

NUME : COSTEA PAULA MARIA

GRUPA : 302210

PROFESOR LABORATOR : DAN MITREA

**ASSIGNMENT 3**

**ORDER MANAGEMENT**

Cuprins

[1 . Obiective 3](#_Toc70286381)

[2 . Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc70286382)

[3 . Proiectare 4](#_Toc70286383)

[4 . Implementare 5](#_Toc70286384)

[5.Rezultate 8](#_Toc70286385)

[6.Concluzii 11](#_Toc70286386)

[7.Bibliografie 11](#_Toc70286387)

# 1 . Obiective

Obiectivul principal al acestei teme este de a implementa un program Java , care comunica cu baza de date si care realizeaza operatii cu baza de date ( operatii precum inserarea , stergerea , editarea , introducerea intr-o lista a tuturor elementelor ) . O baza de data este o colectie organizata de informatii . Cand bazele de date sunt mai complexe , ele sunt deseori implementate utilizand tehnici formate de proiectare si de modelare .

Sistemul de manangement al bazei de date este un softwer ce interactioneaza cu utilizatorii , cu aplicatiile , si cu baza de date in sine pentru a extrage si analiza informatiile .

# 2 . Analiza problemei, modelare, scenarii, cazuri de utilizare

👉 Analiza :

Principalul scop este de a implementa un program original care executa corect operatiile specifice bazei de date , utilizand paradigmele Programarii Orientate pe Obiect . Pentru a realiza acest lucru , trebuie sa ne conectam la baza noastra de date , sa implementam operatii de baza cu baza de date , pentru a putea introduce in aceasta clienti , produse , comenzi si facturi , si multe alte operatii ( stergere, listare , editare ) .

De asemenea , trebuie sa folosim anumite tehnici pentru a crea o metoda care primeste o lista de obiecte si genereaza antetul tabelului , extragand proprietatile obiectului si apoi populeaza tabelul cu valorile elementelor din lista .

👉 Modelarea :

Primul pas pe care trebuie sa il facem pentru a putea implementa cu succes programul, este sa ne conectam la baza de date . Aceasta conectare s-a realizat in pachetul “ Conexiune ” , in clasa „ConnectionFactory“ .

Apoi , trebuie sa adaugam in pachetul „Model” , cate o clasa pentru fiecare tabel pe care il avem in baza de date ( atributele fiecarei clase trebuie sa fie exact de acelasi timp ca si coloanele din tabelele bazei de date ).

Mai departe , in pachetul “ Dao ” , in clasa „AbstractDAO” , am implementat operatiile bazei de date ( insert, delete, edit, list, findById ) generic, urmand ca, „ClientDao”, „ProductDao” si „OrderDao” sa o mosteneasca pe aceasta.

Apoi , in pachetul “ Bll “ , am implementat o clasele „ClientBll”, „ProductBll” si „OrderBll” , unde am introdus metodele care tin de logica programului ( metode in care se apeleaza de fapt metodele care executa interogarile dorite dar din pachetul „Dao” ) .

In continuare, in pachetul “ Prezentare “ , am implementat 5 clase. Prima clasa este „ClientView” care reprezinta fereastra pentru operatiile care se vor executa pe tabela „Client”. Urmatoarele doua sunt „ProductView” si „OrderView”, care reprezinta ferestrele pentru operatiile care se vor executa pe tabela „Product”, respectiv „Order”. Urmatoarea clasa este „View” care este mai mult un „meniu” de unde putem selecta pe care tabela dorim sa afectuam operatii. Si nu in ultimul rand, clasa “ Controller “, reprezinta clasa care traduce interacțiunile utilizatorului cu vederea în acțiuni pe care le va executa ‘modelul’.

In final, in pachetul „Incepe”, am implementat doua clase: clasa „Reflection”, care contine metoda pentru generarea unui tabel cu datele dintr-o tabela si clasa „Main”, care porneste executia programului.

👉 Scenarii

Daca programul ruleaza corect, la inceput se va deschide o interfata grafica de unde utilizatorul poate selecta pe care dintre cele trei tabele doreste sa efectueze operatii. Dupa selectare, se va deschide din nou, o alta fereastra de unde se poate alege operatia care se doreste aplica asupra tabelei dorite. Spre deosebire de inserarea in tabelele „Client” si „Product”, la tabela „Order” a fost necesara o validare, astfel ca se putea introduce o noua comanda doar in cazul in care: id-ul clientului era unul valid (era inregistrat in tabela „Client”, id-ul produsului era unul valid (de asemenea, trebuia sa fie inregistrat in tabela „Product”) iar cantitatea de produs dorita trebuia sa fie mai mica sau cu egala cu cantitatea de produs disponibila.

In cazul generarii comenzii , se scriu de asemenea intr-un fisier text datele referitoare la comanda inregistrata anterior.

👉 Cazuri de utilizare

In timpul rularii programului , se vor putea urmari rezultatele operatiilor efectuate apasand butonul „ViewAll...” ( specific penru fiecare tabela/clasa ). In cazul in care intr-o comanda cantitatea de produs dorita este mai mare decat cantitatea de produs existenta, va aparea un mesaj de eroare : „Nu avem suficienta cantitate de produs!” . Se va genera , la sfarsitul programului, o factura , pentru fiecare client care a dat o comanda valida , care contine numele clientului , numele produsului, cantitatea si pretul total pe care trebuie sa il achite.

# 3 . Proiectare

Pentru a respecate paradigmele Programarii Orientate pe Obiect , am impartit programul in mai multe pachete care contin, de asemenea, mai multe clase .

Primul pachet implementat este pachetul “ Model “ , care contine 4 clase ( cate o clasa pentru fiecare tabela din baza de date ):

⮊ Clasa “Client” : contie urmatoarele atribute si metode :

* id , nume , adresaClient;
* getters si setters pentru fiecare atribut;
* un constructor cu 2 parametrii ( nume , adresaClient );
* un constructor cu 3 parametrii ( id, nume, adresaClient);
* un constructor fara parametrii.

⮊ Clasa “ Product “ : contine urmatoarele atribute si metode :

* id, nume, cantitate, pret;
* getters si setters pentru fiecare atribut;
* un constructor cu 3 parametrii ( nume, cantitate, pret )
* un constructor fara parametrii.

⮊ Clasa “ Comanda “ : contine urmatoarele attribute si metode :

* id, cantitate, idClient, idProduct;
* getters si setters pentru fiecare atribut;
* un constructor cu 3 parametrii ( cantitate , idClient , idProduct ).
* un constructor fara parametrii.

In al doilea pachet ( “ Conexiune “ ) am implementat clasa “ ConnectionFactory “ . Aici am declarat metode prin care am realizat conectarea dintre baza noastra de date si programul Java .

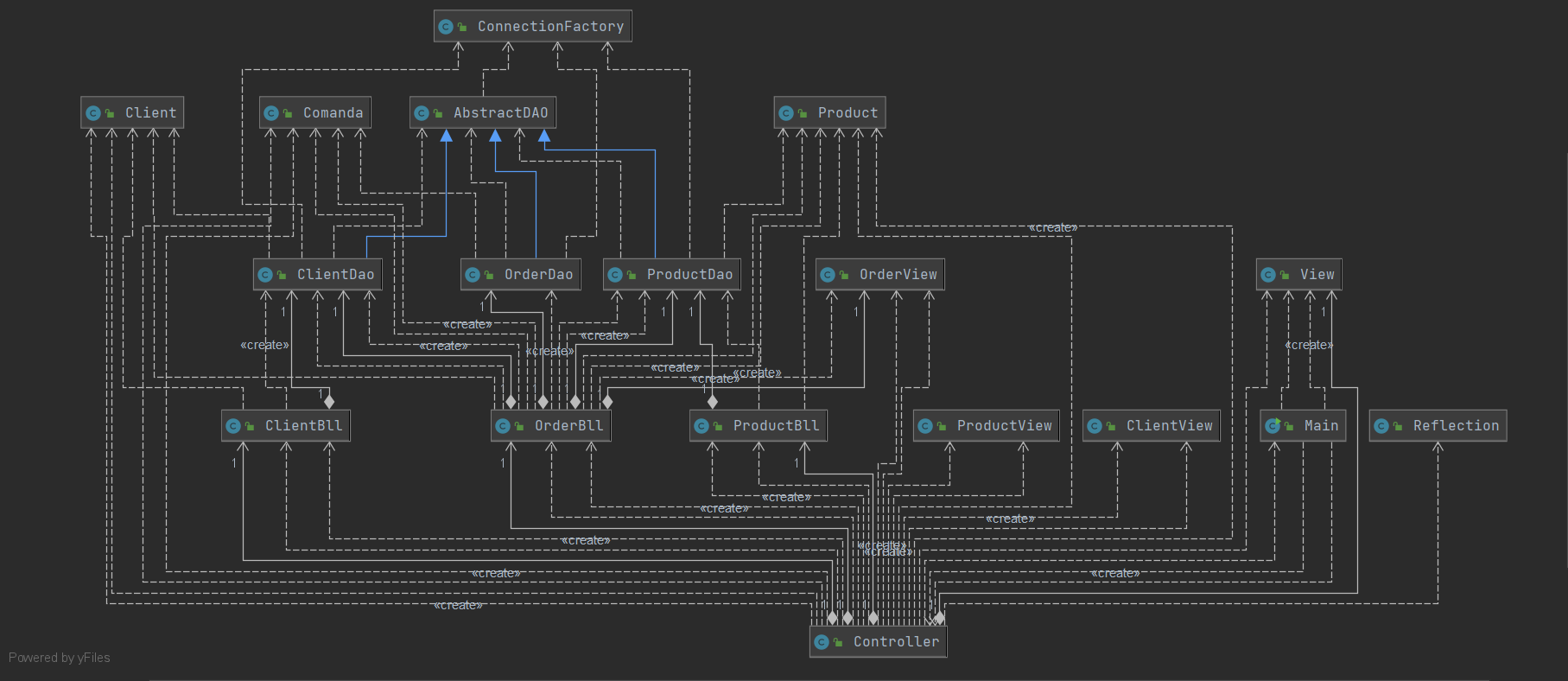
Mai apoi , in pachetul “ Dao “ , am implementat clasele: „AbstractDao” „ClientDao”, „ProductDao” si „OrderDao” . In clasa „AbstractDao”, se gasesc metode care realizeaza operatii cu baza noastra de date : inserare . stergere , editare si listare . Aceste metode sunt realizate in asa fel incat sa poata sa fie aplicate pe oricare dintre cele 3 tabele. Clasele „ClientDao”, „ProductDao” si „OrderDao” mostenesc aceasta clasa pentru a-i putea utiliza metodele. Pe langa acestea, se mai gaseste si o metoda findById . Aceasta returneaza un client, o comanda, respectiv un produs , cu id-ul egal cu cel al id-ului dat ca parametru .

In continuare , avem pachetul “ Bll ” , care contine clasele „ClientBll”, „ProductBll”, si „OrderBll” . Aici intalnim metode care alcatuiesc logica programului ( “insertClientB “ , “ deleteClientB “ , “updateClientB“ , “ updateClientAdresaB “ , “ viewAllClientsB “ , “ findClientByIdB “ , “insertProductB “ , “ deleteProductB “ , “updateProductB“ , “ updateProductCantitateB “ , “ updateProductPretB “ , “ findProductByIdB “ , “ viewAllProductsB “,“insertOrderB “ , “ deleteOrderB “ , “ validateOrder“ , “viewAllOrdersB “ , “ findOrderByIdB “ ) .

Urmatorul pachet pe care il voi prezenta este pachetul “ Prezentare “ , unde am implementat 5 clase. Prima clasa este „ClientView” care reprezinta fereastra pentru operatiile care se vor executa pe tabela „Client”. Urmatoarele doua sunt „ProductView” si „OrderView”, care reprezinta ferestrele pentru operatiile care se vor executa pe tabela „Product”, respectiv „Order”. Urmatoarea clasa este „View” care este mai mult un „meniu” de unde putem selecta pe care tabela dorim sa afectuam operatii. Si nu in ultimul rand, clasa “ Controller “: aceasta este clasa care se ocupa de realizarea „evenimentelor” in momentul in care se apasa pe un anumit buton. Aici, pentru un ascultator adaugat se realizeaza un anumit eveniment al carei rezultat este afisat pe ecran.

Ultimul pachet, „Incepe”, este pachetul care contine clasa „Main” din care porneste executia programului.

Diagrama UML :



# 4 . Implementare

⮊ Pachetul “ Model “

👉 Clasele pachetului : “ Client “ , “ Product “ , “Comanda “ , avand fiecare getters si setters pentru fiecare din atributele lor .

⮊ Pachetul “ Conexiune “

👉 Clasa “ ConnectionFactory “ : aceasta clasa contine un obiect cu o singura instanta ( ceea ce inseamna ca putem instania un singur obiect de acest tip – ne putem conecta la o singura baza de date la un moment dat ) . Aceasta clasa contine metode pentru creearea unei conexiuni , oferind o conexiune activa si , in final , inchizand conexiunea , Statement – ul si ResultSet – ul .

⮊ Pachetul “ Dao “

Clasa “ AbstractDao“ : 🞠 metoda “ findById (int id ) “ : Metoda gaseste elementul din lista totala , dupa un id dat ca parametru .

🞠 metoda “ insert ( T t ) “ : Metoda introduce in baza de date un element “ t “ ,dat ca parametru.

🞠 metoda “ delete ( int id ) “ : Metoda sterge elementul cu id-ul „id”, dat ca parametru, din baza de date.

🞠 metoda “ update ( String nume, int id ) “ : Metoda modifica atributele elementului din baza de date, dupa valoarea data ca parametru.

🞠 metoda “ viewAll ( ) “ : Metoda gaseste toate elementele dintr-o tabela.

🞠 metoda “ createSelectQuery ( String field ) “ : Metoda creeaza query-ul pentru selectarea anumitor elemente din tabela.

🞠 metoda “ createInsertQuery ( String field ) “ : Metoda creeaza query-ul pentru inserarea unui element in tabela.

🞠 metoda “ createDeleteQuery ( String field ) “ : Metoda

creeaza query-ul pentru stergerea unui element din tabela.

🞠 metoda “ createViewAllQuery ( ) “ : Metoda creeaza query-ul pentru selectarea tuturor elementelor dintr-o tabela.

🞠 metoda “ createUpdateQuery ( String coloana, String field2 ) “ : Metoda creeaza query-ul pentru editarea unui element din tabela.

Clasa “ClientDao” : aceasta clasa mosteneste clasa “ AbstractDao” si preia toate metodele din aceasta. Pe langa acestea, se mai gaseste si metoda „updateAdresa ( String adresa, int id) ” care modifica adresa clientului cu id-ul „ id ”.

Clasa “ProductDao” : aceasta clasa mosteneste clasa “ AbstractDao” si preia toate metodele din aceasta. Pe langa acestea, se mai gasesc si metodele „updateCantitate ( int cantitate, int id) ”, care modifica cantitatea produsului cu id-ul „ id ” si „ updatePret (int pret, int id )” , care modifica pretul produsului cu id-ul „id”.

Clasa “OrderDao” : la fel ca si clasele mentionate mai sus, si aceasta clasa mosteneste clasa “AbstractDao” si preia toate metodele din aceasta.

⮊ Pachetul “ Bll “

👉 clasa “ ClientBll “ : 🞠 metoda “findClientById ( ) ” : Aceasta metoda apeleaza metoda din ClientDao pentru cautarea unui client din baza de date prin intermediul atributului clientDao.

🞠metoda “insertClientB ( ) ”: Se apeleaza metoda din ClientDao pentru inserarea unui client in baza de date prin intermediul atributului clientDao. 🞠 metoda “ deleteClientB ( ) ”: Se apeleaza metoda din ClientDao pentru stergerea unui client din baza de date prin intermediul atributului clientDao. 🞠 metoda “ viewAllClientsB ( ) : Se apeleaza metoda din ClientDao pentru cautarea tuturor clientilor din baza de date prin intermediul atributului clientDao.

🞠 metoda “ updateClientB ( ) “ : Se apeleaza metoda din ClientDao pentru editarea numelui unui client din baza de date prin intermediul atributului clientDao.

🞠 metoda “updateClientAdresaB ( ) ” : Se apeleaza metoda din ClientDao pentru editarea adresei unui client din baza de date prin intermediul atributului clientDao.

👉 clasa “ ProductBll “ : 🞠 metoda “findProductById ( ) ” : Aceasta metoda apeleaza metoda din ProductDao pentru cautarea unui produs din baza de date prin intermediul atributului productDao.

🞠metoda “insertProductB ( ) ”: Se apeleaza metoda din ProductDao pentru inserarea unui produs in baza de date prin intermediul atributului productDao. 🞠 metoda “ deleteProductB ( ) ”: Se apeleaza metoda din ProductDao pentru stergerea unui produs din baza de date prin intermediul atributului productDao. 🞠 metoda “ viewAllPoductsB ( ) : Se apeleaza metoda din ProductDao pentru cautarea tuturor produselor din baza de date prin intermediul atributului productDao.

🞠 metoda “ updateProductB ( ) “ : Se apeleaza metoda din ProductDao pentru editarea numelui unui produs din baza de date prin intermediul atributului productDao.

🞠 metoda “updateProductCantitateB ( ) ” : Se apeleaza metoda din ProductDao pentru editarea cantitatii unui produs din baza de date prin intermediul atributului productDao.

👉 clasa “ OrderBll “ : 🞠 metoda “findOrderById ( ) ” : Aceasta metoda apeleaza metoda din OrderDao pentru cautarea unei comenzi din baza de date prin intermediul atributului orderDao.

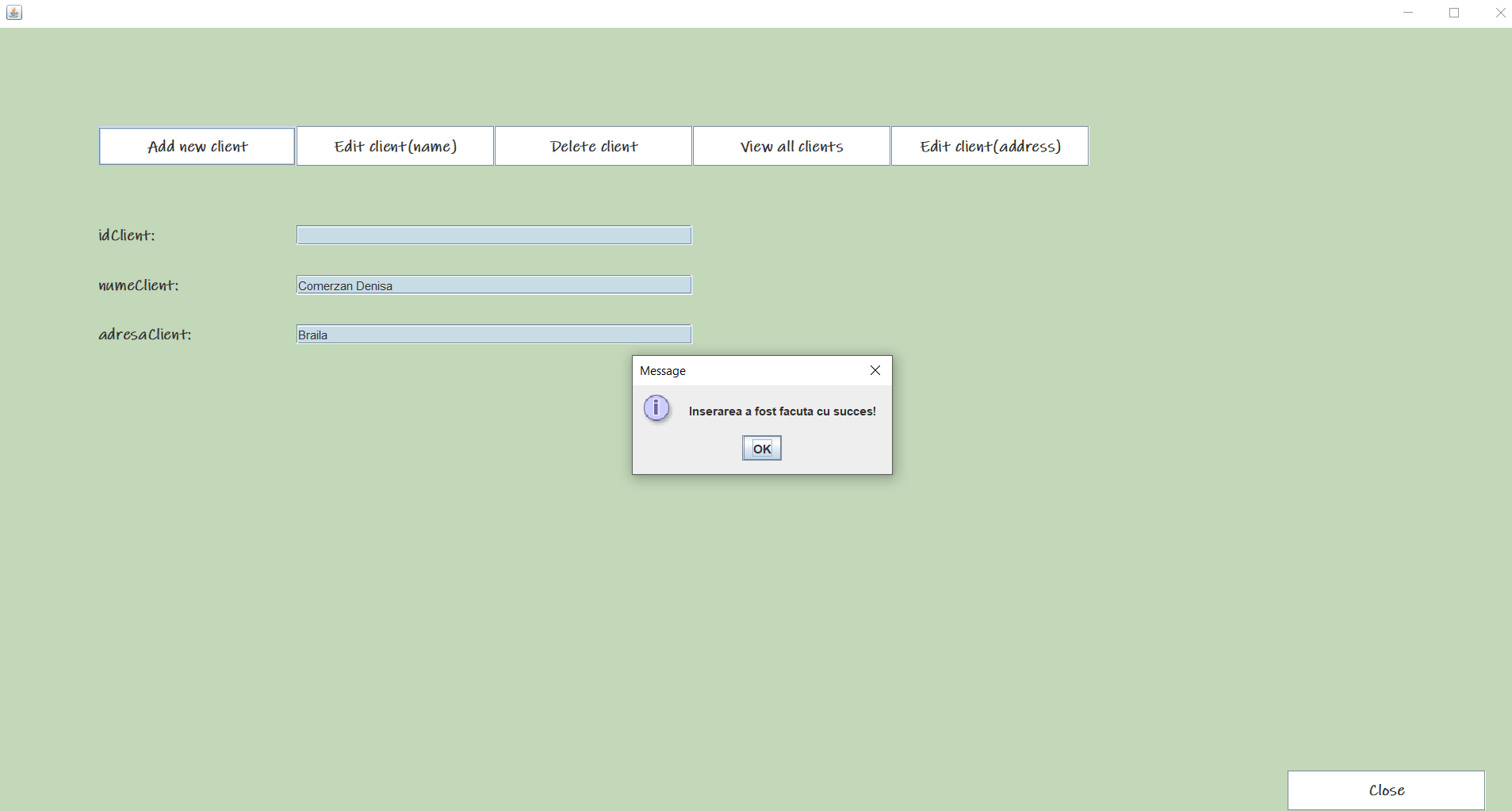
🞠metoda “insertOrder ( ) ”: In aceasta metoda , in cazul in care comanda este valida , adaugam in baza de data comanda respectiva. 🞠 metoda “ deleteOrderB ( ) ”: Se apeleaza metoda din OrderDao pentru stergerea unei comenzi din baza de date prin intermediul atributului orderDao. 🞠 metoda “ viewAllOrdersB ( ) : Se apeleaza metoda din OrderDao pentru cautarea tuturor comenzilor din baza de date prin intermediul atributului orderDao.

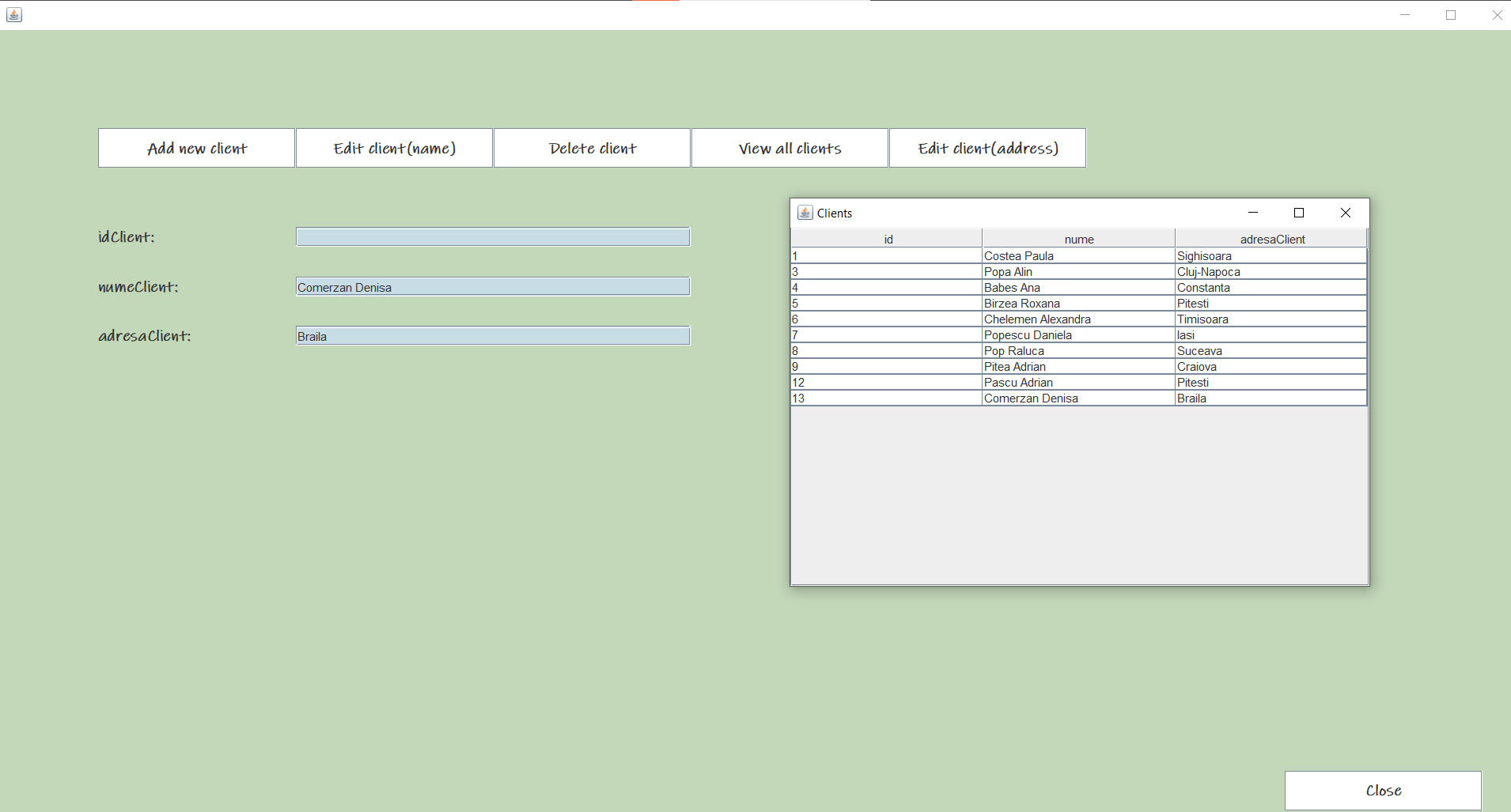
🞠 metoda “ validateOrder ( ) “ : Metoda "validateOrder" verifica daca exista vreun client cu id-ul "idClient" si vreun produs cu id-ul "idProduct",si daca cantitatea pe care vrem sa o comandam este mai mica decat cantitatea disponibila a produsului respectiv.

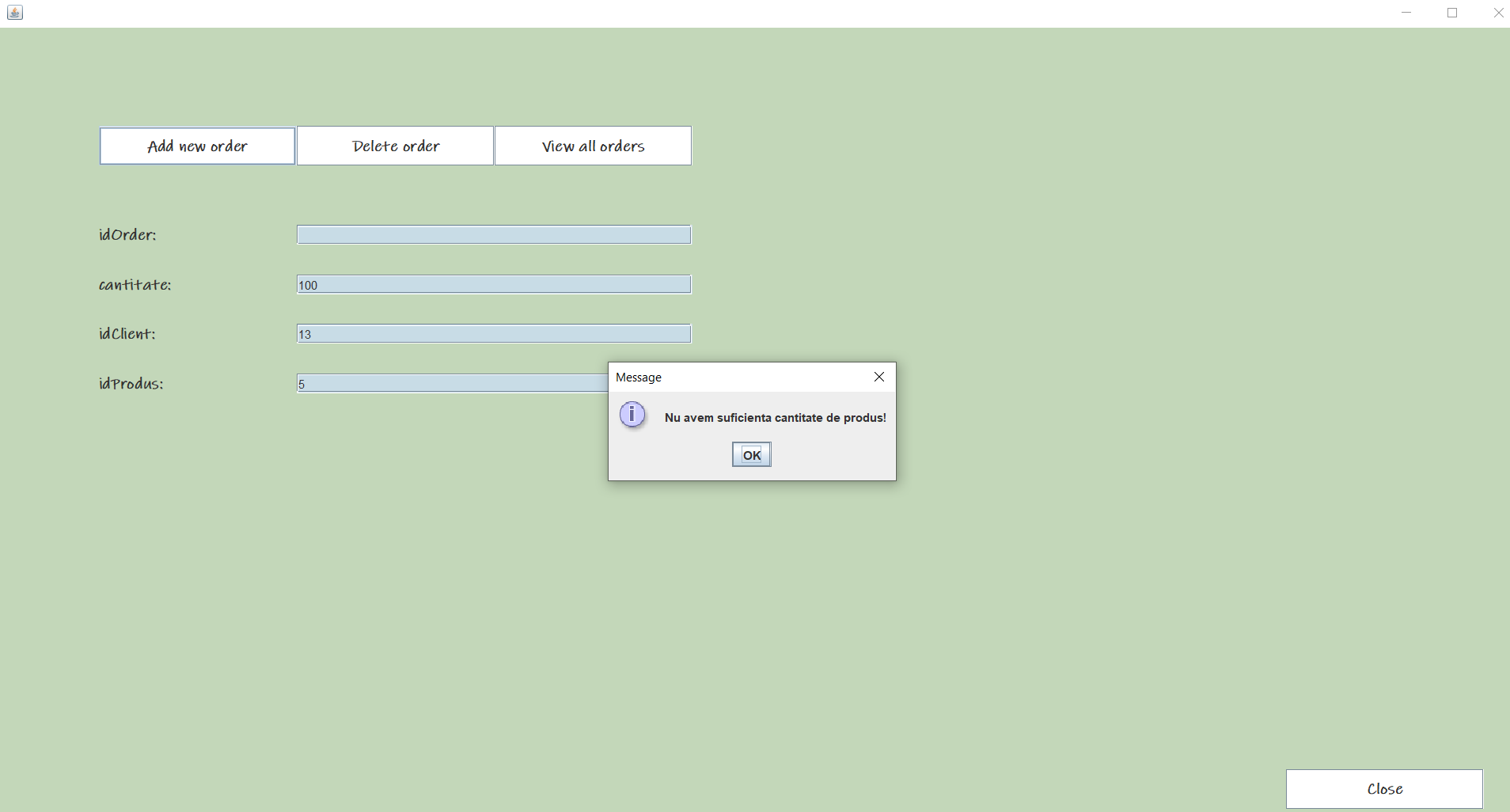
⮊Pachetul “ Prezentare “ : contine 5 clase. Prima clasa este „ClientView” care reprezinta fereastra pentru operatiile care se vor executa pe tabela „Client”. Urmatoarele doua sunt „ProductView” si „OrderView”, care reprezinta ferestrele pentru operatiile care se vor executa pe tabela „Product”, respectiv „Order”. Urmatoarea clasa este „View” care este mai mult un „meniu” de unde putem selecta pe care tabela dorim sa afectuam operatii. Si nu in ultimul rand, clasa “ Controller “, reprezinta clasa care traduce interacțiunile utilizatorului cu vederea în acțiuni pe care le va executa ‘modelul’.

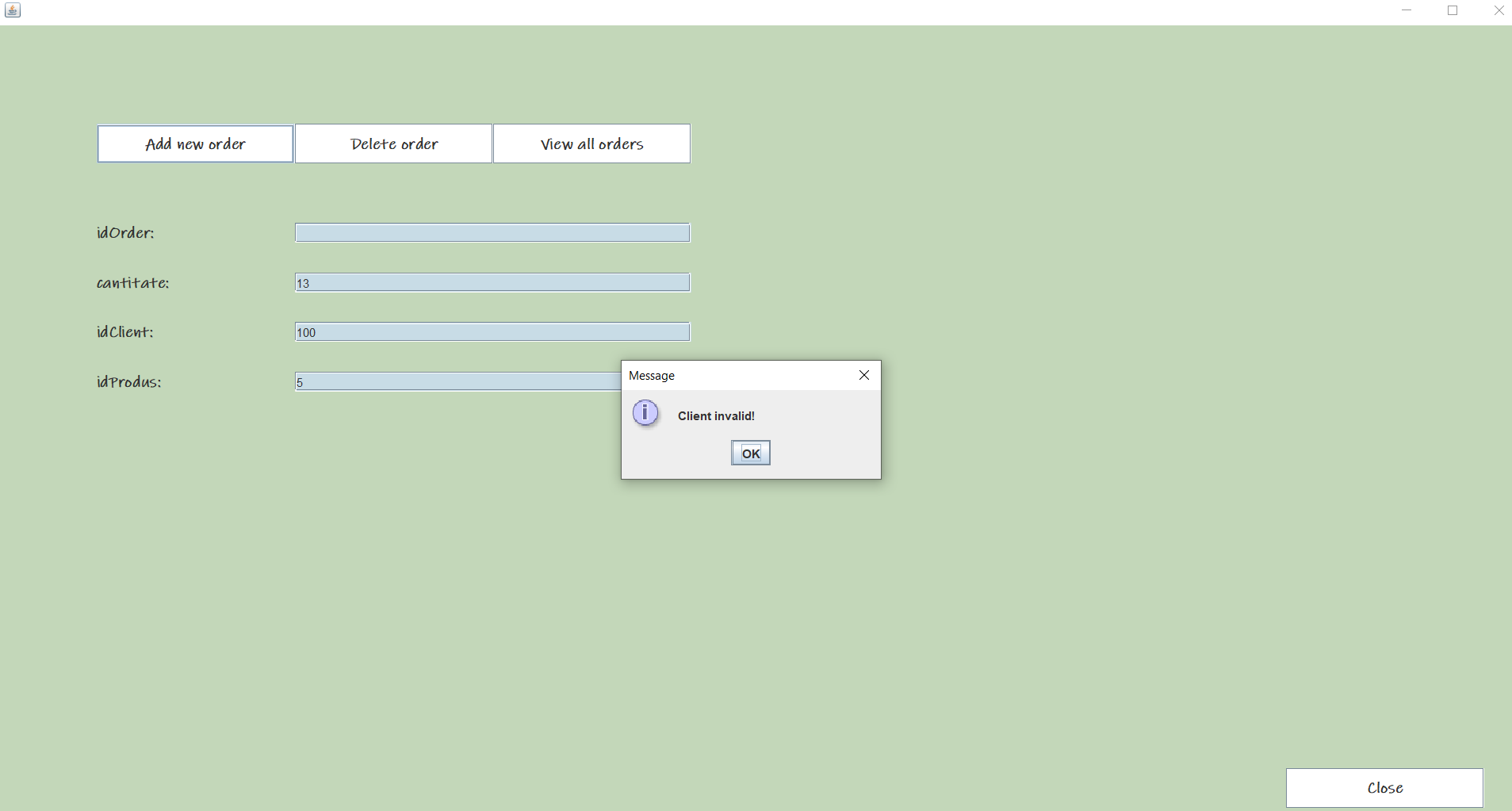
* Pachetul „Incepe”, am implementat doua clase: clasa „Reflection”, care contine metoda pentru generarea unui tabel cu datele dintr-o tabela si clasa „Main”, care porneste executia programului.

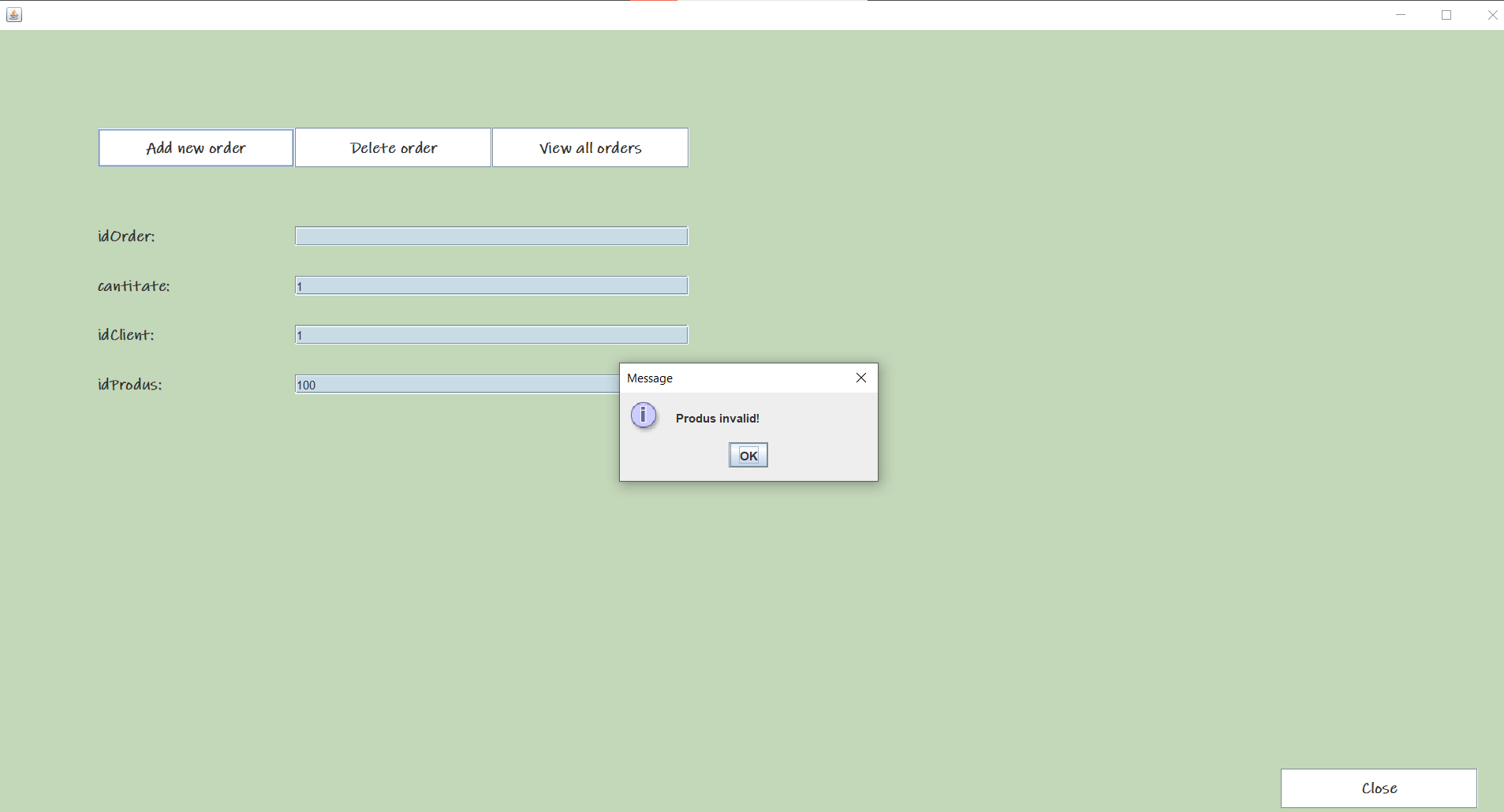
# 5.Rezultate

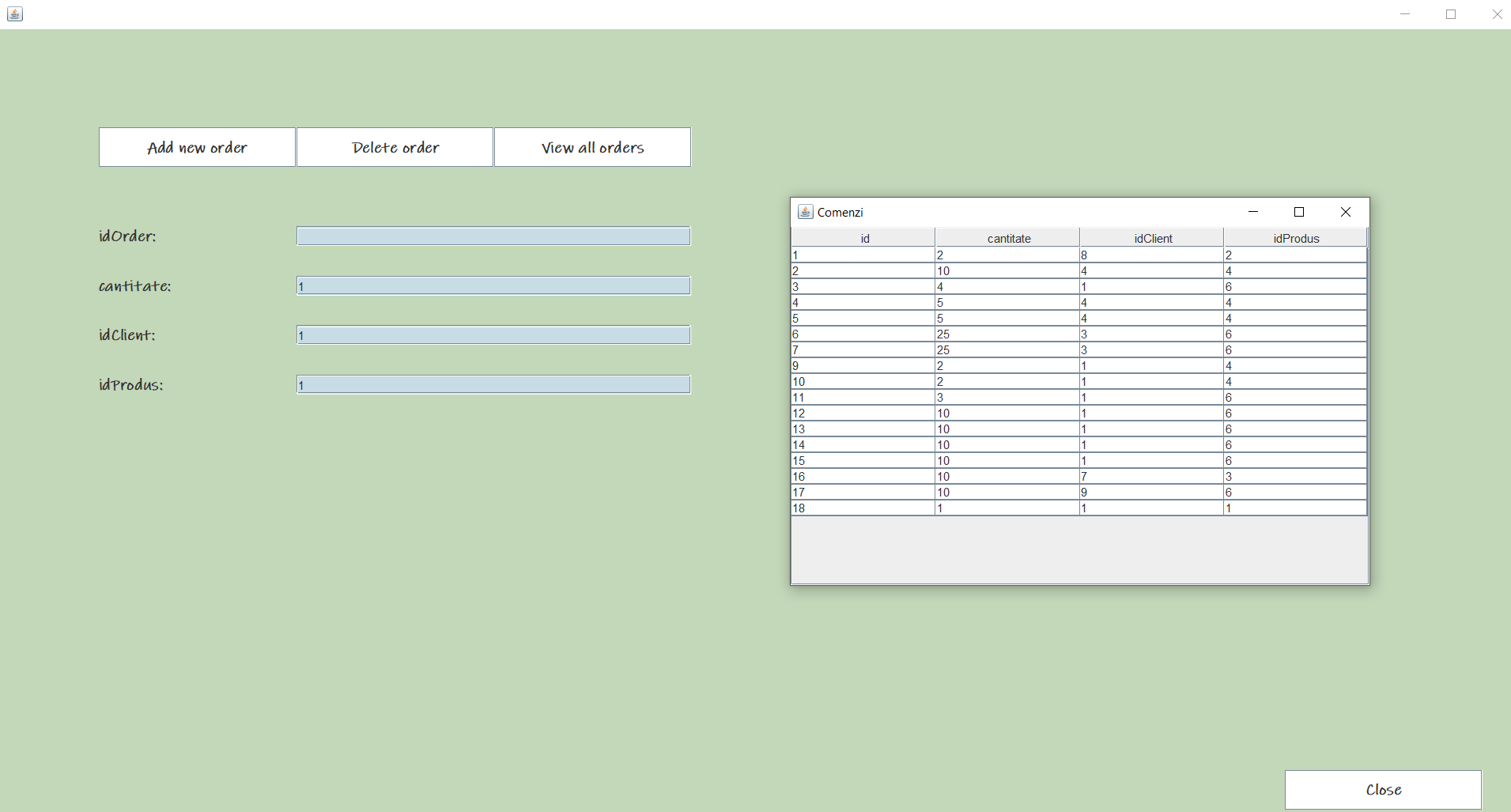


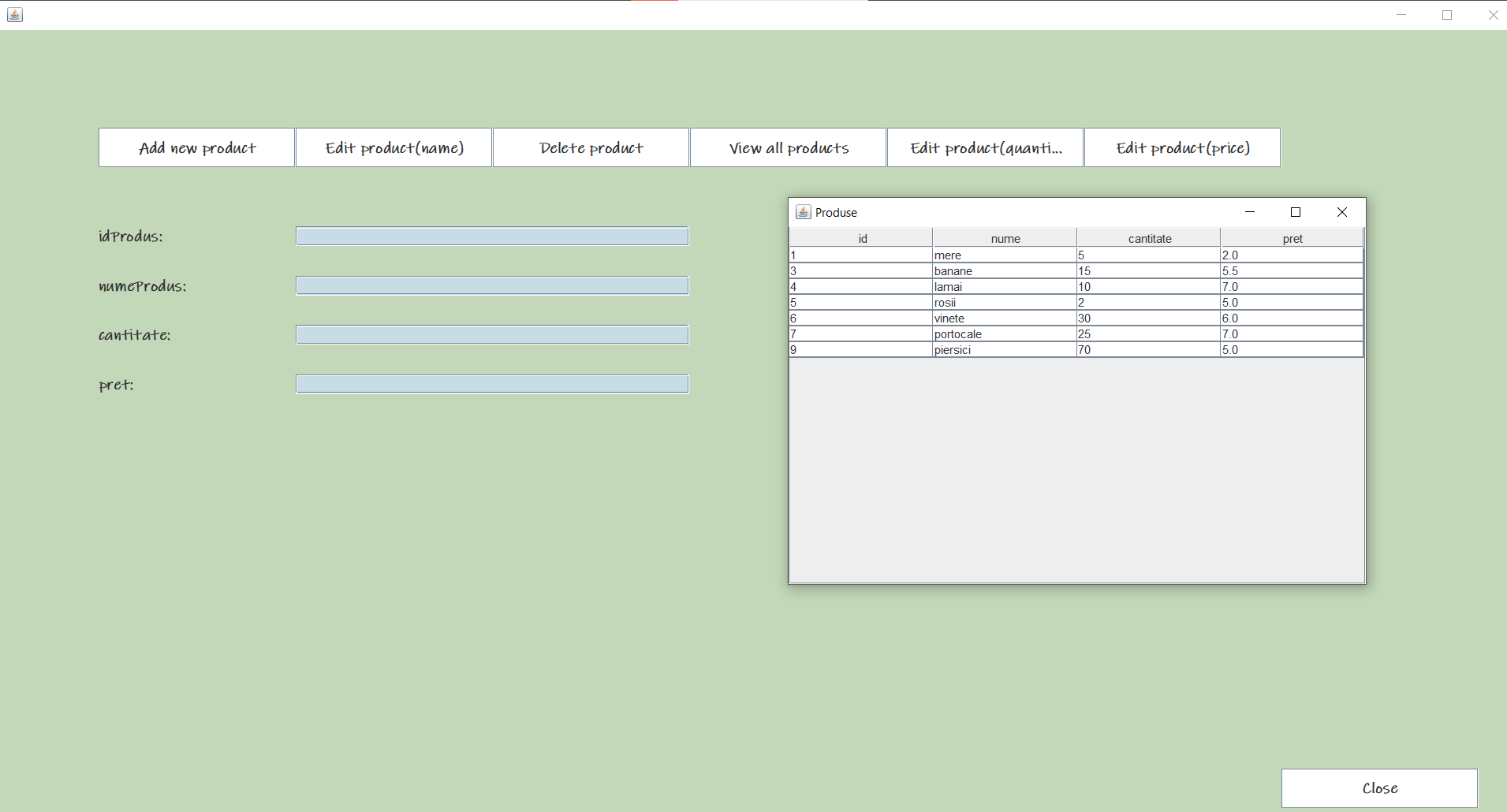












# 6.Concluzii

Aceasta tema e foarte utila , invatand prin intermediul ei sa gestionam o baza de date prin intermediul unui program Java .

# 7.Bibliografie

<https://en.wikipedia.org/wiki/Javadoc>

<https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html>

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

<http://theopentutorials.com/tutorials/java/jdbc/jdbc-mysql-create-database-example/>