DOCUMENTATIE

TEMA *3*

Orders Management

NUME STUDENT: Chindris Alexandra - Maria

GRUPA: 30226

# CUPRINS

[1. Obiectivul temei 3](#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc95297886)

[3. Proiectare 11](#_Toc95297887)

[4. Implementare 14](#_Toc95297888)

[5. Rezultate 20](#_Toc95297889)

[6. Concluzii 21](#_Toc95297890)

[7. Bibliografie 21](#_Toc95297891)

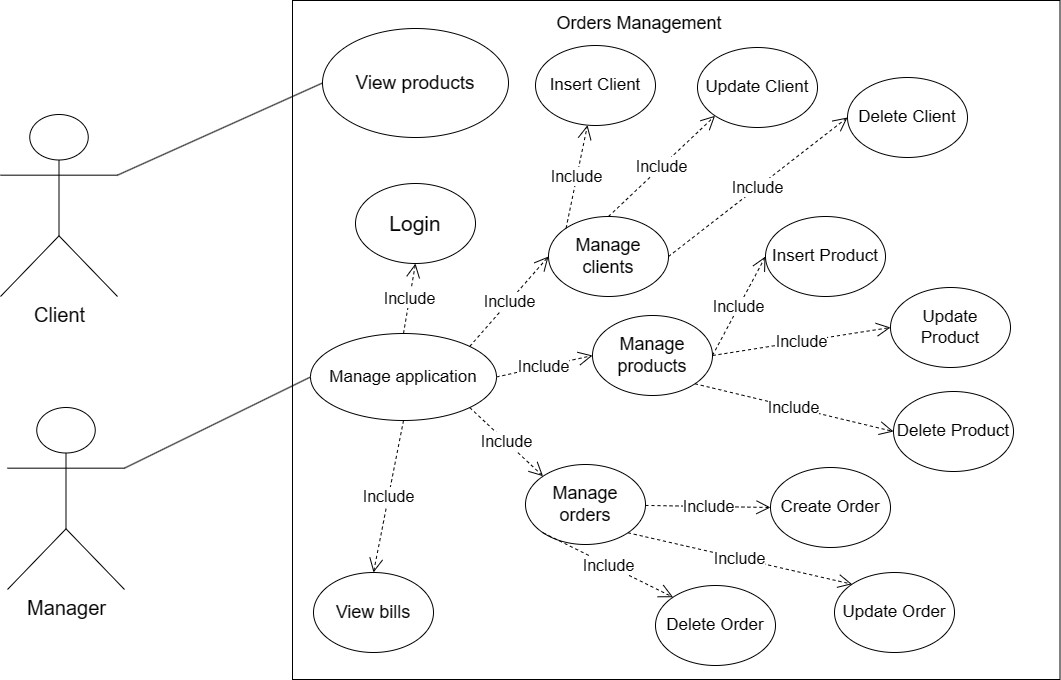
# Obiectivul temei

* Obiectivul principal : Proiectarea și implementarea unei aplicații pentru gestionarea comenzilor clienților pentru un depozit.
* Obiectivele secundare:
  + Analizarea problemei și identificarea cerințelor - determinam cerintele functionale si cele non-functionale. [Capitolul 2]
  + Proiectarea aplicației de gestiune a comenzilor - determinam clasele de care avem nevoie impreuna cu atributele si metodele lor si le organizam in pachete. Determinam legaturile dintre pachete si clase. [Capitolul 3]
  + Implementarea aplicației de gestiune a comenzilor – implementam clasele, cat si algoritmii de aduagarea in coada. [Capitolul 4]
  + Testarea aplicației de gestiune a comenzilor [Capitolul 5]

# Analiza problemei, modelare, scenarii, cazuri de utilizare

* Cerințe funcționale:
  + Aplicația trebuie să permită angajatorului să adauge un client nou, sa-i actualizeze datele si eventual sa-l stearga.
  + Aplicația trebuie să permită angajatorului să adauge un produs nou, sa-i actualizeze datele si eventual sa-l stearga.
  + Aplicația trebuie să permită angajatorului să adauge o comanda noua, sa o actualizeze si eventual sa o stearga.
  + Aplicația trebuie să permită angajatorului sa vizualizeze facturile create in urma crearii comenzii.
  + Aplicația trebuie să permită clientului sa vizualizeze produsele.
* Cerințe non-funcționale:
  + Aplicația trebuie sa fie ușor de folosit de utilizator.
  + Aplicația trebuie să precizeze ce date se doresc introduce de utilizator.

*Diagrama use case*:



1. **Use case:** View Products

**Primary actor:** Client

**Main Success Scenario:**

1. Clientul acceseaza sectiunea adresata lui apasand butonul “Client”.
2. Aplicatia va afisa produsele de vanzare impreuna cu pretul si numarul de bucati disponibile pe stoc.
3. **Use case:** Manage application

**Primary actor:** Manager

**Main Success Scenario:**

1. Managerul acceseaza sectiunea adresata lui apasand butonul “Manager”.
2. Aplicatia va afisa un formular de logare.

1. **Use case:** Login

**Primary actor:** Manager

**Main Success Scenario:**

1. Managerul trebuie sa introduca username-ul si parola pentru autentificare
2. In cazul autentificarii reusite, managerul va putea sa selecteze ce doreste sa gestioneze: client, produse, comenzi sau sa vada facturile associate comenzilor plasate.

**Alternative Sequence:**

1. Datele de autentificare nu sunt valide.
2. Se va afisa o fereastra care anunta managerul despre incorectitudinea datelor si ii este permis sa mai incerce autentificare.
3. **Use case:** Manage clients

**Primary actor:** Manager

**Main Success Scenario:**

1. Managerul acceseaza sectiunea de gestionare a clientilor apasand butonul “Clients Management”.
2. Aplicatia va afisa clientii din baza de date si va pune la dispozitie cele 3 operatii asupra clientiilor: insert, update si delete.
3. **Use case:** Manage products

**Primary actor:** Manager

**Main Success Scenario:**

1. Managerul acceseaza sectiunea de gestionare a produselor apasand butonul “Products Management”.
2. Aplicatia va afisa produsele din baza de date si va pune la dispozitie cele 3 operatii asupra produselor: insert, update si delete.
3. **Use case:** Manage orders

**Primary actor:** Manager

**Main Success Scenario:**

1. Managerul acceseaza sectiunea de gestionare a comenzilor apasand butonul “Orders Management”.
2. Aplicatia va afisa comenzile din baza de date si va pune la dispozitie cele 3 operatii asupra comenzilor: insert, update si delete.
3. **Use case:** View bills

**Primary actor:** Manager

**Main Success Scenario:**

1. Managerul acceseaza sectiunea de vizionare a facturilor emise apasand butonul “Bills”.
2. Aplicatia va afisa facturile din baza de date emise dupa efectuarea comenzilor.
3. **Use case:** Insert Client

**Primary actor:** Manager

**Main Success Scenario:**

1. Se apasa butonul “Insert” si se va deschide o fereastra care va cere datele necesare adaugarii unui nou client.
2. Managerul trebuie sa introduca numele, adresa si email-ul noului client si sa apese butonul “Actualizeaza”.
3. In fereastra curenta se va afisa noul client, impreuna cu cei care existau deja.

**Alternative Sequence:**

1. Daca datele introduse pentru adaugarea noului client nu sunt valide se va afisa o fereastra care va semnala acest lucru, iar managerul poate corecta greseala.
2. **Use case:** Update Client

**Primary actor:** Manager

**Main Success Scenario:**

1. Se va selecta randul clientului care se doreste a fi actualizat.
2. Se apasa butonul “Update”.
3. Se va afisa o fereastra unde pot fi introduse noile date ale clientului, datele actualizate.
4. Managerul poate sa introduca doar datele care necesita sa fie actualizate, lasand restul campurilor goale si la final trebuie sa apese butonul “Actualizeaza”.
5. In fereastra curenta se va afisa clientul cu datele actualizate.

**Alternative Sequence:**

1. Daca datele introduse pentru actualizarea clientului nu sunt valide se va afisa o fereastra care va semnala acest lucru, iar managerul poate corecta greseala.
2. Daca nu a fost selectat niciun client, dar se apasa butonul “Update” se va semnala eroarea printr-un mesaj corespunzator.
3. **Use case:** Delete Client

**Primary actor:** Manager

**Main Success Scenario:**

1. Se va selecta randul clientului care se doreste a fi sters.
2. Se apasa butonul “Delete”.
3. In fereastra curenta vor aparea doar restul clientilor.

**Alternative Sequence:**

1. Daca nu a fost selectat niciun client, dar se apasa butonul “Delete” se va semnala eroarea printr-un mesaj corespunzator.
2. **Use case:** Insert Product

**Primary actor:** Manager

**Main Success Scenario:**

1. Se apasa butonul “Insert” si se va deschide o fereastra care va cere datele necesare adaugarii unui nou produs.
2. Managerul trebuie sa introduca numele produsului, cantitatea disponibila si pretul noului produs si sa apese butonul “Actualizeaza”.
3. In fereastra curenta se va afisa noul produs, impreuna cu cele care existau deja.

**Alternative Sequence:**

1. Daca datele introduse pentru adaugarea noului produs nu sunt valide se va afisa o fereastra care va semnala acest lucru, iar managerul poate corecta greseala.
2. **Use case:** Update Product

**Primary actor:** Manager

**Main Success Scenario:**

1. Se va selecta randul produsului care se doreste a fi actualizat.
2. Se apasa butonul “Update”.
3. Se va afisa o fereastra unde pot fi introduse noile date ale produsului, datele actualizate.
4. Managerul poate sa introduca doar datele care necesita sa fie actualizate, lasand restul campurilor goale si la final trebuie sa apese butonul “Actualizeaza”.
5. In fereastra curenta se va afisa produsul cu datele actualizate.

**Alternative Sequence:**

1. Daca datele introduse pentru actualizarea produslui nu sunt valide se va afisa o fereastra care va semnala acest lucru, iar managerul poate corecta greseala.
2. Daca nu a fost selectat niciun produs, dar se apasa butonul “Update” se va semnala eroarea printr-un mesaj corespunzator.
3. **Use case:** Delete Product

**Primary actor:** Manager

**Main Success Scenario:**

1. Se va selecta randul produsului care se doreste a fi sters.
2. Se apasa butonul “Delete”.
3. In fereastra curenta vor aparea doar restul produselor.

**Alternative Sequence:**

1. Daca nu a fost selectat niciun produs, dar se apasa butonul “Delete” se va semnala eroarea printr-un mesaj corespunzator.
2. **Use case:** Insert Order

**Primary actor:** Manager

**Main Success Scenario:**

1. Se apasa butonul “Insert” si se va deschide o fereastra care va cere datele necesare crearii unei comenzi.
2. Managerul trebuie sa aleaga un client din lista cu clientii existenti, un produs din lista cu produse, cantitatea dorita si sa apese butonul “Actualizeaza”.
3. In fereastra curenta se va afisa noua comanda creata, impreuna cu cele care existau deja.

**Alternative Sequence:**

1. Daca datele introduse pentru adaugarea noii comenzi nu sunt valide se va afisa o fereastra care va semnala acest lucru, iar managerul poate corecta greseala. Ca de exemplu: cantitatea comandata este negativa sau nu este un numar sau nu este disponibila pe stoc.
2. **Use case:** Update Order

**Primary actor:** Manager

**Main Success Scenario:**

1. Se va selecta randul comanezii care se doreste a fi actualizata.
2. Se apasa butonul “Update”.
3. Se va afisa o fereastra unde pot fi introduse noile date ale comenzii, datele actualizate.
4. Managerul poate sa introduca doar datele care necesita sa fie actualizate, lasand restul campurilor goale si la final trebuie sa apese butonul “Actualizeaza”.
5. In fereastra curenta se va afisa comanda cu datele actualizate.

**Alternative Sequence:**

1. Daca datele introduse pentru actualizarea comenzii nu sunt valide se va afisa o fereastra care va semnala acest lucru, iar managerul poate corecta greseala.
2. Daca nu a fost selectata nicio comanda, dar se apasa butonul “Update” se va semnala eroarea printr-un mesaj corespunzator.
3. **Use case:** Delete Order

**Primary actor:** Manager

**Main Success Scenario:**

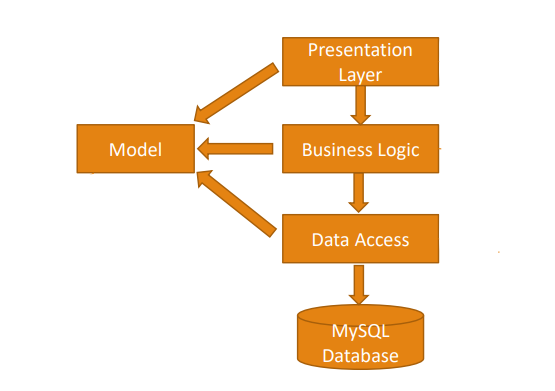
1. Se va selecta randul comenzii care se doreste a fi sters.
2. Se apasa butonul “Delete”.
3. In fereastra curenta vor aparea doar restul comenzilor.

**Alternative Sequence:**

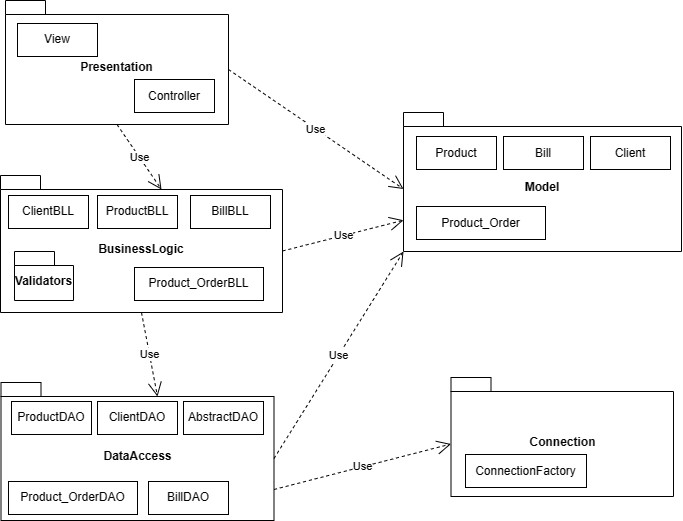
1. Daca nu a fost selectata nicio comanda, dar se apasa butonul “Delete” se va semnala eroarea printr-un mesaj corespunzator.

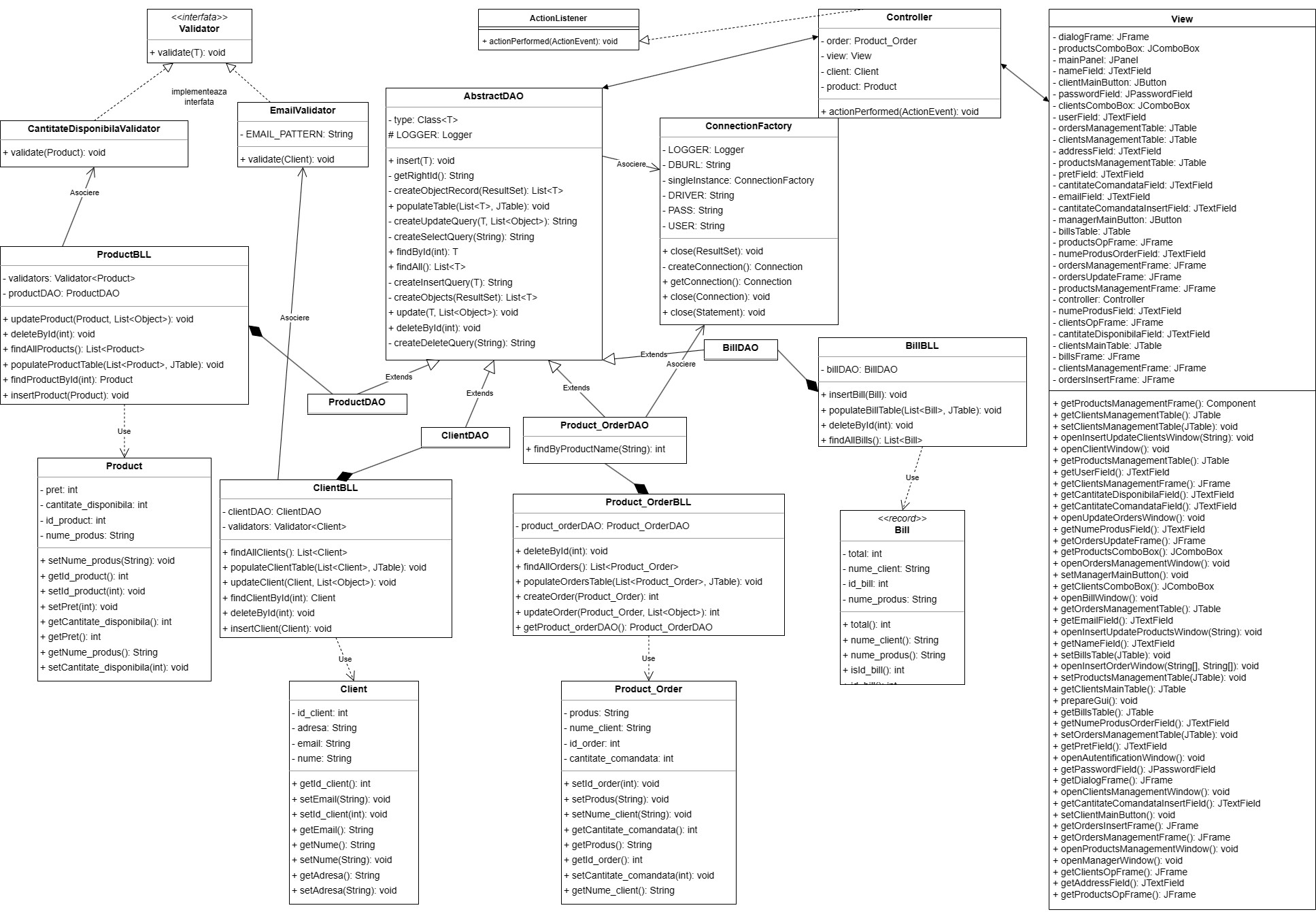
# Proiectare

*Design – Conceptual Architecture:*



*Diagrama de pachete:* -> Use = relatie de dependent.





* + - **CantitateDisponibilaValidator** si **EmailValidator** implementeaza interfata **Validator**=> relatie de **implementare**
    - **Controller** implementeaza interfata **ActionListener** => relatie de de **implementare**
    - **ClientDAO** este o parte a lui **ClientBLL**, au aceasi perioada de viata => relatie de **compozitie**
    - **ProductDAO** este o parte a lui **ProductBLL**, au aceasi perioada de viata => relatie de **compozitie**
    - **Product\_OrderDAO** este o parte a lui **Product\_OrderBLL**, au aceasi perioada de viata => relatie de **compozitie**
    - **BillDAO** este o parte a lui **BillBLL**, au aceasi perioada de viata => relatie de **compozitie**
    - **ClientDAO, ProductDAO, Product\_OrderDAO, BillDAO** extind clasa **AbstractDAO** => relatie de **mostenire**
    - **ProductBLL** foloseste **CantitateDisponibilaValidator** si **ClientBLL** foloseste **EmailValidator** => relatie de **asociere**

# Implementare

Clasa Client

* + - Are un id\_client, numele clientului, adresa si email-ul acestuia.

Clasa Product

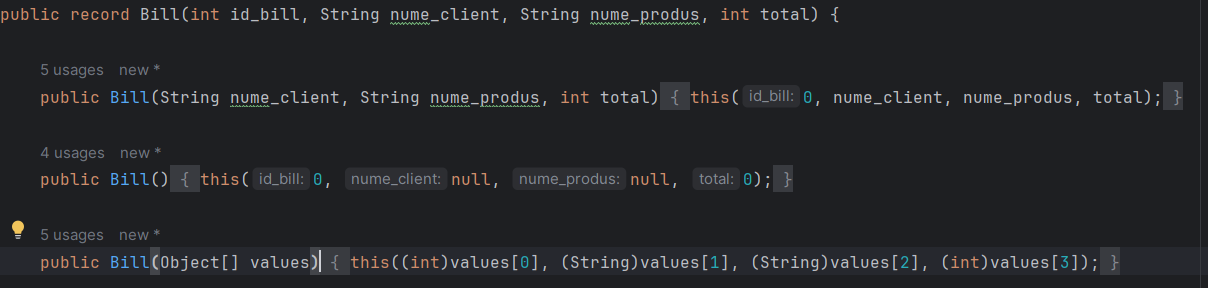
* + - Contine un id al produsului, numele prodususlui, cantitatea disponibila si pretul acestuia.

Clasa Product\_Order

* + - Contine id-ul comenzii, numele clientulu, numele produsului si numarul de bucati comandate.

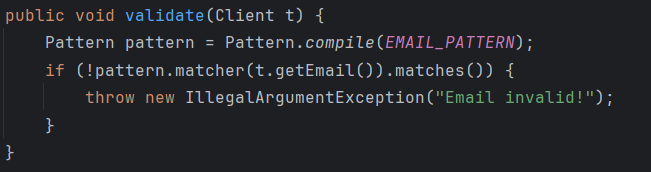
Clasa Bill

* + - Este o clasa imutabila, realizata folosind Java records. Fiind o clasa imutabila, nu putem avea metode de tip setter.
    - Contine id-ul facturii, numele clientului, numele produsului comandat si pretul total al comenzii.



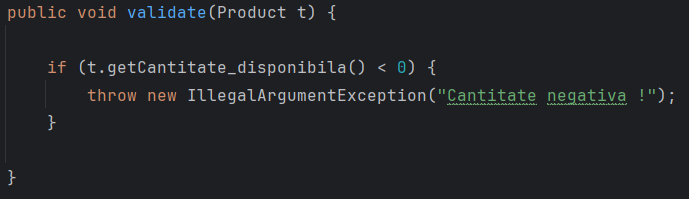
Clasa EmailValidator

* + - Implementeaza interfata Validator, odata cu metoda validate(Client).
    - Metoda verifica daca clientul are o adresa de email ce respecta formatul specific.



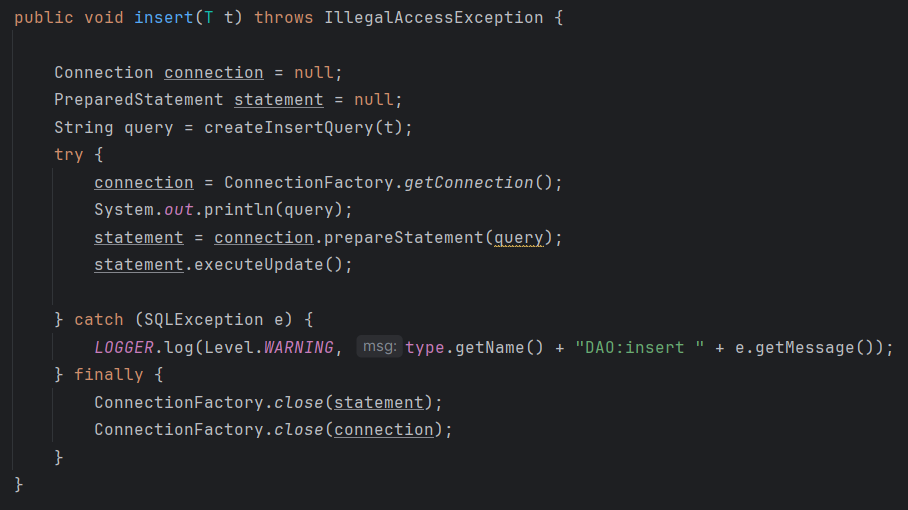
Clasa CantitateDisponibilaValidator

* + - Implementeaza interfata Validator, odata cu metoda validate(Product).
    - Metoda verifica daca produsul are o cantitate reprezentata de un numar intreg pozitiv.



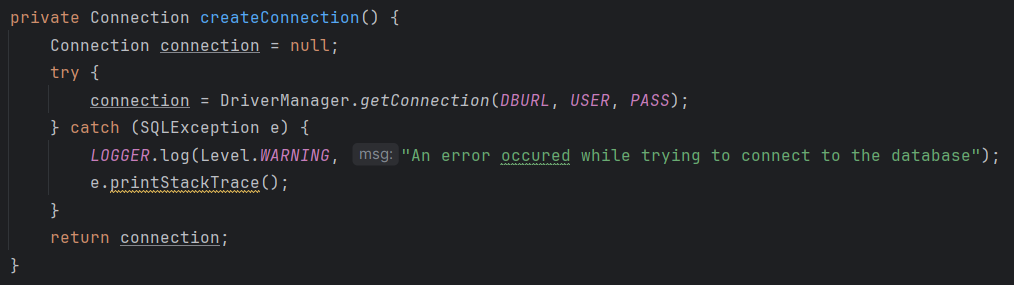
Clasa AbstractDAO

* + - Aceasta este o clasă generică care implementează prin reflecție forma generală a interogărilor. Deoarece interogările sunt aproape toate la fel, această clasă implementează interogările comune. Parametrul generic înlocuiește un model de clasă specific, astfel încât metodele puse în aplicare de această clasă pot fi utilizate de orice altă subclasă sau de orice instanțiere de tip T.
    - Interogările sunt construite mai întâi ca un șir, apoi se aplică conexiunea la baza de date și se execută interogarea.



Clasa ConnectionFactory

* + - Realizeaza conexiunea aplicatiei la baza de date.
    - Aplicația este conectată la o bază de date MySQL numită "baza\_de\_date" prin intermediul utilizatorului root.





* + - Această clasă implementează un șablon Singleton pentru a asigura că există o singură instanță.

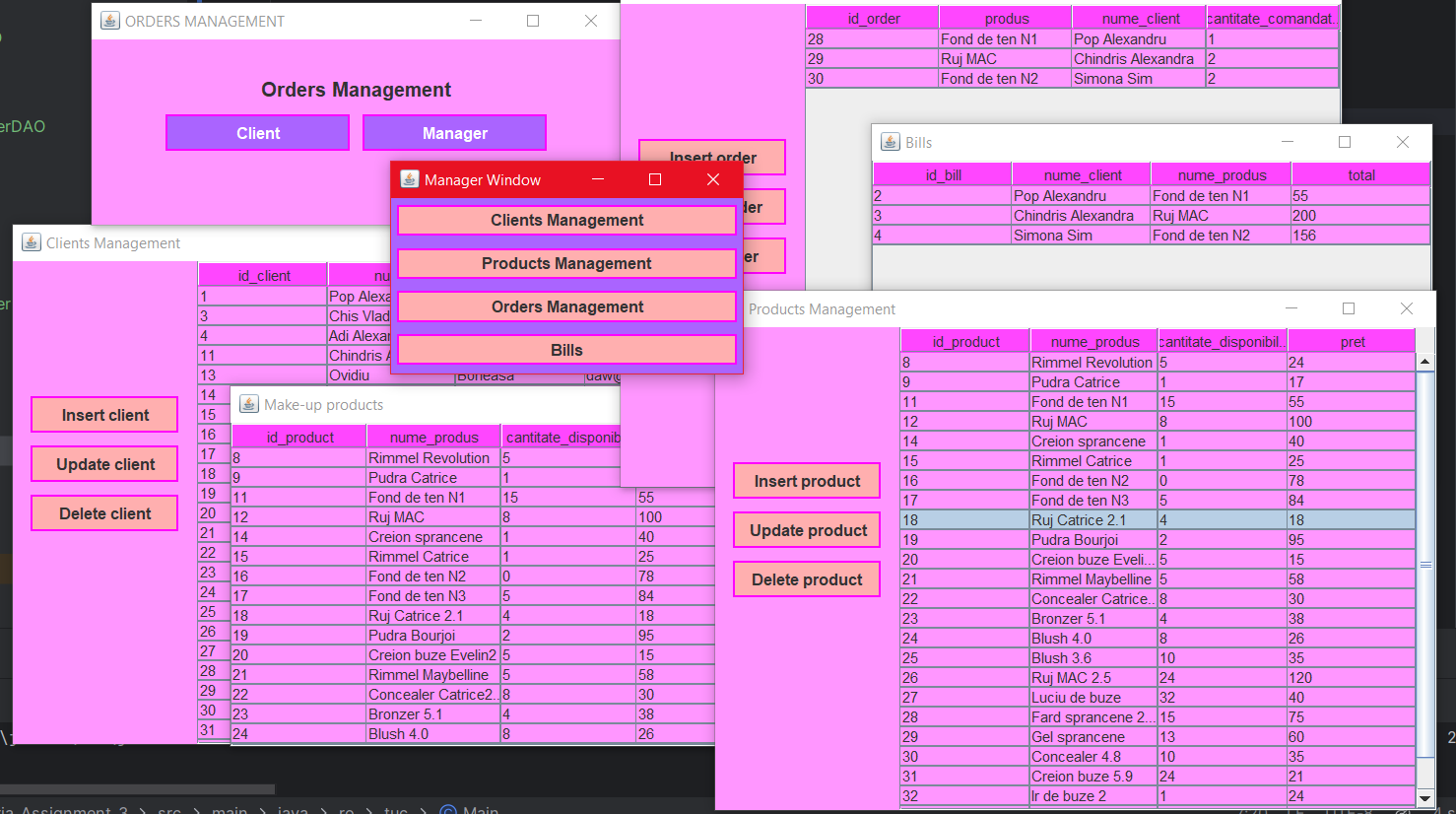
Clasele ClientDAO, ProductDAO, Product\_OrderDAO, BillDAO

* + - Aceste clase extind clasa AbstractDAO și sunt chemate să efectueze operațiile CRUD pe fiecare tabel specificat în numele clasei. Ele sunt mai mult un filtru între logica de afaceri și AbstractDAO. Ele cheama doar super-metodele din clasa părinte cu câmpurile specifice ale clasei copil.

Clasele ClientBLL, ProductBLL, Product\_OrderBLL, BillBLL

* + - Aceste clase formează logica de afaceri a aplicației. Aici aplicația verifică introducerea cererii și decide ce operațiune CRUD ar trebui aplicată.

# Rezultate



# Concluzii

* + - *Posibile dezvoltari:* interfata poate sa fie imbunatatita si pentru partea de login sa se creeze un table in baza de date.

# Bibliografie

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

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

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

- <https://www.baeldung.com/javadoc>

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