**Tehnici de programare fundamentale**

**Tema 3**

**Aplicatia OrderManagement**

Pentek Tamas

Grupa: 30228

**1. Obiectivul temei**

Obiectul principal al temei este crearea unei aplicatii pentru a procesa comenzi facute de clienti la un depozit, la o companie. Aceasta aplicatie trebuie sa foloseasca baza de date, trebuie sa aiba o interfata grafica prin care utilizatorul poate sa faca niste operatii cu clienti, cu produse. Langa aceasta utilizatorul poate sa faca o comanda alcatuita din mai multe produse, la fiecare comanda aplicatie trebuie sa genereze o factura cu produsele comandate.

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

Utilizatorul poate sa aleaga dintre mai multe operatii. Cand utilizatorul deschide aplicatie are posibilitatea de a face: operatii cu clienti de exemplu a adauga un nou client sau a sterge un client; operatii cu produse de exemplu adaugarea unui nou produs sau actualizarea pretului la mai multe produse; o comanda cu mai multe produse, poate sa aleaga clientul pentru care face comanda si la sfarsit poate sa genereze o factura cu produse si cu suma totala.

Utilizatorul este ajutat prin mai multe guidelineuri, de exemplu: daca introduce o valoare gresita apare un mesaj de eroare, daca comanda contine mai multe produse decat sunt pe stoc, apare o fereastra in care scrie exact cantitatile fiecarui produs. Un alt feedback pentru utilizator este in momentul finalizarii comenzilor: daca comanda este valida si se poate efectua, apare un mesaj de informare cu textul corespunzator.

Utilizatorul poate sa aleage din 3 operatii mai complexe: operatii cu clienti, operatii cu produse, aduagarea unei comanda. Operatii cu clienti si cu produse contin mai multe operatii mici la randul lor.

In ceea ce urmeaza, sunt prezentate cateva operatii ca o diagrama use-case si prin descriere use-case sub forma unei liste, pe care daca utilizatorul urmareste va efectua operatia dorita.

**Use Case: Adaugarea unei comenzi**

**Primary Actor: utilizator**

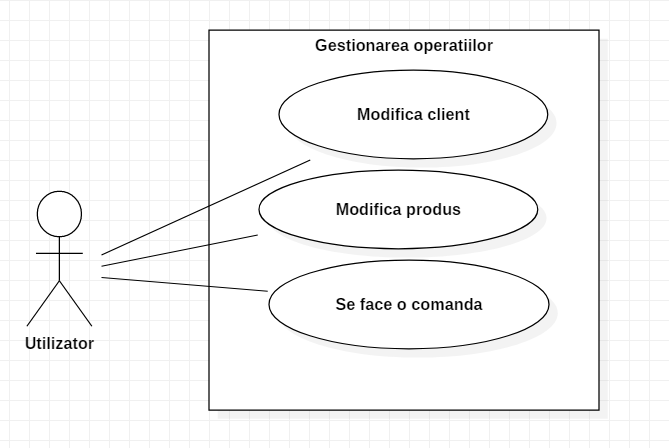
**Main Success Scenario:**

1. Utilizatorul deschide aplicatia
2. Utilizatorul alege butonul „Create an order”
3. Apare o noua fereastra, prima data utilizatorul selecteaza clientul
4. Utilizatorul adauga cu butonul „+” cate produse doreste
5. Selecteaza produsul dorit
6. Introduce cantitatea din produsul respectiv
7. Selecteaza tipul de plata
8. Utilizatorul apasa butonul „Finish!”
9. Apare o noua fereastra de informare care confirma comanda „Order Completed Successfully!”
10. Utilizatorul apasa butonul „OK”
11. Utilizatorul are posibilitatea de a vedea factura salvata pe hard disk

**Alternative Sequences:**

a.) Utilizatorul nu alege tipul de plata si apasa butonul „Finish!”:

1. Apare o noua fereastra cu un mesaj de eroare
2. Utilizatorul apasa butonul „OK”
3. Utilizatorul continua cu punctul 7. din Main Success Scenario



**Use Case: Adaugare client**

**Primary Actor: utilizator**

**Main Success Scenario:**

1. Utilizatorul deschide aplicatia
2. Utiliatorul alege butonul „Client Operations”
3. Apare o noua fereastra si utilizatorul selecteaza operatia de aduagare
4. Introduce datele clientului: ID Client, first name, last name, address, phone number, IBAN number, e-mail address
5. Utilizatorul apasa butonul „OK”
6. Programul verifica datele si adauga clientul in tabelul Client
7. Apare o fereastra noua care confirma adaugarea noului client: „New Client Inserted Successfully!”

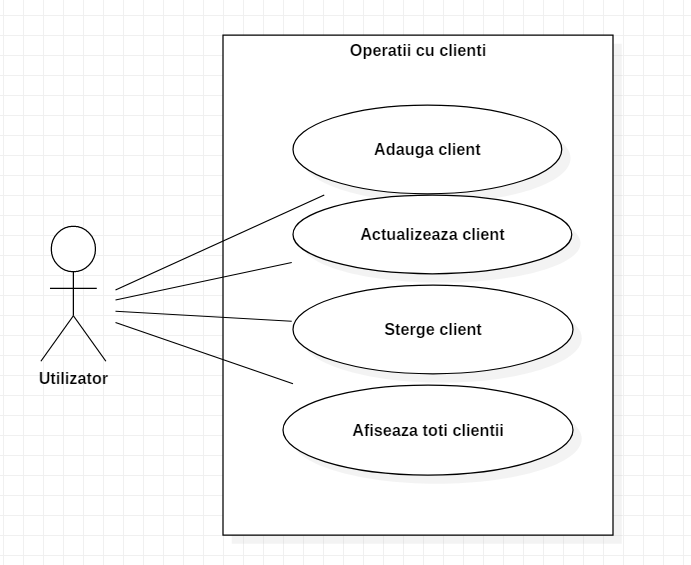
**Alternative Sequences:**

a.) Utilizatorul nu introduce nici o valoare si apasa butonul „OK”:

1. Programul afiseaza intr-o fereastra noua un mesaj de eroare
2. Utilizatorul apasa butonul „OK”
3. Utilizatorul continua cu punctul 4. din Main Success Scenario

b.) Utilizatorul introduce un ID de client care exista deja si apasa butonul „OK”:

1. Programul afiseaza intr-o fereastra noua mesajul care indica faptul ca a introdus valori dublicate
2. Utilizatorul apasa butonul „OK”
3. Utilizatorul continua cu punctul 4. din Main Success Scenario



**Use Case: Stergere produs**

**Primary Actor: utilizator**

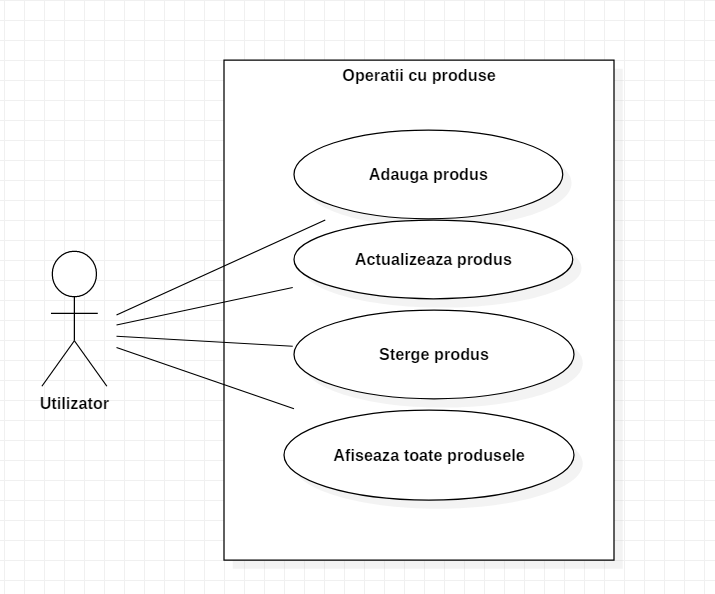
**Main Success Scenario:**

1. Utilizatorul deschide aplicatia
2. Utiliatorul alege butonul „Product Operations”
3. Apare o noua fereastra si utilizatorul selecteaza operatia stergere
4. Utilizatorul introduce conditia de stergere
5. Apasa butonul „OK”
6. Programul verifica corectitudinea conditiei si se face stergerea din tabelul Product
7. Programul afiseaza intr-o fereastra noua un mesaj de informare: „Deleted Succesfully 1 Field(s)!

**Alternative Sequences:**

a.) Utilizatorul introduce o conditie gresita si apasa butonul „OK”:

1. Programul afiseaza intr-o fereastra noua un mesaj de eroare: „Something went wrong!”
2. Utilizatorul apas butonul „OK”
3. Utilizatorul continua cu punctul 4. din Main Success Scenario



**3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)**

Acest proiect este impartit in 6 pachete. Pachetele sunt facute in asa fel incat sa respecte proiectarea Layered Architecture. Primul pachet este DatabaseProject.OrderManagement.Connection care contine o singura clasa DBConnection in care se face conexiunea cu baza de date si alte operatii cu baza de date, cum ar fi inchiderea bazei de date si inchiderea statement-ului. Conexiunea cu baza de date este facuta intr-un obiect Singleton, care inseamna ca avem un singur obiect a clasei DBConnection declarat private. Acest obiect privat face conexiunea cu baza de date si prin acest fapt garanteaza ca exista o singura conexiune cu baza de date la orice moment, deci orice modificare ce face utilizatorul are efect pe aceeasi baza de date.

Al doilea pachet este DatabaseProject.OrderManagement.Model care contine 4 clase care sunt de tipul celei patru tabele folosite, adica fiecare obiect al acestor clase salveaza, contine un rand din tabelul respectiv. Fiecare clasa are variable de instante corespunzatoare cu tipul de date folosit in tabel.

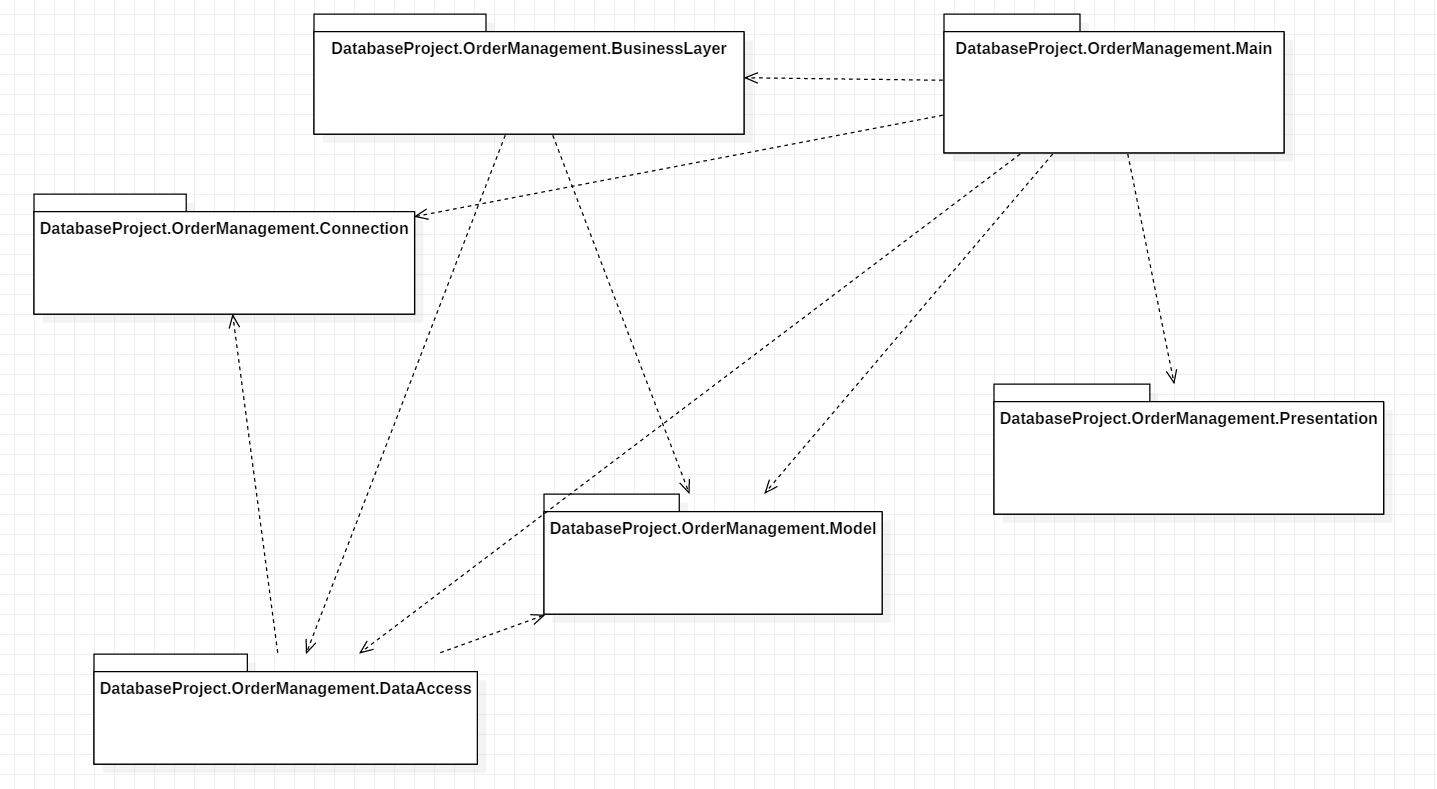
Al treilea pachet este DatabaseProject.OrderManagement.DataAccess care contine 5 clase, aceste clase au rol de a accesa baza de date si de a salva datele din baza de date in obiecte corespunzatoare. O clasa AbstractDAC contine metode care se poate aplica in general pentru fiecare tabel, iar cele 4 clase(ClientDAC, ProductDAC, CompanyOrderDAC, OrderListDAC) extind clasa AbstractDAC si contine date si metode specifice pentru fiecare tabel in parte.

Al patrulea pachet este DatabaseProject.OrderManagement.BusinessLayer care este alcatuit din 3 clase, cele 3 clase reprezinta operatiile principale a aplicatie: operatii cu clienti, operatii cu produse si plasarea unei comenzi. Aceste clase folosesc metodele si datele din pachetele Model si DataAccess. BusinessLayer-ul foloseste metodele din DataAccess pentru a accesa date din baza de date si clasele din Model pentru a salva, a stoca datele obtinute din baza de date.

Al cincilea pachet este DatabaseProject.OrderManagement.Presentation care este alcatuit din 12 clase, fiecare clasa este utilizat la interfata grafica a aplicatiei. Operatiile sunt modularizate in diferite ferestre, in diferite panel-uri, fiecare clasa facand un singur task, iar la final in clasa MainWindow sunt legate impreuna toate componentele a interfetei.

Ultimul pachet este DatabaseProject.OrderManagement.Main, acesta contine o singura clasa MainClass de unde porneste aplicatia. In aceasta clase avem metoda main() aici este declarat fereastra principala a interfetei grafice.

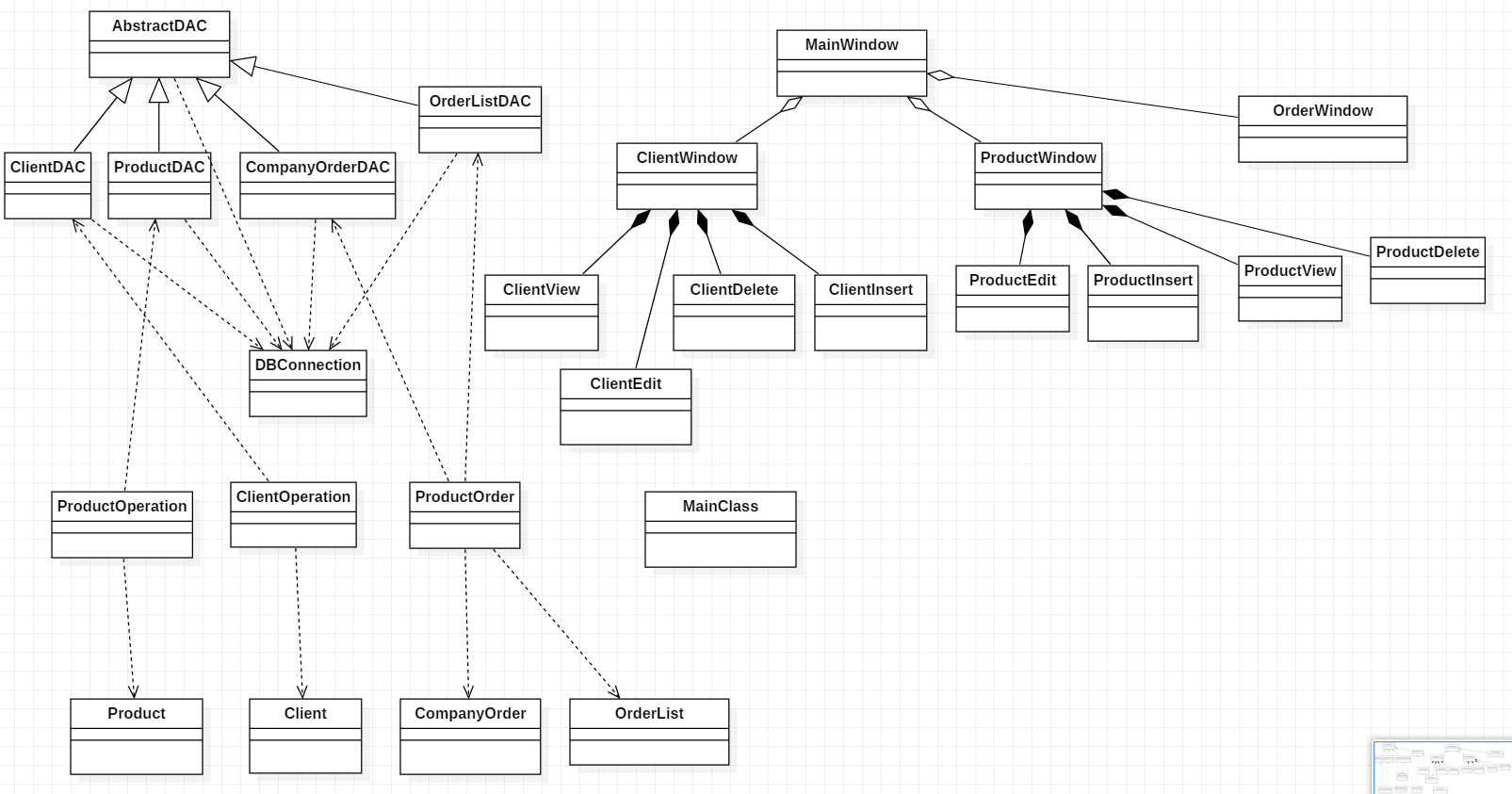
Diagrama UML de pachete arata in felul urmator:



Din figura rezulta ca pachetul Main depinde de toate celelalte pachete, ceea ce este absolut normal, fiindca fara celelalte pachete nu putem rula aplicatia, daca rulam fara un pachet nu vom avea rezultatul dorit. Un alt lucru ce rezuta de pe figura este faptul ca pachetul DataAccess depinde de Connection, pentru ca sa acceseaza datele trebuie sa faca conexiunea cu baza de date si depinde de Model, deoarece datele accesate trebuie salvate in obiecte corespunzatoare. BusinessLayer depinde de DataAcces, deoarece fara acesta nu avem datele care putem prelucra si depinde de Model pentru ca datele prelucrate sunt salvate in obiecte corespunzatoare, ceea ce gasim in pachetul Model.

Clasele respecta proiectarea OOP, prin structura si prin componente, variabile si operatii folosite. In pachetul Presentation avem pentru fiecare operatie o clasa, de exemplu: pentru aduagarea unui nou clinet avem clasa ClientInsert, pentru stergerea unui produs avem ProductDelete si pentru plasarea comenzii avem OrderWindow. In baza de date avem 4 tabele diferite, de aceea in pachetul Model vom avea 4 clase, pentru fiecare tabel o clasa si anume: Client, Product, CompanyOrder si OrderList. Clasele din BusinessLayer sunt clase cele mai importante si complexe, deoarece aici este legatura intre conexiunea cu baza de date cu datetele accesate din pachetul DataAcces si salvate in obiecte care au ca si tip clase din pachetul Model. In BusinessLayer avem 3 clase: ClientOperation in care gasim toate operatiile efectuate cu clientii, cum ar fi adaugarea, stergerea, actualizarea; ProductOperation care contine toate metodele care au efect asupra produse; ProductOrder in care se afla operatiile facute pentru a face o comanda si pentru a crea si a salva o factura.

Diagrama UML de clase este prezentat pe figura urmatoare.



**4. Implementare**

Clasa ProductOrder este o clasa foarte importanta, aici gasim toate metodele care sunt folosite la crearea facturii si la plasarea comenzii. Prima data ,cand utilizatorul vrea sa faca o noua comanda, introducem un rand in tabelul Companyorder: introducem IDul de comanda, IDul de client, tipul de plata si data in care s-a facut comanda. Acest lucru se face metoda addOrder din clasa ProductOrder:

**public** **void** addOrder(**int** ID, **int** IDClient, String payment\_type, String date) {

CompanyOrderDAC coDAC = **new** CompanyOrderDAC();

coDAC.insert(ID, IDClient, payment\_type, date);

}

Dupa aceea urmeaza actualizarea cantitatilor dupa ce comanda a fost facuta. Metoda updateStock cauta produsul dupa nume si dupa aceea actualizeaza stocul cu noua valoare calculata. Deci daca comanda contine 10 bucati dintr-un produs, atunci metoda scade 10 din cantitatea curenta si salveaza noua valoare in baza de date.

**public** **int** updateStock(**int** newValue, String name) {

ProductDAC pDAC = **new** ProductDAC();

**int** rez = pDAC.stockUpdate(newValue, name);

**return** rez;

}

Dupa aceea se face inserarea in tabelul Orderlist, care contine comanda mai detaliata dupa ID de comanda: in acest tabel apare IDul fiecarui produs comandat, pretul produsului, cantitatea si suma total care trebuie platit pentru produsul respectiv. Inserarea din clasa OrderListDAC arata in felul urmator:

**public** **void** insert(**int** ID, **int** IDProduct, **int** price, **int** quantity) {

Connection connection = **null**;

PreparedStatement statement = **null**;

**int** result = 0;

String query = insertInto();

connection = DBConnection.*getConnection*();

**try** {

statement = connection.prepareStatement(query);

statement.setInt(1, ID);

statement.setInt(2, IDProduct);

statement.setInt(3, price);

statement.setInt(4, quantity);

result = statement.executeUpdate();

} **catch** (SQLException ex) {

System.***out***.println("Error Insert OrderList");

} **finally** {

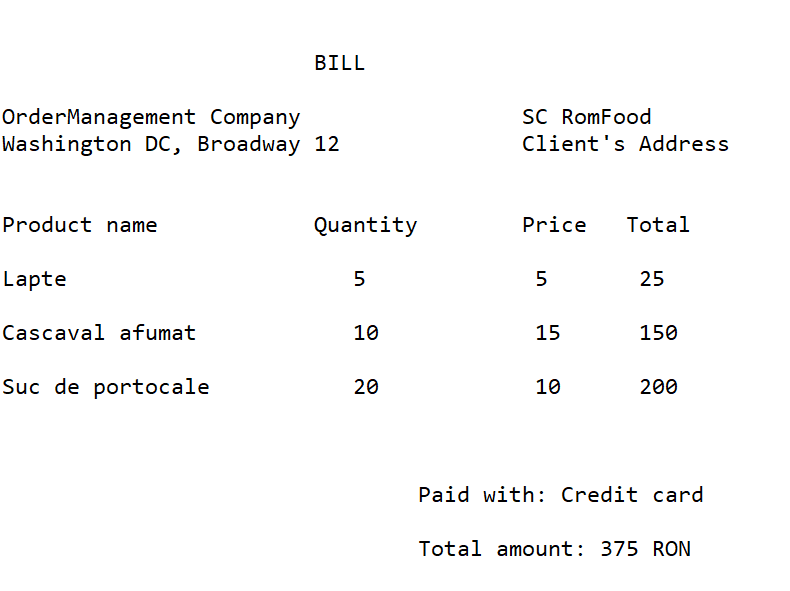
DBConnection.*close*(connection);

DBConnection.*close*(statement);

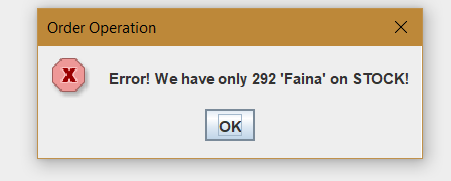
}

}

Dupa cele doua inserari are loc crearea facturii si calcularea pretului total. Factura va fi salvalta pe hard disk si este usor de gasit, deoarece numele facturii se termina in IDul de comanda, deci daca stim numarul comenzii gasim foarte repede factura respectiva. Factura care contine 3 diferite produse, cu cantitati diferite si care a fost platita cu card arata in felul urmator:



Daca utilizatorul comanda o cantitate mai mare dintr-un produs un mesaj de eroare va fi afisat, in care scrie exact cantitatea maxima din produsul respectiv, astfel ajutand pentru utilizator sa nu greseasca de doua sau de mai multe ori.



La afisarea tabelului are un rol foarte important metoda createTable care returneaza un Jtable, dar langa aceasta continutul tabelului este generat prin metoda findAll care se gaseste in clasa AbstractDAC. Datele care sunt puse in tabelul afisat sunt salvate din tabelul din baza de date. Prima data se face conexiunea cu baza de date printr-u obiect de tip Connection. Dupa aceea se executa o interogare care are sintaxa urmatoare: „SELECT \* FROM Client;”. Dupa executarea interogarii datele sunt salvate intr-un obiect de tip ResultSet, iar ca sa lucram mai usor si mai rapid cu aceste date metoda createObjects creeaza din acest obiect o lista de tipul Client, in cazul in care lucram cu tabelul Client, si metoda returneaza aceasta lista care pana la urma ajunge la metoda createTable prin care va fi creat tabelul Client. Este foarte important sa inchidem conexiunea folosita, ca sa nu avem probleme cand folosit data viitoare conexiunea respectiva. Langa conexiune inchidem si obiectele de tip PreparedStatement si ResultSet. Folosim try catch pentru ca poate sa apare o exceptie de tip SQLException, de aceea este important sa tratam si cazul in care interogarea esueaza. Metoda findAll este arata in felul urmator:

**public** List<T> findAll() {

Connection connection = **null**;

PreparedStatement statement = **null**;

ResultSet resultSet = **null**;

String query = selectAll();

connection = DBConnection.*getConnection*();

List<T> obj = **new** ArrayList<T>();

**try** {

statement = connection.prepareStatement(query);

resultSet = statement.executeQuery();

obj = createObjects(resultSet);

} **catch** (SQLException e) {

System.***out***.println("Error findAll");

} **finally** {

DBConnection.*close*(connection);

DBConnection.*close*(statement);

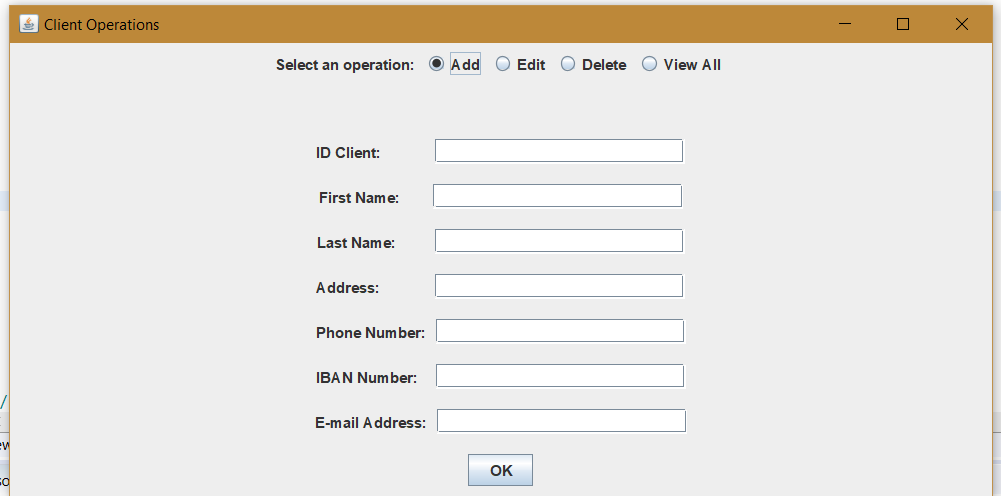
DBConnection.*close*(resultSet);

}

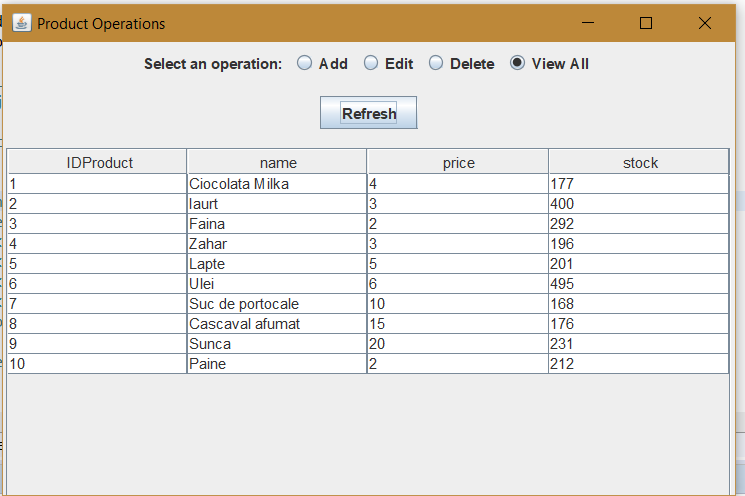
**return** obj;

}

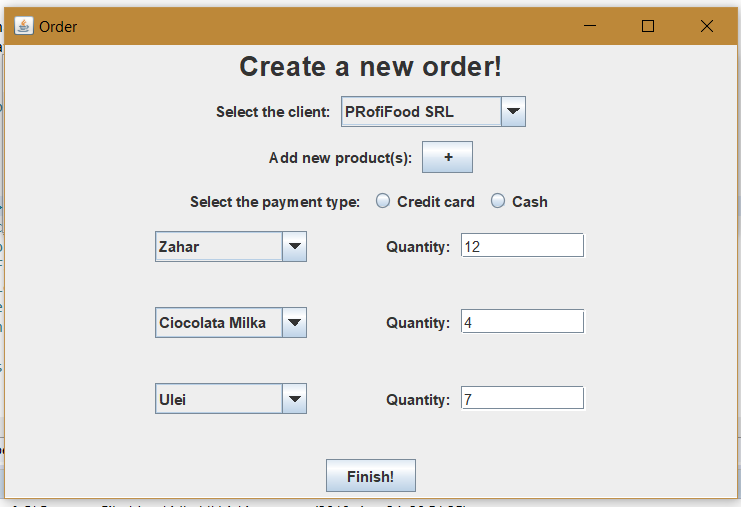
Interfata grafica este alcatuita din 12 clase, dintre care avem 4 clase importante. Clasa MainWindow leaga toate componentele grafice impreuna, aceasta fiind fereastra principala. Pornind de aceasta fereastra putem sa ajungem la 3 ferestre diferite: ClientWindow, ProductWindow si OrderWindow. ClientWindow contine toate operatiile cu clientii, ProductWindow contine toate operatiile cu produse, iar pe OrderWindow este plasat comanda. Cele trei ferestre arata in felul urmator:



ClientWindow, adaugare client nou



ProductWindow, afisarea tabelului Product



OrderWindow, comanda cu 3 produse, fara alegerea tipul de plata

**5. Concluzii**

Aceasta aplicatie este foarte folositoare, mai ales daca utilizatorii sunt persoane care lucreaza la o companie care vinde multe lucruri, o companie care are clienti si produse si mai multe date care trebuie salvate intr-o baza de date. Aplicatie reprezinta o versiune mai simpla a unor aplicatii pe care si companiile mai mari le folosesc. Aceasta tema a fost folositoare deoarece am recapitulat notiunile legate de baze de date, am invatat cum se face conexiunea cu o baza de date din Java, cum se lucreaza cu fisiere si cum putem sa cream un tabel.

Aceast aplicatie se poate dezvolta cu foarte mult, se mai pot adauga si alte operatii, de exemplu cautarea si afisarea clientilor dupa adresa sau dupa ID de client, putem sa adaugam mai multe tabele, putem sa dezvoltam cu parte de logistica, adica sa proiectam cum vor fi livrate comenzile la adresa clientilor.

**6. Bibliografie**

* Pentru diagrame StarUML
* Tema3\_HW3\_Indications
* https://stackoverflow.com/questions/2839321/connect-java-to-a-mysql-database
* https://www.javatpoint.com/example-to-connect-to-the-mysql-database
* https://www.geeksforgeeks.org/singleton-class-java/
* https://docs.oracle.com/javase/7/docs/api/javax/swing/JTable.html