

Sistem de gestionare a comenzilor

Halasz Lorand Daniel

Grupa 30226

Cuprins

[1. Obiectivul temei 3](#_Toc511961133)

[2. Analiza si modelarea problemei 4](#_Toc511961134)

[2.1. Cazuri de utilizare si scenarii 4](#_Toc511961135)

[2.2. Asumptii 5](#_Toc511961136)

[3. Proiectare 6](#_Toc511961137)

[3.1. Decizii de proiectare 6](#_Toc511961138)

[3.2. Diagrama UML 7](#_Toc511961139)

[3.3. Structuri de date 8](#_Toc511961140)

[3.4. Proiectare clase 8](#_Toc511961141)

[3.5. Pachete 9](#_Toc511961142)

[3.6. Interfata utilizator 10](#_Toc511961143)

[3.7. Modul de tratare a erorilor 11](#_Toc511961144)

[4. Implementare 12](#_Toc511961145)

[1. Clasa **Customer** 12](#_Toc511961146)

[2. Clasa **Product** 13](#_Toc511961147)

[3. Clasa **Orders** 13](#_Toc511961148)

[4. Clasa **Company** 14](#_Toc511961149)

[5. Clasa **CustomerAdmin** 15](#_Toc511961150)

[6. Clasa **ProductAdmin** 15](#_Toc511961151)

[7. Clasa **OrderProcessing** 16](#_Toc511961152)

[8. Clasa **CompanyAdmin** 17](#_Toc511961153)

[9. Clasa **Reflection** 17](#_Toc511961154)

[10. Clasa **BazaDeDate** 18](#_Toc511961155)

[11. Clasa **MainFrame** 19](#_Toc511961156)

[5. Rezultate 20](#_Toc511961157)

[6. Concluzii 21](#_Toc511961158)

[7. Bibliografie 22](#_Toc511961159)

1. Obiectivul temei

In ziua de azi, comertul este unul dintre cele mai dezvoltate domenii, fie ca vorbim de comertul online sau de comertul traditional. In ambele cazuri, trebuie sa se tina o evidenta a vanzarilor produselor, mai exact a comenzilor onorate. Un astfel de raport este util din mai multe puncte de vedere, intrucat cu ajutorul lui se pot obtine diferite informatii, precum cantitatea de produse ramase in stoc, perioadele din an in care vanzarile cresc, ajutand inclusiv la dezvoltarea unui plan de marketing, pentru atragerea unui numar cat mai mare de clienti, lucrul mult dorit de catre fiecare comerciant. Obiectivul principal al temei consta in propunerea, proiectarea si implementarea unui sistem de gestionare a comenzilor pentru un depozit. Obiectivul principal poate fi descompus in mai multe obiective secundare, si anume:

* Implementarea diferitelor operatii pe tabele, precum:
  + adaugare
  + modificare
  + stergere
  + vizualizare continului tabelului
* implementarea unei interfete grafice
* crearea unor noi comenzi si actualizarea stocurilor de produse
* utilizarea bazei de date relationale
* utilizarea unor tehnici de reflection pentru a crea o metoda care primeste o lista de obiecte si genereaza tabel
* crearea unor facturi

1. Analiza si modelarea problemei

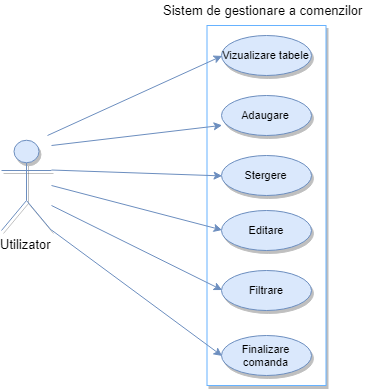
Analiza problemei consta in identificarea domeniului problemei. De cele mai multe ori, problema fiind una complexa urmeaza a fi descompusa in subprobleme pentru a ne usura munca. Aceste subprobleme pot fi, la randul lor, descompuse in alte subprobleme, acest pas repetandu-se pana in momentul in care se ajunge la un nivel de dificultate minim, care poate fi rezolvat. Urmatorul pas consta in identificarea domeniului solutiei. Acest lucru este realizabil prin aplicarea in repetate randuri a unui proces de abstractizare, pana cand se obtine ceea ce se doreste.

Odata analizate aceste lucruri se formeaza o idee generala asupra problemei. Urmatorul pas consta in descrierea cadrului de cerinte functionale, ceea ce se reduce la prezentarea modului in care sistemul reactioneaza in anumite situatii si la anumite intrari. In cazul de fata, pentru sistemul de gestionare a comenzilor se cunoaste faptul ca trebuie sa poata efectua anumite operatii ( de adaugare, de stergere, de modificare si de filtrare a campurilor din tabele) si sa ofere o interactiune mai usoara pentru utilizator, prin intermediul unei interfete grafice. Tot in cadrul analizei problemei se considera faptul ca in momentul in care se realizeaza o comanda, este necesara o actualizare a stocului de produse, iar in cazul in care nu se dispune de stocul necesar onorarii comenzii sa se afiseze un mesaj care sa indice acest lucru.

Procesul de dezvoltare software se realizeaza pornind de la anumite nevoi. Acestea sunt translatate in cerinte, care urmeaza a fi validate de un client. Pe urma se concepe design-ul prin proiectarea de ansamblu si de detaliu. Se implementeaza componentele si se testeaza separat, dupa care acestea se integreaza in sistem, care la randul sau trebuie validat si testat ca intreg.

* 1. Cazuri de utilizare si scenarii

Cazurile de utilizare reprezinta un set de scenarii legate de modul in care este utilizat sistemul. Acestea ilustreaza modul in care un actor (in cazul de fata un utilizator) interactioneaza cu sistemul. Diagrama de utilizare (diagrama use-case) ne ajuta sa descoperim entitatile sitemului, atributele, actorii si comportamentul sistemului.

 Pentru inceput, daca utilizatorul doreste sa vizualizeze tabelele, este necesar sa apese pe butonele din dreptul fiecarui tabel, care poarta denumirea de “Vizualizare”, urmata de numele tabelului pe care doreste sa il vada. Apoi, in functie de ceea ce doreste utilizatorul sa faca, acesta trebuie sa introduca datele necesara dupa care trebuie sa apese pe butonul corespunzator comenzii alese.

Introducerea datelor in mod corect reprezinta un best scenario, de aceea utilizarea sitemului in mod corect presupune introducerea datelor respectand semnificatia campurilor. Totusi, acest lucru nu este sigur si ne putem astepta la diferite lucruri, dintre care:

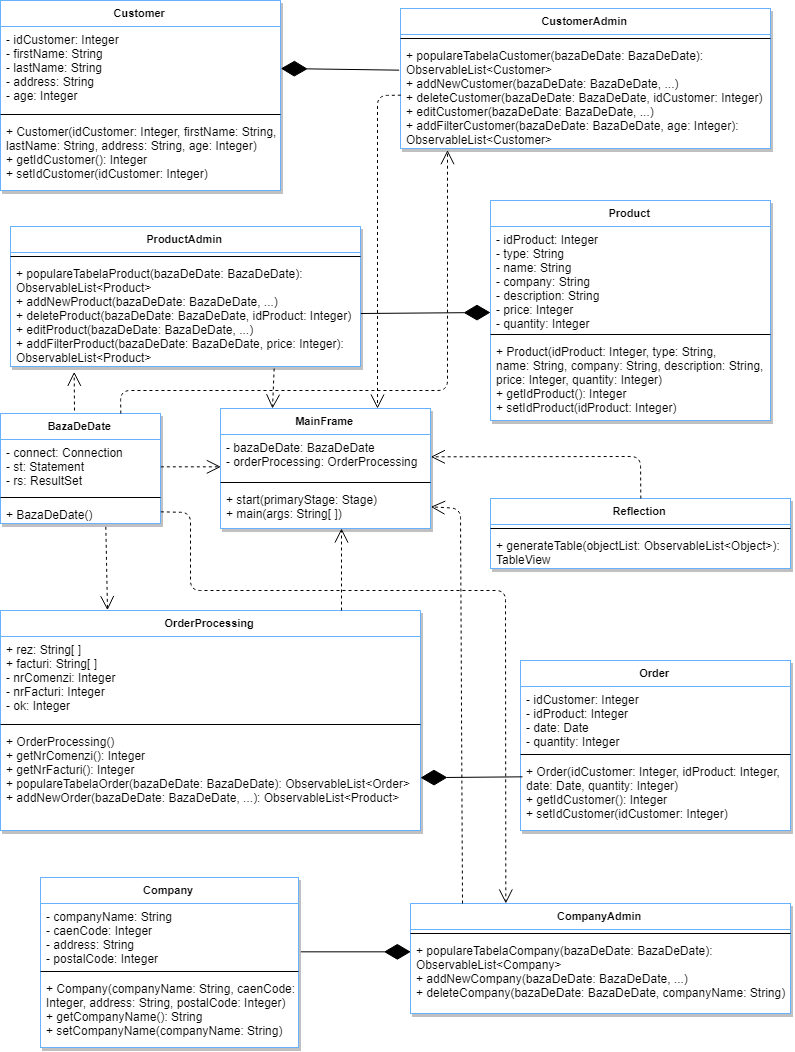
* Introducere unui text in locul unei valori
* Nerespecarea formatului datei
* Introducerea unui nou client, cu un id identic cu id-ul altui client deja existent. Acest lucru este valabil pentru toate tabelele.
* Neintroducere unor date indispensabile, precum id-ul produsului in cadrul unei comenzi sau a numelui unei companii
  1. Asumptii

Asumptiile stabilesc cadrul in care se dezvolta aplicatia. In cadrul proiectului s-a stabilit urmatoarea asumptie: presupunem ca datele de intrare sunt introduse in mod corect, conform specificatiilor anterioare.

1. Proiectare
   1. Decizii de proiectare

Pornind de la specificatia proiectului, am luat decizia descompunerii problemei in mai multe clase: clasa MainFrame ocupandu-se de lansarea aplicatiei si de implementarea interfetei grafice, clasele Customer, Product, Order si Company reprezentand clasele reprezentative fiecarui tabel, clasa BazaDeDate care creeaza legatura cu baza de date, clasele CompanyAdmin, CustomerAdmin, OrderProcessing, ProductAdmin, Reflection, clase care se ocupa de administrarea tabelelor si clasa Reflection care contine metoda generateTable, care, cu ajutorul metodelor de reflection, returneaza un tabel generat pe baza unei liste de obiecte primite ca parametru al metodei.

* 1. Diagrama UML

Urmatoarea diagrama UML descrie clasele si obiectele sistemului si relatiile dintre acestea:

* 1. Structuri de date

Pentru definirea campurilor din tabele se utilizeaza o liste obiecte. Acest lucru este realizabil utilizand colectii, implicit clasa FXCollections care implementeaza toate metodele din interfata ObservableList<T>.

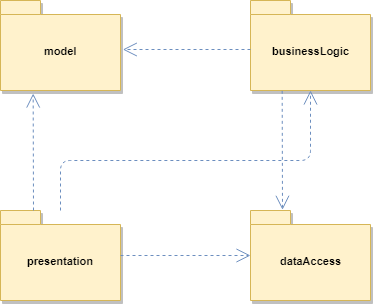
* 1. Proiectare clase

Sistemul de gestionare a comenzilor contine 11 clase:

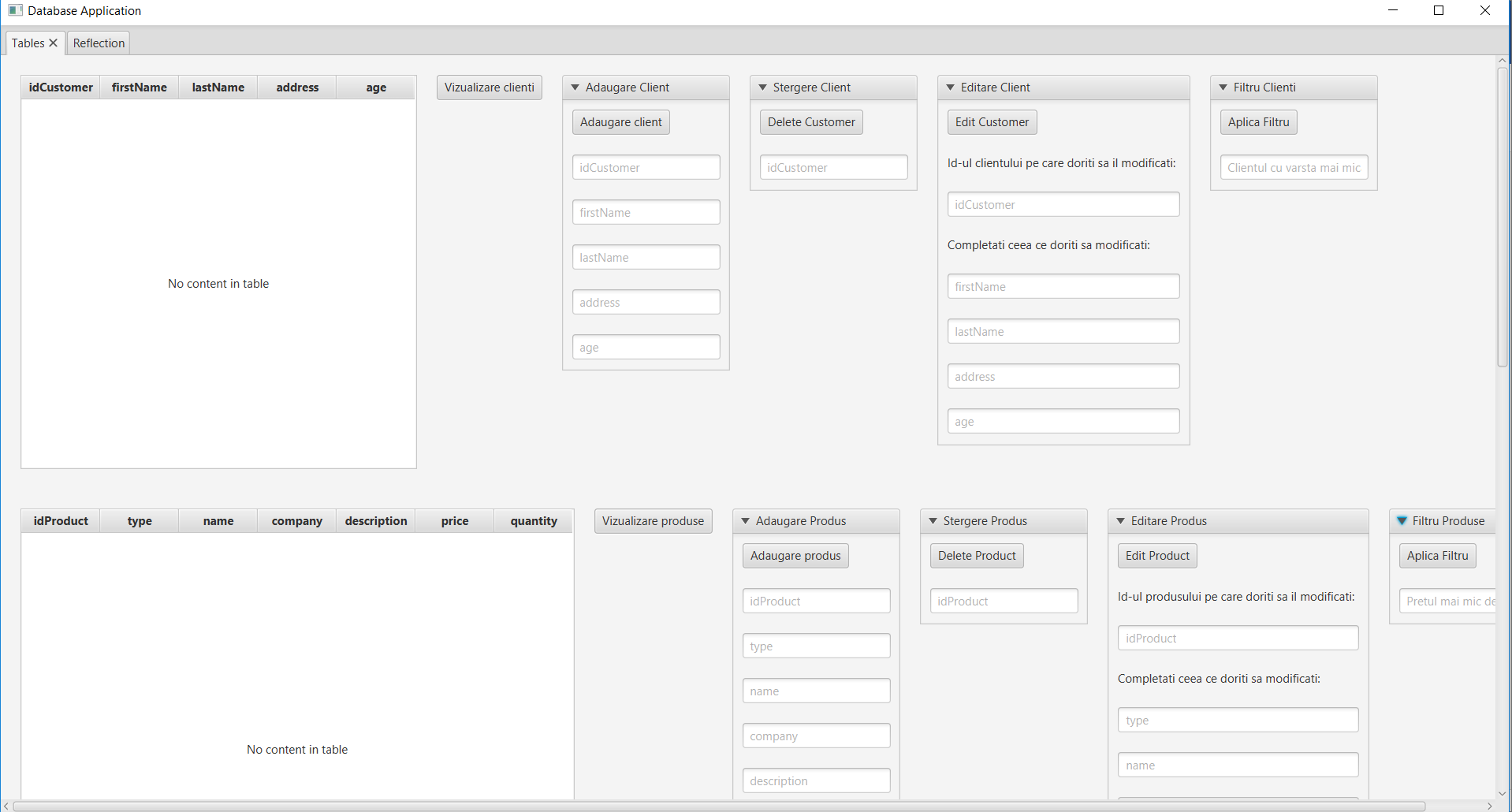
1. Clasa **Customer** – se foloseste pentru a stoca toti clientii. Aceasta contine un constructor, iar pentru fiecare atribut al clasei, cate un getter si un setter.
2. Clasa **Product** – se foloseste pentru a stoca toate produsele aflate in catalogul depozitului. Clasa Product contine un constructor si, de asemenea, cate un getter si un setter pentru atributele clasei.
3. Clasa **Orders** – stocheaza toate comenzile inregistrate. Aceasta contine un constructor, iar pentru fiecare atribut al clasei, cate un getter si un setter.
4. Clasa **Company** – memoreaza companiile din cadrul tabelului Company, impreuna cu datele acestora. Clasa contine un constructor si metode de getter si setter pentru fiecare atribut al clasei.
5. Clasa **CustomerAdmin** – clasa utilizata pentru administrarea clientilor, care contine metodele necesare pentru vizualizarea tabelului si pentru implementarea operatiilor pe tabelul Customer.
6. Clasa **ProductAdmin** – clasa asemanatoare cu clasa prezentata anterior, clasa CustomerAdmin, dar se diferentiaza, prin tabelul care sta la baza acesteia si anume tabelul Product
7. Clasa **OrderProcessing** – clasa utilizata pentru procesarea comenzilor, care contine metodele necesare pentru vizualizarea tabelului si pentru adaugarea unor noi comenzi.
8. Clasa **CompanyAdmin** – clasa utilizata pentru administrarea firmelor, care contine metodele de adaugare si de stergere a unei companii, si metoda de vizualizare a tabelului.
9. Clasa **Reflection** – clasa care contine metoda generateTable, care, cu ajutorul metodelor de reflection, returneaza un tabel generat pe baza unei liste de obiecte primite ca parametru al metodei.
10. Clasa **BazaDeDate** – clasa care face legatura cu baza de date
11. Clasa **MainFrame** – clasa care sta la baza interfetei grafice si care practic construieste un mod mai usor de interactionare a utilizatorului cu sistemul.
    1. Pachete

Diagramele de pachet ne ajuta la descompunerea sistemelor mari in subsisteme, lucru necesar pentru a putea realiza o implementare sistematica. Sistemul de gestionare a comenzilor este impartit in 4 pachete:

1. Pachetul model – reprezinta modelele de date ale aplicatiei, mai exact clasele Customer, Product, Order si Company
2. Pachetul businessLogic – contine logica aplicatiei, mai exact clasele CustomerAdmin, ProductAdmin, OrderProcessing, CompanyAdmin si Reflection.
3. Pachetul presentation – contine interfata grafica a proiectului, mai exact clasa MainFrame
4. Pachetul dataAccess – contine clasa care prin care se realizeaza accesul la baza de date, mai exact clasa BazaDeDate



* 1. Interfata utilizator

Interfata grafica reprezinta un mod mai usor de interactionare dintre utilizator si program. Fara aceasta, utilizarea aplicatiei de gestionare a comenzilor pentru un depozit devine mai greoaie si destul de dificila pentru un utilizator nespecializat. Interfata este realizata cu ajutorul clasei MainFrame care extinde clasa Application.

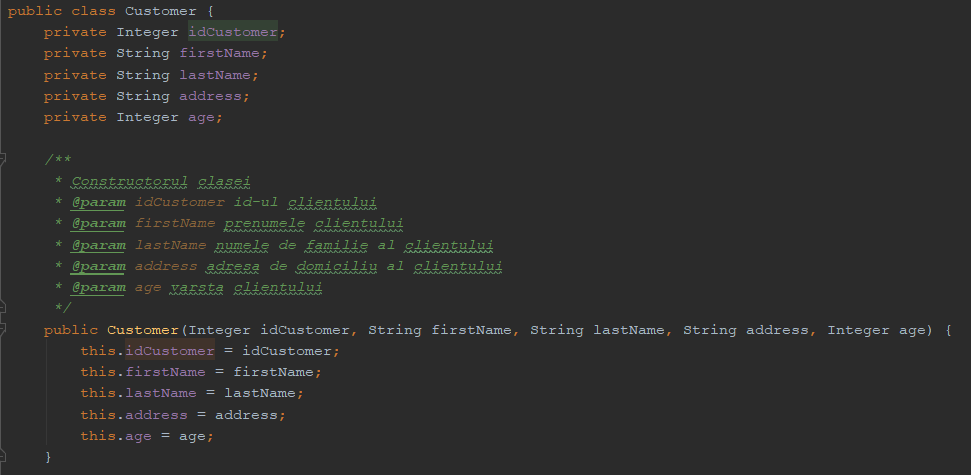
Interfata este compusa dintr-o fereastra cu titlul „Database Application” care este compusa din 2 tab-uri, primul fiind tab-ul pentru tabele, iar cel de-al doilea fiind pentru exemplificarea metodei care utilizeaza tehnicile de reflection. Primul tab este impartit in 4 zone, astfel:

1. Zona 1 – zona rezervata pentru tabelul Customer, care contine butoanele si textField-urile necesare operatiilor specificate
2. Zona 2 – zona utilizata pentru vizualizarea si producerea operatiilor asupra tabelului Product, care contine butoanele si textField-urile necesare
3. Zona 3 – zona in care se pot realiza noi comenzi si, totodata, zona in care se pot vizualiza vechile comenzi
4. Zona 4 – zona in care sunt prezentate, adaugate si sterse diferite companii
   1. Modul de tratare a erorilor

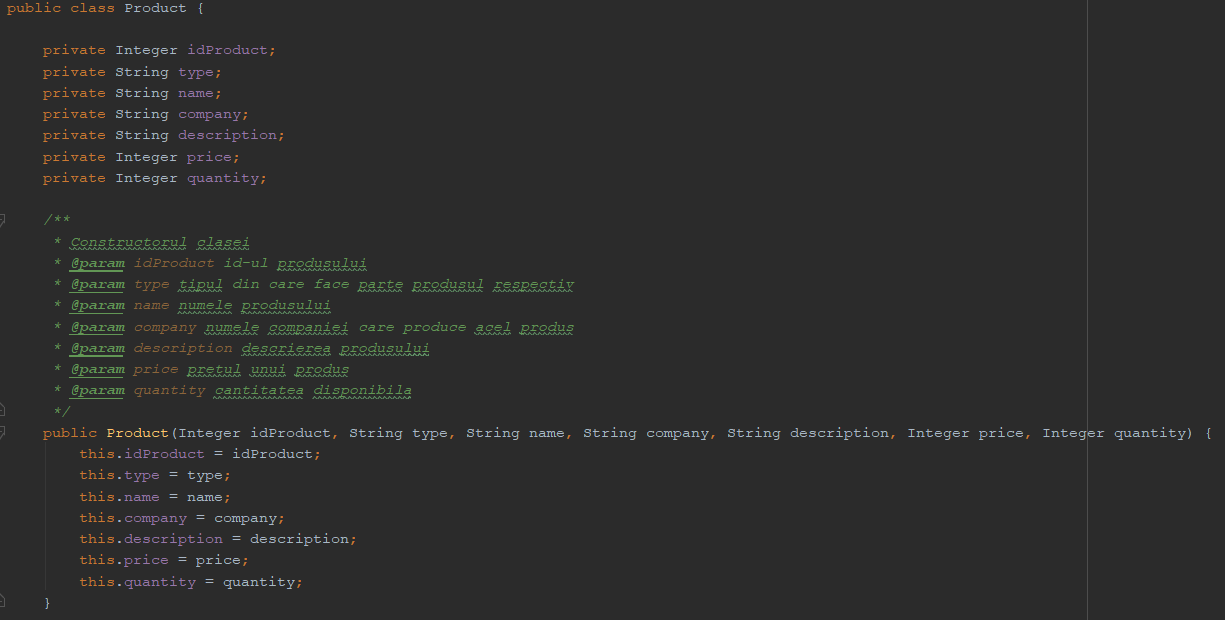
Ca in orice sistem, aparitia unor erori este foarte posibila si trebuie luata in calcul. In cazul de fata, o eroare destul de posibila este introducerea unei comenzi facute de un client care nu exista sau procesarea unei comenzi pentru un produs care lipseste din depozit sau a carui stoc este depasit de cantitatea solicitata. In aceste cazuri, se va afisa un mesaj specific.

1. Implementare

Dupa stabilirea obiectivului principal si a obiectivelor secundare si dupa analiza, modelarea si proiectarea proiectului urmeaza implementarea lui.

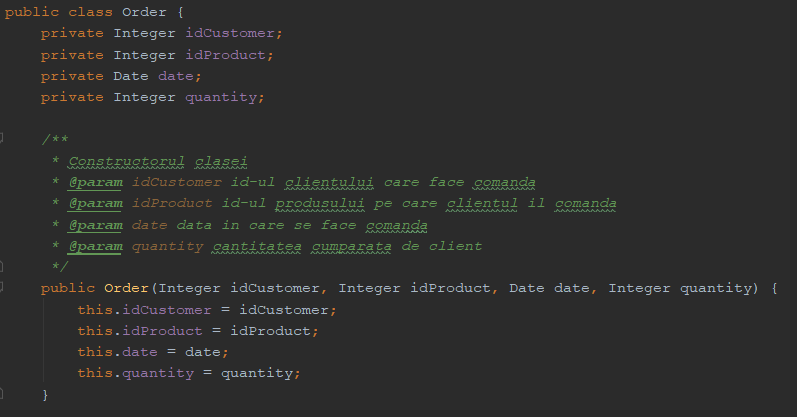
1. Clasa **Customer**

Clasa Customer, are ca si atribute id-ul, prenumele, numele de familie, adresa si varsta clientului. Aceasta contine un constructor, iar pentru fiecare atribut al clasei, cate un getter si un setter, metode necesare pentru procesul de incapsulare.

1. Clasa **Product**

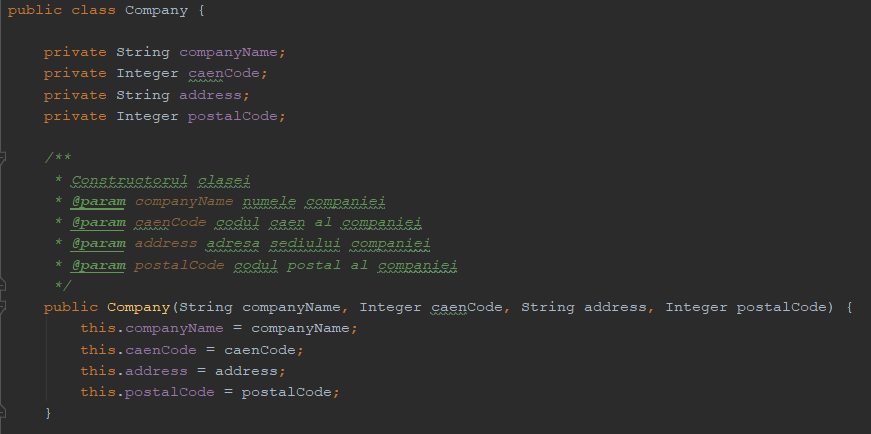
Clasa Product, are ca si atribute id-ul, tipul, nume, compania, descrierea, pretul si cantitatea fiecarui produs. Aceasta contine un constructor, iar pentru fiecare atribut al clasei, cate un getter si un setter.

1. Clasa **Orders**

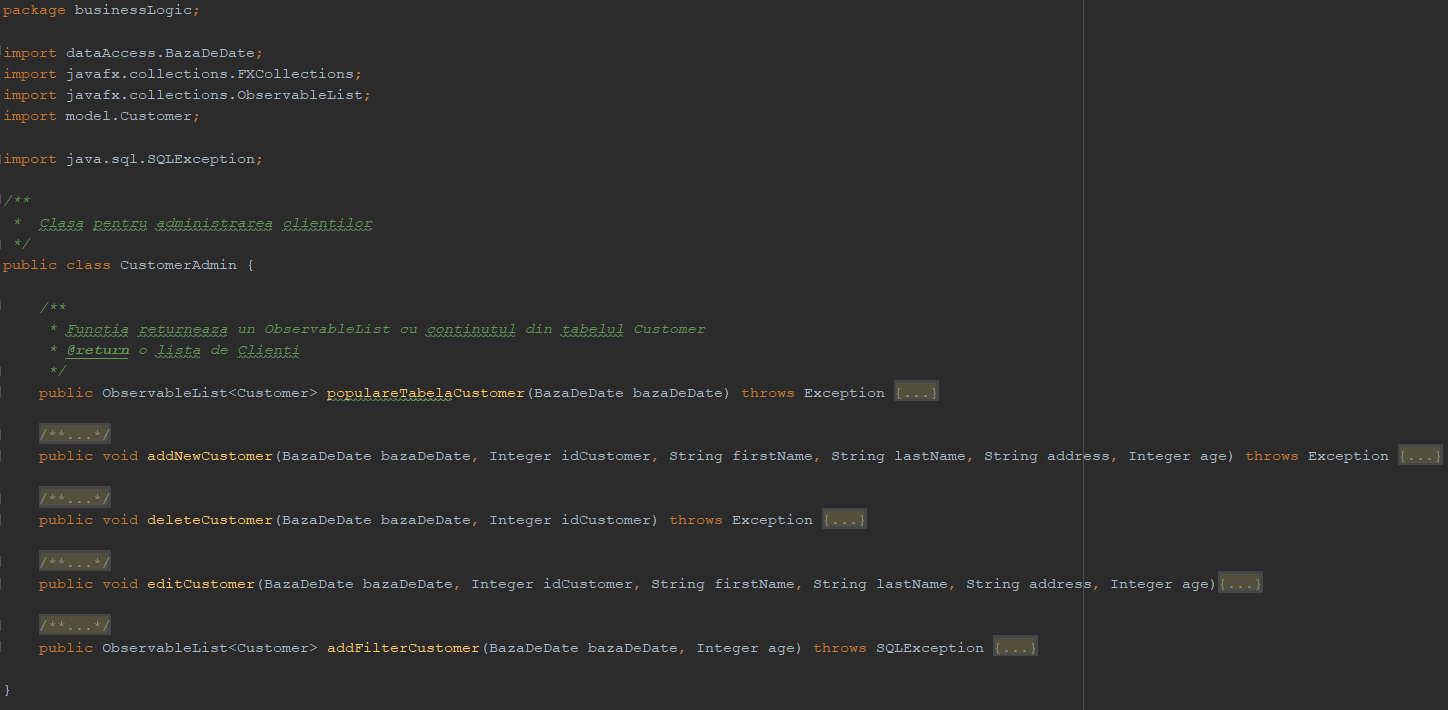


Clasa orders stocheaza toate comenzile inregistrate. Are ca atribute id-ul clientului care face comanda, id-ul produsului pe care acesta il comanda, data in care se face comanda si cantitatea cumparata. Aceasta contine un constructor, iar pentru fiecare atribut al clasei, cate un getter si un setter.

1. Clasa **Company**

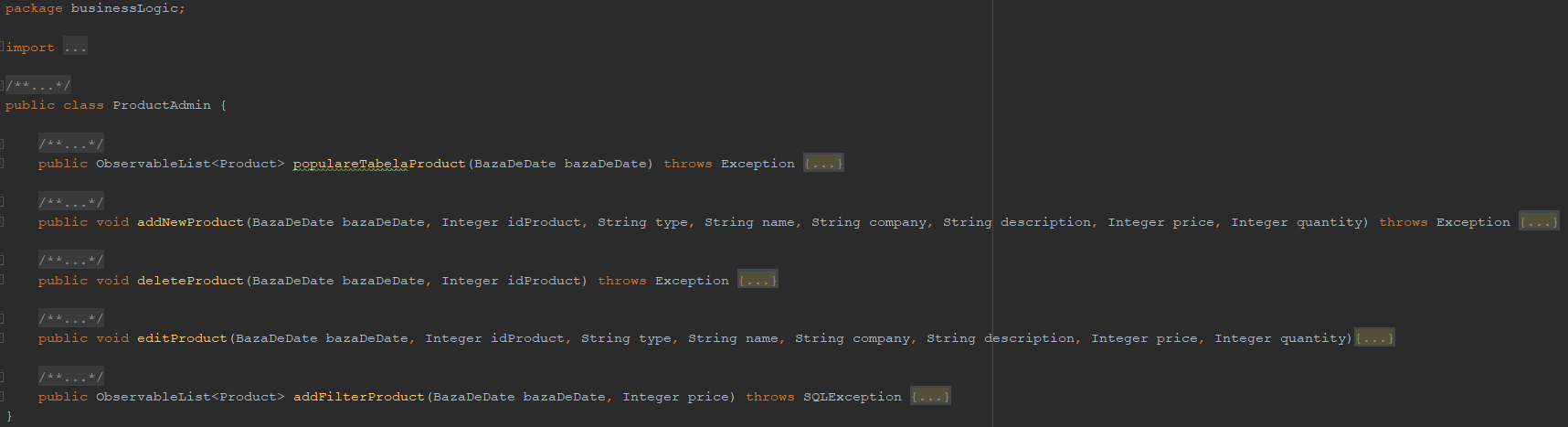


Clasa Company memoreaza companiile din cadrul tabelului Company, impreuna cu datele acestora. Aceasta are ca atribute numele companiei, codul caen, adresa sediului si codul postal al companiei. Clasa contine un constructor si metode de getter si setter pentru fiecare atribut al clasei.

1. Clasa **CustomerAdmin**

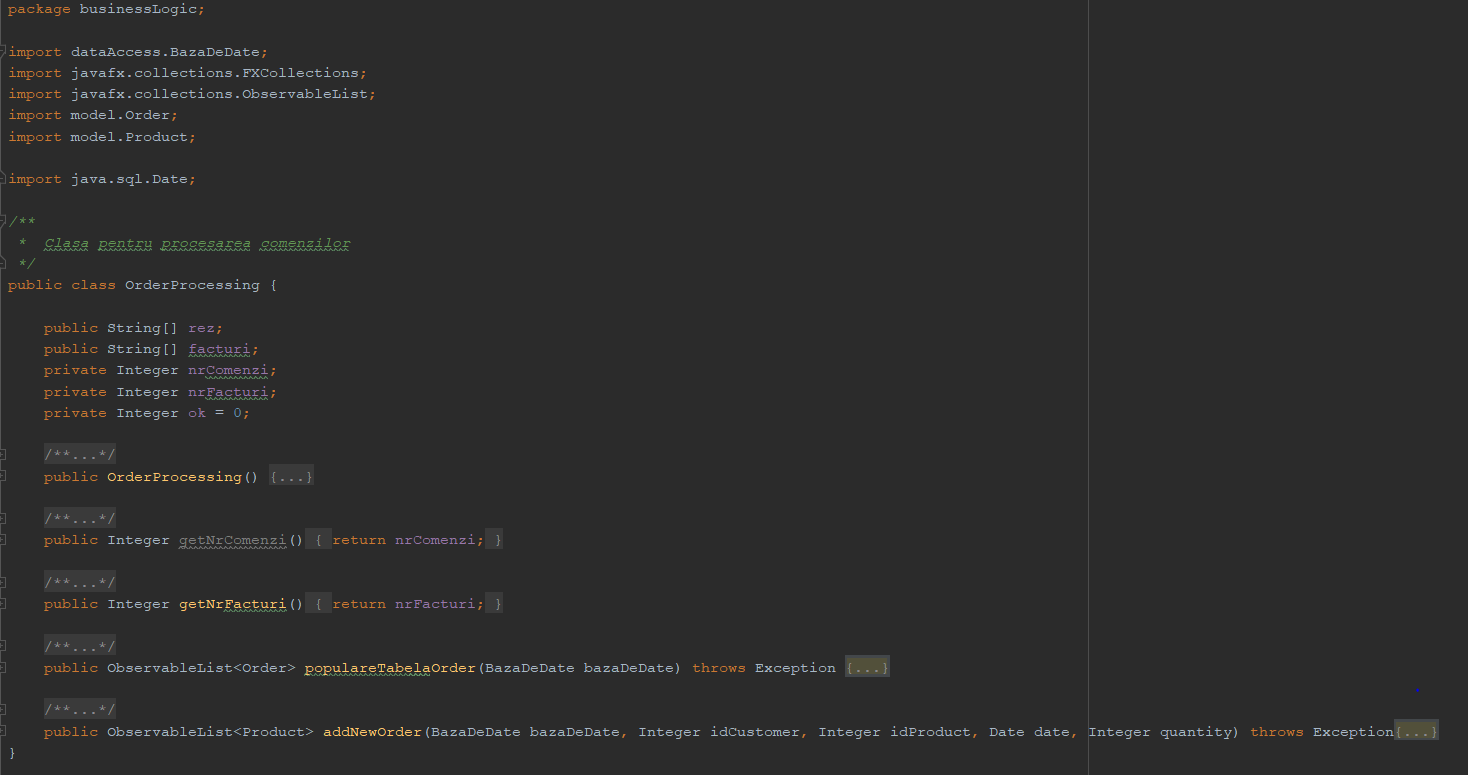
Clasa CustomerAdmin este utilizata pentru administrarea clientilor. Aceasta contine metodele necesare pentru vizualizarea tabelului, pentru adaugarea, pentru stergerea si pentru editarea unui client si pentru filtrarea acestora in functie de varsta.

1. Clasa **ProductAdmin**



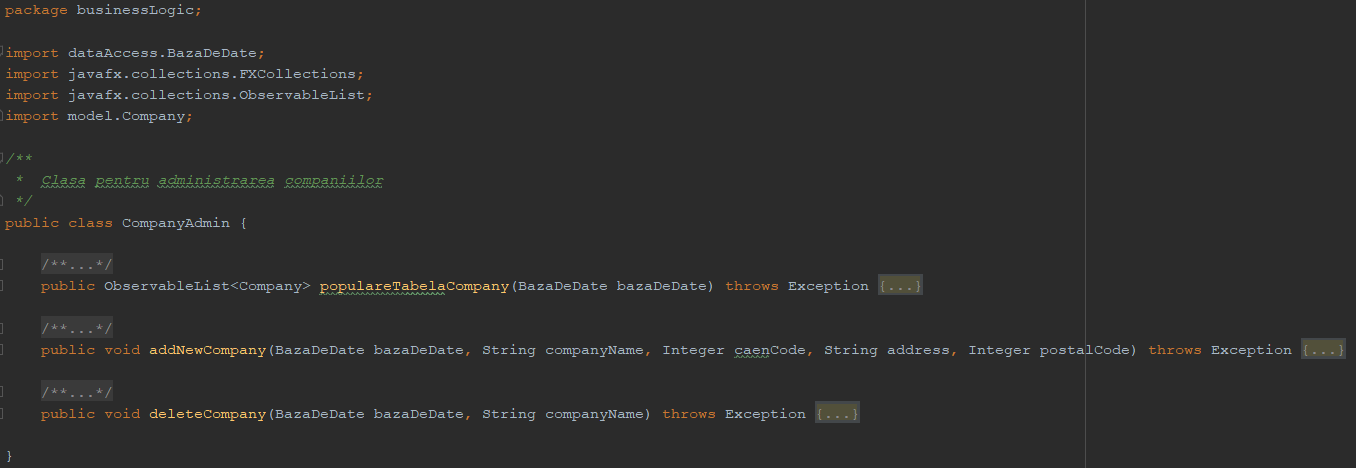
Clasa ProductAdmin este asemanatoare cu clasa prezentata anterior, doar ca in cazul de fata tabelul care sta la baza operatiilor este tabelul Product.

1. Clasa **OrderProcessing**



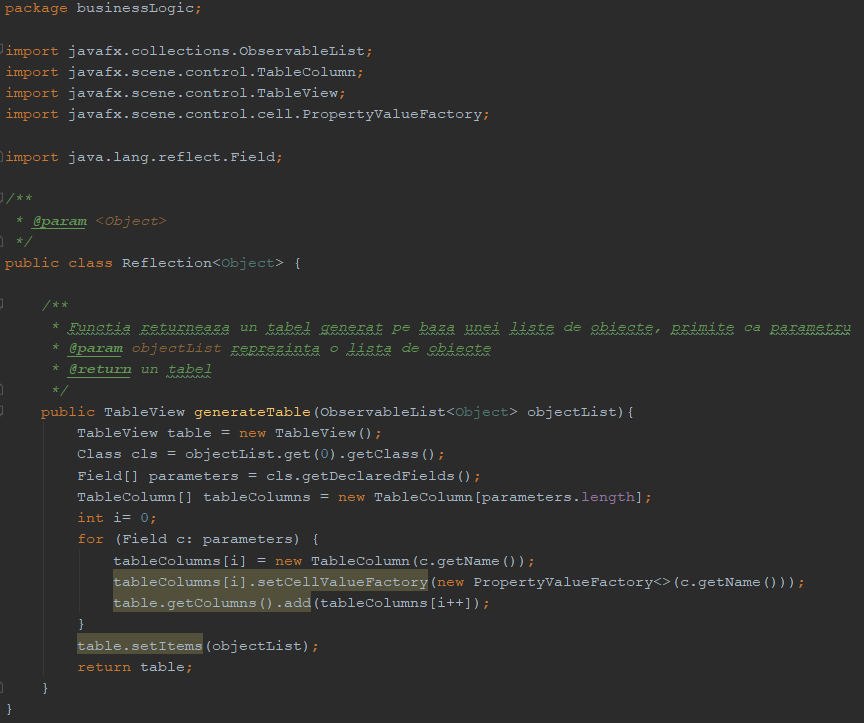
Clasa OrderProcessing este utilizata pentru procesarea comenzilor. Aceasta clasa contine metodele necesare pentru vizualizarea tabelului si pentru adaugarea unor noi comenzi.

1. Clasa **CompanyAdmin**



Clasa CompanyAdmin este utilizata pentru administrarea firmelor. Aceasta contine metodele de adaugare si de stergere a unei companii, si metoda de vizualizare a tabelului.

1. Clasa **Reflection**



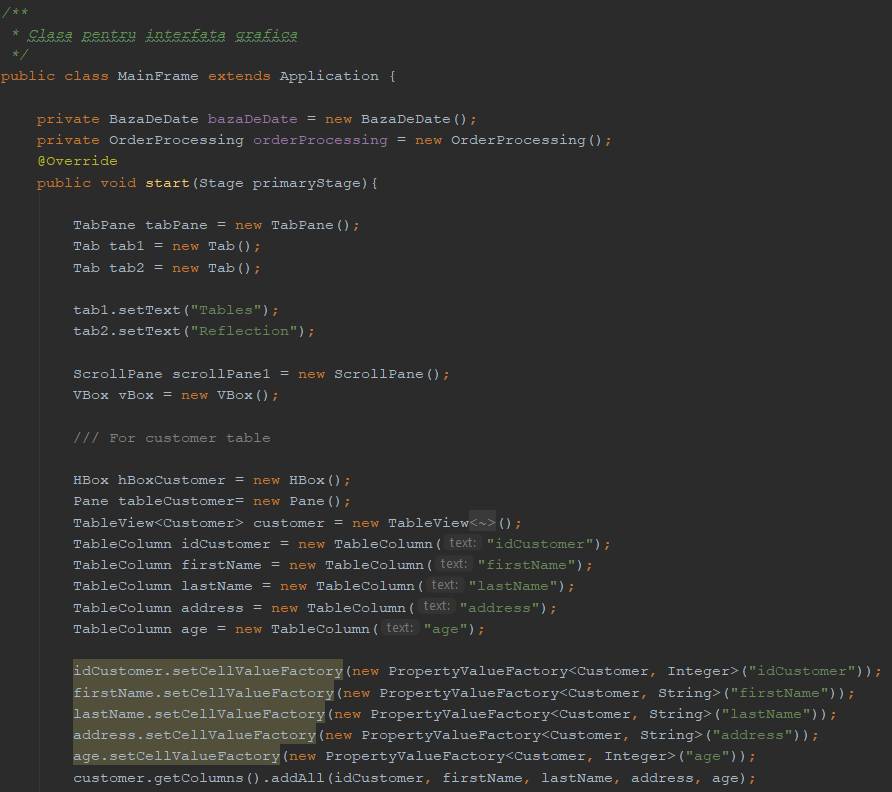
Clasa Reflection contine metoda generateTable, care, cu ajutorul metodelor de reflection, returneaza un tabel generat pe baza unei liste de obiecte primite ca parametru al metodei.

1. Clasa **BazaDeDate**



Clasa BazaDeDate face legatura cu baza de date.

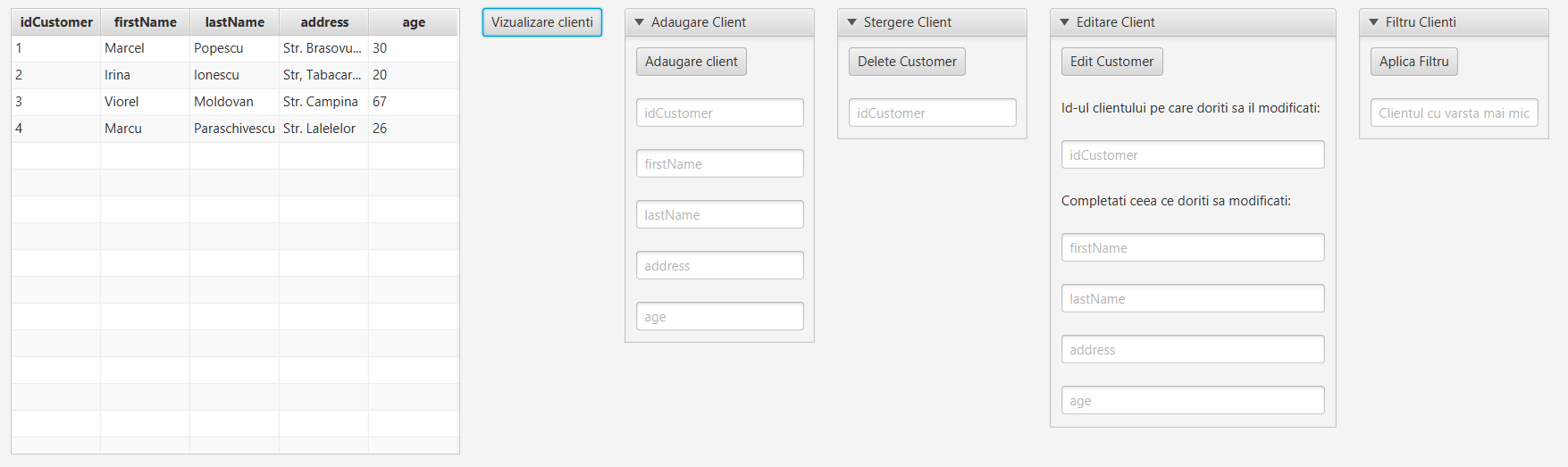
1. Clasa **MainFrame**



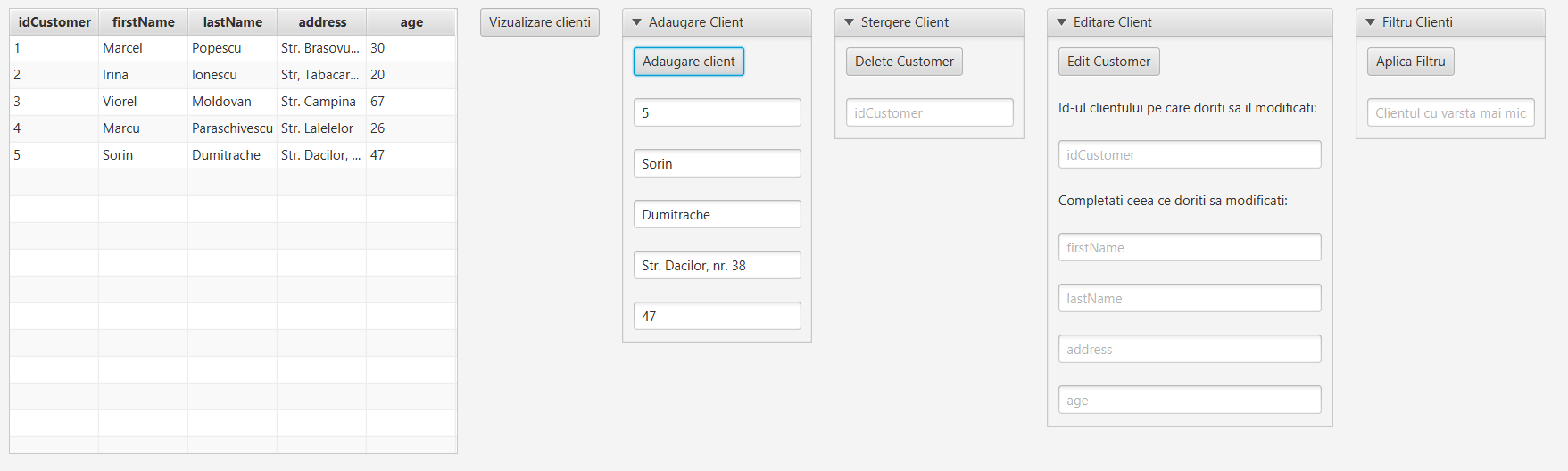
Clasa MainFrame realizeaza practic toata interfata grafica. Aceasta clasa extinde clasa Application si contine metoda main, in interiorul careia se utilizeaza metoda launch. Dupa care urmeaza suprascrierea metodei start care creeaza interfata grafica. Mai apoi, se suprascrie metoda handle pentru fiecare buton in parte.

1. Rezultate

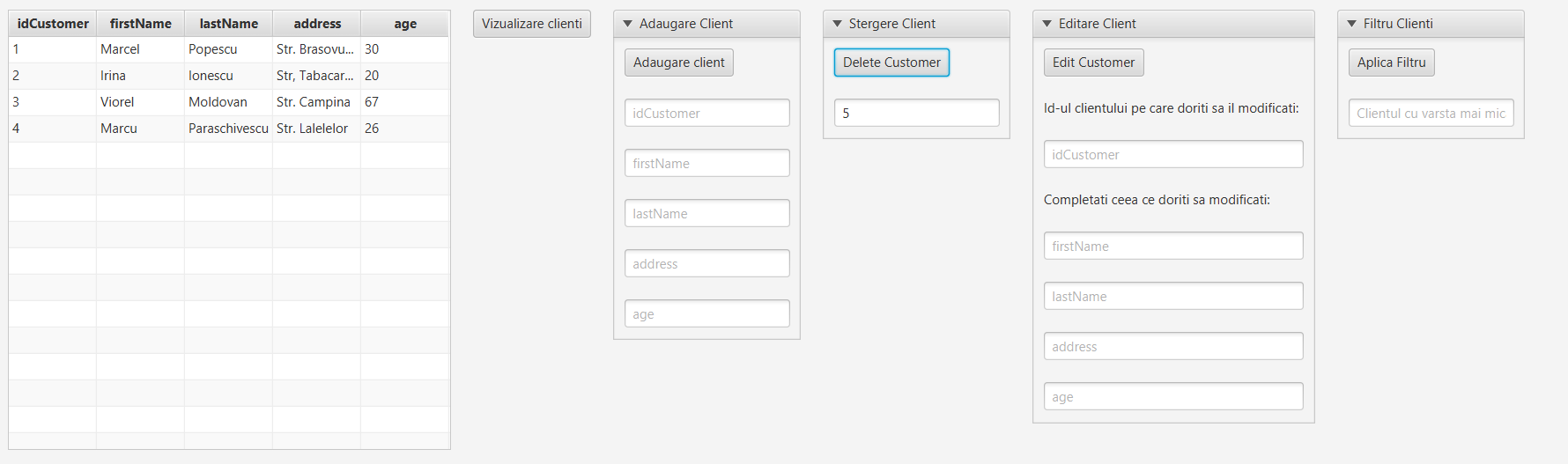
Dupa introducerea datelor dorite se pot apasa butoanele corespunzatoare operatiei alese sau pe butonul de vizualizare date, in cazul in care se doreste doar vizualizarea tabelelor.



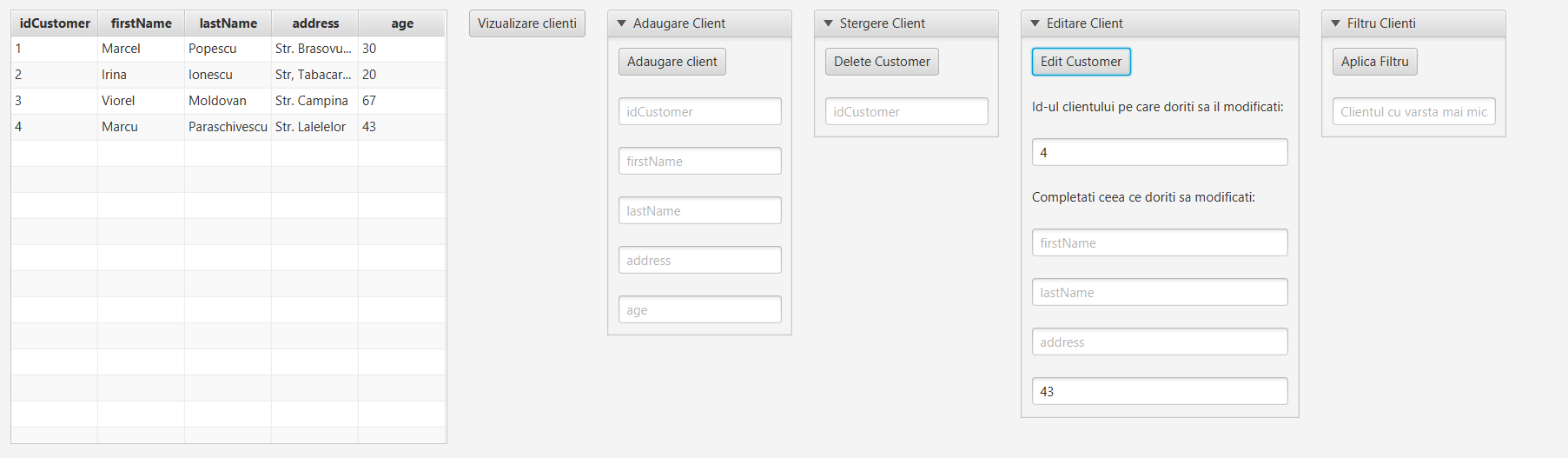
* Adaugare



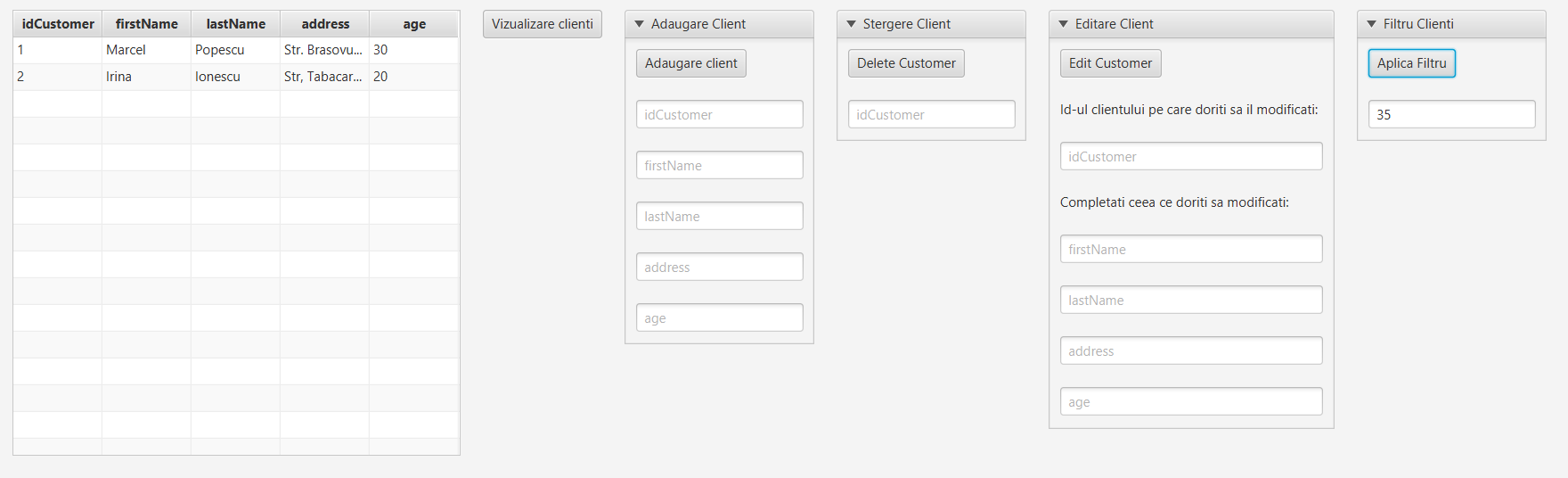
* Stergere



* Editare



* Filtrare



1. Concluzii

O concluzie foarte importanta, a acestui proiect, consta in importanta descompunerii problemei in subprobleme si usurinta cu care o problema poate fi rezolvata in acest mod. De asemenea este foarte importanta realizarea unei analize amanuntite si formarea unei idei generale asupra problemei inaintea de a incepe implementarea ei.

Din acest proiect am invatat lucruri precum:

* Imbogatirea cunostintelor in materie de interfata grafica
* Utilizarea colectiilor intr-un cadru mai amanuntit
* Importanta utilizarii proceselor de modularizare, abstractizare si incapsulare
* Crearea unei conexiuni cu o baza de date
* Complexitatea realizarii unui proiect de la 0 care, pe langa partea de rezolvarea a problemei, necesita de asemenea o interfata grafica, o unitate de testare si o documentatie laborioasa a intregului proiect
* Scrierea intr-un fisier text
* Utilizarea metodelor de reflection
* Utilizarea instrumentului Javadoc pentru crearea documentatiei claselor

Sistemul de gestionare a comenzilor unui depozit are o gama larga de posibilitati de dezvoltare ulterioara, dintre care amintim:

* Crearea unei baze de date mai complexe
* Dezvolarea sistemului la o scara mai larga
* Posibilitatea de a putea crea anumite interogari
* Imbunatatirea sistemului de filtrare
* Introducerea unor noi functionalitati, precum: schimbarea direct din aplicatie a structurii tabelelor sau crearea unei proceduri de autentificare la baza de date
* Dezvoltarea interfetei grafice

1. Bibliografie
2. <http://www.coned.utcluj.ro/~salomie/PT_Lic/>
3. <http://users.utcluj.ro/~igiosan/teaching_poo.html>
4. <https://www.draw.io/>
5. <https://docs.oracle.com/javase/8/javase-clienttechnologies.htm>