și salvate în baza de date.
5. Editarea unui produs: Modificările asupra unui produs existent sunt făcute prin interfața de editare.
6. Plasarea unei comenzi: Utilizatorul selectează un produs și un client, introduce cantitatea dorită și plasează comanda. Dacă stocul este insuficient, se afișează un mesaj de avertizare.
7. Generarea unei facturi: Odată ce comanda este finalizată, se generează automat o factură imutabilă, care este stocată în tabelul Bill.

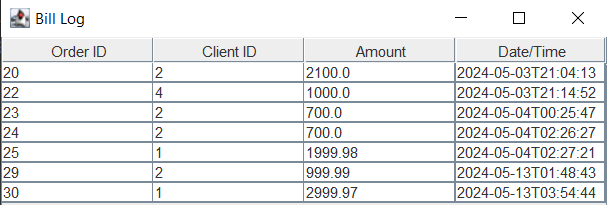
Fiecare caz de utilizare implică interacțiuni directe cu baza de date prin intermediul claselor de acces la date, care folosesc tehnici de reflecție pentru a dinamiza crearea și interogarea datelor. Utilizarea unei arhitecturi pe straturi și a principiilor de programare orientată obiect asigură că aplicația este bine organizată și ușor de întreținut.





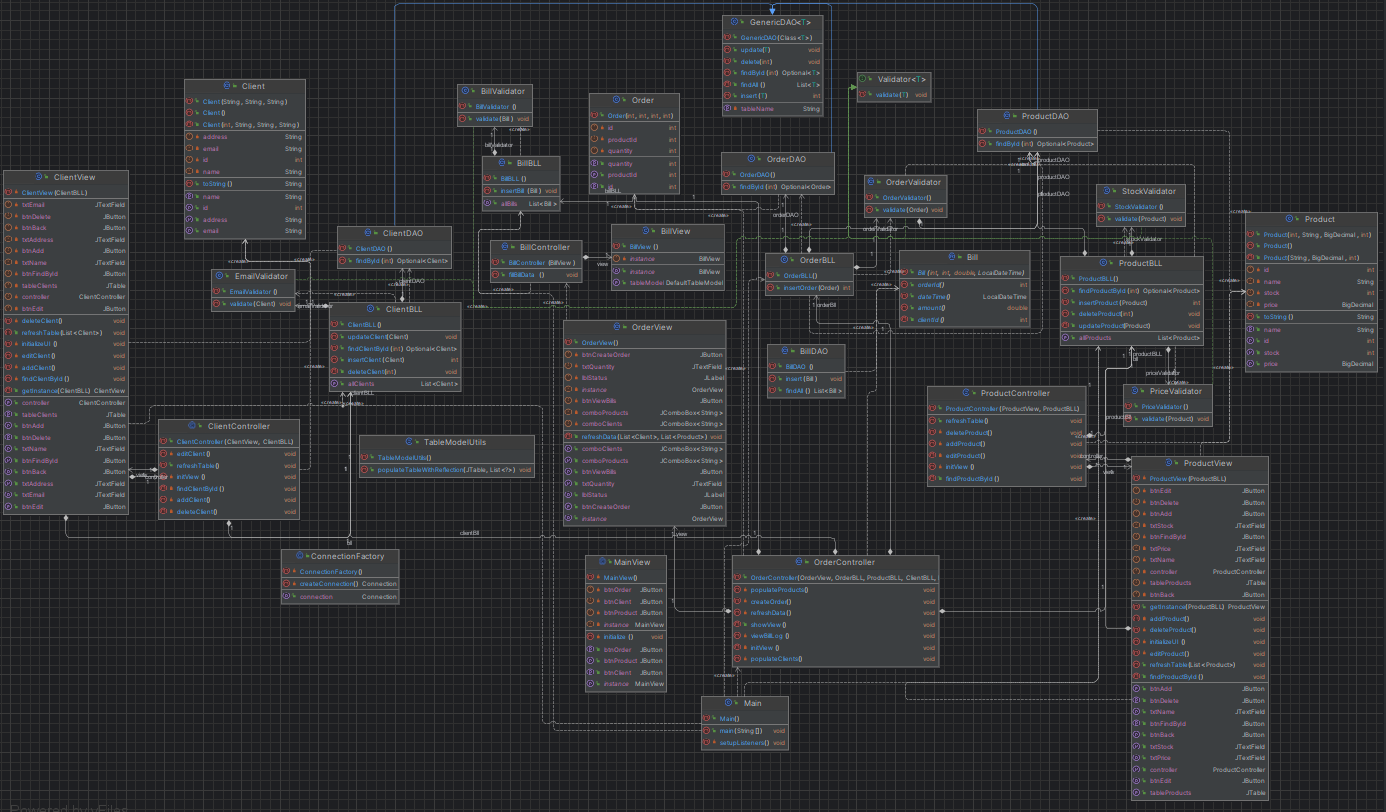




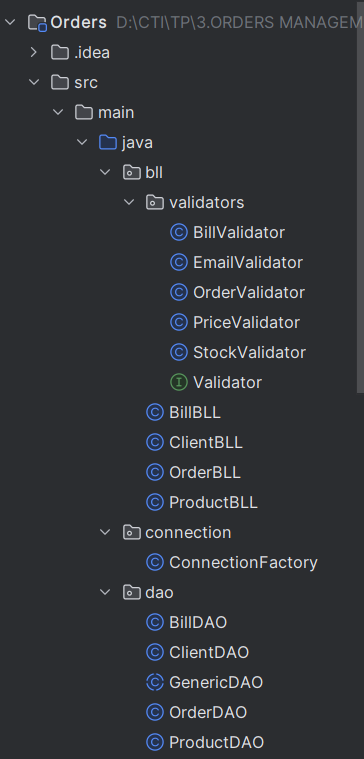


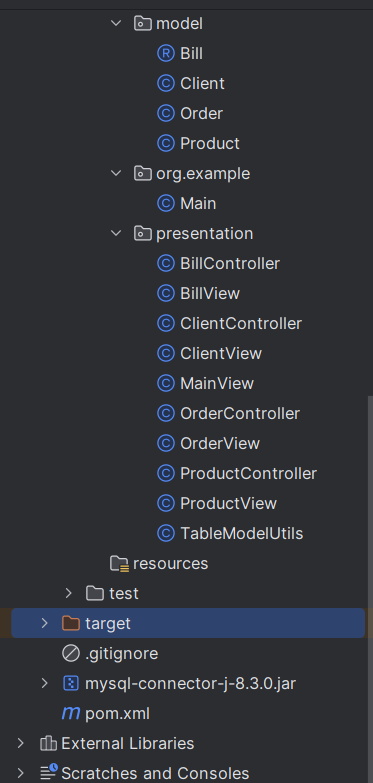
# Proiectare

***Clasele***



***Pachetele***

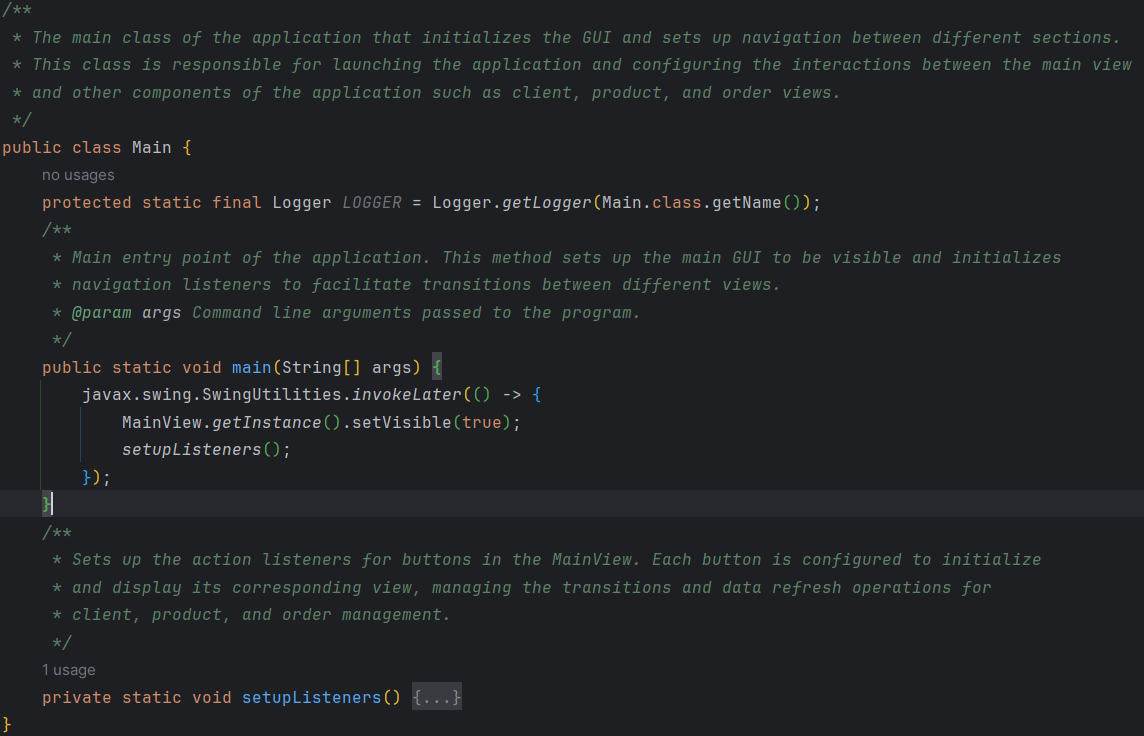




# Implementare pachete

## ***Org.example – Main***

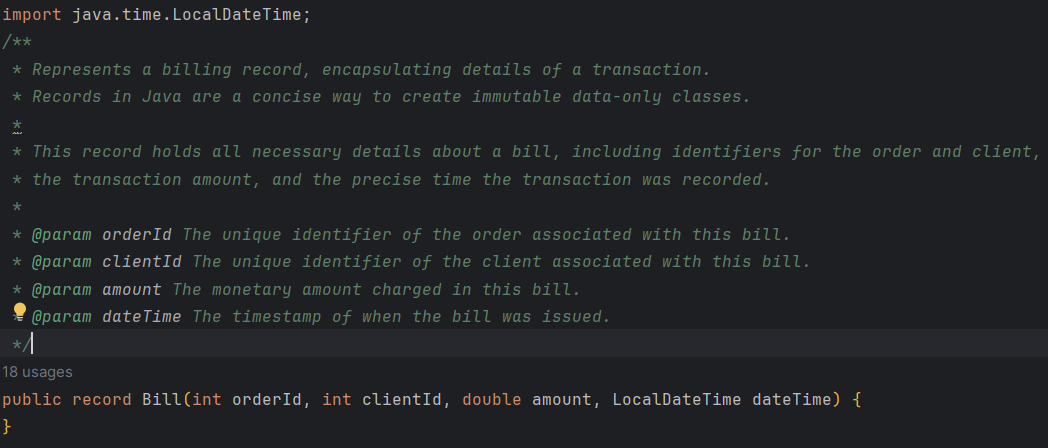
Clasa **Main** din pachetul **org.example** servește drept punct de intrare în aplicația de gestionare a comenzilor. Aceasta conține metoda **main**, care este metoda standard de lansare pentru orice aplicație Java.



## ***Model***

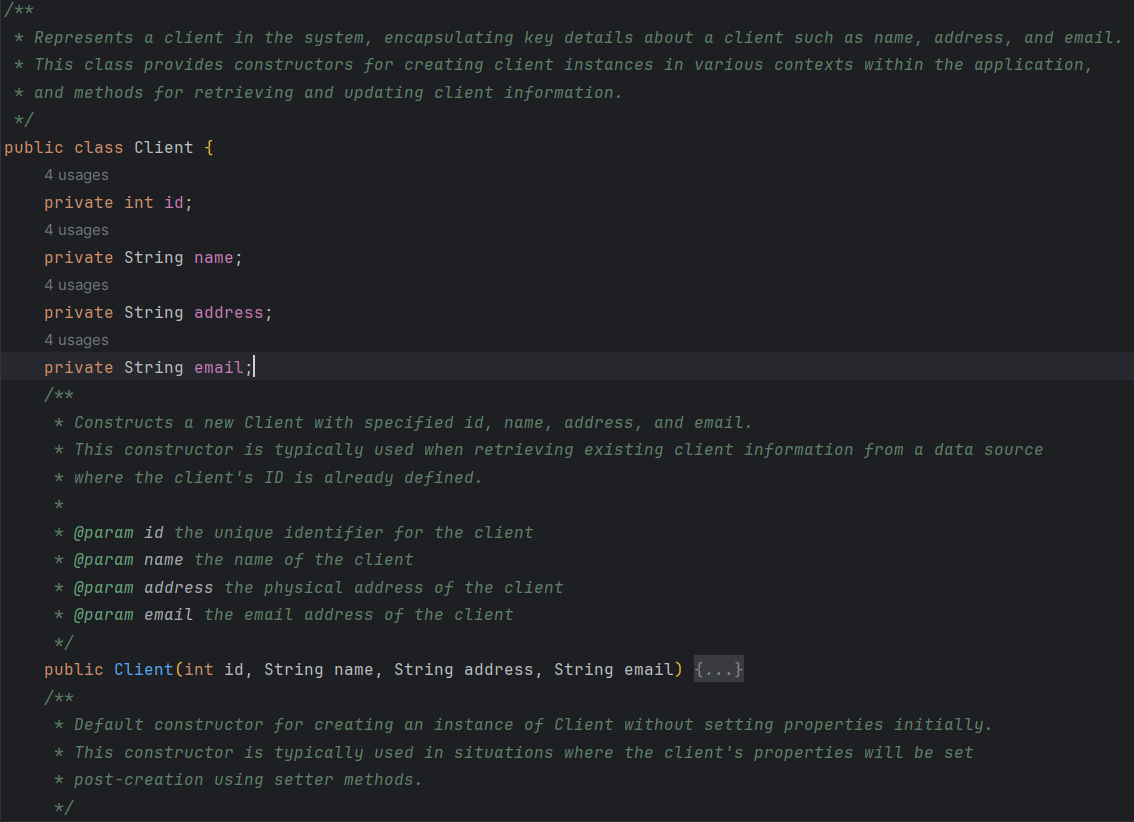
### Bill

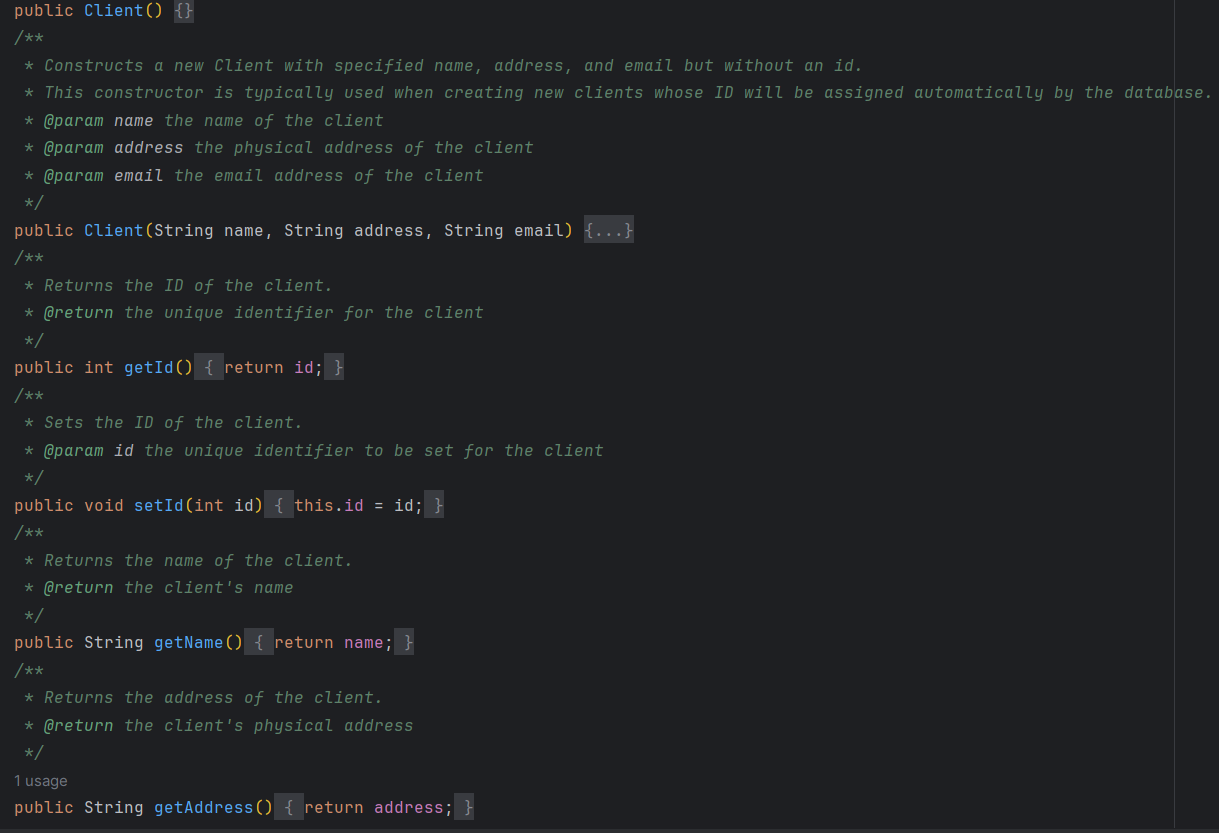
Modelul pentru facturi.

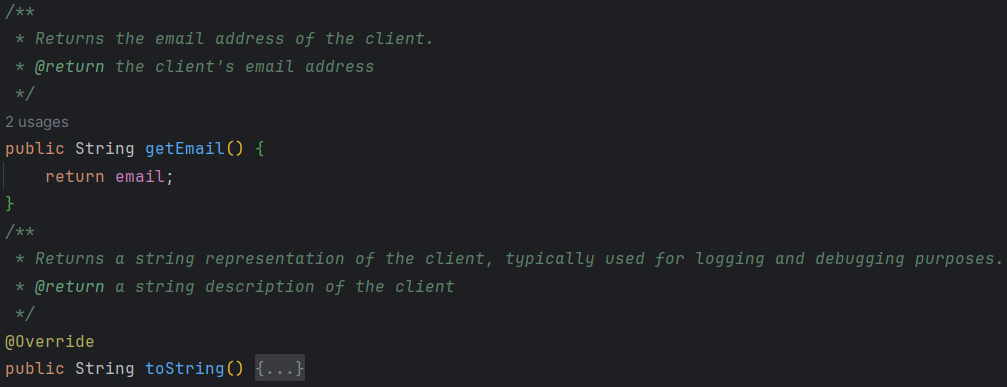


### Client

Modelul pentru clienți.

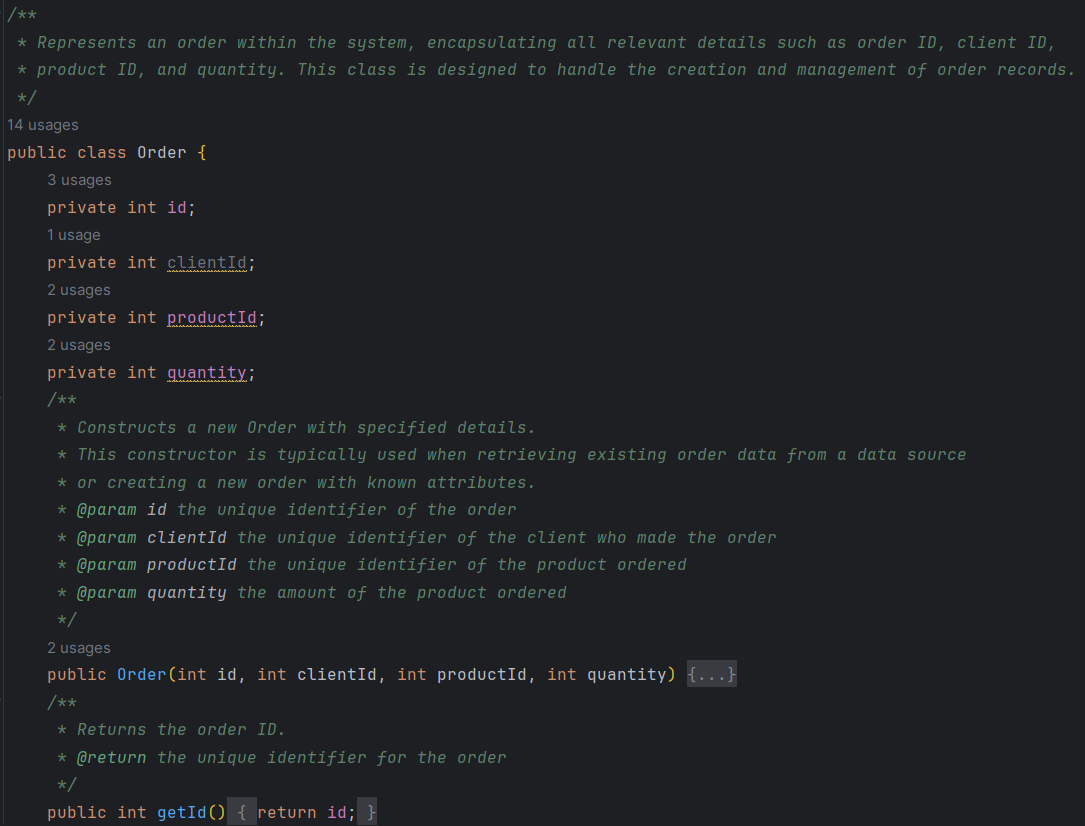


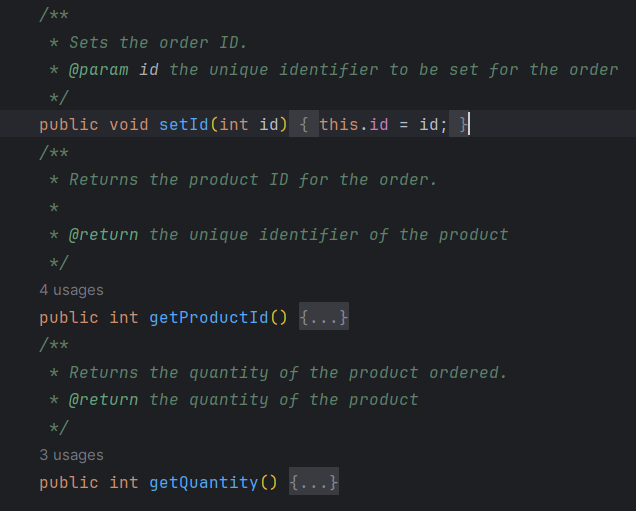




### Order

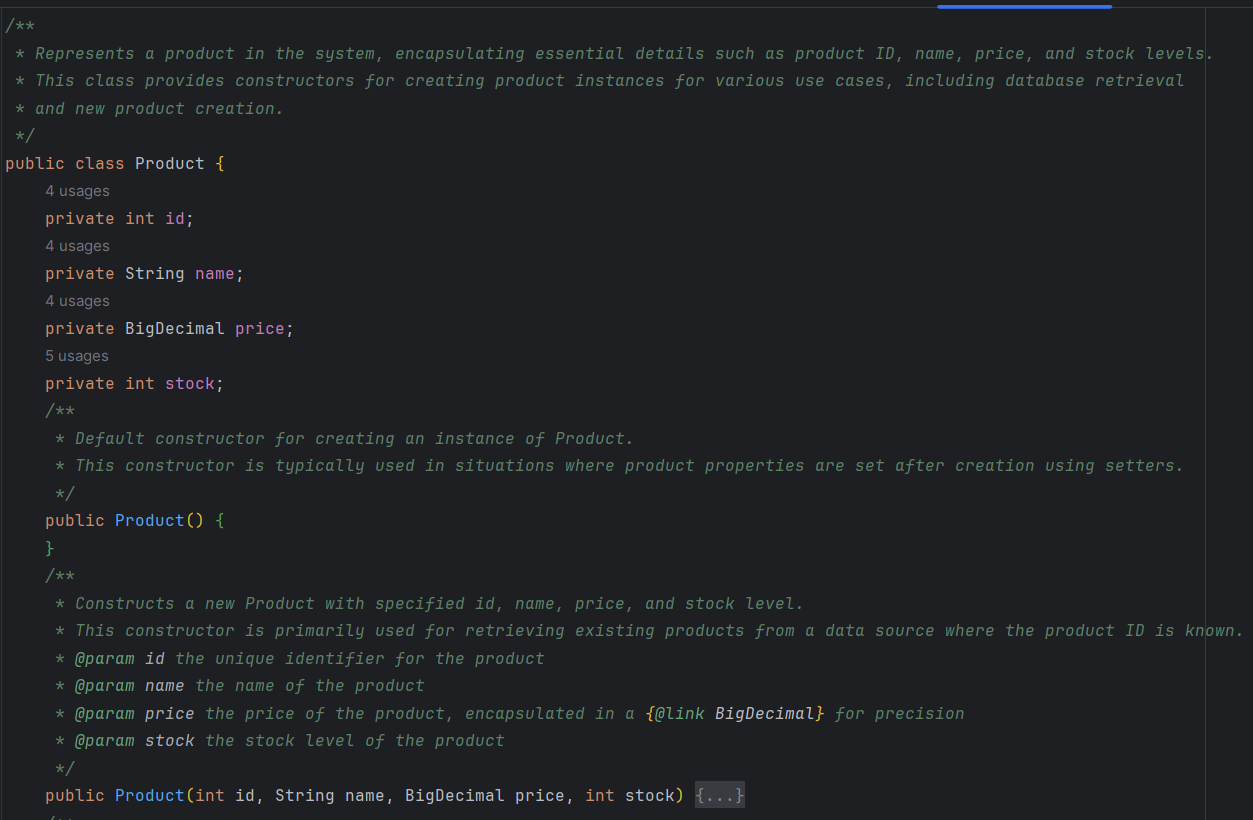
Modelul pentru comenzi.

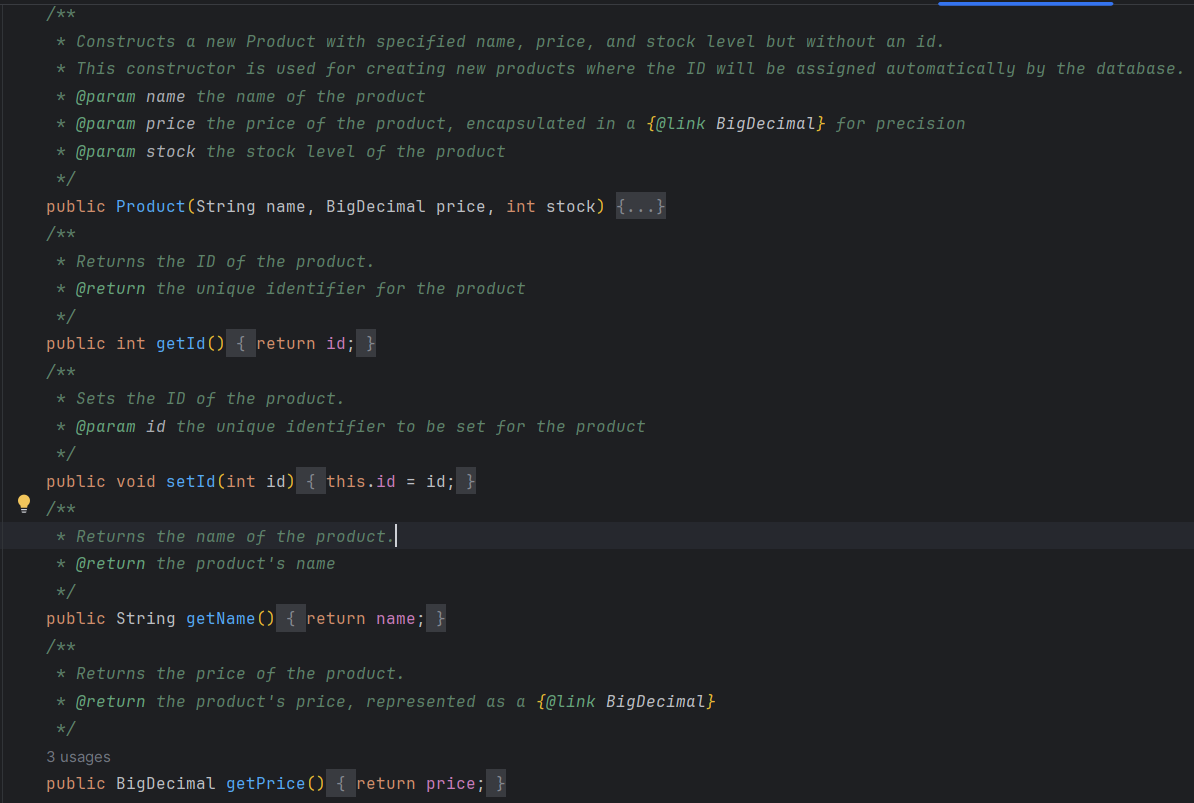


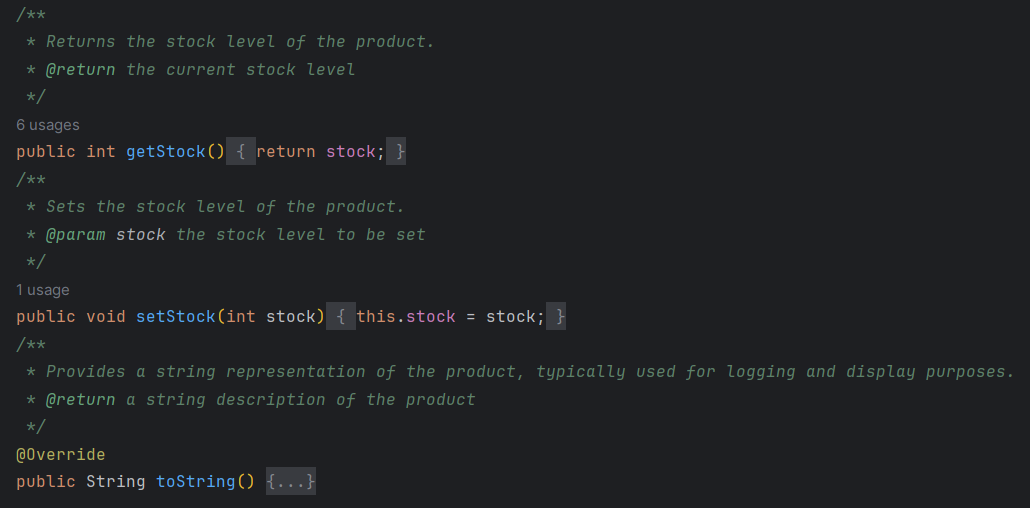


### Product

Modelul pentru produse.





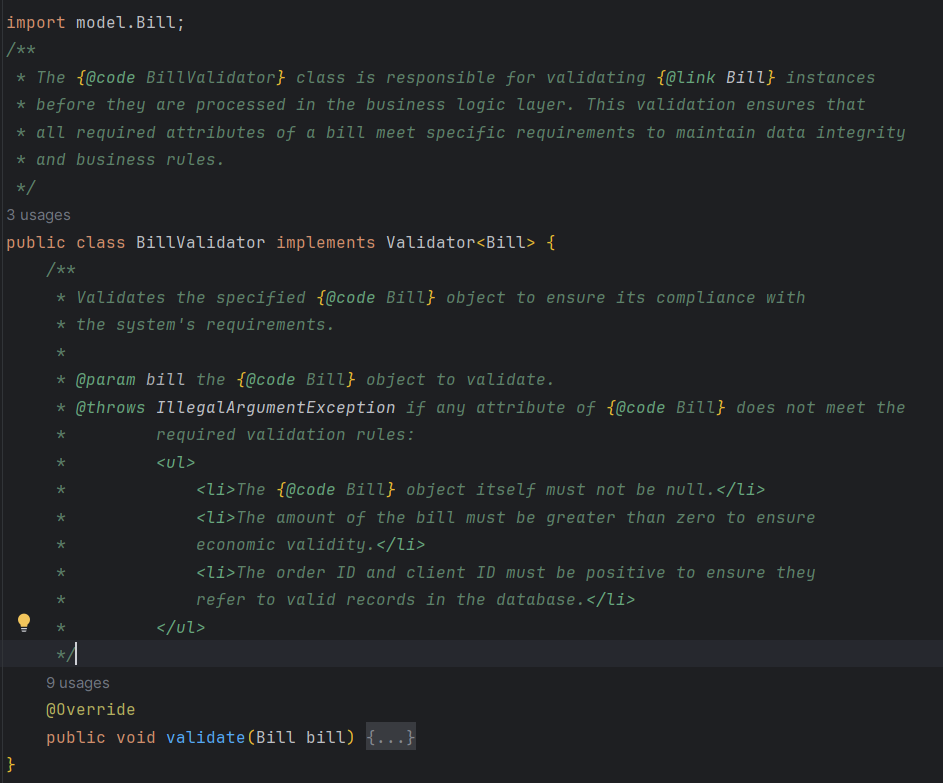


## ***Bll***

### Validators

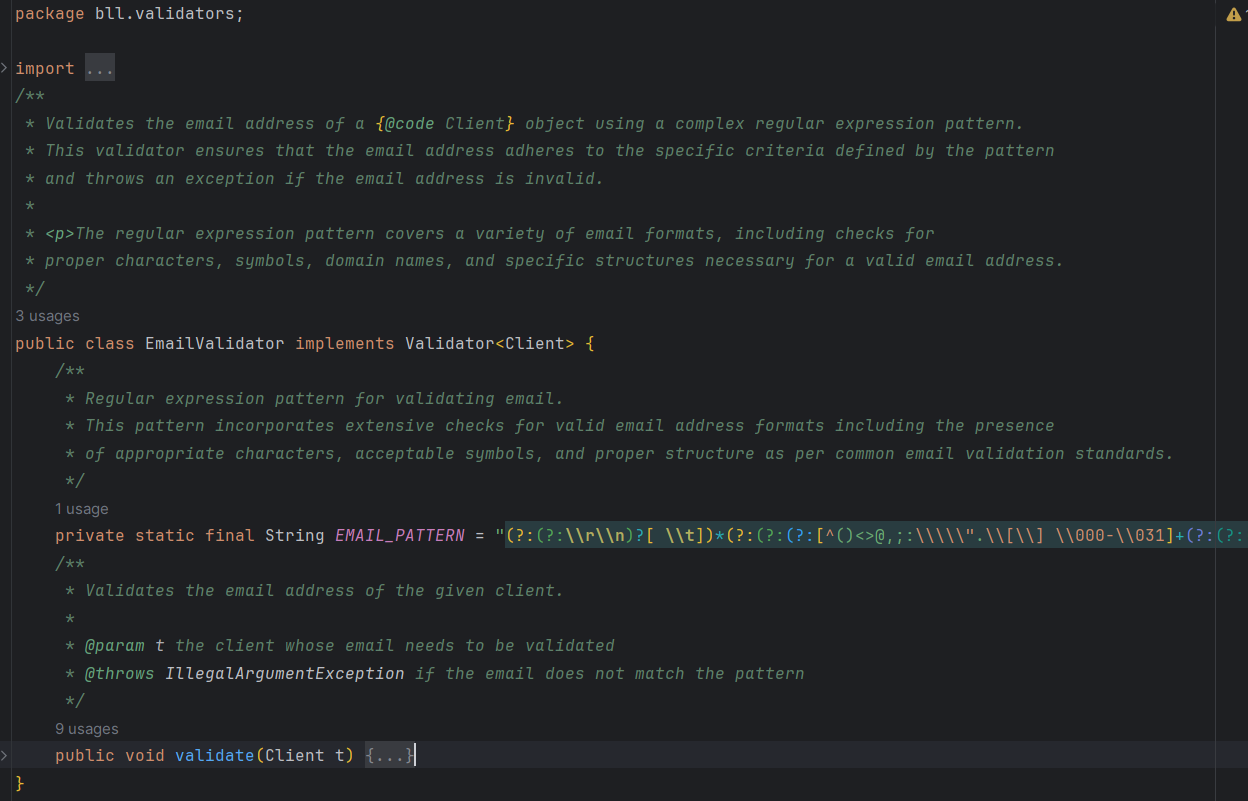
#### BillValidator

Validare specifică pentru facturi.



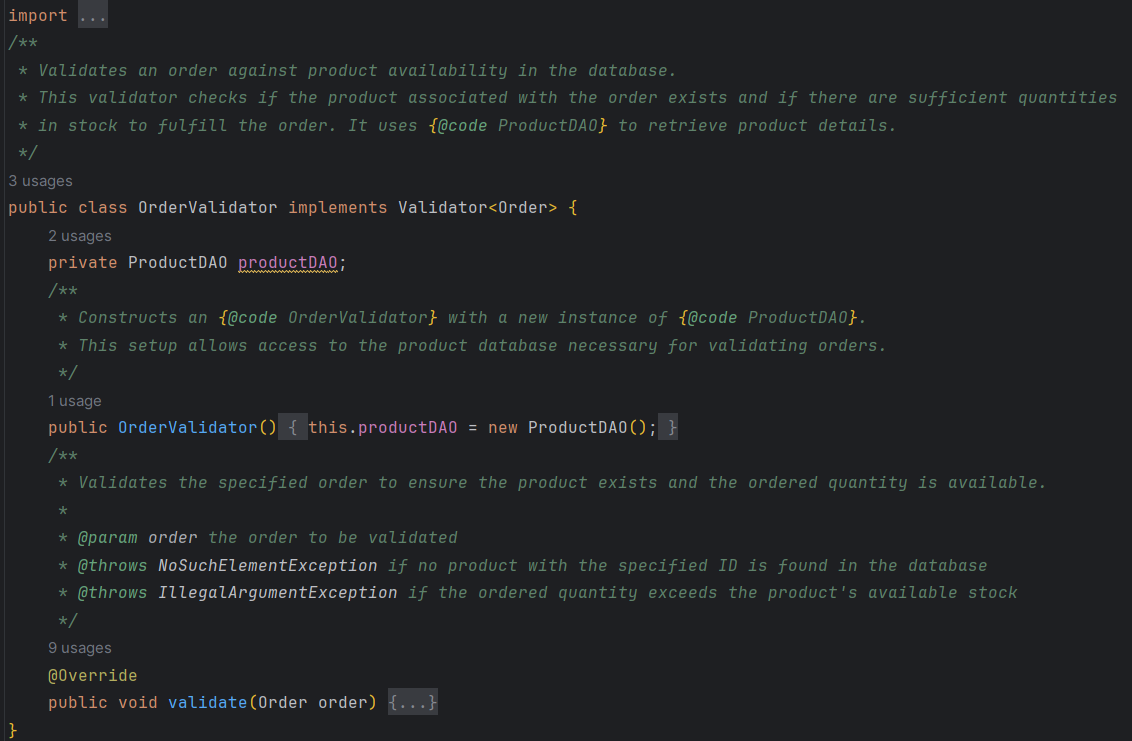
#### EmailValidator

Validare pentru adresele de email.



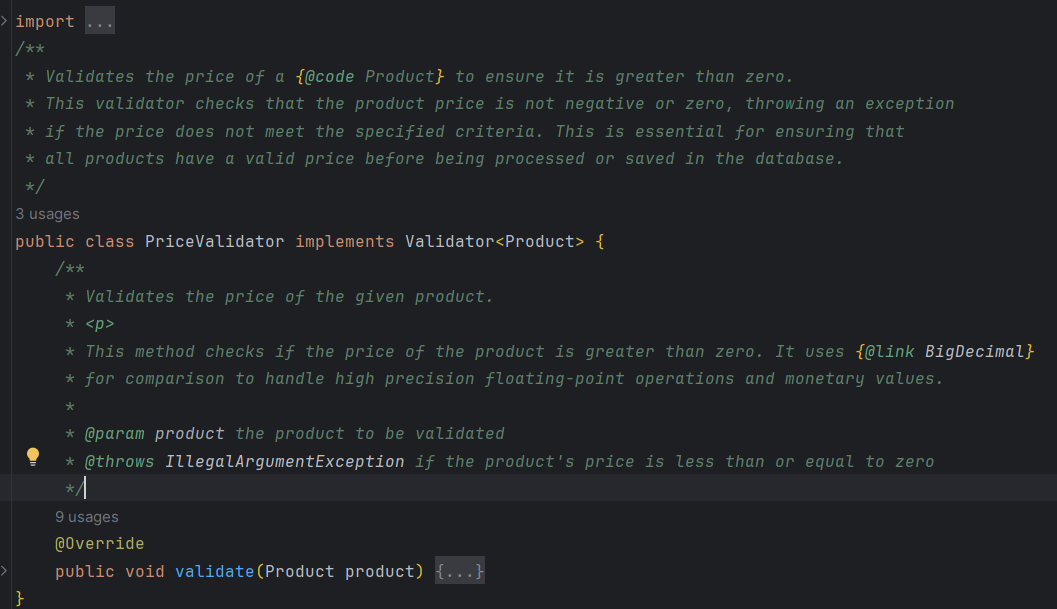
#### OrderValidator

Validare pentru comenzile plasate.



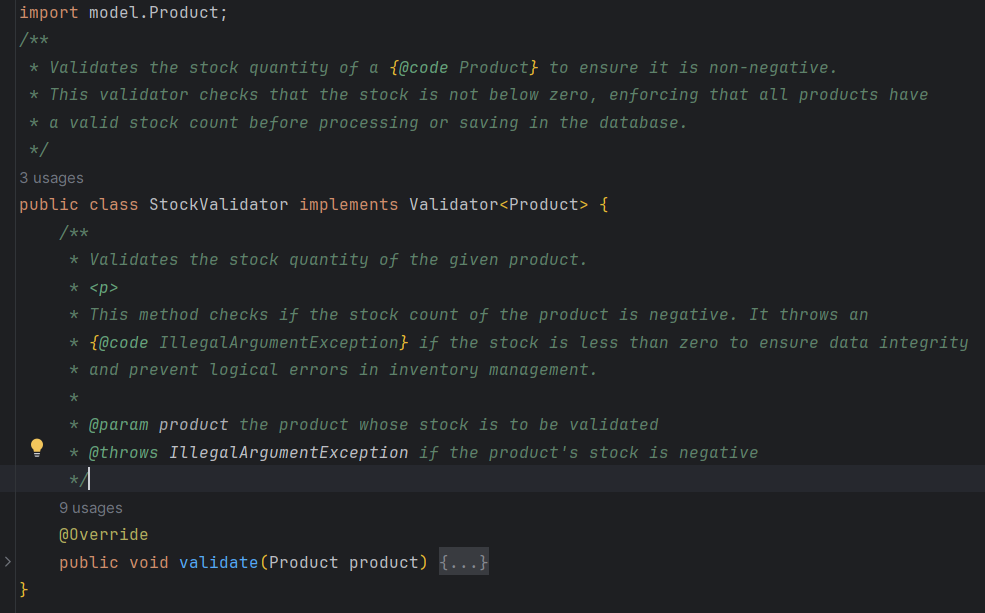
#### PriceValidator

Validare pentru prețurile produselor.



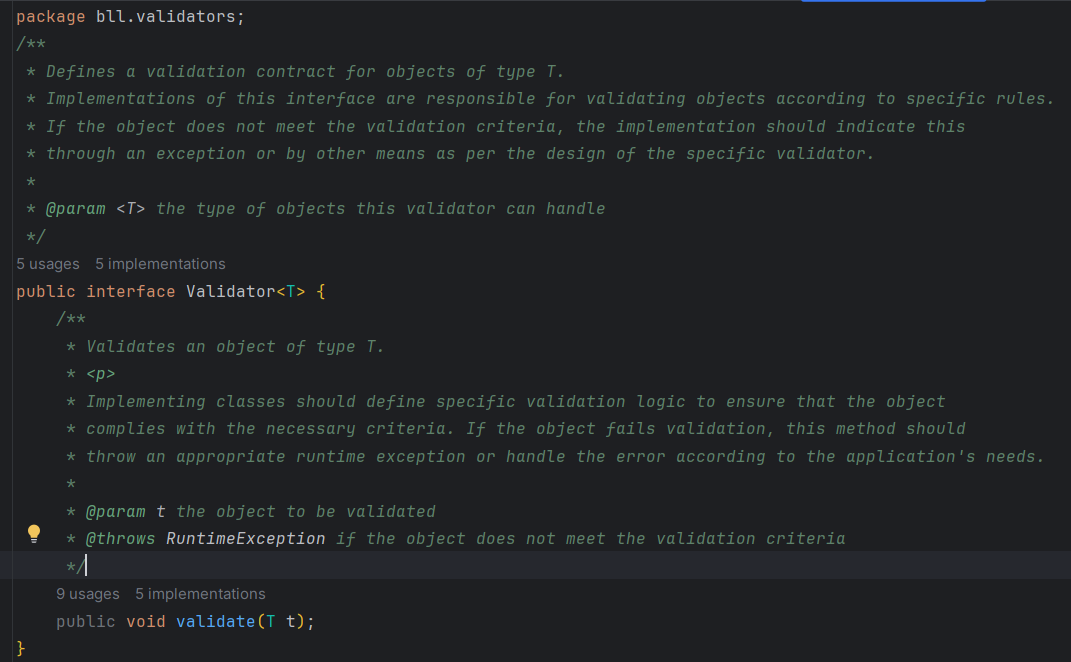
#### StockValidator

Validare pentru stocurile de produse.



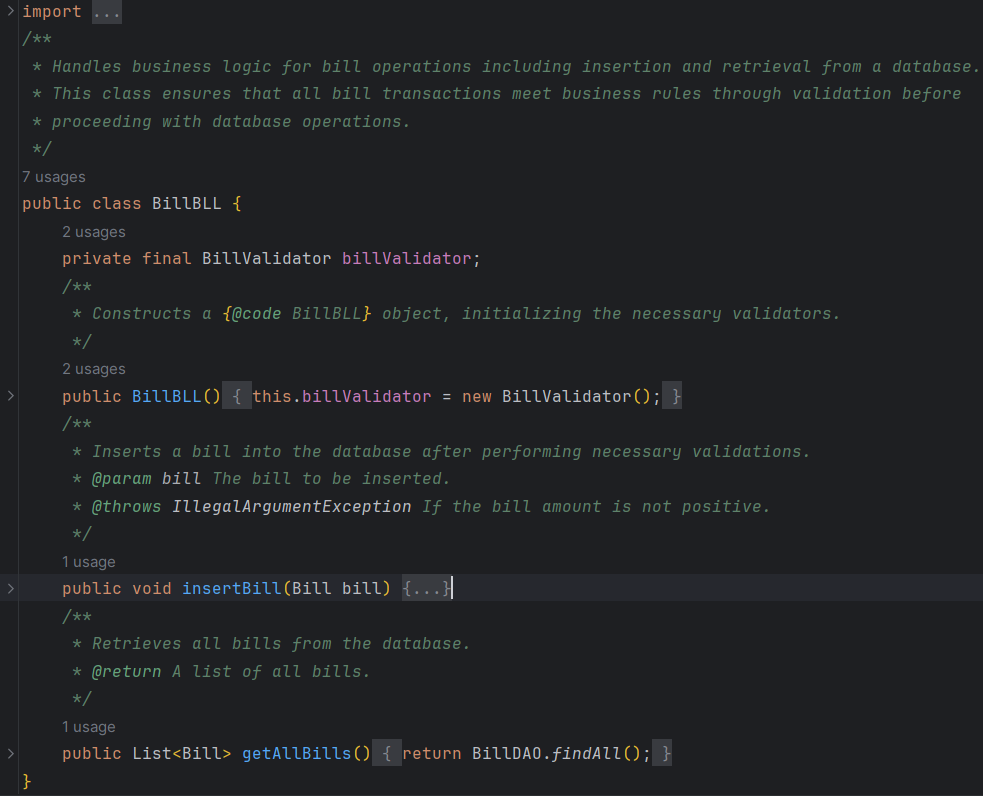
#### Validator

Interfață pentru validatoare.



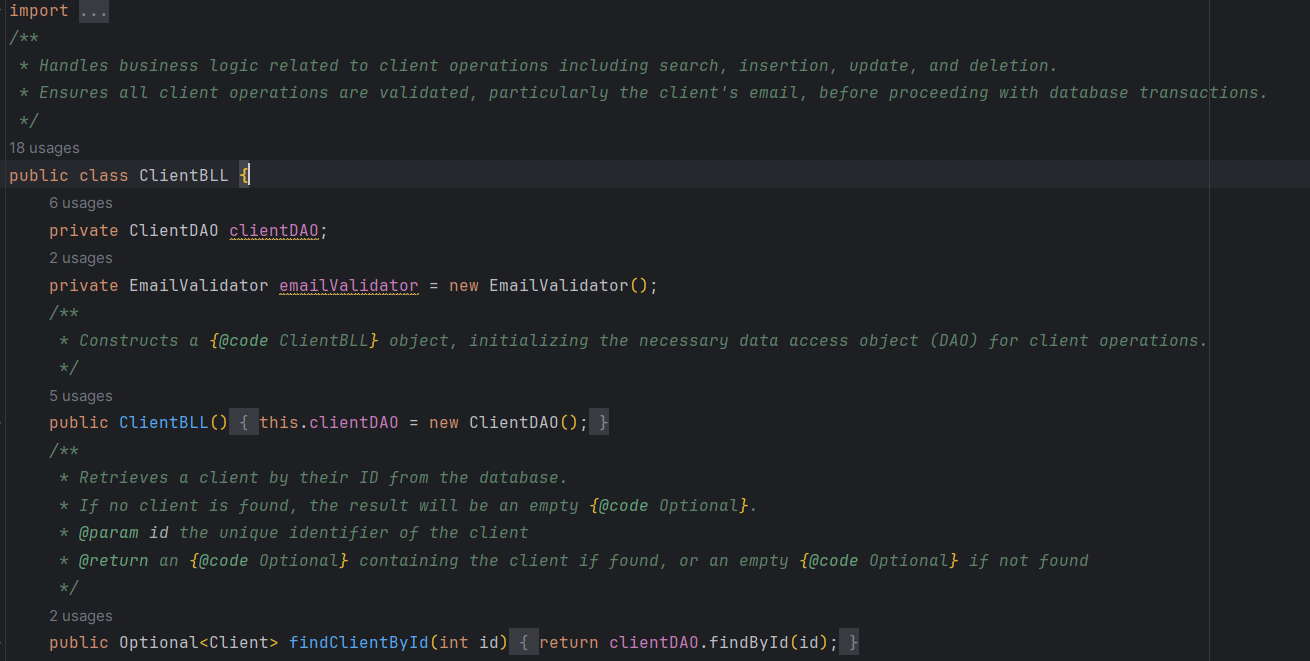
### BillBll

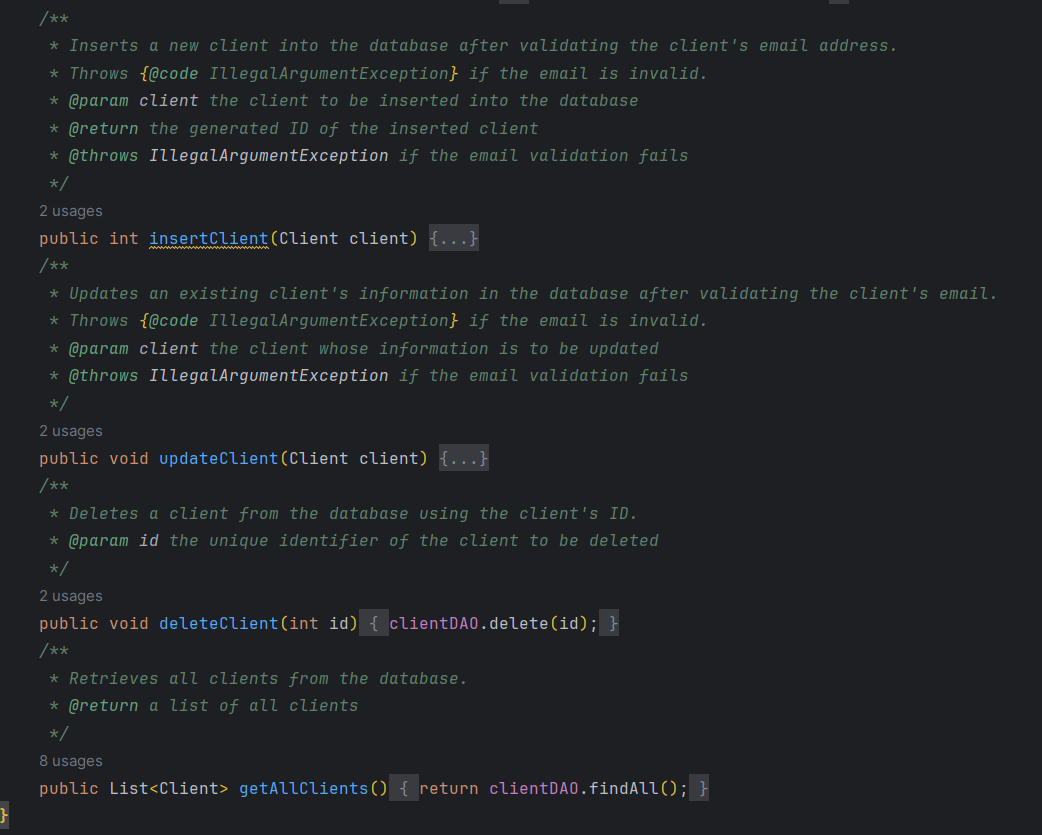
Logică de afaceri pentru gestionarea facturilor.



### ClientBll

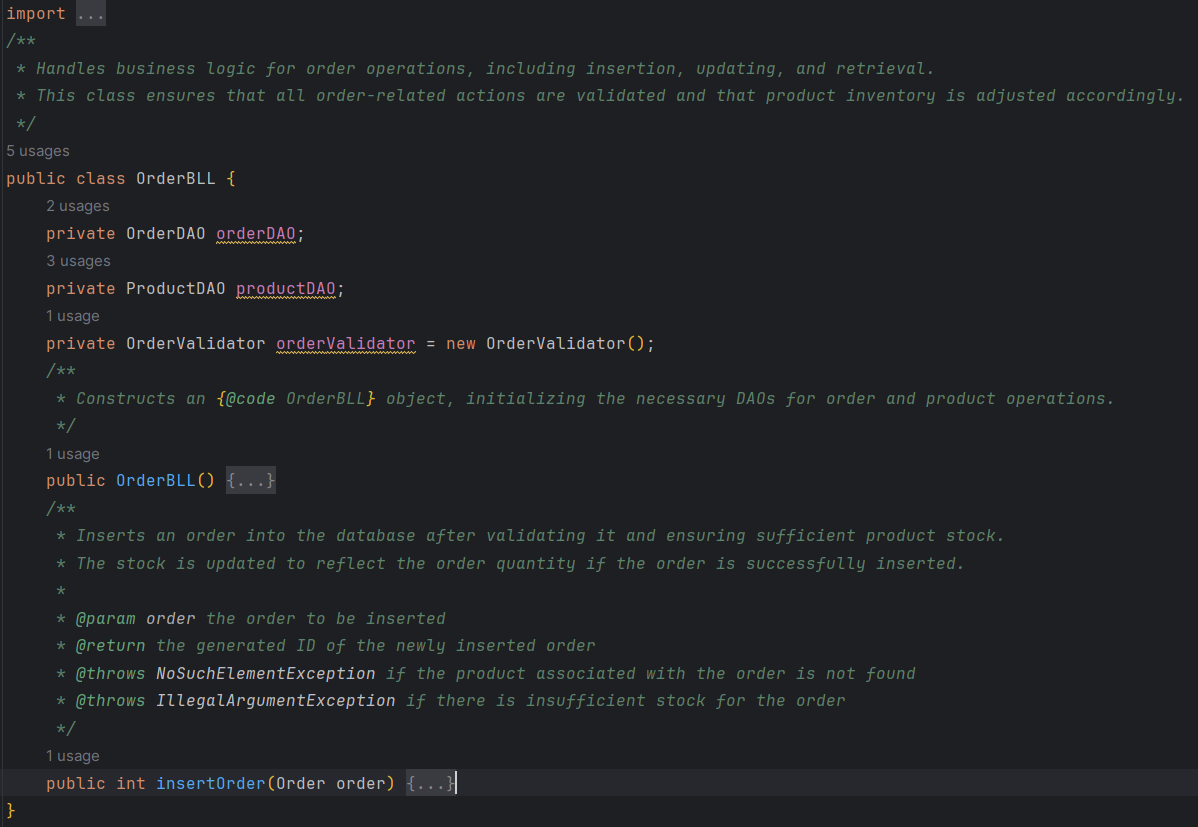
Logică de afaceri pentru gestionarea clienților.





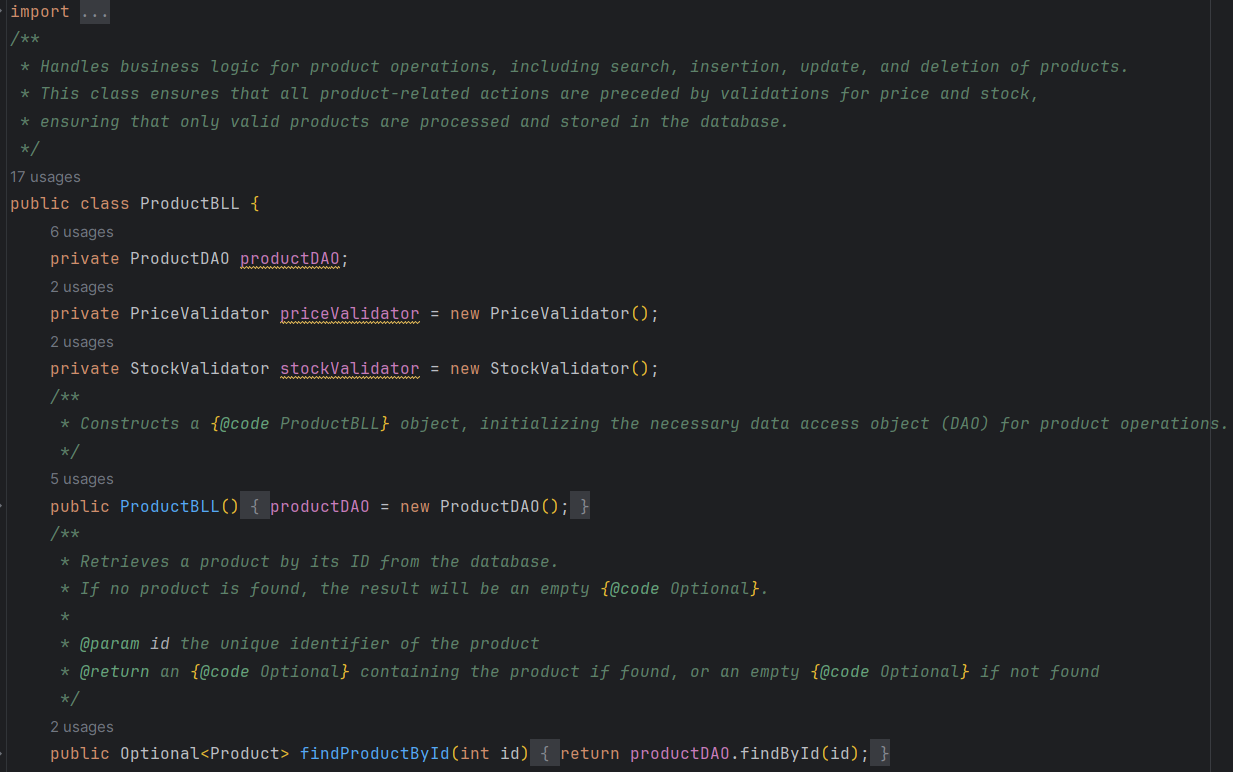
### OrderBll

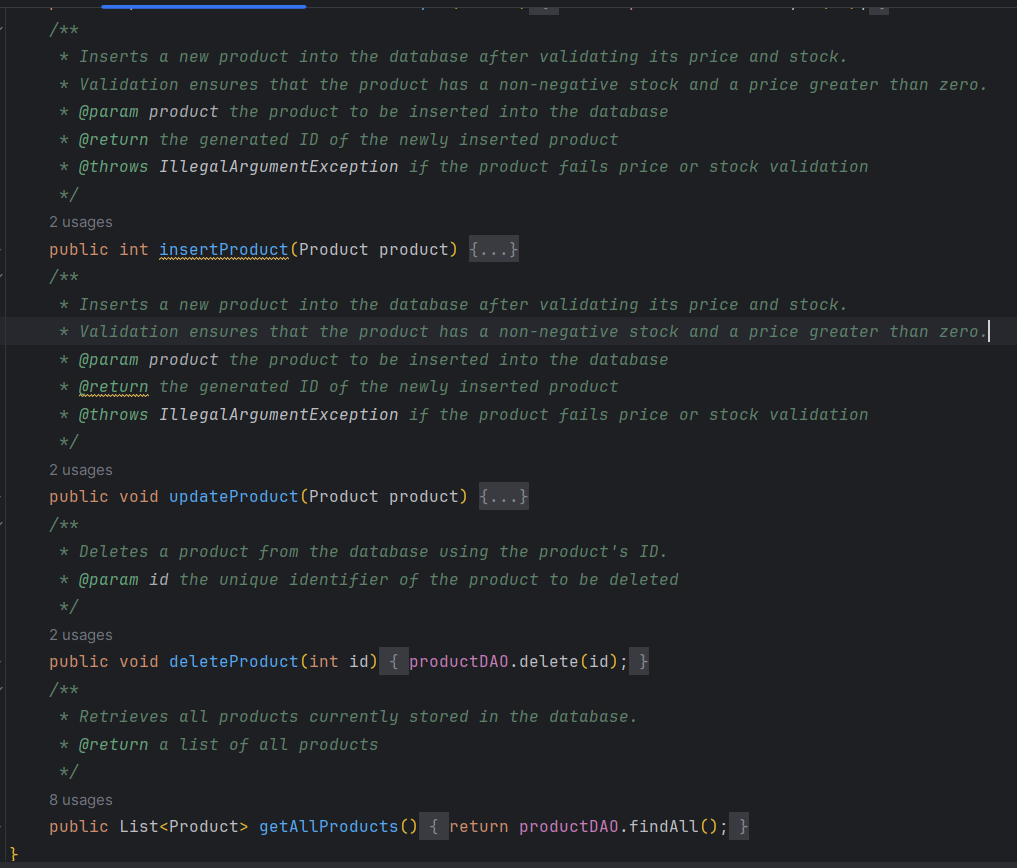
Logică de afaceri pentru procesarea comenzilor.



### ProductBll

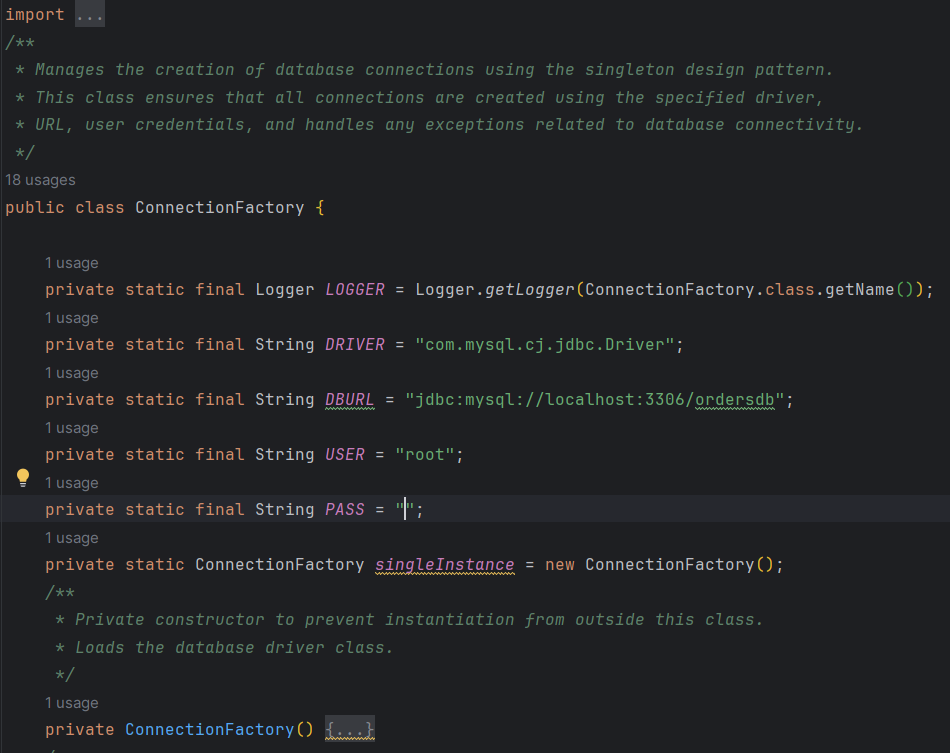
Logică de afaceri pentru gestionarea produselor.

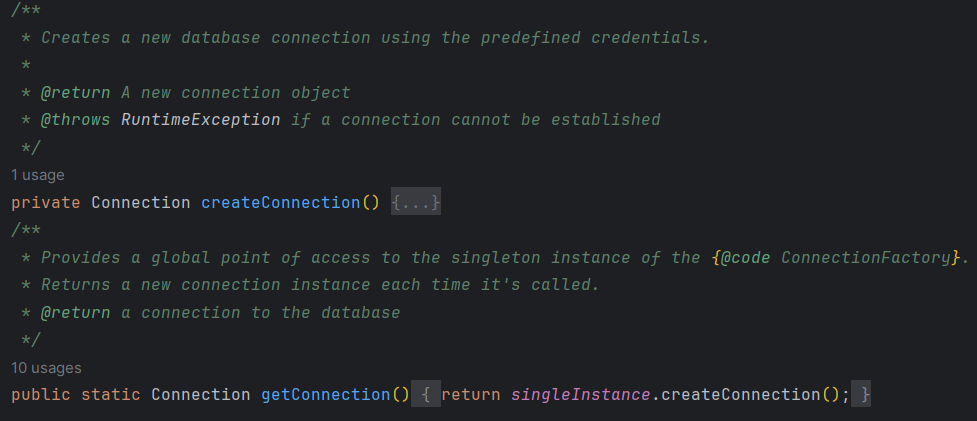




## ***Connection – ConnectionFactory***

Clasa responsabilă pentru stabilirea conexiunii la baza de date.

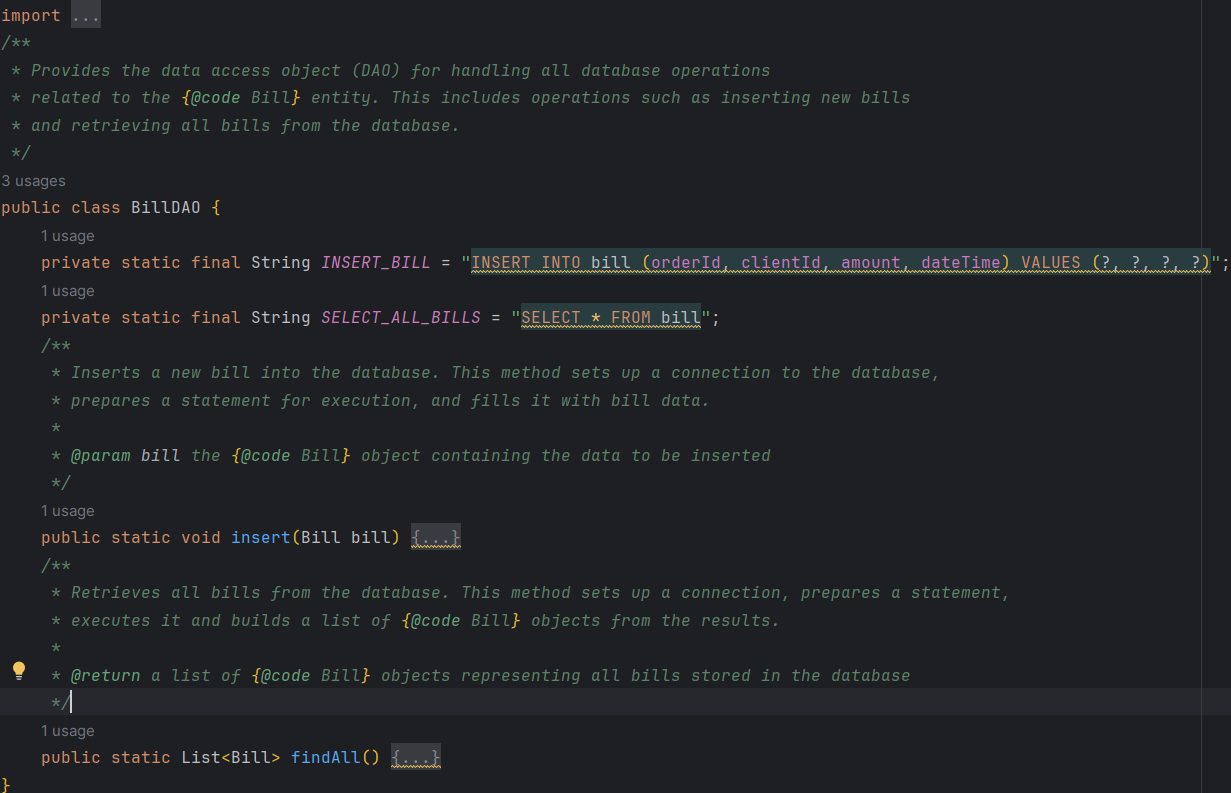




## ***Dao***

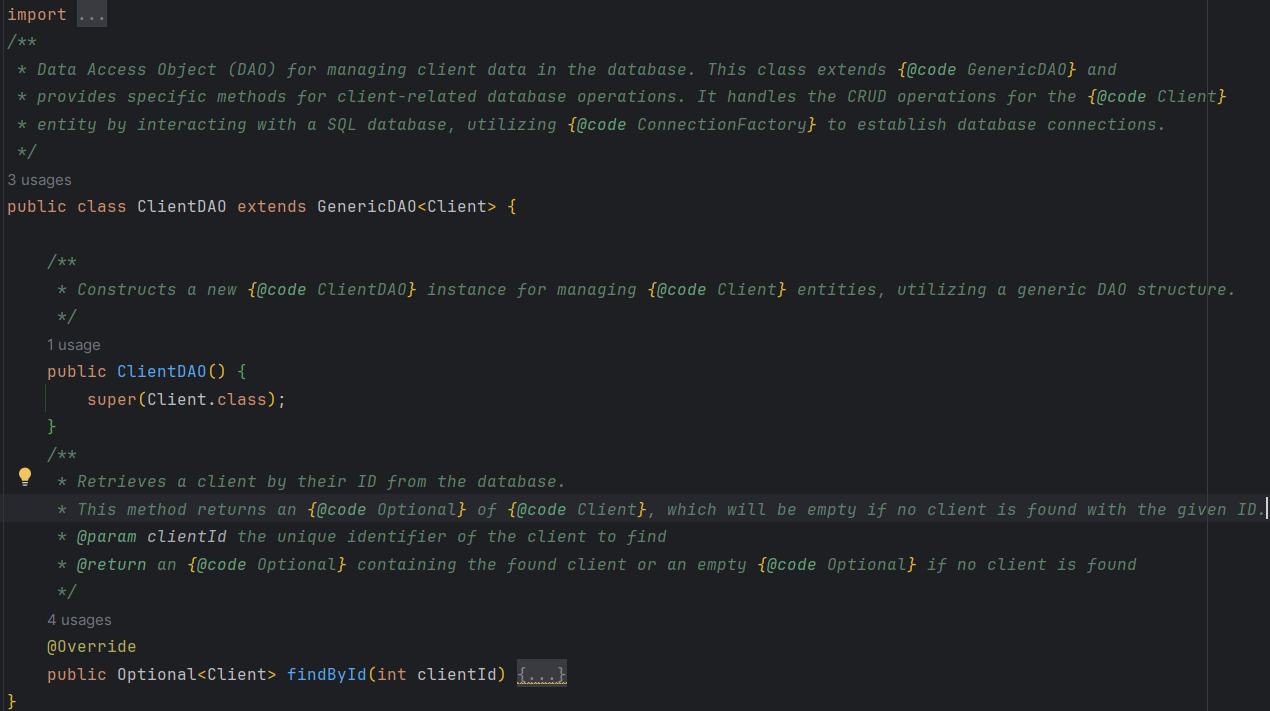
### BillDao

Acces la date pentru facturi.



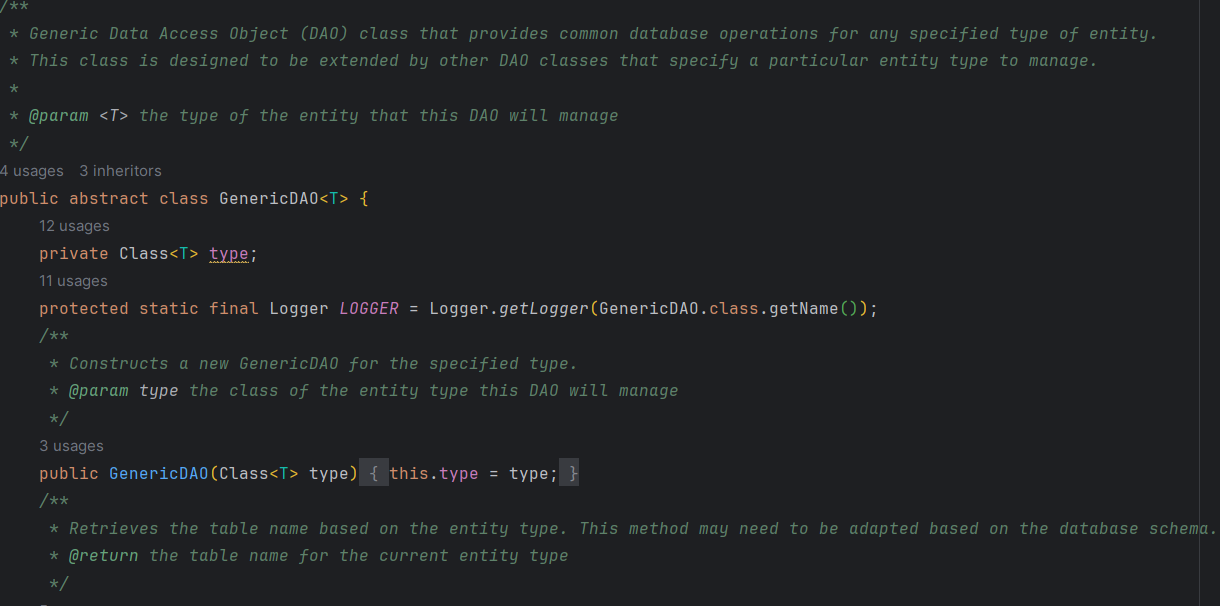
### ClientDao

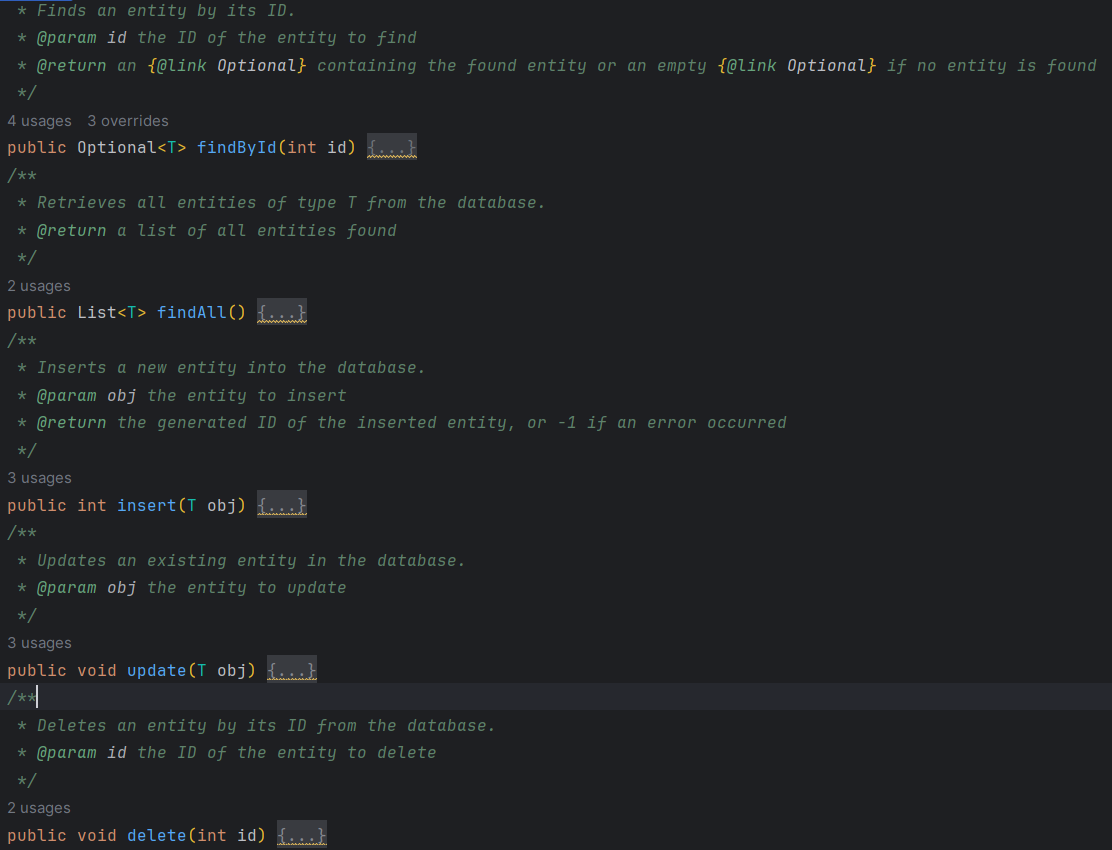
Acces la date pentru facturi.



### GenericDao

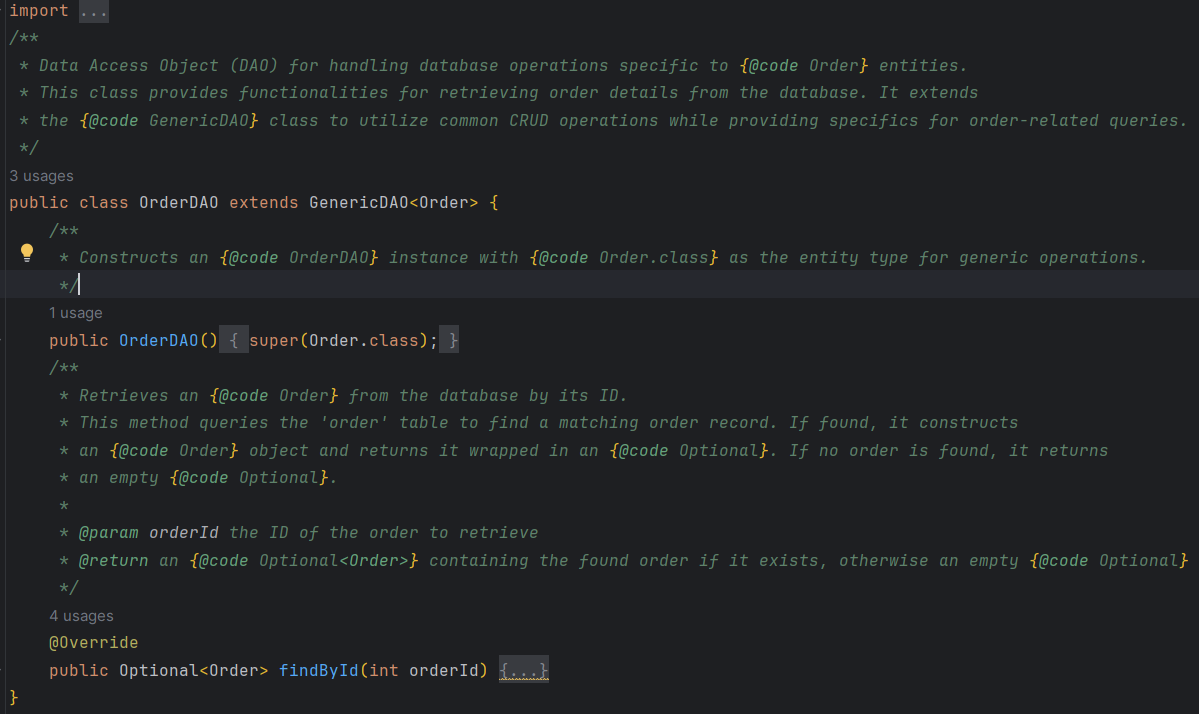
O clasă generică pentru accesul la date, utilizabilă pentru mai multe modele.





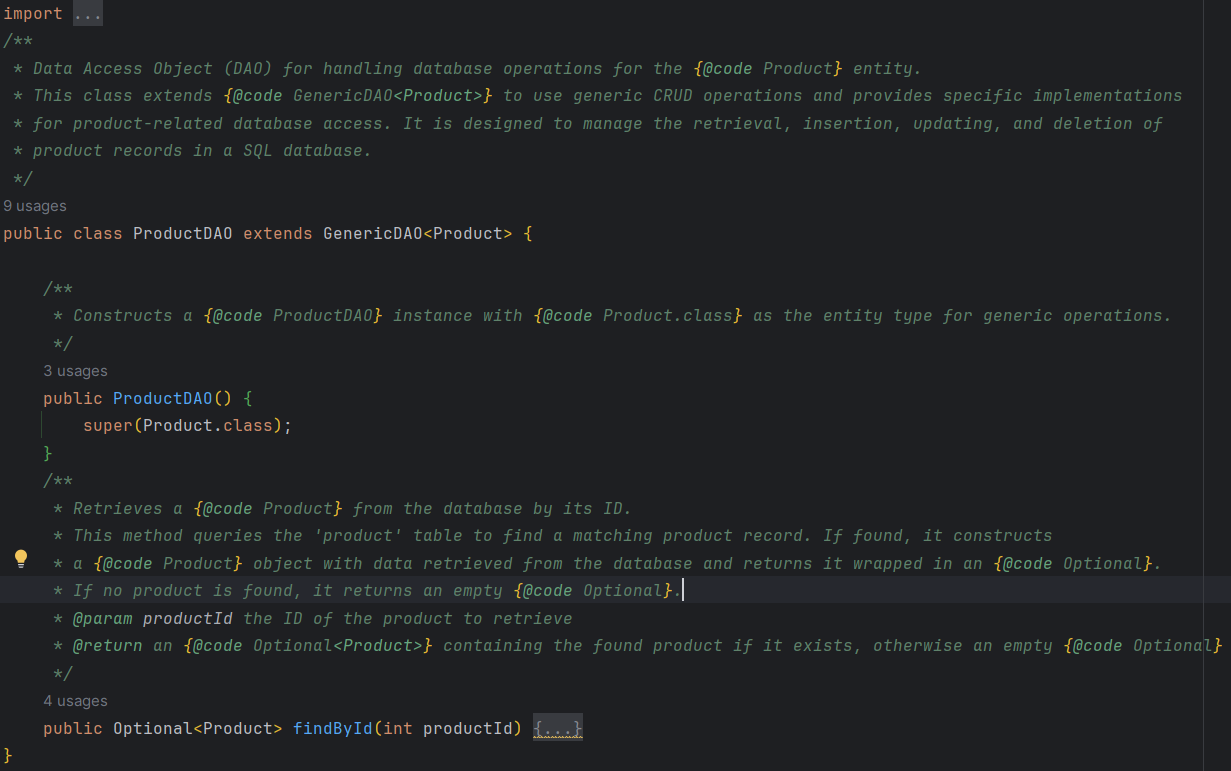
### OrderDao

Acces la date pentru comenzile plasate.



### ProductDao

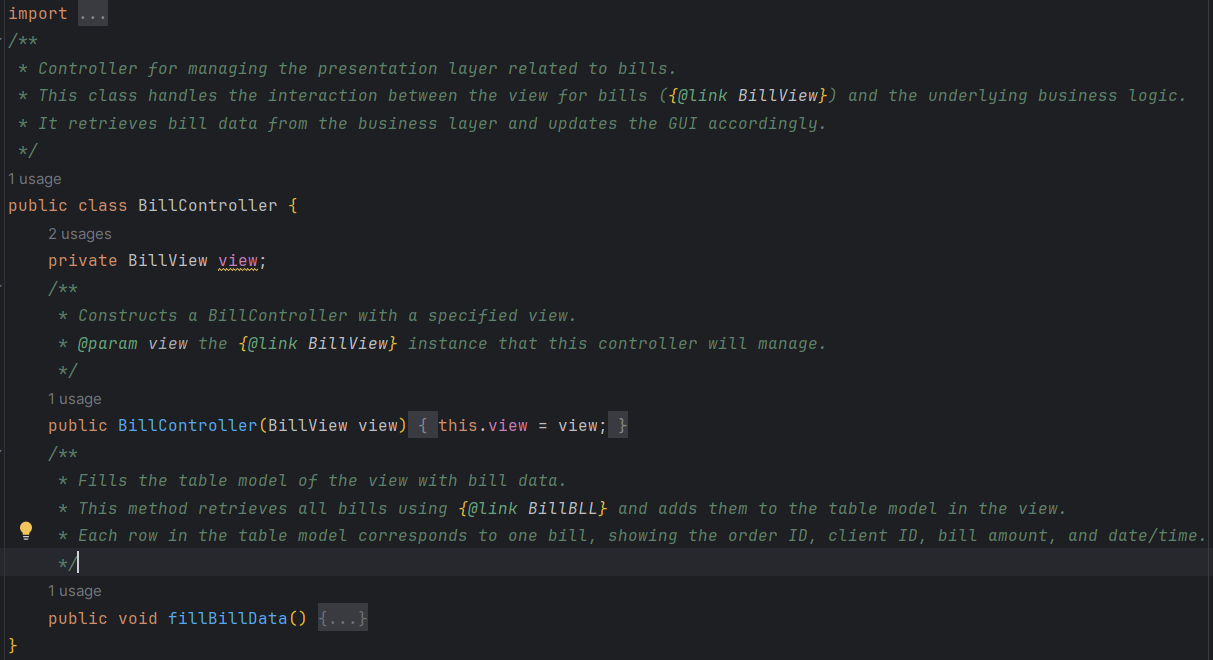
Acces la date pentru produse.



## ***Presentation***

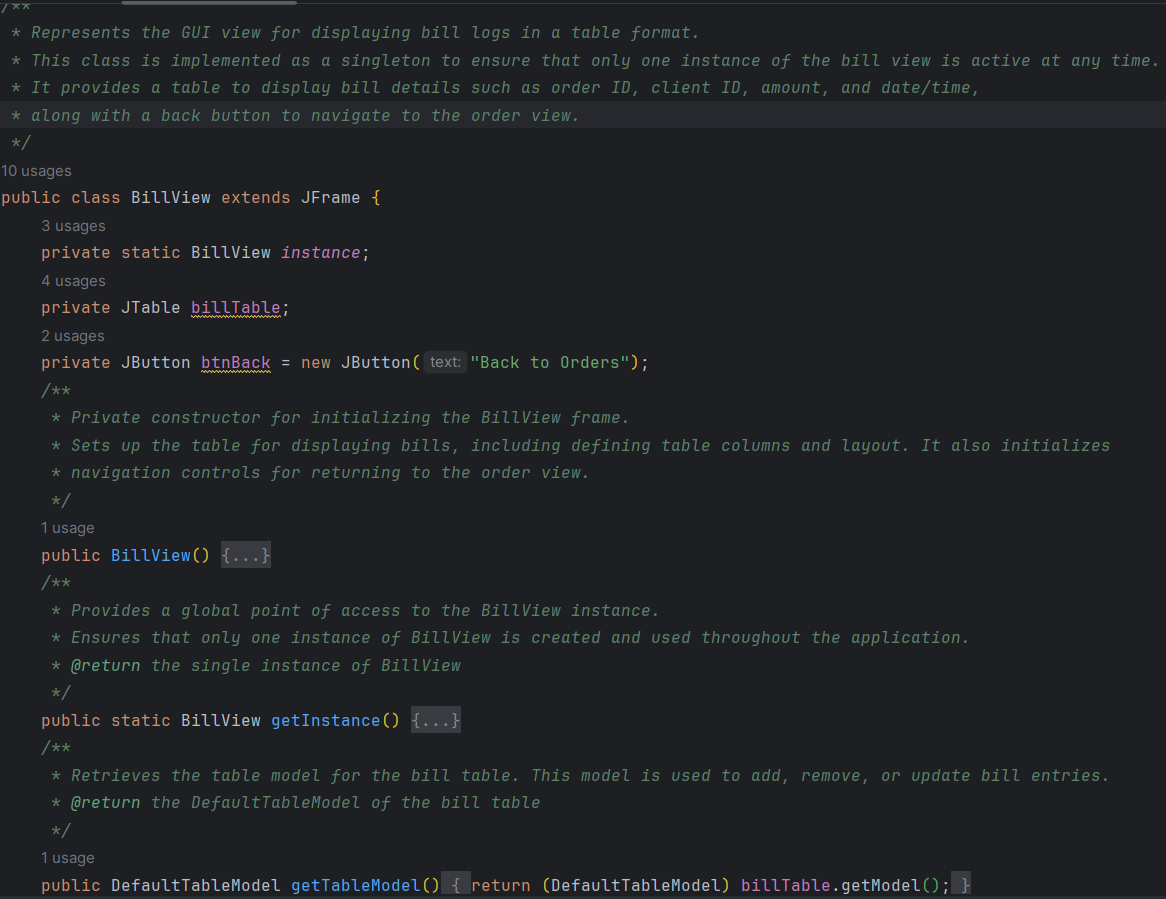
### BillController

Controler pentru gestionarea UI pentru facturi.



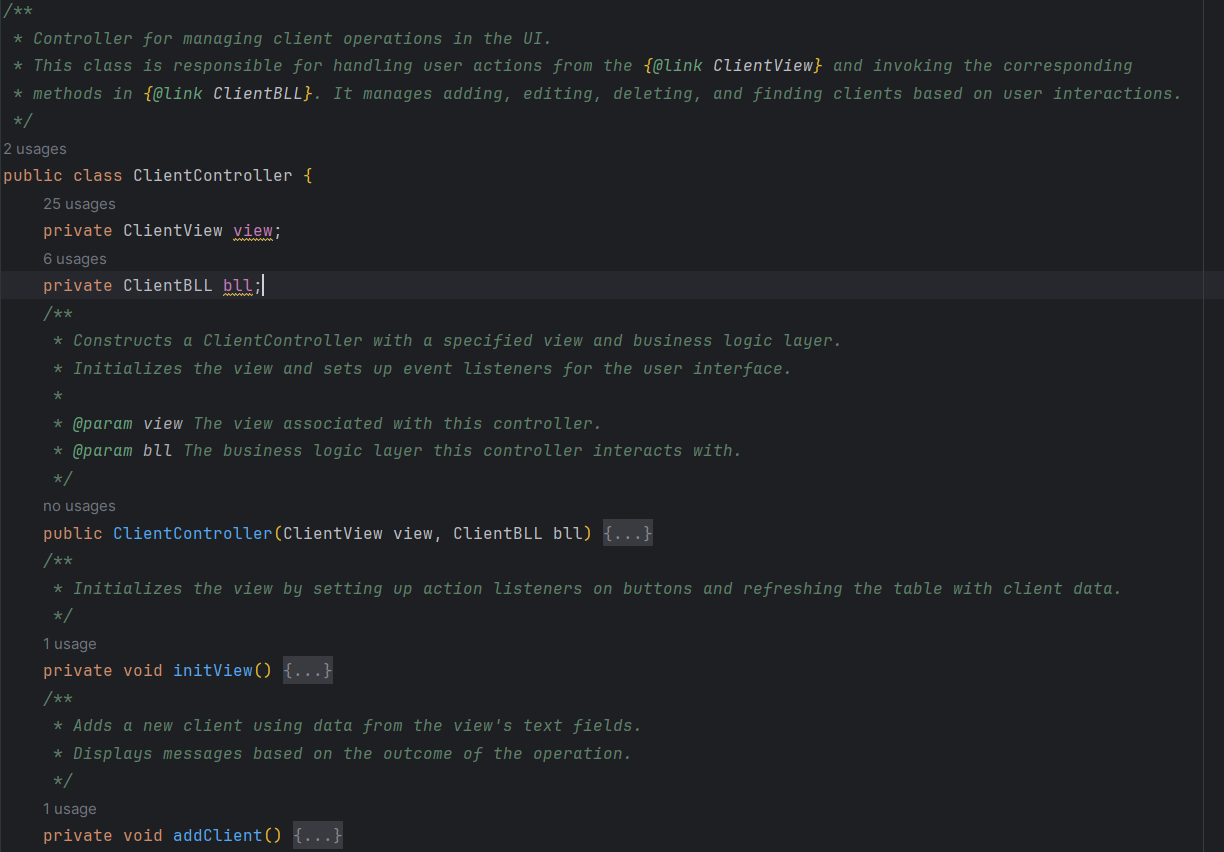
### BillView

Interfață utilizator pentru facturi.



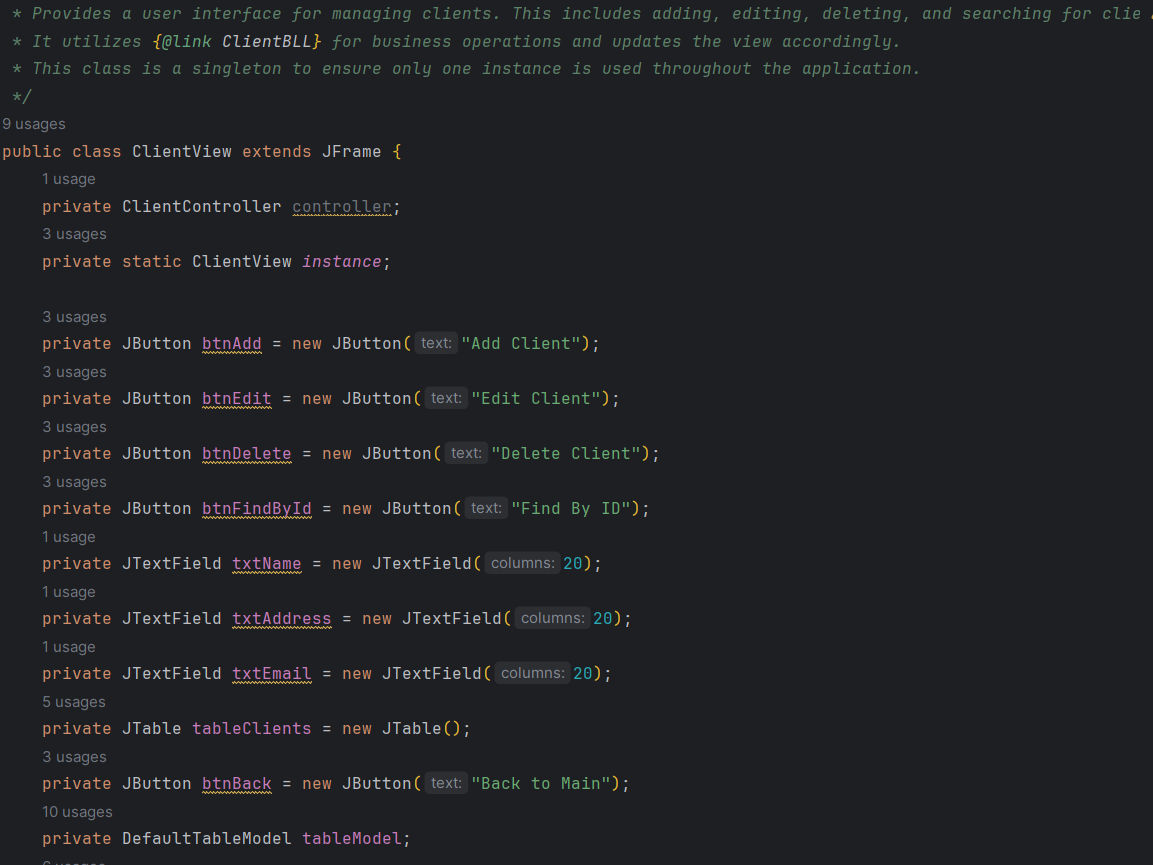
### ClientController

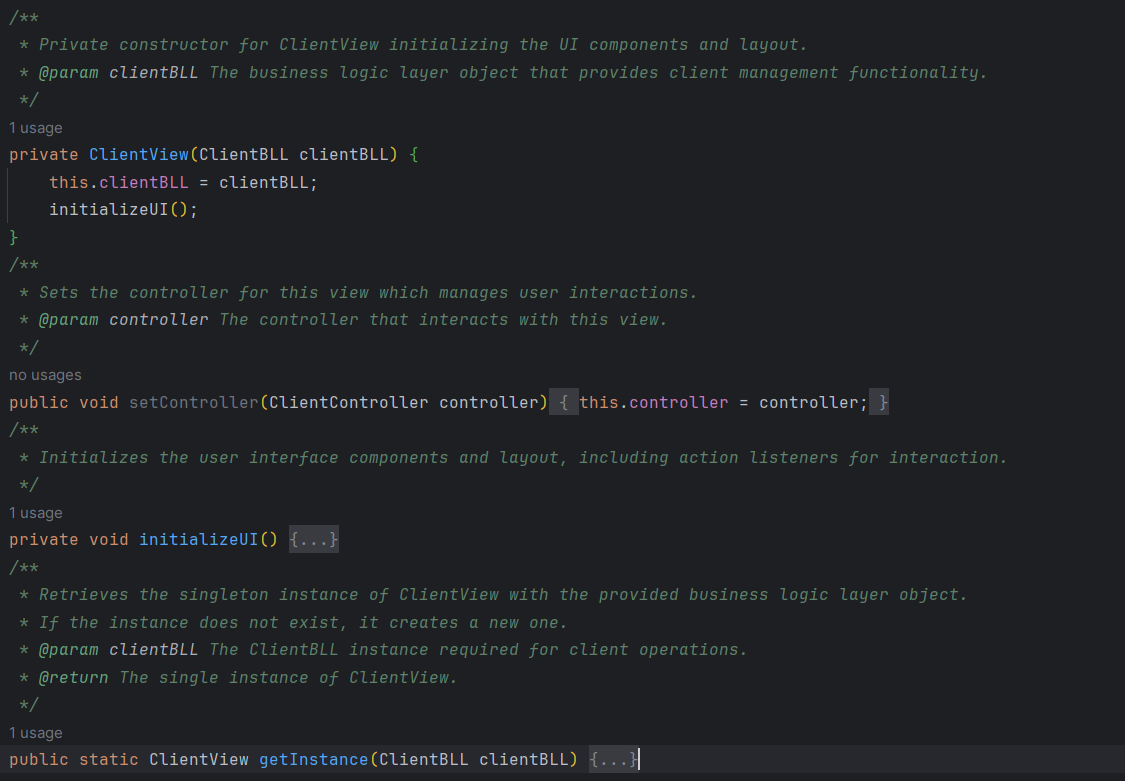
Controler pentru gestionarea UI pentru clienți.



### ClientView

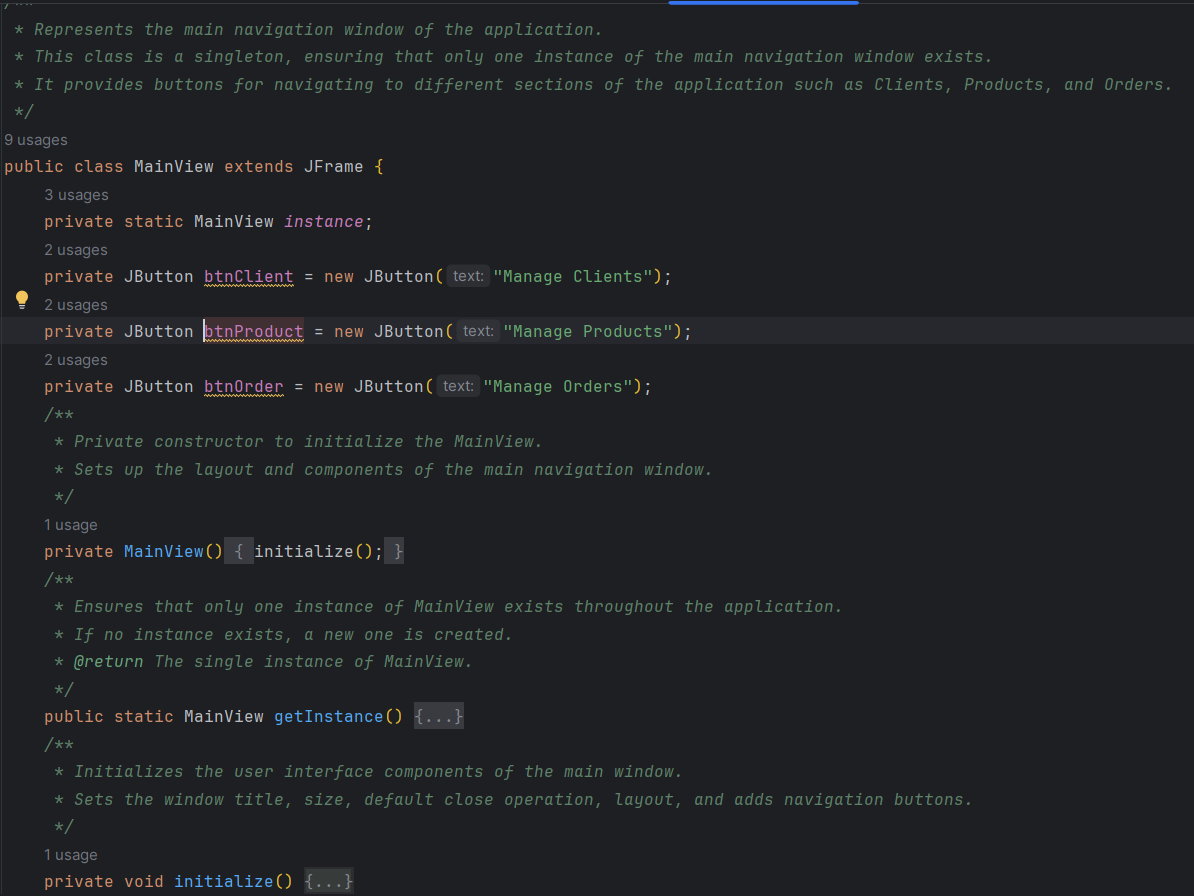
Interfață utilizator pentru clienți.





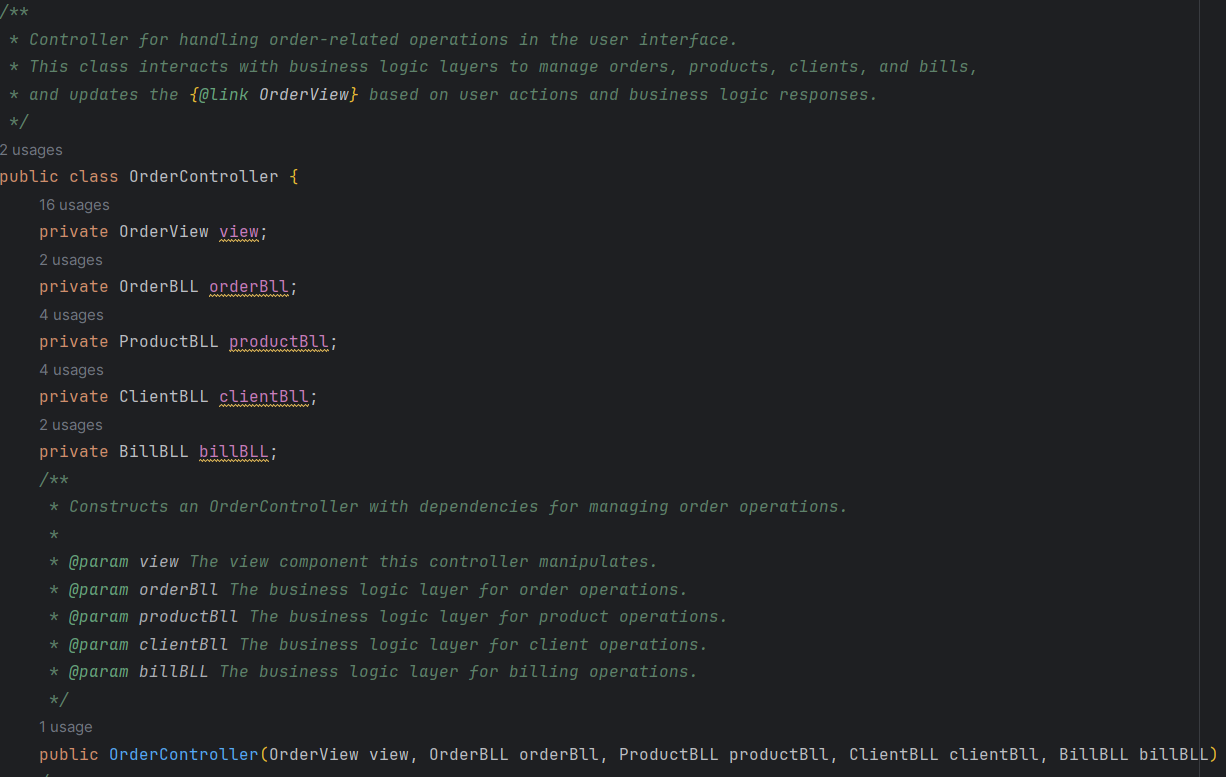
### MainView

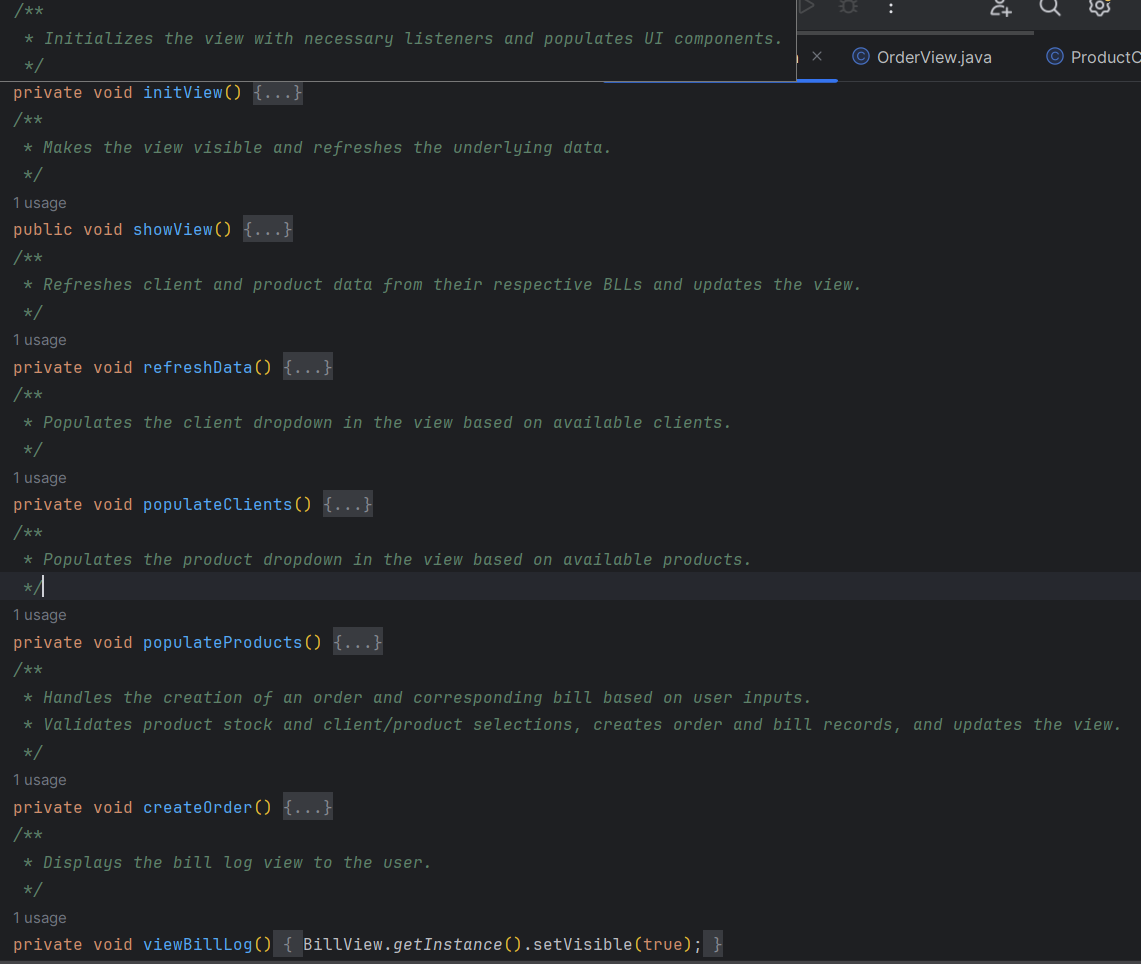
Interfața principală a aplicației.



### OrderController

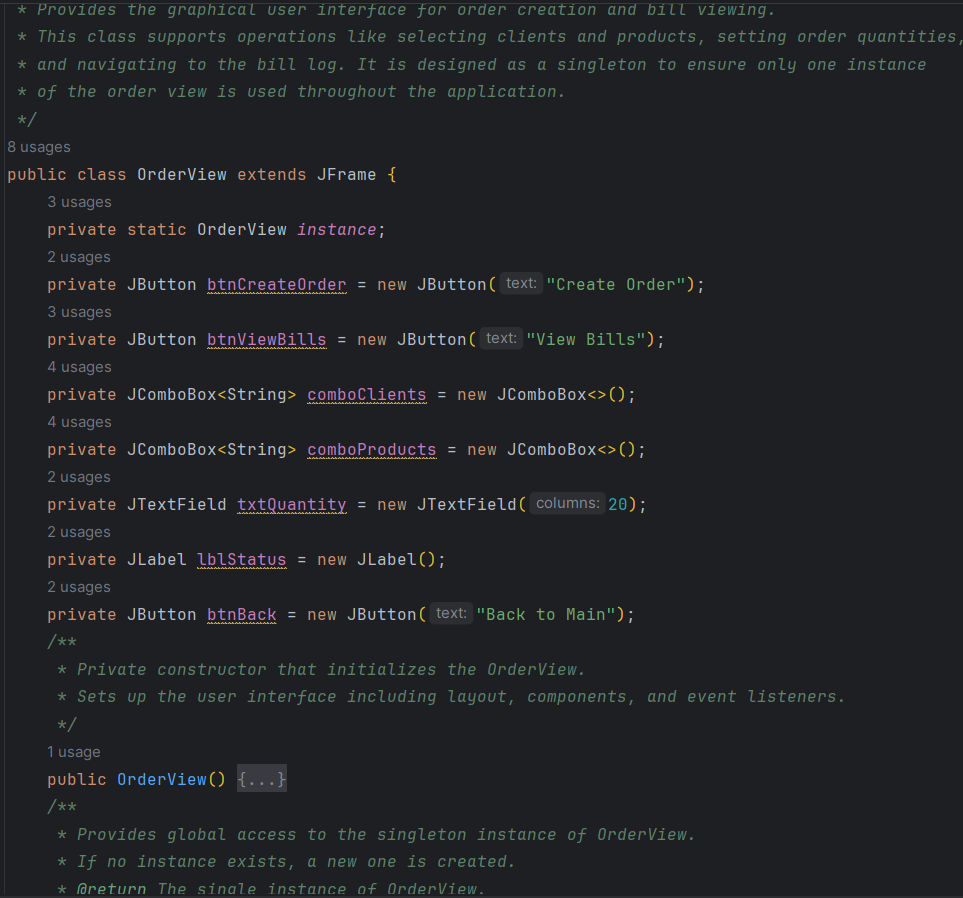
Controler pentru gestionarea UI pentru comenzi.

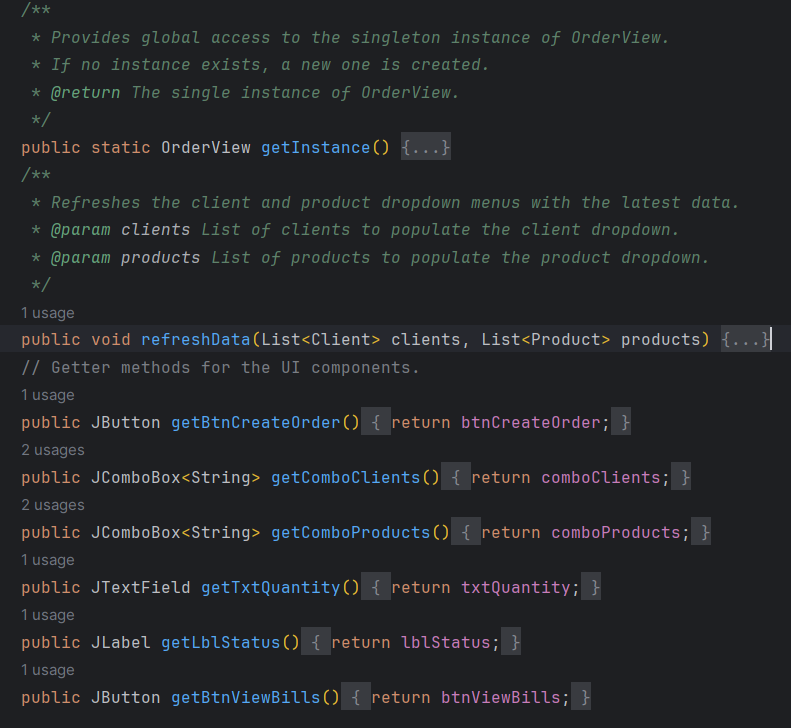




### OrderView

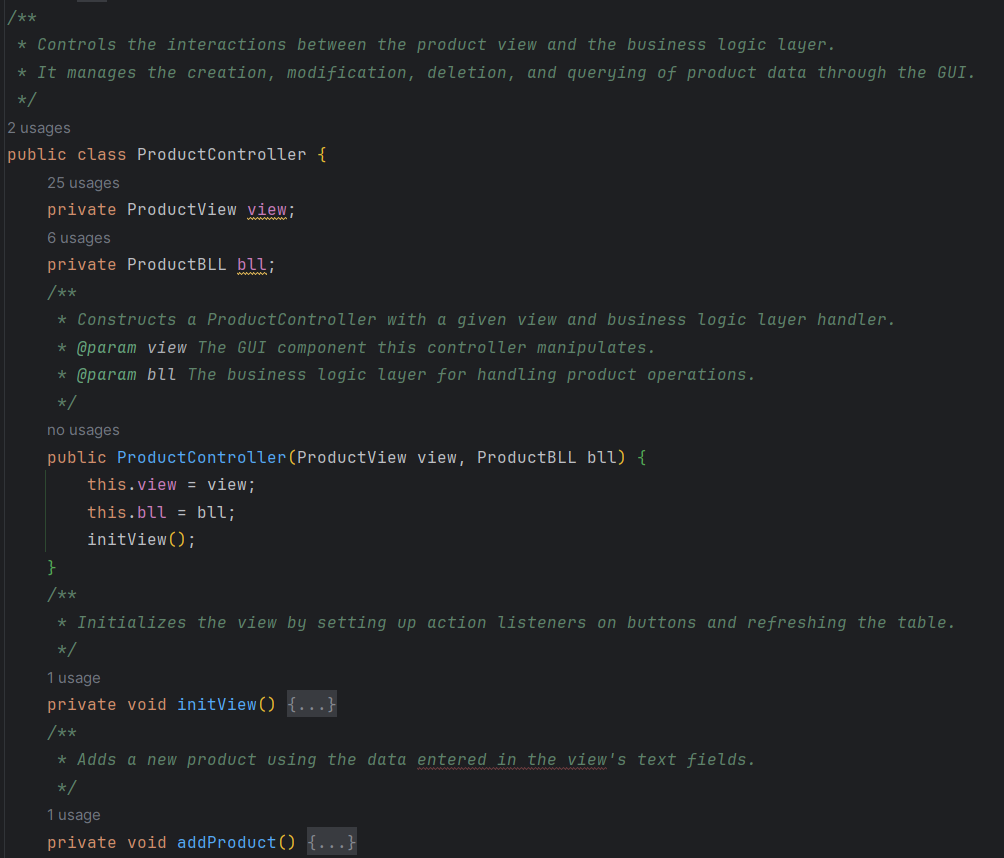
Interfață utilizator pentru comenzi.

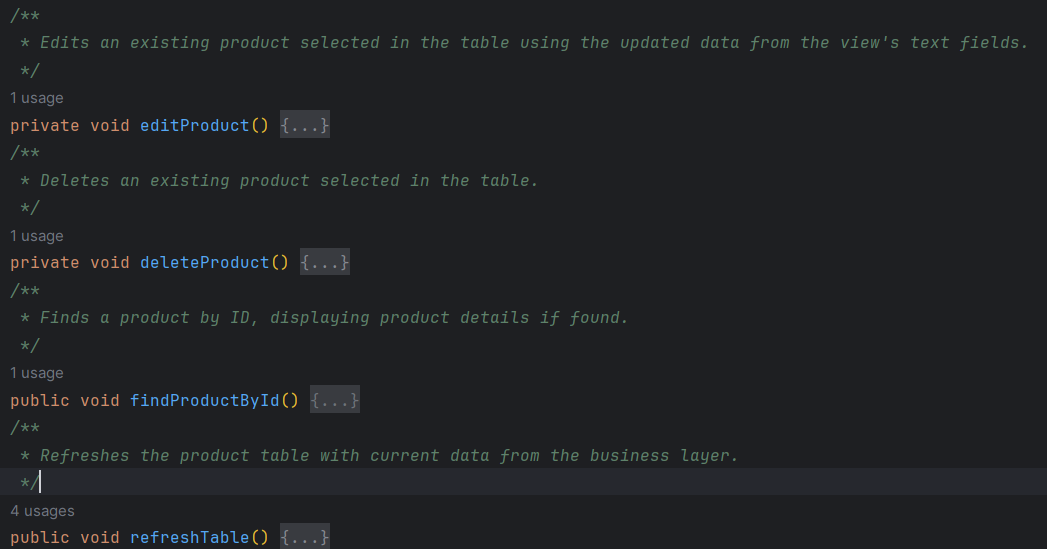




### ProductController

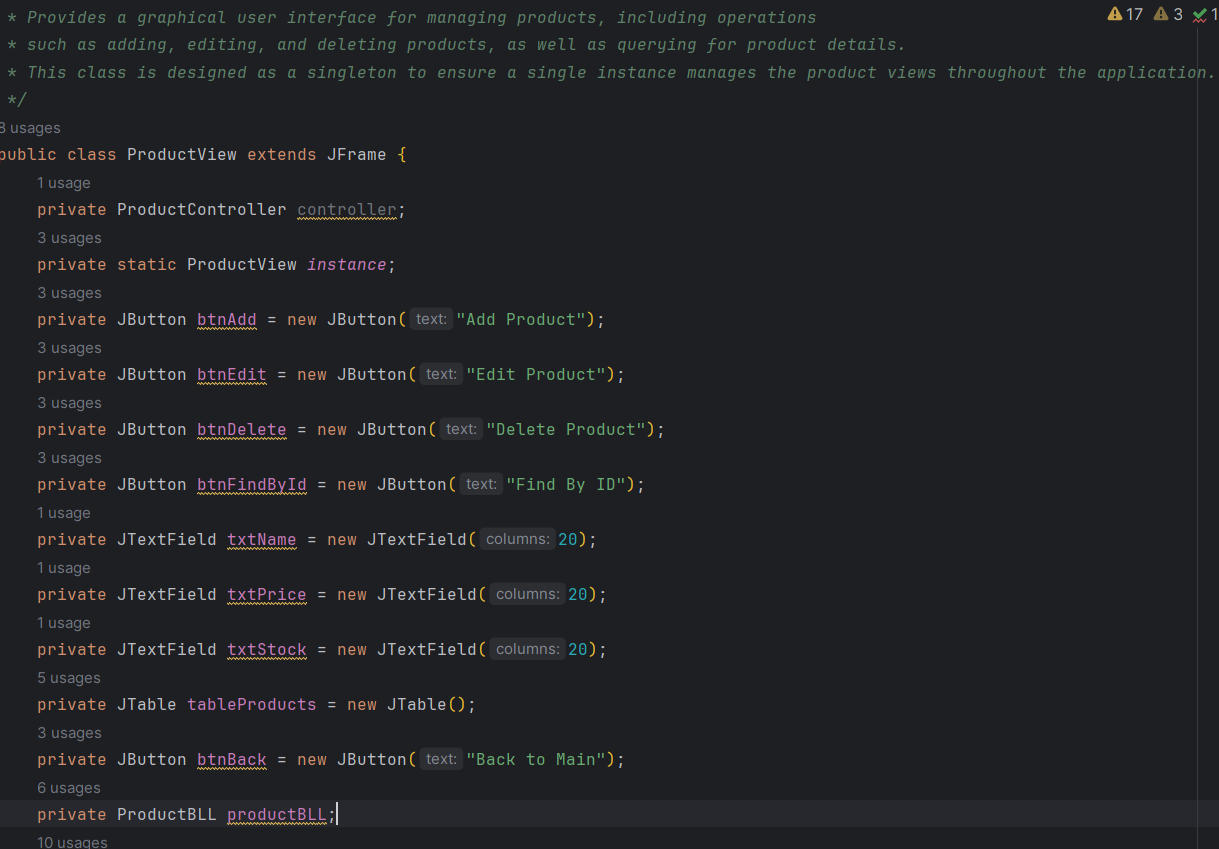
Controler pentru gestionarea UI pentru produse.

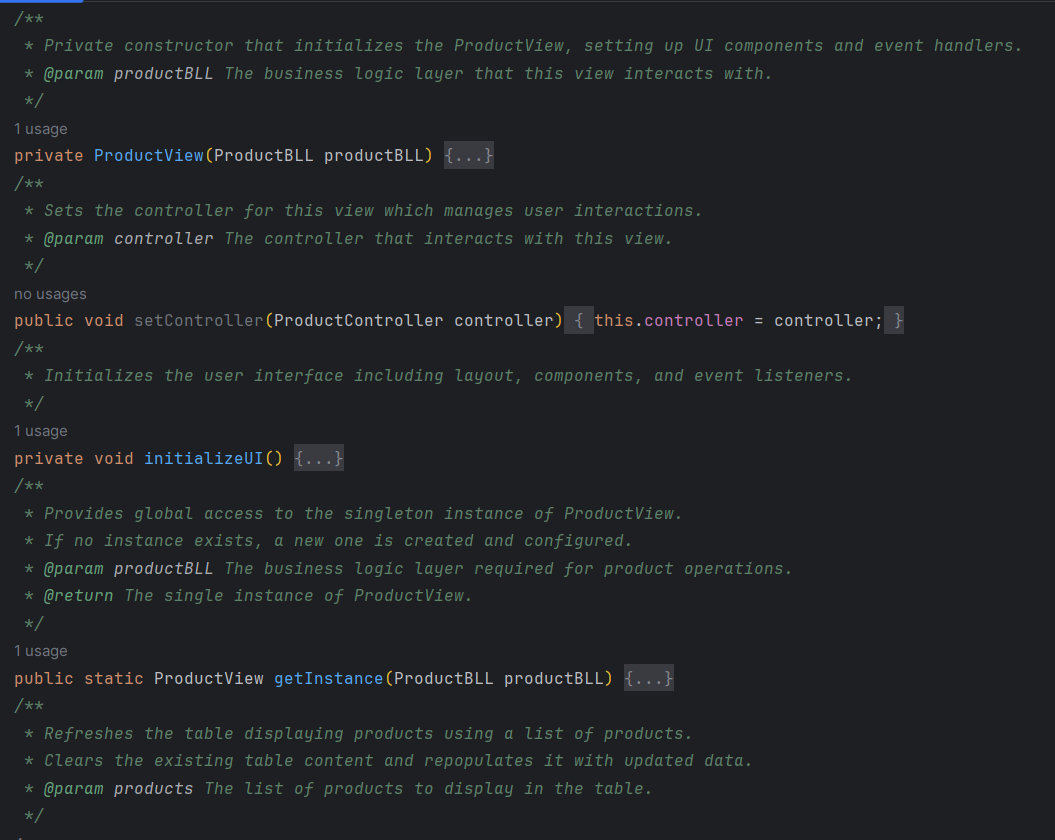




### ProductView

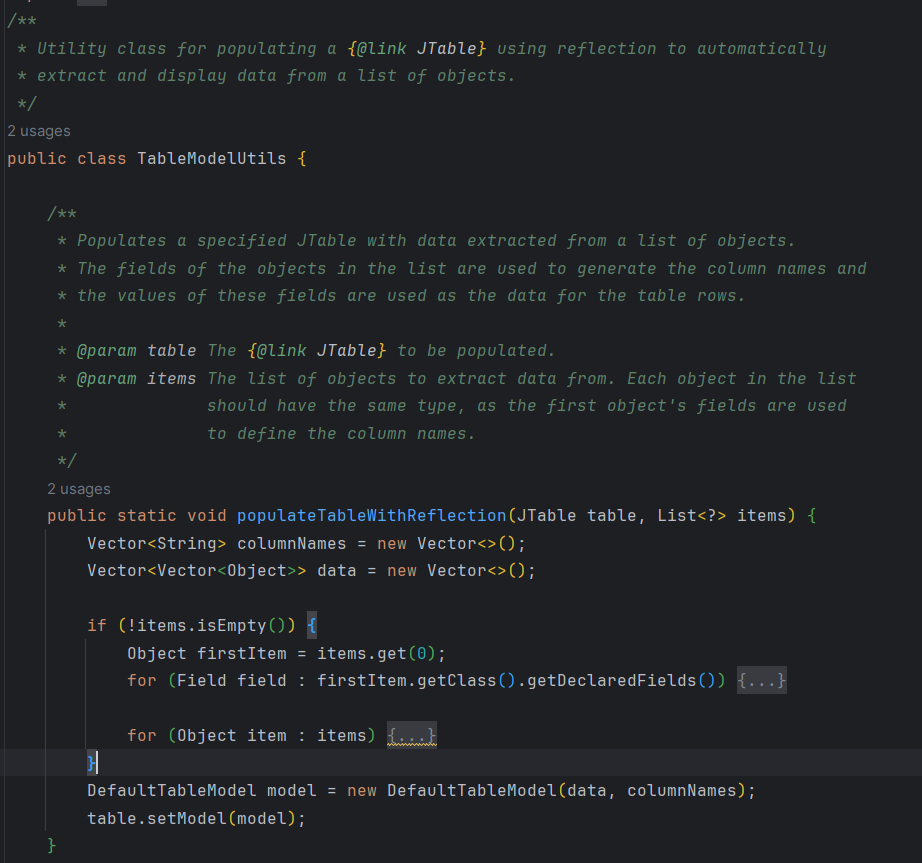
Interfață utilizator pentru produse.





### TableModelUtils

Utilitare pentru modelele de tabel în interfața utilizator.



# Rezultate

Implementarea aplicației de gestionare a comenzilor pentru un depozit a dus la rezultate semnificative în termeni de funcționalitate și ușurință în utilizare. Aplicația oferă o interfață grafică intuitivă și eficientă, care permite utilizatorilor să gestioneze clienți, produse și comenzi într-un mod eficient. Utilizatorii pot adăuga, edita și șterge clienți cu ușurință, iar toate modificările sunt reflectate corect în baza de date. La fel, produsele pot fi adăugate, actualizate sau șterse, cu informații actualizate privind stocul, facilitând gestionarea inventarului. Procesarea comenzilor este simplă și directă, aplicația verificând automat disponibilitatea produselor și actualizând stocurile după fiecare comandă, cu mesaje de avertizare pentru stoc insuficient afișate când este necesar.

Interfața utilizatorului este clară și bine organizată, permițând o navigare ușoară între diferitele module ale aplicației, cum ar fi clienți, produse și comenzi. Documentația este completă și oferă instrucțiuni detaliate despre utilizarea aplicației, inclusiv scenarii comune și soluționarea problemelor, ceea ce îmbunătățește experiența utilizatorilor. În plus, documentația Javadoc este generată corect, oferind descrieri utile pentru clase și metode, ceea ce facilitează înțelegerea și întreținerea codului.

Implementarea tehnicilor de reflecție pentru dinamizarea creării interfețelor și a interogărilor în baza de date a redus cantitatea de cod necesară și a minimizat erorile asociate cu duplicarea codului. Prin această implementare, aplicația oferă o soluție robustă pentru gestionarea operațiunilor unui depozit, îmbunătățind eficiența proceselor de afaceri și contribuind direct la creșterea productivității și la reducerea erorilor umane sau întârzierilor în procesarea comenzilor.

# Concluzii și dezvoltări ulterioare

Implementarea aplicației de gestionare a comenzilor pentru un depozit a validat eficacitatea utilizării unei arhitecturi pe straturi și a bazelor de date relaționale pentru o gestionare organizată și eficientă a datelor. Interfața utilizatorului, care este optimizată și intuitivă, a îmbunătățit semnificativ eficiența în operarea zilnică, crescând astfel productivitatea și minimizând posibilitățile de erori umane. Utilizarea tehnicilor de reflecție pentru generarea dinamică a interfețelor și interogărilor de bază de date a contribuit la eficientizarea dezvoltării și a redus riscul de erori asociate cu duplicarea codului.

Privind spre viitor, dezvoltarea aplicației ar putea beneficia de mai multe îmbunătățiri pentru a spori utilitatea și scalabilitatea acesteia. Integrarea cu alte sisteme de management al depozitelor ar putea oferi o gestionare mai fluidă a stocurilor și comenzi provenite din diverse canale. Un modul avansat de raportare ar permite utilizatorilor să genereze rapoarte personalizate, oferind astfel date valoroase pentru luarea deciziilor strategice.

Dezvoltarea unei aplicații mobile sau a unei interfețe web responsive ar îmbunătăți accesibilitatea sistemului, permitând gestionarea afacerii de oriunde și oricând. Implementarea tehnologiilor de inteligență artificială pentru anticiparea necesarului de stocuri ar putea optimiza gestionarea inventarului, prevenind problemele de sub-stocare sau supra-stocare.

De asemenea, îmbunătățirea măsurilor de securitate, inclusiv autentificarea multifactor și criptarea datelor, ar proteja informațiile sensibile ale companiei și ale clienților. În cele din urmă, asigurarea conformității cu normele de sustenabilitate și reglementările actuale ar putea ajuta compania să își îmbunătățească practicile de afaceri și să rămână competitivă pe piață.

Aceste dezvoltări ar extinde funcționalitățile aplicației, consolidându-i rolul ca instrument esențial în managementul operațional al unui depozit și sprijinind adaptabilitatea și creșterea continuă a afacerii într-un mediu economic în schimbare.

# 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>