

**FACULTATEA DE AUTOMATICA SI CALCULATOARE**

**DEPARTAMENTUL CALCULATOARE**

**DISCIPLINA TEHNICI DE PROGRAMARE**

**Documentatie Tema 3**

**Aplicatie pentru managementul comenzilor unor produse pentru un depozit**

Pavel Madalina Adriana

Grupa 30224

**CUPRINS**

**1. OBIECTIVUL TEMEI ……………………………………………………………………..3**

**2. ANALIZA PROBLEMEI, MODELARE, SCENARII, CAZURI DE UTILIZARE …...4**

**3. PROIECTARE (DECIZII DE PROIECTARE, DIAGRAME UML, STRUCTURI DE DATE, PROIECTARE CLASE, INTERFETE, RELATII, PACKAGES, ALGORITMI, INTERFATA UTILIZATOR) …………………………..……………………………………5**

**4. IMPLEMENTARE ………………………………………………………………………..10**

**5. REZULTATE …………………………………….………………………………………..12**

**6. CONCLUZII .……………………………………………………………………………....14**

**7. BIBLIOGRAFIE …………………………………………………………………………..14**

**1.OBIECTIVUL TEMEI**

Enuntul temei este urmatorul :

Implementati o aplicatie Order Management pentru procesarea comenzilor unor clienti pentru un depozit. Bazele de date relationale sunt folosite pentru a stoca produsele, clientii si comenzile. Mai mult, aplicatia trebuie sa foloseasca minim urmatoarele clase :

* Clase Model – reprezinta modelele de date ale aplicatiei
* Clase Business Logic – contine logica aplicatiei
* Clase Presentation – clase care contin interfata grafica
* Clase Data Access – clase care contin accesul la baza de date

Alte clase pot fi adaugate pentru o functionalitate completa a aplicatiei.

Obiectivul principal al aplicatiei este acela de a gestiona comenzile plasate de clienti in cadrul unui depozit.

Obiectivele secundare sunt urmatoarele :

* Dezvoltarea de use-case-uri si scenarii (Capitolul 2)
* Alegerea structurilor de date (Capitolul 3)
* Impartirea pe clase, utilizarea interfetelor, evidentierea relatiilor (Capitolul 3)
* Dezvoltarea algoritmilor utilizati (Capitolul 4)
* Testarea solutiei (Capitolul 5)

**2.ANALIZA PROBLEMEI, MODELARE, SCENARII, CAZURI DE UTILIZARE**

Aplicatia incearca sa imbunatateasca gestionarea datelor cu privire la produsele existente intr-un depozit, la clientii care plaseaza comenzi si cu privire la comenzile inregistrate. Aplicatia trebuie sa actualizeze toate datele modificate in urma efectuarii unei operatii asupra bazei de date. Baza de date are rolul de a stoca si a organiza informatiile, iar aplicatia are rolul de a se asigura ca informatiile introduse si stocate in baza de date sunt consistente.

Aplicatia ofera posibilitatea de a introduce date referitoare la produsele care se gasesc in depozit, de a introduce date referitoare la clientii care au plasat comenzi, de a introduce date referitoare la comenzile plasate, de a edita fiecare camp (mai putin id-ul care pentru toate este unic si odata introdus, acesta nu poate fi modificat) al tabelelor, de a vizualiza toate datele existente intr-o anumita tabela si de a sterge un anumit produs, un anumit client sau o anumita comanda, utilizand doar id-ul.

**3. PROIECTARE (DECIZII DE PROIECTARE, DIAGRAME UML, STRUCTURI DE DATE, PROIECTARE CLASE, INTERFETE, RELATII, PACKAGES, ALGORITMI, INTERFATA UTILIZATOR)**

Pentru implementarea acestei teme am realizat o conexiune la baza de date, in cadrul careia sunt stocate datele de interes ale aplicatiei. Am utilizat tehnica de programare Reflection pentru a accesa variabilele instanta ale claselor model. Operatiile efectuate asupra bazei de date sunt definite generic, la fel si crearea tabelelor si popuarea acestora (pentru vizualizarea datelor), iar specializarea se realizeaza la run-time, tot prin intermediul Reflection.

Aplicatia este structurata pe layere :

* Data Access Layer – are rolul de a efectua operatii direct asupra bazei de date, prin intermediul conexiunii la aceasta; acest layer poate accesa campurile claselor care realizeaza modelul bazei de date si este singurul layer care poate accesa aceste date si care poate comunica cu baza de date.
* Business Logic Layer – are rolul de a descrie logica aplicatiei, acesta face legatura cu interfata grafica din care se preiau datele si Data Access Layer, care realizeaza operatiile asupra bazei de date; acest layer preia datele, se asigura ca acestea sunt valide si doar in acest caz le transmite mai departe pentru a fi procesate si stocate in baza de date.
* Presentation Layer – acest layer contine interfata grafica, care este alcatuita din mai multe ferestre, necesitate impusa de logica aplicatiei; prin intermediul interfetei grafice se introduc datele in aplicatie, date ce sunt mai intai validate si mai apoi transmise layer-ului Business Logic.

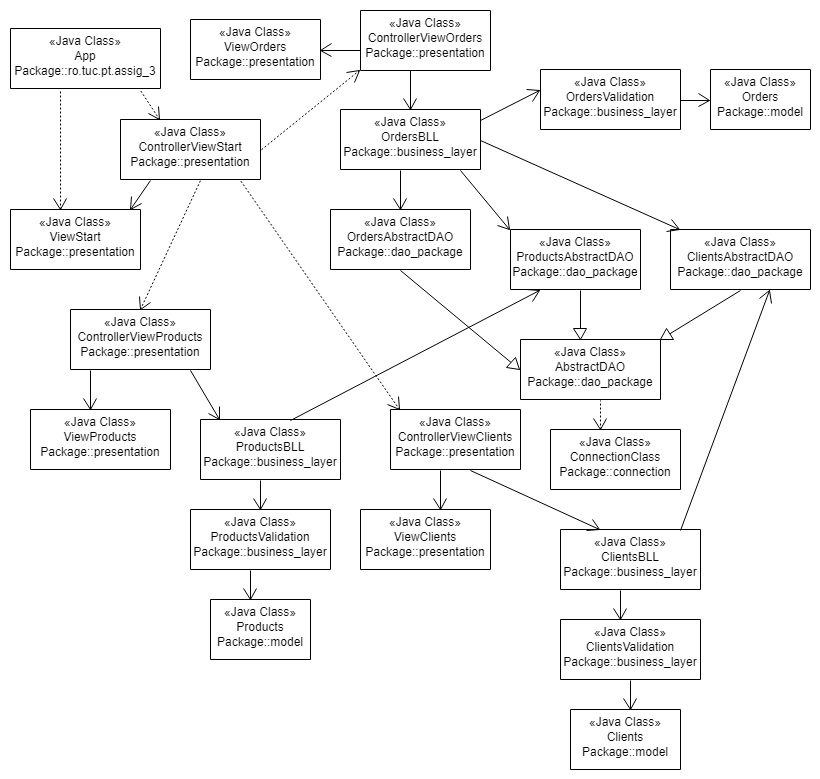
Clasele utilizate in realizarea acestei aplicatii sunt :

1. **Clasa ClientsBLL** – contine 2 variabile instanta : un obiect de tipul ClientsAbstractDAO, care are rolul de a efectua legatura dintre Business Logic Layer si Data Access Layer, si un obiect de tipul ClientsValidation, care are rolul de a valida datele referitoare la un client inainte de a se efectua o operatie pe acestea. Pe langa constructor, clasa contine si cate o metoda pentru fiecare operatie (adaugare client, editarea numelui clientului, stergerea unui client si vizualizarea tuturor clientilor din tabela Clients)**.**
2. **Clasa ClientsValidation** – contine o singura variabila instanta, de tipul Clients, care caracterizeaza un client. Pe langa constructor, clasa contine si o metoda care se ocupa cu validarea datelor referitoare la un client.
3. **Clasa OrdersBLL** - contine 4 variabile instanta : 3 obiecte de tipul ClientsAbstractDAO, ProductsAbstractDAO, respectiv OrdersAbstractDAO, care au rolul de a efectua legatura dintre Business Logic Layer si Data Access Layer, si un obiect de tipul OrdersValidation, care are rolul de a valida datele referitoare la o comanda inainte de a se efectua o operatie pe acestea. Pe langa constructor, clasa contine si cate o metoda pentru fiecare operatie (adaugare comanda, editarea clientului comenzii, editarea produsului comenzii, editarea cantitatii comenzii, stergerea unei comenzi si vizualizarea tuturor comenzilor din tabela Orders)**.**
4. **Clasa OrdersValidation** – contine o singura variabila instanta, de tipul Orders, care caracterizeaza o comanda. Pe langa constructor, clasa contine si o metoda care se ocupa cu validarea datelor referitoare la o comanda.
5. **Clasa ProductsBLL** – contine 2 variabile instanta : un obiect de tipul ProductsAbstractDAO, care are rolul de a efectua legatura dintre Business Logic Layer si Data Access Layer, si un obiect de tipul ProductsValidation, care are rolul de a valida datele referitoare la un produs inainte de a se efectua o operatie pe acestea. Pe langa constructor, clasa contine si cate o metoda pentru fiecare operatie (adaugare produs, editarea numelui produsului, editarea pretului produsului, editarea stocului produsului, stergerea unui produs si vizualizarea tuturor produselor din tabela Products)**.**
6. **Clasa ProductsValidation** – contine o singura variabila instanta, de tipul Products, care caracterizeaza un produs. Pe langa constructor, clasa contine si o metoda care se ocupa cu validarea datelor referitoare la un produs.
7. **Clasa ConnectionClass** – contine 4 constante de tipul String, care retin : driver-ul, url-ul, user-ul si parola necesare pentru conexiunea la baza de date, si o variabila statica de tipul ConnectionClass, care reprezinta unica instanta a acestei clase. Clasa contine un constructor privat si metode pentru : obtinerea conexiunii la baza de date, inchiderea conexiunii, inchiderea ResultSet-ului si inchiderea PreparedStatement-ului.
8. **Clasa AbstractDAO<T>** – contine o constanta de tipul Logger si un obiect de tipul Class<T>. Pe langa constructor, clasa contine si metode care creeaza query-urile pentru operatiile de select, select all, insert, update si delete, plus metodele care realizeaza conexiunea la baza de date si executa query-urile.
9. **Clasa ClientsAbstractDAO** – extinde clasa AbstractDAO<Clients>, pentru a inlocui parametrul generic T. Clasa nu are variabile instanta si contine un constructor gol.
10. **Clasa OrdersAbstractDAO** – extinde clasa AbstractDAO<Orders>, pentru a inlocui parametrul generic T. Clasa nu are variabile instanta si contine un constructor gol.
11. **Clasa ProductsAbstractDAO** – extinde clasa AbstractDAO<Products>, pentru a inlocui parametrul generic T. Clasa nu are variabile instanta si contine un constructor gol.
12. **Clasa Clients** – contine 2 variabile instanta : id-ul clientului (de tip int) si numele clientului (de tipul String). Clasa contine 2 constructori (unul cu parametri si celelalt gol), metode de get si set, pentru a accesa si modifica variabilele instanta din exterior, si metoda toString, utilizata pentru afisarea datelor despre un client.
13. **Clasa Orders** – contine 4 variabile instanta : id-ul comenzii, id-ul clientului, id-ul produsului si cantitatea comenzii (de tip int). Clasa contine 2 constructori (unul cu parametri si celelalt gol), metode de get si set, pentru a accesa si modifica variabilele instanta din exterior, si metoda toString, utilizata pentru afisarea datelor despre o comanda.
14. **Clasa Products** – contine 4 variabile instanta : id-ul produsului (de tipul int), numele produsului (de tipul String), pretul produsului (de tipul float) si stocul (de tipul int). Clasa contine 2 constructori (unul cu parametri si celelalt gol), metode de get si set, pentru a accesa si modifica variabilele instanta din exterior, si metoda toString, utilizata pentru afisarea datelor despre un produs.
15. **Clasa ControllerViewClients** – contine 2 variabile instanta : un obiect de tipul ViewClients, care reprezinta fereastra din UI, si un obiect de tipul ClientsBLL, care are rolul de a prelua datele si de a le transmite spre layer-ul urmator. Clasa contine un constructor si clase interne care implementeaza interfata ActionListener, clase care descriu comportamentul fiecarui buton din fereastra pentru clienti : **Clasa ActionListenerAdd, Clasa ActionListenerEdit, Clasa ActionListenerDelete, Clasa ActionListenerViewAll, Clasa ActionListenerBack.**
16. **Clasa ControllerViewOrders** – contine 2 variabile instanta : un obiect de tipul ViewOrders, care reprezinta fereastra din UI, si un obiect de tipul OrdersBLL, care are rolul de a prelua datele si de a le transmite spre layer-ul urmator. Clasa contine un constructor si clase interne care implementeaza interfata ActionListener, clase care descriu comportamentul fiecarui buton din fereastra pentru comenzi : **Clasa ActionListenerAdd, Clasa ActionListenerEdit, Clasa ActionListenerDelete, Clasa ActionListenerViewAll, Clasa ActionListenerBack.**
17. **Clasa ControllerViewProducts** – contine 2 variabile instanta : un obiect de tipul ViewProducts, care reprezinta fereastra din UI, si un obiect de tipul ProductsBLL, care are rolul de a prelua datele si de a le transmite spre layer-ul urmator. Clasa contine un constructor si clase interne care implementeaza interfata ActionListener, clase care descriu comportamentul fiecarui buton din fereastra pentru produse : **Clasa ActionListenerAdd, Clasa ActionListenerEdit, Clasa ActionListenerDelete, Clasa ActionListenerViewAll, Clasa ActionListenerBack.**
18. **Clasa ControllerViewStart** – contine o singura variabila instanta : un obiect de tipul ViewStart, care reprezinta fereastra de start a aplicatiei. Clasa contine un constructor si o clasa interna care implementeaza interfata ActionListener, clasa ce descrie comportamentul combo box-ului care selecteaza fereastra urmatoare : **Clasa ActionListenerComboBox**.
19. **Clasa ViewClients** – extinde clasa JFrame; contine ca variabile instanta obiecte de tipul JTextField pentru id-ul clientului, numele clientului (pentru adaugarea unui nou client), id-ul clientului al carui nume trebuie modificat, noul nume (pentru editarea clientului) si id-ul clientului ce se doreste a fi sters (pentru stergerea unui client existent); obiecte de tipul JButton pentru butoanele corespunzatoare operatiilor de adaugare, editare, stergere a unui client, afisarea tabelei, si butonul back; si 3 obiecte de tipul JTable, DefaultTableModel si JScrollPane pentru afisarea tabelei in interfata grafica. Clasa contine un constructor in care este creata fereastra, metode de get pentru variabilele instanta si metode utilizate pentru a adauga cate un ActionListener fiecarui buton.
20. **Clasa ViewOrders** – extinde clasa JFrame; contine ca variabile instanta obiecte de tipul JTextField pentru id-ul comenzii, id-ul clientului, id-ul produsului, cantitatea comenzii (pentru adaugarea unei comenzi), id-ul comenzii a caror date trebuie modificate, id-ul noului client, id-ul noului produs, noua cantitate (pentru editarea comenzii) si id-ul comenzii ce se doreste a fi stearsa (pentru stergerea unei comenzi existente); obiecte de tipul JButton pentru butoanele corespunzatoare operatiilor de adaugare, editare, stergere a unei comenzi, afisarea tabelei, si butonul back; si 3 obiecte de tipul JTable, DefaultTableModel si JScrollPane pentru afisarea tabelei in interfata grafica. Clasa contine un constructor in care este creata fereastra, metode de get pentru variabilele instanta si metode utilizate pentru a adauga cate un ActionListener fiecarui buton.
21. **Clasa ViewProducts** – extinde clasa JFrame; contine ca variabile instanta obiecte de tipul JTextField pentru id-ul produsului, numele produsului, pretul produsului, stocul (pentru adaugarea unui nou produs), id-ul produsului al caror date trebuie modificate, noul nume, noul pret, noul stoc (pentru editarea produsului) si id-ul produsului ce se doreste a fi sters (pentru stergerea unui produs existent); obiecte de tipul JButton pentru butoanele corespunzatoare operatiilor de adaugare, editare, stergere a unui produs, afisarea tabelei, si butonul back; si 3 obiecte de tipul JTable, DefaultTableModel si JScrollPane pentru afisarea tabelei in interfata grafica. Clasa contine un constructor in care este creata fereastra, metode de get pentru variabilele instanta si metode utilizate pentru a adauga cate un ActionListener fiecarui buton.
22. **Clasa ViewStart** - extinde clasa JFrame; contine ca variabila instanta un obiect de tipul JComboBox<String>, care permite selectarea urmatoarei ferestre ce se doreste a fi deschisa. Clasa contine un constructor in care este creata fereastra, o metoda de get pentru variabila instanta si o metoda utilizata pentru a adauga un ActionListener combobox-ului.
23. **Clasa App** – nu contine variabile instanta si nici constructori. Metoda main a acestei clase lanseaza in executie aplicatia.

Clasele prezentate la numerele 1.-6. fac parte din pachetul business\_layer, cea prezentata

a numarul 7. face parte din pachetul connection, cele prezentate la numerele 8.-11. fac parte din pachetul dao\_package, cele prezentate la numerele 12.-14. fac parte din pachetul model, cele prezentate la numerele 15.-22. fac parte din pachetul presentation, iar cea prezentata la numarul 23. face parte din pachetul ro.tuc.pt.assig\_3.

Diagrama UML este urmatoarea :



**4. IMPLEMENTARE**

Metodele implementate in fiecare clasa sunt :

1. Clasa ClientsBLL

* void addNewClient(Clients client) – adauga clientul in baza de date daca acesta este valid
* void editClientName(int id, String newName) – modifica numele unui client existent in baza de date, daca numele nu depaseste mai mult de 30 de caractere.
* void deleteClient(int id) – sterge un client existent in functie de id-ul acestuia (unic)
* Object[][] viewAll() – returneaza tuplele tabelei Clients pentru a putea fi afisata in JTable.

1. Clasa ClientsValidation

* boolean isValid() – returneaza true daca id-ul clientului este pozitiv si numele nu depaseste 30 de caractere; in caz contrar, returneaza false.

1. Clasa OrdersBLL

* void addNewOrder(Orders order) – adauga comanda in baza de date daca aceasta este valida
* void editOrderClient (int id, int newClientID) – modifica id-ul clientului corespunzator comenzii, daca id-ul exista
* void editOrderProduct (int id, int idProduct) – modifica id-ul produsului corespunzator comenzii, daca id-ul exista, si reface stocul produsului vechi
* void editOrderQuantity (int id, int newQuantity) – modifica cantitatea comenzii si actualizeaza stocul produsului
* void deleteOrder(int id) – sterge o comanda existenta in functie de id-ul acesteia (unic) si actualizeaza stocul produsului
* Object[][] viewAll() – returneaza tuplele tabelei Orders pentru a putea fi afisata in JTable.

1. Clasa OrdersValidation

* boolean isValid() – returneaza true daca id-ul comenzii, id-ul clientului, id-ul produsului si cantitatea sunt pozitive; in caz contrar, returneaza false.

1. Clasa ProductsBLL

* void addNewProduct(Products product) – adauga produsul in baza de date daca acesta este valid, daca exista deja, doar actualizeaza stocul
* void editProductName(int id, String newName) – modifica numele produsului, daca acesta nu depaseste 30 de caractere
* void editProductPrice(int id, float newPrice) – modifica pretul produsului, daca acesta este un numar pozitiv
* void editProductStock(int id, int newStock) – modifica stocul produsului, daca acesta este un numar pozitiv
* void deleteProduct(int id) – sterge un produs existent in functie de id-ul acestuia (unic)
* Object[][] viewAll() – returneaza tuplele tabelei Products pentru a putea fi afisata in JTable.

1. Clasa ProductsValidation

* boolean isValid() – returneaza true daca id-ul produsului, pretul si stocul sunt pozitive si daca numele produsului nu depaseste 30 de caractere; in caz contrar, returneaza false.

1. Clasa ConnectionClass

* static Connection getConnection() – returneaza conexiunea la baza de date
* static void closeConnection(Connection con) – inchide conexiunea la baza de date
* static void closeResultSet(ResultSet result) – inchide obiectul de tip ResultSet
* static void closePreparedStatement(PreparedStatement statement) – inchide obiectul de tip PreparedStatement

1. Clasa AbstractDAO<T>

* String createSelectQ(String field) – returneaza un String care reprezinta query-ul pentru select
* String createSelectAllQ() – returneaza un String care reprezinta query-ul pentru select all
* String createInsertQ() – returneaza un String care reprezinta query-ul pentru insert
* String createUpdateQ(String field1, String field2) – returneaza un String care reprezinta query-ul pentru update
* String createDeleteQ(String field) – returneaza un String care reprezinta query-ul pentru delete
* T findByID(int id) – returneaza obiectul de clasa T al carui id este dat
* private List<T> createObjects(ResultSet resultSet) – returneaza lista de obiecte pe care le extrage din ResultSet
* void addObject(T tObject) – creeaza un obiect utilizand insert
* void updateField(String fieldNewValue, String fieldCond, Object o1, Object o2) – realizeaza un update pe un obiect utilizand query-ul pentru update
* void deleteObjectByID(int id) – sterge obiectul al carui id este dat utilizand query-ul pentru delete
* List<T> findAll() – returneaza lista obiectelor care se gasesc in ResultSet, dupa efectuarea query-ului de select all

1. Clasa ActionListenerAdd

* void actionPerformed(ActionEvent e) – in functie de controller-ul in care se afla clasa interna, metoda preia datele din interfata si apeleaza metoda de adaugare

1. Clasa ActionListenerEdit

* void actionPerformed(ActionEvent e) – in functie de controller-ul in care se afla clasa interna, metoda preia datele din interfata si apeleaza metoda de editare

1. Clasa ActionListenerDelete

* void actionPerformed(ActionEvent e) – in functie de controller-ul in care se afla clasa interna, metoda preia datele din interfata si apeleaza metoda de stergere

1. Clasa ActionListenerViewAll

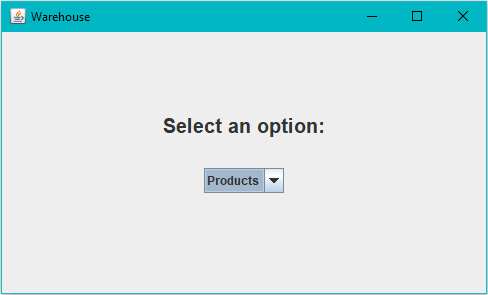
* void actionPerformed(ActionEvent e) – in functie de controller-ul in care se afla clasa interna, metoda preia datele din interfata si apeleaza metoda de afisare

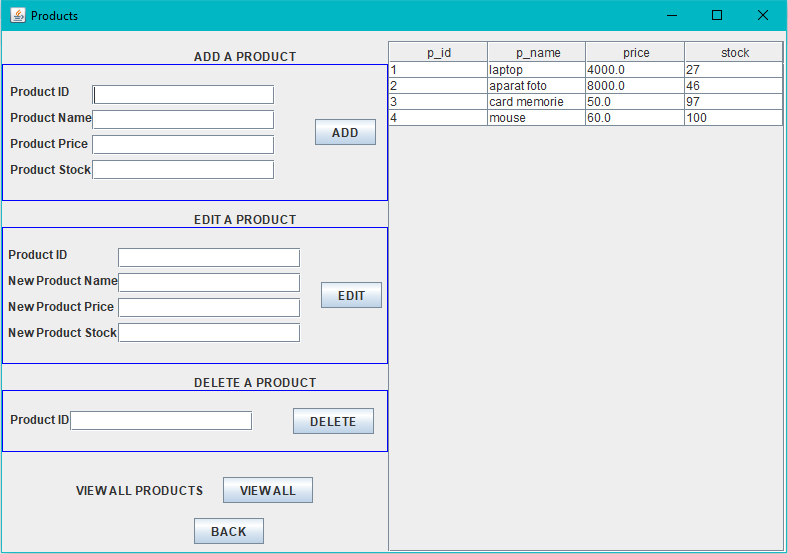
1. Clasa ActionListenerBack

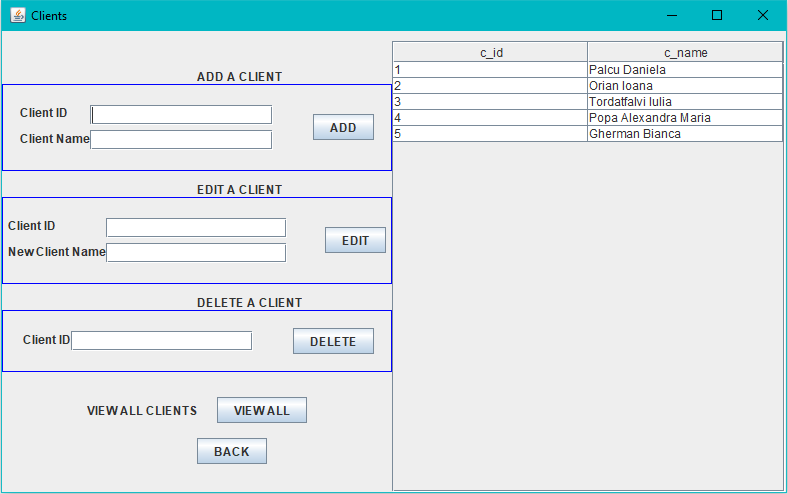
* void actionPerformed(ActionEvent e) – realizeaza revenirea in fereastra principala

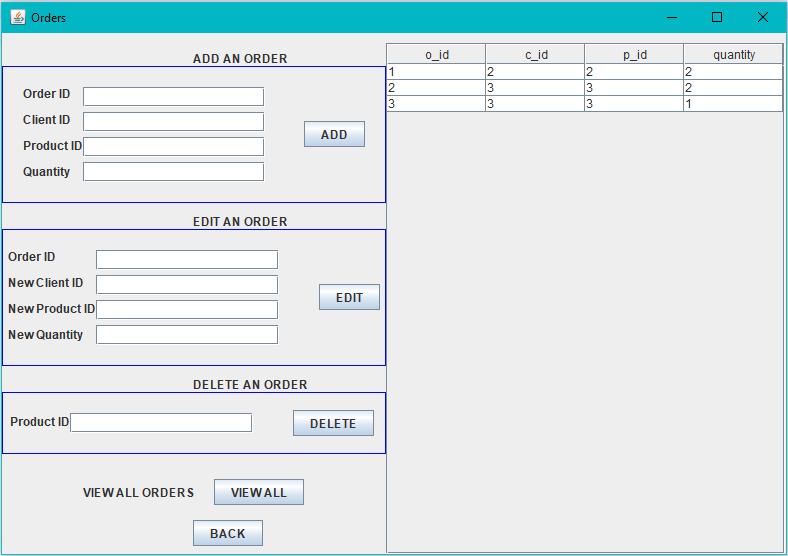
**5. REZULTATE**

Interfata utilizator este urmatoarea :









**6. CONCLUZII**

In concluzie, aceasta tema m-a ajutat sa imi imbunatatesc cunostiintele legate de conceptele de Reflection si baze de date.

Printre posibilitatile de dezvoltare ulterioara se numara : extinderea bazei de date prin adaugarea mai multor tabele, adaugarea unor metode care sa realizeze totalul unor campuri sau metode care sa filtreze informatiile.

**7. BIBLIOGRAFIE**

* <https://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/>
* <http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW3_Tema3/Tema3_HW3_Indications.pdf>